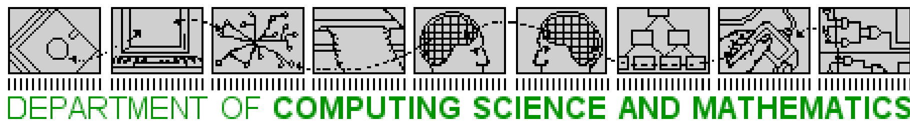
AN INVESTIGATION OF FACTORS INFLUENCING ALGORITHM SELECTION FOR HIGH DIMENSIONAL CONTINUOUS OPTIMISATION PROBLEMS

KEVIN GRAHAM



Doctor of Philosophy

Institute of Computing Science and Mathematics
University of Stirling

November 2019

DECLARATION

I, Kevin Graham, hereby declare that all material in this thesis is my own original work except where reference has been given to other authors. The work presented here has not been submitted for the award or partial award of any other degree at the University of Stirling nor any other institute.

Stirling, November 2019

Kevin Graham

ABSTRACT

The problem of algorithm selection is of great importance to the optimisation community, with a number of publications present in the Body-of-Knowledge. This importance stems from the consequences of the No-Free-Lunch Theorem which states that there cannot exist a single algorithm capable of solving all possible problems. However, despite this importance, the algorithm selection problem has of yet failed to gain widespread attention . In particular, little to no work in this area has been carried out with a focus on large-scale optimisation; a field quickly gaining momentum in line with advancements and influence of big data processing. As such, it is not as yet clear as to what factors, if any, influence the selection of algorithms for very high-dimensional problems (> 1000) - and it is entirely possible that algorithms that may not work well in lower dimensions may in fact work well in much higher dimensional spaces and vice-versa. This work therefore aims to begin addressing this knowledge gap by investigating some of these influencing factors for some common metaheuristic variants.

To this end, typical parameters native to several metaheuristic algorithms are firstly tuned using the state-of-the-art automatic parameter tuner, SMAC. Tuning produces separate parameter configurations of each metaheuristic for each of a set of continuous benchmark functions; specifically, for every algorithm-function pairing, configurations are found for each dimensionality of the function from a geometrically increasing scale (from 2 to 1500 dimensions). The nature of this tuning is therefore highly computationally expensive necessitating the use of SMAC. Using these sets of parameter configurations, a vast amount of performance data relating to the large-scale optimisation of our benchmark suite by each metaheuristic was subsequently generated.

From the generated data and its analysis, several behaviours presented by the metaheuristics as applied to large-scale optimisation have been identified and discussed. Further, this thesis provides a concise review of the relevant literature for the consumption of other researchers looking to progress in this area in addition to the large volume of data produced, relevant to the large-scale optimisation of our benchmark suite by the applied set of common metaheuristics.

All work presented in this thesis was funded by EPSRC grant: EP/J017515/1 through the DAASE project [1].

1 http://daase.cs.ucl.ac.uk/

ACKNOWLEDGMENTS

"Let not your mind run on what you lack as much as on what you already have"
– *Marcus Aurelius*

"Why do we fall, Bruce? So we can learn to pick ourselves up"
– *Thomas Wayne (Batman Begins 2005)*

I am dedicating this thesis to my loving wife Tracy, my sons Carson and Callan and my dad Michael Graham.

My dad sadly passed away from an asbestos related disease at the beginning of my 2nd year - a particularly difficult time for family and myself. Of all the occasions when things got tough during the course of my studies, one of the things that kept me going was his pride that I had gotten to where I was; not through luck or innate ability, but like he did his whole life - through hard work and dedication. Being a family-centric man, I still believe that hard work on its own could not have brought me to this point, but the love and support from my dad and my family throughout my life. We all miss his presence, humour and love but his memory will live on through his wife - my mum, sons and grandchildren for years to come.

My PhD years were not all filled with grief however, I married my wife Tracy and 3 years later we had our first son, Carson Michael Graham. I simply would not be here writing this now if not for the unconditional love and support Tracy has shown me over the years - extending far beyond PhD studies. Whenever I needed to vent, talk things over or was in dire need of mental support I could always count on her to be there for me - even when I needed to be 'rescued' from Nottingham University when I fell ill with severe appendicitis. I really could not have done any of this were it not for her.

I make no secret of the fact about how many times - and how close I came - to throwing in the towel. The illogical side of my psyche had me convinced that the PhD was 'cursed' - I had lost multiple supervisors (through work opportunities), family members, had several failed/abandoned projects, suffered severe illness and was getting nowhere fast despite my efforts. That's until my son Carson was born, and then my son Callan Thomas Graham just short of 2 years later. Every day since then I vowed that I would show them how to not give

up, to keep fighting through obstacles in your way in order to become the best version of yourself and that getting knocked down only teaches us how to get back up - I didn't want their first lesson in life to be to stop moving forward.

And so my sons, this is for you, your mummy and your grandad - Always remember that you are braver than you believe, stronger than you seem, smarter than you think and more loved than you'll ever know.

Finally, I would like to thank my current supervisors Prof. Leslie Smith, Dr. Alexander (Sandy) Brownlee and Dr. David Cairns for all their help and support over the last two years - without whom I could not have reached this point. Also, I'd like to give thanks to a former supervisor - Dr John Woodward, who never gave up trying to organise me, even up to the day he took up his new appointment. Also, I'd like to thank my fellow PhD students for all the years of interesting discussion and laughter. In particular, I'd like to thank Jason Adair (now Dr. Adair) for all the help and support (both personally and work-based), chats and putting up with my long venting sessions on the not-so-rare occasion - and of whom will always be considered as a friend. Of course, I cannot forget my mum who has always shown me love and support throughout my life and who taught ME that getting knocked down only teaches us how to get back up.

LIST OF PUBLICATIONS

[42] Kevin Graham, Jerry Swan, and Simon Martin. The Blackboard Pattern for Metaheuristics. *In Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1265-1267, 2015.

[41] Kevin Graham and Leslie Smith. Comparing Hyper-heuristics with Blackboard Systems. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1141-1145, 2017.

The publications listed here are related to previous research directions no longer being followed and are therefore not relevant to this thesis.

CONTENTS

i INTRODUCTION 1

1 INTRODUCTION TO THE THESIS 2

- 1.1 Introduction . . . 2
- 1.2 Structure of the Thesis . . . 3

ii BACKGROUND & LITERATURE REVIEW 5

2 CHAPTER 2: OPTIMISATION 6

- 2.1 Introduction . . . 6
- 2.2 General Optimisation . . . 7
  - 2.2.1 Terminology . . . 7
  - 2.2.2 A Formal Definition of an Optimisation Problem . . . 9
  - 2.2.3 Problem Difficulty . . . 9
- 2.3 Classical Optimisation Methods . . . 11
  - 2.3.1 Strong Methods . . . 12
  - 2.3.2 Weak Methods . . . 16
- 2.4 Metaheuristic Optimisation . . . 17
  - 2.4.1 What is a Metaheuristic? . . . 17
  - 2.4.2 Exploration vs. Exploitation in Metaheuristics . . . 19
  - 2.4.3 Problems Addressed by Metaheuristics . . . 20
  - 2.4.4 Categorisations of Metaheuristics . . . 20
  - 2.4.5 General Concepts . . . 22
- 2.5 Large-Scale Global Optimisation . . . 31
  - 2.5.1 Cooperative Co-Evolutionary Algorithm . . . 34

3 CHAPTER 3: CONTINUOUS OPTIMISATION BENCHMARK FUNCTIONS 42

- 3.1 Introduction . . . 42
- 3.2 What Makes a Function Difficult to Optimise . . . 42
- 3.3 Constructing a Custom Function Suite Versus Using Existing Benchmark Function Suites . . . 46
- 3.4 Functions . . . 48
  - 3.4.1 Summary of Functions . . . 49

4 CHAPTER 4: METAHEURISTIC APPROACHES USED IN THIS THESIS 50

4.1 Introduction 50
4.1.1 Exposing Algorithm Parameters 51
4.1.2 Solution Representation 52
4.1.3 Neighbourhood Function 52
4.2 Hill Climbing Algorithms 54
4.2.1 Common Variations 55
4.2.2 Global Optimisation with Hill Climbing 58
4.2.3 Random Mutation Hill Climb: Implementation 61
4.3 Genetic Algorithms (GA) 65
4.3.1 The Basic GA Procedure 67
4.3.2 Genetic Operators 69
4.3.3 Steady State GA vs. Generational GA 78
4.3.4 Steady State Genetic Algorithm: Implementation 79
4.4 Particle Swarm Optimisation (PSO) 85
4.4.1 Neighbourhood Topologies 85
4.4.2 Parameters of PSO 87
4.4.3 Boundary-Handling in PSO 90
4.4.4 Particle Swarm Optimisation (PSO): Implementation 92
4.5 Simulated Annealing (SA) 98
4.5.1 Annealing in Condensed Matter Physics 99
4.5.2 Acceptance Probability 99
4.5.3 Cooling Schedules 100
4.5.4 Simulated Annealing (SA): Implementation 101
4.6 Differential Evolution (DE) 107
4.6.1 The Differential Evolution Process 107
4.6.2 Variants of DE 110
4.6.3 Differential Evolution (DE): Implementation 110
4.7 Covarience Matrix Adaption Evolutionary Strategy (CMA-ES) 115
4.7.1 Algorithm Overview 115
4.7.2 CMA-ES for Large-scale Optimisation 116
4.7.3 Covariance Matrix Adaption Evolution Strategy (CMA-ES): Implementation 117
5 CHAPTER 5: THE PROBLEMS OF ALGORITHM SELECTION AND CONFIGURATION 120
5.1 Introduction 120
5.2 Relationships Between Algorithm Selection and Other Areas of Research 121
5.2.1 Meta-learning 121

5.3 Automatic Parameter Tuning . . . 125
5.3.1 Model-free vs. Model-based Methods . . . 126
5.4 SMAC: Sequential Model-based Algorithm Configuration . . . 129
5.4.1 The SMAC Tuning Procedure . . . 129
iii METHOD 132
6 CHAPTER 6: ALGORITHM PARAMETER CONFIGURATION 133
6.1 Introduction . . . 133
6.2 Parameter Tuning Method . . . 134
6.2.1 Parameter Identification and Range Selection . . . 135
6.2.2 Conducting the Tuning Procedures . . . 136
6.2.3 Generation of Tuned Performance Data . . . 136
6.2.4 Obtaining a Final Parameter Configuration Set . . . 137
6.3 Improving the Generality of Empirical Results and Observations . . . 138
6.4 Result of Parameter Tuning with SMAC . . . 139
iv RESULTS AND CONCLUSIONS 141
7 CHAPTER 7: EMPIRICAL RESULTS 142
7.1 Algorithm Tuning Results . . . 142
7.1.1 Comparing SMAC and Irace Tuning Performance . . . 143
7.2 Function Evaluation Budget Comparison . . . 145
7.2.1 Effects Observed when Tuning with 10,000 and 50,000 Function Evaluation Budgets . . . 145
7.3 Performance Comparison Between Algorithm Implementations . . . 154
7.3.1 Random Mutation Hill Climbing (RMHC) . . . 154
7.3.2 Changes to Rank Ordering of Performance at Larger Scales . . . 155
7.4 Conclusions and Discussion . . . 164
7.4.1 Tuning Evaluation Budget Comparisons . . . 164
7.4.2 Performance Comparison Between Algorithm Implementations . . . 166
7.4.3 Rank Ordering Changes at Larger Problem Scales . . . 166
7.5 Contributions . . . 166
7.6 Future Work . . . 167
V APPENDICES 1
A APPENDIX A: 10,000 AND 50,000 EVALUATION COMPARISON PLOTS 2
B APPENDIX B: 10K & 50K EVALUATION TUNING BUDGET DESCRIPTIVE STATISTICS 32

C APPENDIX C: SMALL TO LARGE-SCALE METAHEURISTIC COMPARISON MATERIAL 51
C.1 Metaheuristic Comparison Plots . . . 52
C.1.1 Comparison of Approaches Tuned at the 10,000 Evaluation Budget (With Q1 & Q3 Error Bars) . . . 52
C.1.2 Comparison of Approaches Tuned at the 10,000 Evaluation Budget (No Error Bars) . . . 55
C.1.3 Comparison of Approaches Tuned at the 50,000 Evaluation Budget (With Q1 & Q3 Error Bars) . . . 58
C.1.4 Comparison of Approaches Tuned at the 50,000 Evaluation Budget (No Error Bars) . . . 61
C.2 Graphical Metaheuristic Rankings . . . 64
C.2.1 Rank Ordering By Benchmark Function . . . 64
C.2.2 Rank Ordering: Mean Aggregate and Pooled Mean Over Subsets of the Benchmark Suite . . . 67
C.3 Algorithm Ranking Tables . . . 69
C.3.1 Algorithm Rank Ordering By Benchmark Function . . . 69
C.3.2 Mean Aggregated Algorithm Rank Ordering By All Problems and Problems Classified by Problem Feature Investigated . . . 79
D APPENDIX D: PLOTS SHOWING THE RESULTS OF TUNING EACH ALGORITHM AT A 10,000 AND 50,000 EVALUATION BUDGET COMPARED TO THE CORRESPONDING UNTUNED ALGORITHM 83
D.1 SMAC Parameter Tuning Results . . . 84
D.2 Irace vs. SMAC Parameter Tuning Results . . . 114
E APPENDIX E: DEFINITIONS OF BENCHMARK FUNCTIONS IN THE TEST SUITE 132
E.1 Ackley Function no.1 . . . 132
E.2 Alpine Function no.1 . . . 132
E.3 Bent Cigar Function . . . 133
E.4 Brown Function . . . 134
E.5 Chung-Reynolds Function . . . 135
E.6 Deflected Corrugated Spring Function . . . 136
E.7 Exponential Function . . . 137
E.8 Griewank Function . . . 138
E.9 Inverted Cosine Wave Function . . . 139
E.10 Levy Function . . . 140
E.11 Qing Function . . . 142

E.12 Rastrigin Function 143
E.13 Rosenbrock Function 144
E.14 Schwefel Function no.26 146
E.15 Sphere Function 147
E.16 Sum of Different Powers Function 148
E.17 Sum Squares Function 148
F SUPPLEMENTARY EXPERIMENT MATERIAL 150
F.1 Extended PSO Experiment Results 150

LIST OF FIGURES

Figure 3.1 Barchart Summarising Prevalence of Problems With Certain Features Present in Each Benchmark Test Suite 47
Figure 4.1 Effects of inappropriate fixed step sizes 53
Figure 4.2 Definition of a neighbourhood of a solution sol using the $stepsize$ parameter and the stochastic generation of a neighbouring solution sol_2 64
Figure 4.3 Basic operation of a typical genetic algorithm implementation 68
Figure 4.4 Roulette Wheel Selection example showing a single selection over a population of four individuals. Each individual, A,B,C and D represent a 32.7%, 12.7%, 3.6% and 51% portion of the roulette wheel respectively. Random point r selects the portion representing chromosome D similar to a hypothetical roulette ball 70
Figure 4.5 Tournament selection, where t = 8 (tournament size) 72
Figure 4.6 Uniform Crossover 73
Figure 4.7 One-point Crossover 73
Figure 4.8 Uniform Mutation: where x is transformed to x' 75
Figure 4.9 Velocity and Position Update in PSO 88
Figure 4.10 Production of a mutant vector $\vec{V}_i$ from the scaled difference vector $(\vec{X}_{r_2} - \vec{X}_{r_3})$ 109
Figure 5.1 Taken from [58], This figure shows two steps of SMBO (EGO) for the optimization of a 1D function. The true function is shown as a solid line, and the circles denote our observations. The dotted line represents the mean prediction of a noise-free Gaussian process model (the DACE model), with the grey area representing its uncertainty. Expected improvement, EI, (scaled by the authors for visualisation) is shown by a dashed line. 128
Figure 6.1 Example of a parameter configuration file compiled from the individual problem-dimension configurations obtained from SMAC 137
Figure 6.2 Result Obtained from DE Implementation (tuning at 50,000 function evaluations) vs. Untuned Version 140
Figure 7.1 Examples of Successful Tunings of RMHC and SSGA Using SMAC on Ackleys Function and Alpine n.1 143

Figure 7.2 Example of GA Continuing to Progress Beyond 10,000 Function Evaluations for Ackley's Function . . . 147
Figure 7.3 SSGA Performance Results Comparison (Irace Tuned) Example - 10,000 and 50,000 Tuning Evaluation Budgets . . . 148
Figure 7.4 SSGA: Divergence of Performance between 10,000 and 50,000 Tuning Evaluation Budgets of SMAC and Irace . . . 149
Figure 7.5 Example showing divergence between parameter tuning evaluation budgets for PSO (Alpine n.1 Function) at dimensions 11-16 and re-convergence occuring at dimension 128 . . . 150
Figure 7.6 Plots showing individual values for each tuned parameter from our original 5 parameter configurations sets obtained from SMAC . . . 151
Figure 7.7 Comparison of Performance (Median Solution Quality of 20 Independent Repeats) Against the Chung-Reynolds Function Between Original and Extended Scale PSO Experiments . . . 153
Figure 7.8 Mean Aggregate Ranking for Each Dimensionality and Pooled Mean Rank Over Every n Data Points Over All Benchmark Functions (where $n = 5$) . . . 157
Figure 7.9 Mean Aggregate Ranking for Each Dimensionality and Pooled Mean Rank Over Every n Data Points Over All Uni-modal Functions (where $n = 5$) . . . 159
Figure 7.10 Mean Aggregate Ranking for Each Dimensionality and Pooled Mean Rank Over Every n Data Points Over All Multi-modal Functions (where $n = 5$) . . . 159
Figure 7.11 Mean Aggregate Ranking for Each Dimensionality and Pooled Mean Rank Over Every n Data Points Over All Separable Functions (where $n = 5$) . . . 162
Figure 7.12 Mean Aggregate Ranking for Each Dimensionality and Pooled Mean Rank Over Every n Data Points Over All Non-separable Functions (where $n = 5$) . . . 162
Figure E.1 Ackley Function no.1 in its 2-dimensional form . . . 132
Figure E.2 Alpine Function no.1 in its 2-dimensional form . . . 133
Figure E.3 Bent Cigar Function in its 2-dimensional form . . . 134
Figure E.4 Brown Function in its 2-dimensional form: (a) and (b) shows the function's full domain, and (c) and (d) shows the function with domain limited to $-1 \leqslant x_i \leqslant 1$ . . . 135
Figure E.5 Chung-Reynolds Function in its 2-dimensional form . . . 136

Figure E.6 Deflected Corrugated Spring Function in its 2-dimensional form 137
Figure E.7 Exponential Function in its 2-dimensional form 138
Figure E.8 Griewank Function in its 2-dimensional form 139
Figure E.9 Inverted Cosine Wave Function in its 2-dimensional form 140
Figure E.10 Levy Function in its 2-dimensional form 141
Figure E.11 Qing Function in its 2-dimensional form: (a) and (b) shows the function's full domain, and (c) and (d) shows the function with domain limited to $-2 \leqslant x_i \leqslant 2$ 143
Figure E.12 Rastrigin Function in its 2-dimensional form 144
Figure E.13 Rosenbrock Function in its 2-dimensional form: (a) and (b) shows the function's full domain, and (c) and (d) shows the function with domain limited to $-2.5 \leqslant x_i \leqslant 2.5$ 145
Figure E.14 Schwefel Function in its 2-dimensional form 146
Figure E.15 Sphere Function in its 2-dimensional form 147
Figure E.16 Sum of Different Powers Function in its 2-dimensional form 148
Figure E.17 Sum Squares Function in its 2-dimensional form 149

LIST OF TABLES

Table 3.1 Summary of the Proportion of Problems With Certain Features Present in Various Existing Test Suites Versus the Test Suite Constructed for this Study . . . 47
Table 3.2 Summary of n-Dimensional Benchmark Function Suite . . . 49
Table 7.1 Plot summary of problem dimensionalities where algorithm performance between 10k and 50k evaluations diverge in favour of tuning at 50k function evaluations . . . 146
Table 7.2 Pooled Mean Rank Table Over All Functions . . . 157
Table 7.3 Table of Pooled Mean Ranks Over All Uni-modal Functions . . . 160
Table 7.4 Table of Pooled Mean Ranks Over All Multi-modal Functions . . . 160
Table 7.5 Table of Pooled Mean Ranks Over All Separable Functions . . . 163
Table 7.6 Table of Pooled Mean Ranks Over All Non-separable Functions . . . 163
Table B.1 sep-CMA-ES Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (10,000 Evaluation Tuning) . . . 33
Table B.2 sep-CMA-ES Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (50,000 Evaluation Tuning) . . . 33
Table B.3 sep-CMA-ES Descriptive Statistics: Exponential to Rastrigin Functions (10,000 Evaluation Tuning) . . . 34
Table B.4 sep-CMA-ES Descriptive Statistics: Exponential to Rastrigin Functions (50,000 Evaluation Tuning) . . . 34
Table B.5 sep-CMA-ES Descriptive Statistics: Rosenbrock to Sum Squares Functions (10,000 Evaluation Tuning) . . . 35
Table B.6 sep-CMA-ES Descriptive Statistics: Rosenbrock to Sum Squares Functions (50,000 Evaluation Tuning) . . . 35
Table B.7 DE Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (10,000 Evaluation Tuning) . . . 36
Table B.8 DE Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (50,000 Evaluation Tuning) . . . 36
Table B.9 DE Descriptive Statistics: Exponential to Rastrigin Functions (10,000 Evaluation Tuning) . . . 37

Table B.10 DE Descriptive Statistics: Exponential to Rastrigin Functions (50,000 Evaluation Tuning) 37
Table B.11 DE Descriptive Statistics: Rosenbrock to Sum Squares Functions (10,000 Evaluation Tuning) 38
Table B.12 DE Descriptive Statistics: Rosenbrock to Sum Squares Functions (50,000 Evaluation Tuning) 38
Table B.13 GA Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (10,000 Evaluation Tuning) 39
Table B.14 GA Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (50,000 Evaluation Tuning) 39
Table B.15 GA Descriptive Statistics: Exponential to Rastrigin Functions (10,000 Evaluation Tuning) 40
Table B.16 GA Descriptive Statistics: Exponential to Rastrigin Functions (50,000 Evaluation Tuning) 40
Table B.17 GA Descriptive Statistics: Rosenbrock to Sum Squares Functions (10,000 Evaluation Tuning) 41
Table B.18 GA Descriptive Statistics: Rosenbrock to Sum Squares Functions (50,000 Evaluation Tuning) 41
Table B.19 PSO Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (10,000 Evaluation Tuning) 42
Table B.20 PSO Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (50,000 Evaluation Tuning) 42
Table B.21 PSO Descriptive Statistics: Exponential to Rastrigin Functions (10,000 Evaluation Tuning) 43
Table B.22 PSO Descriptive Statistics: Exponential to Rastrigin Functions (50,000 Evaluation Tuning) 43
Table B.23 PSO Descriptive Statistics: Rosenbrock to Sum Squares Functions (10,000 Evaluation Tuning) 44
Table B.24 PSO Descriptive Statistics: Rosenbrock to Sum Squares Functions (50,000 Evaluation Tuning) 44
Table B.25 RMHC Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (10,000 Evaluation Tuning) 45
Table B.26 RMHC Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (50,000 Evaluation Tuning) 45
Table B.27 RMHC Descriptive Statistics: Exponential to Rastrigin Functions (10,000 Evaluation Tuning) 46

Table B.28 RMHC Descriptive Statistics: Exponential to Rastrigin Functions (50,000 Evaluation Tuning) . . . 46
Table B.29 RMHC Descriptive Statistics: Rosenbrock to Sum Squares Function (10,000 Evaluation Tuning) . . . 47
Table B.30 RMHC Descriptive Statistics: Rosenbrock to Sum Squares Function (50,000 Evaluation Tuning) . . . 47
Table B.31 SA Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (10,000 Evaluation Tuning) . . . 48
Table B.32 SA Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (50,000 Evaluation Tuning) . . . 48
Table B.33 SA Descriptive Statistics: Exponential to Rastrigin Functions (10,000 Evaluation Tuning) . . . 49
Table B.34 SA Descriptive Statistics: Exponential to Rastrigin Functions (50,000 Evaluation Tuning) . . . 49
Table B.35 SA Descriptive Statistics: Rosenbrock to Sum Squares Functions (10,000 Evaluation Tuning) . . . 50
Table B.36 SA Descriptive Statistics: Rosenbrock to Sum Squares Functions (50,000 Evaluation Tuning) . . . 50
Table C.1 Algorithm Rankings for Ackley Function over 2 to 1448 Dimensions (1=Best, 5=Worst) . . . 70
Table C.2 Algorithm Rankings for Alpine n.1 Function over 2 to 1448 Dimensions (1=Best, 5=Worst) . . . 70
Table C.3 Algorithm Rankings for Bent Cigar Function over 2 to 1448 Dimensions (1=Best, 5=Worst) . . . 71
Table C.4 Algorithm Rankings for Brown Function over 2 to 1448 Dimensions (1=Best, 5=Worst) . . . 71
Table C.5 Algorithm Rankings for Chung-Reynolds Function over 2 to 1448 Dimensions (1=Best, 5=Worst) . . . 72
Table C.6 Algorithm Rankings for Deflected Corrugated Spring Function over 2 to 1448 Dimensions (1=Best, 5=Worst) . . . 72
Table C.7 Algorithm Rankings for Exponential Function over 2 to 1448 Dimensions (1=Best, 5=Worst) . . . 73
Table C.8 Algorithm Rankings for Griewank Function over 2 to 1448 Dimensions (1=Best, 5=Worst) . . . 73
Table C.9 Algorithm Rankings for Inverted Cosine Wave Function over 2 to 1448 Dimensions (1=Best, 5=Worst) . . . 74

Table C.10 Algorithm Rankings for Levy Function over 2 to 1448 Dimensions (1=Best, 5=Worst) 74
Table C.11 Algorithm Rankings for Qing Function over 2 to 1448 Dimensions (1=Best, 5=Worst) 75
Table C.12 Algorithm Rankings for Rastrigin Function over 2 to 1448 Dimensions (1=Best, 5=Worst) 75
Table C.13 Algorithm Rankings for Rosenbrock Function over 2 to 1448 Dimensions (1=Best, 5=Worst) 76
Table C.14 Algorithm Rankings for Schwefel Function over 2 to 1448 Dimensions (1=Best, 5=Worst) 76
Table C.15 Algorithm Rankings for Sphere Function over 2 to 1448 Dimensions (1=Best, 5=Worst) 77
Table C.16 Algorithm Rankings for Sum of Different Powers Function over 2 to 1448 Dimensions (1=Best, 5=Worst) 77
Table C.17 Algorithm Rankings for Sum Squares Function over 2 to 1448 Dimensions (1=Best, 5=Worst) 78
Table C.18 Mean Aggregated Algorithm Rankings Over All Benchmark Functions 80
Table C.19 Mean Aggregated Algorithm Rankings Over All Uni-modal Functions 80
Table C.20 Mean Aggregated Algorithm Rankings Over All Multi-modal Functions 81
Table C.21 Mean Aggregated Algorithm Rankings Over All Separable Functions . 81
Table C.22 Mean Aggregated Algorithm Rankings Over All Non-separable Functions 82

LIST OF ACRONYMS

acp Algorithm Configuration Problem

apt Automatic Parameter Tuning

asp Automatic Selection Problem

bcd Binary Coded Decimal

ccd Central Composite Design

cdiw Chaotic Descending Inertial Weight

cma-es Covariance Matrix Adaption - Evolution Strategy

criw Chaotic Random Inertial Weight

de Differential Evolution

ga Genetic Algorithm

gga Generational Genetic Algorithm

ils Iterated Local Search

nfl no free lunch (theorem)

pso Particle Swarm Optimisation

psosa Particle Swarm Optimisation Simulated Annealing

pso-tvac Particle Swarm Optimisation - Time Varying Acceleration Coefficients

rmhc Random Mutation Hill Climbing

rws Roulette Wheel Selection

sa Simulated Annealing

sga Simple Genetic Algorithm

shc Stochastic Hill Climbing

smac Sequential Model-based Algorithm Configuration

ssga Steady-State Genetic Algorithm

sus Stochastic Universal Sampling

vns Variable Neighbourhood Search

Part I

INTRODUCTION

1

INTRODUCTION TO THE THESIS

1.1 INTRODUCTION

Human beings are a species of optimisers. Throughout history, advances in civilisation can be said to have been built, almost solely, on our ability to optimise the various processes essential to our survival. From our early ancestors gradually improving on their tools for hunting or gathering to developing more efficient means to automatically mass produce items of necessity during the industrial period, we owe much of our current, and relatively comfortable, state of being to our insatiable desire to 'find something better'.

As our mathematics has developed, an ever increasing number of processes of increasing complexity are being modelled. Ever more complex models from the physical sciences, natural sciences and human sciences have provided us with new and interesting insights and new optimisation problems to solve. From molecular biology, for example, we have the problem of protein folding...

However, many of these real world problems can have very large search spaces; that is, the number of possible solutions to the problem can be incredibly large - increasing exponentially with the dimensionality of the problem. This makes it difficult for more traditional and exact optimisation methods, e.g., exhaustive search, to effectively search through all these possible solutions in a time that is acceptable. In terms of optimisation in continuous spaces; that is, where the problem variables are from the set of real numbers, these traditional approaches often make use of function derivatives. However, many continuous problems have search spaces that are inherently non-differentiable and thus inhibiting the use of these strategies.

For exactly these reasons, heuristic based optimisation methods - able to effectively search within arbitrarily large search spaces in a reasonable length of time - have become increasingly popular with several thousand associated publications even in the last year alone (2017). However, a fundamental theorem of any search based optimisation strategy - the No Free Lunch Theorem (NFL) - states that no single search algorithm, heuristic or otherwise, can be shown to exhibit higher than average performance when all possible problems are considered.

The consequences of the NFL therefore pose a problem to the users and developers of search based optimisation algorithms; specifically, what algorithm or algorithms will perform best on a given problem or set of problems of a certain type?

This problem, dubbed the *Algorithm Selection Problem* (ASP) by Rice [112] in 1975, has been of some interest to researchers however despite its importance it has failed to gain widespread attention in the research community. Furthermore, of the works that do exist, none were found to address the issue of algorithm selection for large-scale optimisation problems - those of very high dimensionality. This is an important omission, and one we attempt to begin addressing - at least partially - because the exponential increase in search space volume that invariably results from increases to problem dimensionality means that the effectiveness - at least in terms of solution quality - of all search algorithms, including heuristic algorithms, diminishes with dimensionality. Some strategies may however remain reasonably effective on certain problems at higher dimensions - where their rate of decreasing performance with dimensionality is slower - but no data is yet available to even attempt to make such discriminations.

1.2 STRUCTURE OF THE THESIS

We begin in Chapter 2 our review of the available subject matter. In this chapter we cover the subject of optimisation along with descriptions of some more 'classical' algorithmic approaches to addressing continuous optimisation problems.

In Chapter 3, we provide a description of our selected suite of 17 benchmark functions used throughout the experiments in this thesis. Along with the descriptions, we also provide visual (2D) representations of each function we generated through sampling of the functions implemented. A review of the metaheuristic search-based optimisation algorithms being used in these studies is then provided in Chapter 4.

We then briefly cover both the problem of algorithm selection and automatic parameter tuning (APT), in particular of the former - our selected tuning method, Sequential Model-based Algorithm Configuration (SMAC) in Chapter 5.

Beginning our method part with Chapter 6, attention is firstly given to the metaheuristic implementations developed for use in our experiments, including, for each approach, a list

and discussion of the various ‘tunable parameters’ configured by our selected APT described in Chapter 5. To supplement this description, and to provide more specificity to our use of SMAC, Chapter 7 provides a brief description of the steps taken to tune each metaheuristic for each of our benchmark functions at each dimensionality.

Finally, in Chapter 8, we blend our experimental methodology with individual findings and discussions and for each, provide our conclusions. These conclusions are then summarised at the end of the chapter along with a discussion of future work directions.

Part II

BACKGROUND & LITERATURE REVIEW

2

CHAPTER 2: OPTIMISATION

2.1 INTRODUCTION

The task of optimisation has become one of the main staples of modern day science, finding utility in almost every scientific field of enquiry. In fact, Schwefel in [118] states:

> "There is scarcely a modern journal, whether of engineering, economics or the social sciences, in which the concept 'optimization' is missing from the subject index."

Given our long relationship with optimisation and its many successes, having been paramount to our continued advancement as a species for thousands of years, this is not too surprising.

However, as problems have become increasingly more complex, compounded with the fact that in many cases we do not know the mathematical formulation of our problem, we are less able to rely on some of our more 'classical' or exact methods to optimisation. This has led to the popularisation of stochastic optimisation methods, specifically those techniques such as metaheuristics and hyper-heuristics, that are more able to deal with these intractable problems in a reasonable time scale albeit with no guarantee of exact optimality. Therefore, metaheuristics and other stochastic approaches have their place in situations where a 'good enough' solution is acceptable or no known exact algorithm can find the optimal solutions in reasonable runtimes.

This chapter begins by discussing some of the background and terminology of optimisation in general, including: a definition of optimisation, an overview of how problem difficulty is defined and what this means to optimisation on the whole and a comparison of the main categories of optimisation: discrete vs. continuous and local vs. global optimisation. Next we present an overview of some of the more classical and exact methods to optimisation and how these approaches are less able to perform efficiently when problems being approached become larger and more complex. This leads us give a relatively concise discussion on the topic of metaheuristics, specifically how they are applied to help solve continuous optimisation problems.

2.2.1 *Terminology*

Here we present some terminology useful to further discussions in this chapter.

2.2.1.1 *Objective Function*

An objective function is an encoding of the goal of some problem as a mathematical function or computational model used to measure the quality of a given solution in respect to the problem. It essentially represents the interface between an algorithm and the real problem and plays a crucial role in the guidance of a search algorithm though a given search space - without it the algorithm would have no clue as to how the solutions produced are performing. The problem goal, or goals in the case of multi-objective problems, can be defined as either being a minimisation goal, where the objective is to produce solutions with low values with respect to the objective function, or a maximisation problem where higher valued solutions are sought. When indirect solution representations are used within the algorithm - those that do not yet represent a solution in their current form i.e., they are encoded so must first be converted (decoded) into a form that can be directly evaluated by the objective function [17].

2.2.1.2 *Feasible vs. Infeasible Solutions*

Feasible solutions are all solutions within the search space of the problem which satisfy the constraints on the problems parameters - specifically, the hard constraints (see below) [17]. Conversely, infeasible solutions all violate at least one of the problems hard constraints.

2.2.1.3 *Hard vs. Soft Constraints*

Hard constraints placed on a problem are conditions which have to be satisfied in order for a solution to be feasible [17]. On the other hand, soft constraints are those which we would like to have satisfied but which are not essential; so a solution violating one or more soft-constraints would still be considered feasible [17]. A common way of handling soft constraints when searching for a solution to an optimisation problem is to have solutions incur a penalty if a soft constraint is not satisfied; possibly weighted by desirability. Therefore, the objective function of many problems are represented by the summation of penalties for the soft constraints [17]. This means that solutions that do not violate the soft constraints will be considered as being of a higher quality [17].

2.2.1.4 *Local vs. Global Optimisation*

The difference between local and global optimisation lies in the scope of the optimisation task. Local optimisation looks to optimise within a local feasible region [97], that is, to discover a local minimiser (or optimum) - the smallest objective value within some feasible neighbourhood in the problem domain. A neighourhood function N defines the neighbourhood of a solution $s \in S$ as a mapping $N : S \to 2^S$ which assigns to each s a set of solutions $N(s) \subset S$ [135]. A solution $s' \in N(s)$ is refered to as a neighbour of s [135]. Therefore, given a domain S and a feasible neighbourhood of the domain $N(s) \subset S$, local optimisation seeks to find a local optimum $x^* \in N$ such that $f(x^*) \leqslant f(x) \forall x \in N$ [135]. Global optimisation on the other hand is concerned with discovering the smallest objective value or values over the entire feasible domain of a function - a global optimum, also often referred to as the 'best solution'. As before, given a domain S, global optimisation seeks to find a global optimum $s^* \in S$ such that $f(s^*) \leqslant f(s) \forall s \in S$ [135]. The main outcome of optimisation is therefore to discover a global optimum s_* of which many such solutions may exist [135]. Where many global optima may be present (it is not always known), the outcome may be defined as attempting to discover all of the global optima in order to generate alternative choices of solution [135].

2.2.1.5 *Best Solution*

Using a term like the best solution implies that there must be more than one solution, all of which are considered to have differing levels of value [50]. The meaning of best, in terms of quality, is quite often depends on the actual problem, the solution method and any allowed tolerances [50]. Some types of problem have exact answers where the term 'best' has an absolute definition [50] - in optimisation terms, 'best' in these instances refers to a problems single global optimum. Other problems however can have several global optima, so 'best' in these situations is more of a relative term [50]

2.2.1.6 *Discrete vs. Continuous Optimisation*

Discrete optimisation involves the optimisation of problems which are characterised by a finite number of states accepted by their input parameters; on the other hand, continuous problems are typically those that take their solutions from an uncountably infinite set of solutions whose parameters are members of the set of real values - of which there are no discontinuities [97]. However, given that digital computer technologies cannot provide the kind of precision required of a 'true' continuous solution, the set of possible solutions will be far less than uncountably infinite albeit still intractably large.

Although the practical optimisation of both types of problem can be approached using similar methods - at least in terms of inexact heuristic methods - there are nuances found for each type of problem that need to be taken into account in order to optimise effectively. For example, continuous optimisation problems are considered easier to solve than discrete problems as their typical smoothness make it possible to work out the functions behaviour at all points close to a point x by using objective and constraint information [97]. On the other hand, the behaviour of the objective function may change drastically between points - which are deemed close by some measure of distance [97].

2.2.2 *A Formal Definition of an Optimisation Problem*

Burke and Kendall in [17] define optimisation as trying to find the best solution possible from amongst all possible solutions. They state further that optimisation can therefore be considered as the task of modelling the problem to be solved as a mathematical evaluation (or objective) function representing the quality of a solution and then search through the space of all possible solutions in order to find one that either minimises or maximises this function [17].

The canonical form for an optimisation problems is usually stated as the minimisation of the objective function subject to constraints placed on its variables and expressed as [97, 6]:

$$\min f(x) : x \in \mathbb{R}^n \quad \text{subject to} \begin{cases} g_j(x) \geqslant 0, & j = \{1, 2, \dots, J\} \\ h_k(x) = 0, & k = \{1, 2, \dots, K\} \end{cases} \tag{2.1}$$

$$\text{such that:} \quad x_i^{(L)} \leqslant x_i \leqslant x_i^{(U)}, \quad i = \{1, 2, \dots, n\}$$

where $f(x)$ is the function being optimised, $g_j(x)$ are J inequality constraints and $h_k(x)$ represent the K equality constraints [6].

2.2.3 *Problem Difficulty*

Michalewicz and Fogel in [90] state several different reasons why real-world optimisations are difficult to solve effectively:

- The search space is too vast to allow the use of an exhaustive search

- The problem is so complex that in order to determine any answer whatsoever requires the use of such simplified models of the problem domain that any solution found is essentially useless
- The objective function is noisy or varies over time therefore requiring that an entire series of solutions are required as opposed to a single solution
- The search space is so heavily constrained that even finding one feasible solution to the problem is difficult

2.2.3.1 *Complexity and P vs. NP*

Complexity refers to the study of how difficult an optimisation problem is to solve [17]. Here, problems are classified according to the properties of optimisation algorithms where roughly speaking, a problem is classified as being 'hard' if there exists no known fast solver and 'easy' otherwise [29].

The first key concept in computational complexity is that of problem size which is rooted in the dimensionality of a given problem - that is, the number of problem parameters - and the size of the set of values of which each of the problem parameters can be set [29].

The second key concept relates to algorithms rather than to the problems themselves, that of the running-time [29]. Running-time is the number of "elementary" operations required of an algorithm before it terminates when running against a given problem - in general, the intuition of computational complexity is that larger problems will require more computational time to solve, although this is not always true [29]. The best known definitions of the 'hardness' of a problem is a relationship between the size of the problem being solved to the worst-case runtime of the algorithm to be used to solve it. This is encapsulated as a formula that defines an upper bound on runtime of the algorithm as a function of the problem size [29]. Basically, shorter running times are expected when this formula is a polynomial or longer when the formula is 'super-polynomial' such as exponential [29].

The final key concept of computational complexity relates to the concept that it is possible to transform one problem into another through an appropriate mapping which may or may not be reversible. This is referred to as *problem reduction* [29].

These key concepts can now be described more formally as [29]:

1. A problem can be said to belong to the class P (polynomial) if there exists at least one algorithm capable of solving it in polynomial time [29]
2. A problem can be said to belong to class NP (Non-deterministic polynomial) when it can be solved by some algorithm, specifically a non-deterministic turing machine, with no runtime guarantees, and its solution can be tested in polynomial time [29]. P can be considered as a subset on NP since a solver with a polynomial runtime can also be used to test solutions in polynomial time [29]
3. A problem is said to belong to the class NP-complete when it belongs to the class NP and another problem in NP can be reduced to this particular problem by an algorithm that runs in polynomial time [29]
4. A problem belongs to the class NP-hard if (i) it is at least as difficult as any problem in NP-complete, but (ii) where solutions cannot be necessarily tested in polynomial time - the canonical example being the halting problem [29]

One of the 'grand' challenges in complexity theory is to provide a proof that $P = NP$ or conversely that $P \neq NP$, in other words, is the class P and the class NP in fact the same class - or not, in the latter case [29]. It should be noted that the proof required of the latter case will need to be acquired through the application of complex mathematics, but for the former, 'simply' showing that an algorithm exists that can solve at least problem of class NP-complete would provide enough of a proof [29].

2.3 CLASSICAL OPTIMISATION METHODS

Before continuing on further to discuss stochastic optimisation, it is worth outlining some of the existing 'classical' mathematical optimisation techniques. Wehrens and Buydens [142] categorise the following methods under two headings: *strong* methods and *weak* methods.

Strong methods make certain assumptions about the structure of the solutions space; if the assumptions made are correct then strong methods are both fast and reliable - however, it they are incorrect the methods will never manage to find the global optimum even through increased repetitions [142]. They are often used in the final part of the optimisation of difficult problems where the region of the global optimum has been found but an exact solution has yet to be discovered [142]. Additionally, they find utility where problem dimensionality is low [142].

Weak problems, on the other hand, make very little - if any - assumptions about the search space but at a cost to their effectiveness and/or performance and should only be used when there is no other option [142]. Each essentially samples the search space in the hope of discovering good solutions quickly [142] The quintessential example of an weak method is that of exhaustive enumeration of the search space, a technique whose use is infeasible for all but the smallest of problem instances [142]. They tend to make use of a random component in generating solution instances, given the fact they also use no assumptions - repeatability then tends to be low as repeated runs of these approaches typically produce very different results [142].

Between these two classes lies an *intermediate* class encapsulating approaches such as metaheuristics and other stochastic algorithms. Metaheuristics will be discussed later in this chapter in section 2.4.5.3 and specific methods described in Chapter 4.

2.3.1 *Strong Methods*

2.3.1.1 *Gradient-based Optimisation Methods*

GRADIENT ASCENT The first gradient-based method that will be discussed is that of Gradient Ascent - or Descent if dealing with minimisation. Here, the basic idea is to find the slope of the function to be optimised, from the current point, and move up the hill towards the maxima at its peak [82]. Progress continues until the slope of the function reaches zero - which may or may not have occurred around a function maxima. Since progress cannot continue where the slope equals zero, it is possible for gradient ascent to converge elsewhere, such as: (i) at minima of the function and (ii) at ***saddle points***. The gradient ascent method does not require the computation or knowledge of $f(x)$, but it does however make the assumption that the first derivative $f'(x)$ can be known and calculated [82]. The procedure for this method is quite straightforward for both the 1-dimensional and the n-dimensions cases.
In 1-dimension, a random initial point is selected. We then continually add to it a portion of its slope i.e., $x \leftarrow x + \alpha f'(x)$ - where α is a small positive value [82]. This process continues until the slope reaches zero and therefore x is unable to change further [82]. For the n-dimensional case, the slope at the current point is simply replaced with the gradient so that $\bar{x} \leftarrow \bar{x} + \alpha \nabla f(\bar{x})$. There are a couple of known problems with gradient descent, one has already been discussed - that of search progress being unable to continue when a slope not found at an optima equals zero. The other issue is speed of convergence. The reason behind this second issue is that as the slope approaches zero at the maximum, the value for x will overshoot the peak - landing

on the other side of the hill - and may do this many times before reaching the maximum. The cause of this is due to the step size α being based solely on the slope of the function at the current position. If a slope is very steep, α will be large even if it is not required. One way to deal with this is to tune the value of α to the particular problem instance being tackled. Another way, if the second derivative of the function, $f''(x)$, can also be calculated, would be to abandon the gradient ascent method altogether, opting instead for Newtons Method, a variation of gradient ascent described next.

NEWTON'S METHOD This variation on gradient ascent/descent involves the use of the second derivative, i.e. $x \leftarrow x - \alpha \frac{f'(x)}{f''(x)}$ of the function such that α is dampened as the algorithm approaches a slope of zero [82]. This means that the algorithm will now converge to any kind of zero slope - maxima, mimima, saddle points and to points of inflection [82]. Additionally, the multidimensional case of the second derivative is not as straightforward to compute as the gradient $\nabla f(\bar{x})$ was for the first derivative, but is instead a multi-dimensional Hessian matrix $H_f(\bar{x})$ comprised of partial second derivatives over each dimension [82]. This extra complexity is compounded by the fact that newtons method divides by the second derivative, requiring that inverse hessian matrix be calculated [82].

Since the second derivative is being used, the algorithm maintains all the information required to identify whether it has reached a maximum solution (or minimum point in the case of minimisation) as opposed to other points of zero slope discussed previously. This is due to the fact that the second derivative will have negative value when a maximum point is met and positive otherwise [82].

The use of the functions second derivative still does not solve the overall problem of local search methods however, in that it can still converge to local optima rather than the global optima [82]. One simple method of constructing a global search algorithm from both gradient ascent and newtons method is to place these algorithms in a loop such that as a local optima is discovered, the algorithm may be restarted from a random position in the search space - keeping track of the best optimum found so far - in the hope of finding better optima which hopefully lead to the global optima [82]. This is essentially the same approach taken for derivative-free hill climbing algorithms when looking to search for global optima - see Chapter 4 Section 4.2.2.1 for more details on using restarts for global optimisation.

2.3.1.2 *Response Surface Methods*

Originating from the area of experimental design, response surface methods make the assumption that the response surface (objective function landscape / fitness landscape) can be parameterised under a simple functions containing a single optimum [142]; that is, these

methods aim to estimate the response surface by choosing parameters in an intelligent way [142]. A common implementation is known as a central composite design (CCD) which makes use of 2^N points (factorial points) [70] combined with 2N points (axial points) [70] and one central points [142, 70].

2.3.1.3 *Simplex Methods*

The Nelder-Mead Simplex algorithm, or simplex algorithm, was introduced in 1965 by Nelder and Mead in [94] for multidimensional unconstrained optimisation problems. The approach achieves this without derivative information - using only the returned objective function values at given points - and do not attempt to approximate a gradient at any point [122]. From its name, the Nelder-Mead Simplex algorithm is simplex based [1] Simplex-based algorithms begin with an initial working simplex composed of $n+1$ solution points considered as the vertices of the simplex and a corresponding set of objective function values $f_j := f(x_j) : j = \{1, 2, \ldots, n\}$ [122]. It is a requirement of the method that the initial vertices do not lie in the same hyperplane, that is, the simplex should be non-degenerate [122].

To execute the algorithm, a sequence of transformation operators are applied to the simplex which are determined by testing one or more test points and comparing these with the objective values of the vertices [122]. The general procedure can be stated simply as [122]

1. Generate an initial working simplex
2. While termination criteria not satisfied:
    a) calculate the information relevant to testing termination criteria
    b) transform the current working simplex
3. Return the objective value of the best vertex of the final simplex

There are four fundamental transformation operators, each controlled by a parameter: reflection α, contraction β, expansion γ and shrink δ [122]. Each of the four parameters α, β, γ and δ must satisfy the following constraints [122]:

- $0 < \alpha < \gamma > 1, 0 < \beta < 1, 0 < \delta < 1$

The most common values of these parameters that satisfy the constraints are: $\alpha = 1$, $\beta = 0.5$, $\gamma = 2$ and $\delta = 0.5$ [122].

1 A simplex S in a search space R^n is defined as the convex hull of $n+1$ vertices in R^n [122]

TRANSFORMATION STEPS Transformation of the current simplex follows three steps [123, 122]:

1. determine the worst, second-worst and best vertices in the simplex: h, s and l respectively as in:

   $f_h = \max_j f_j, \quad f_s = \max_{j \neq h} f_j \quad f_l = \min_{j \neq h} f_j$

2. A new centroid is computed for the best side - opposite from the worst vertex - by:
   $c = \frac{1}{n} \sum_{\substack{j=0 \\ j \neq h}}^{n} x_j$

3. Calculate the new working simplex by:

   a) First attempt to replace the worst vertex with a new vertex through reflection...

      - If the new vertex x_r is better than the second best vertex x_s but is not better than the best vertex x_l then simply accept the new vertex, replacing x_h and continue on to the next iteration

      - If the new reflected vertex x_r is actually better than the best vertex x_l, attempt to explore further by expanding x_r to x_e by:
        $x_c = c + \gamma(x_r - c)$

      - The vertex x_h is the replaced by the better of the vertices x_r and x_e

   b) If the reflected point x_r was found to be worse than the second best vertex x_s then this suggests that the region of x_r is not a promising direction so we perform contraction on the simplex by: $\quad x_c = c + \beta(x_h - c)$

      - if the new contracted vertex x_c is better than x_h then x_h is replaced by x_c and move to the next iteration, that is: $x_h \leftarrow x_c$

      - If the contracted vertex x_c is worse than x_h we must resort to the final transformation, the shrink operator which transforms the entire simplex by maintaining the current best point x_l and modifying all others relative to it. The jth new point is calculated using:
        $x_j = x_l + \delta(x_j - x_l)$

Termination is then determined by the following criteria:

1. Iteration budget has been exhausted

2. The minimum size for the simplex ha been reached

3. An acceptable objective value for one of the vertices has been reached

The main advantage of simplex methods according to [122] is that it succeeded in achieving a good minimisation of the objective function over a number of numerical trials and does so over a relatively low number of objective function evaluations. However, a disadvantage given is that "numerical breakdown" of the algorithm occurs in practice resulting in requiring an "enormous" number of iteration and making very little progress toward the minimum - even when the simplex is nowhere near the minimum [122].

2.3.2 *Weak Methods*

2.3.2.1 *Random Search*

This very simple approach involves sampling the search space at random, maintaining the best found and terminating when the time budget is exceeded [142]. As such, this is not an approach that is widely used [142] given the multitude of other more successful methods available - of which some are close in terms of their simplicity (See Hill Climbing Approaches, Chapter 4 - Section 4.2). In fact, the only class of problem that this approach is expected to perform as well as any other is that of needle-in-the-haystack problems - where the entire search space is completely 'flat' except for a single point, and so no information involving the proximity of the optimum from the current location can be obtained by any existing algorithm [142].

2.3.2.2 *Sampling Methods*

A somewhat more feasible weak method is to sample the search space using a grid of a pre-determined size [142] - aptly referred to as *grid search* methods. When promising regions are identified, grids with smaller spacings may be placed in these locations in order to perform a more fine grained search [142]. These approaches suffer more than most from the curse of dimensionality (Chapter 3 - Section 3.2) where as the number of problem dimensions increases the number of samples required to form the grids increases exponentially [142]. Another disadvantage is that since points between the samples are not known so good solutions and regions smaller than the spacing of the grid may be missed [142].

2.4 METAHEURISTIC OPTIMISATION

2.4.1 *What is a Metaheuristic?*

The name metaheuristic is derived from a combination of the Greek prefix 'meta', meaning beyond - in terms of high-level and 'heuristic' from the greek word 'heuriskein' meaning 'to search' [39]. This combination can therefore translated roughly to 'high-level search'. Although this etymology may provide us with a loose idea of what it may mean to be a metaheuristic, unfortunately, no one definition of metaheuristic available in the literature seems to capture everything that it means to be a metaheuristic and so several will be presented in this section for comparison.

As stated by Burke and Kendall in [17], Glover first used the term metaheuristics in 1986 in [39] and defines it as[2]:

> "A meta-heuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality. The heuristics guided by such a meta-strategy may be high level procedures or may embody nothing more than a description of available moves for transforming one solution into another, together with an associated evaluation rule"

Stutzle in [131], found in [11], offers another quite explanatory definition:

> "Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristics, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descent by allowing the local search to escape from local optima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more "intelligent" way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decisions made during the

2 there are other earlier papers making use of the term metaheuristic

search. But, the main difference to pure random search is that in these algorithms randomness is not used blindly but in an intelligent, biased form.”

A less formal definition of metaheuristics, as defined by Sean Luke in [82], is that metaheuristics comprise a general class of optimisation algorithms - forming the main subfield of stochastic optimisation - that make use of some level of randomness in order to find the solution to ’hard’ problems [82].

Blum and Roli in [11] provide a summary of the fundamental properties in which they feel characterise metaheuristics:

- “Metaheuristics are strategies that “guide” the search process”
- “The goal is to efficiently explore the search space in order to find (near-)optimum solutions”
- “Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes”
- “Metaheuristic algorithms are approximate and usually non-deterministic”
- “The may incorporate mechanisms to avoid getting trapped in confined areas of the search space”
- “The basic concepts of metaheuristics permit an abstract level description”
- “Metaheuristics are not problem specific”
- “Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy”
- “Todays more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search”

They continue to then offer their own short definition of a metaheuristic, as [11]:

> “In short we could say that metaheuristics are high level strategies for exploring search spaces by using different methods. Of great importance hereby is that a dynamic balance is given between diversification and intensification . . . ”

For our purposes, we define a metaheuristic to be an algorithm making use of some for of randomness, comprised of one or more high-level heuristics whose task it is to drive and control a set of lower-level search heuristics in their quest to discover ‘good enough’ solutions to ‘hard’ optimisation problems.

2.4.2 *Exploration vs. Exploitation in Metaheuristics*

A common theme in all metaheuristics is that of the trade-off between exploration and exploitation [6] - also referred to a diversification and intensification respectively [11]. Exploration refers to the force in a metaheuristic, provided by its operators, that diversifies solutions maintained by the algorithm from the search space [6]. This divergent behaviour of the algorithm encourages a global search behaviour [6]. Exploitation on the other hand refers to how well the metaheuristic operators are able to use information from previously discovered solutions in order to focus the search on specific regions of the search space [6]. Metaheuristics exist on a spectrum depending on how much of each property is displayed i.e., some metaheuristic approaches can be more exploitative than explorative and vice-versa [6].

On the extreme ends of the scale, an example of a purely explorative search can be observed in the trivial random search algorithm which involves randomly selecting solutions in the search space for a given number of iterations [6]. Examples of purely exploitative search are hill climbing algorithms (Chapter 4 - Section 4.2) that perform small incremental steps in the space while new improvements are discovered or no further steps can be taken without finding a worsening solution [6]. Most metaheuristics, however, offer the ability to tune these aspects through the parameters used by their operators [6].

The exploitative and explorative nature of metaheuristics, along with their effective use of randomness, give them several advantages over classical (exact) optimisation methods. Bandaru and Deb illustrate this in [6] with the following list:

- Metaheuristics are able to find 'good enough' solutions to computationally 'easy' problems with a large input complexity [6]
- They can find 'good enough' solutions for NP-hard problems (see Section 2.2.3.1) [6]
- As opposed to most classical optimisation approaches, metaheuristics make use - and indeed, do not require - gradient information about the objective function and can therefore be applied to non-analytic, black-box or simulation-based objective functions [6]
- Most metaheuristic algorithms are capable of escaping from local optima [6]
- Due to the above ability to escape local optima, metaheuristics can better deal with uncertainties related to the objectives [6]

- Multiple objectives can be handled by most metaheuristics given minor algorithmic modifications [6]

2.4.3 *Problems Addressed by Metaheuristics*

The types of problem addressed by metaheuristics are those where

- The size of the solution space is large enough to make the use of exhaustive or exact algorithms infeasible,
- There is no known deterministic algorithm to solve the problem at all or within a reasonable time frame, and
- Problems where there is little knowledge of the domain that can be used to construct an optimal solution[82].
- A near-optimal solution to a problem is acceptable as opposed to the searching for the exact optimal solution [6]. Metaheuristics do not come with any guarantees of optimality.

2.4.4 *Categorisations of Metaheuristics*

Several different categorisations of metaheuristics, based on a variety of properties, have been made over the years. Unfortuanately, there is little consensus as to a single scheme for categorising metaheuristics from the list given in the next subsections, they are often used interchangeably in the literature and even combined for increased specificity.

2.4.4.1 *Nature-inspired vs. Non-nature Inspired Metaheuristics*

One method used to classify metaheuristics is by basing this classification on the conceptual origins of the approach [11]. In metaheuristic research, these origins roughly fall into the category of either being inspired by natural processes (Genetic Algorithms (GA) and Particle Swarm Optimisation (PSO)) or "artificial" (Tabu Search and ILS) in the sense that nature was not queried for insight. Blum and Roli [11] do not consider this classification very meaningful for two reasons. The first is that many 'hybrid' approaches do not fit cleanly into either class - and in fact, may fit both simultaneously [11]. The second reason given is that it is often difficult to attribute even some non-hybrid metaheuristics to either class, for example, the use of memory in Tabu search could also be considered as being nature inspired [11].

2.4.4.2 *Single-solution vs. Population-based Metaheuristics*

Another characteristic that may be used for classifying metheuristics is whether one or more solutions are being utilised at once [11]. Metaheuristic algorithms that operate only on a single solution are often referred to as *trajectory methods* which include the set of local search algorithms such as: Iterated Local Search (ILS), Simulated Annealing (SA) and Tabu Search [11]. What all these algorithms have in common is their property of describing the path , or trajectory, through the search space [11]. In contrast, population based metaheuristics such as: Particle Swarm Optimisation (PSO), Genetic Algorithms (GA) and Differential Evolution (DE), focus on the descriptions of the evolution of multiple, often related, solutions in the search space [11]. Another way of referring to these classifications is as *Local vs. Global Optimisation Metaheuristics* - typically single solution metaheuristics are only employed for local search whereas population-based approaches are able to search for a globally optimum solution to a given problem.

2.4.4.3 *Dynamic vs. Static Objective Function*

Metaheuristics can also be classified by the way in which they might make use of the objective function [11]. Some approaches use the objective function in the same way it is provided - as a 'black-box' - where others such as guided local search make modifications to the objective function as the search progresses [11]. The logic behing this approach is to enable the algorithm to escape from local optima by changing the search landscape itself [11].

2.4.4.4 *One vs. Many Neighbourhood Structures*

The majority of metaheuristics make use of only a single neighbourhood structure throughout the search process. What this means is that the topology of the search landscape does not change [11]. Others, the canonical example being Variable Neighbourhood Search (VNS), make use of a set of neighbourhood topologies allowing for the algorithm to switch between then in order to diversify the search [11].

2.4.4.5 *Memory vs. Memoryless Metaheuristics*

Another classification scheme is to focus on whether or not the metaheuristic makes use of search history / trajectory information - these days considered to be a fundamental component of sucessful metaheuristics [11]. Memoryless algorithms, such as Simulated Annealing (SA), carry out a Markov process - where the information used to decide where the search should explore next is taken exclusively from the current solution [11]. For Memory algorithms, a further delineation can be found in the various ways that an algorithm may make use of

memory - usually considered between the categories of short or long-term memory [11]. Short-term memory approaches typically maintain only information about the previous few iterations, the last few solutions found and decisions made [11]. Long-term memory approaches on the other hand make use of a large set of various 'synthetic' parameters about the search [11].

2.4.5 *General Concepts*

2.4.5.1 *Solution Representations Used in Metaheuristics*

In order to develop solutions to a given problem; that is, to provide input to the objective function representing the problem, a suitable solution representation for use in optimising the problem in the context of metaheuristics must be selected. There are several drawbacks and advantages to the use of each type of solution representation discussed here, however the selection of a representation is often heavily influenced by the problem being solved. Any change to solution representation will often require a subsequent change to the objective function and in effect, the nature of the search space will also change. For example, it may be tempting to use a representation simply because it feels more natural and intuitive but some search spaces based on a representation (and it's corresponding objective function) may well be more rugged, sparse, deceptive or of a larger scale than other spaces defined by other representations and functions. The solution representation also determines how the search operators might create neighbourhoods and how the objective function will be evaluated; for example, a representation requiring heavy or complex conversion by the objective function will require more time and computational resources. Objective functions are considered to be one of the main bottlenecks in search, so it is always prudent to minimise computation costs here where possible. All in all, it is important to keep the nature of the resulting search space in mind when selecting a representation as the efficiency of search is very much determined by it. In the following paragraphs, since we work primarily with a vector-based representation[3], we outline several of the more common vector-based representations and how they are typically used.

(A) BINARY REPRESENTATION The simplest of the common representations used, the *binary representation*, is one in which each solution is held as a vector of bits where each bit or sub-sequence of bits represents some parameter of the problem being solved [29]. For a given problem, a practitioner should: (i) ensure that it is clear how the binary string is to be

3 By vector, we mean a one-dimensional array of a fixed length, as per Luke in [82]

interpreted, that is, how it is to be decoded into a meaningful solution and (ii) in ensuring (i), that each possible binary vector encodes a valid (but no necessarily feasible) solution to the problem given and conversely that each possible solution can be encoded [29]. For some problem types, the binary representation is a natural one - especially those considering boolean decisions (yes/no) [29, 111, 135] - thus ensuring the above considerations are met. An canonical example is a *0/1 Knapsack Problem* where a solution can be encoded as simply as whether or not an item is included in the knapsack or not - a binary decision [29, 111, 135]. It is also common that a binary vector be used to represent other data types such as integer or real-valued numbers; for example, a bit string of length $l = 80$ can be used to encode ten integer values using 8 bits each or five 16 bit real-valued numbers [29] - a scheme known as Binary Coded Decimal (BCD) [143]. BCD represents each decimal digit as a string of bits (4 or more bits). So a string of length 80 could code 20 decimal digits (each length 4), or 10 decimal digits (each length 8), or 5 decimal digits (each length 16). The sign is usually coded as another single bit. However, these schemes tend to have the disadvantage that consecutive decimal numbers are considered neighbours in the context of decimal space but not in the context of binary space [143], that is, the Hamming distance between two consecutive decimal numbers, for example integers, often does not equal 1 [29] forming what are known as Hamming Cliffs: where adjacent integers are represented by complimentary bit vectors and therefore share few or no bits in common [143, 82]. This problem can be helped by a binary to decimal mapping variation, *Gray Coding*, where consecutive integers will always have a Hamming Distance of 1 [29, 143].

(B) INTEGER REPRESENTATION Used in situations where individual solution parameters can naturally take one of n discrete values, the Integer representation has been used for solutions representations, such as: of unordered sets i.e., $\{1 = \mathtt{red}, 2 = \mathtt{green}, 3 = \mathtt{blue}\}$ or the definition of a discrete metric space [82]. When designing an integer-based solution representation of a problem, it is worth considering if there are any natural relationships between all the possible values that a solution parameter can have [29]. This is often more obvious for ordinal parameters, such as for the integers themselves, than for cardinal parameters - such as compass directions that do not appear to have a natural ordering [29].

(C) REAL-VALUED REPRESENTATION Usually, the most natural encoding for parameters that take their values from a continuous rather than discrete space is a real-valued or floating-point representation [29]. Given that precision on current digital computer technology is limited, it is technically more accurate to refer to this representation as a floating-point representation [29]; however, for consistency we will continue with using the term *real-valued*

for the remainder of this thesis. An example of a natural class of problem for a real-valued representation is that of continuous function optimisation, where the parameters of a solution represent a continuous point within the metric space of a given continuous function. When recombination operators for real-valued representations are considered, it is possible to generalise many of the operators used for discrete spaces to real-valued representations [29], for example: one-point, n-point crossover and uniform crossover (Chapter 4, Section 4.3.2.2); however these *discrete recombination operators* - as they are for discrete representations - are unable to introduce new values for the parameters thus leaving this task to the mutation operators [29]. Other options are to use recombination operators specifically suited to real-valued solutions, ones that are able to generate new parameter values [29]. Operators of this type that generate new values between two solutions to be recombined are known as *intermediate* or *arithmetic* recombination operators [29]. Alternatively operators that generate new parameter values close to - and can exceed - one of the solutions are known as *blend* recombination operators [29]. Unlike recombination, mutation operators for real-valued solutions cannot be extended naturally from discrete representations and so specialised mutation operators have been developed; one such example is that of *uniform* mutation (Chapter 4, Section 4.3.2.3).

2.4.5.2 *Metaheuristics and the 'No Free Lunch Theorem'*

In the fields of optimisation and metaheuristics, the no free lunch theorem (NFL) states that [146, 145]:

> When averaged across all possible problems, the performance of all algorithms is equal, irrespective of the choice of evaluation criteria.

There are a couple of consequences to this statement. Firstly, since all algorithms are equally good on average across all problems, this suggests there must be some subset of problems on which an algorithm performs well and another subset on which it performs poorly [4]. Secondly, it suggests that it is not correct to state that one algorithm is better than any other and that comparisons between algorithms must take into account the types of problems the algorithm performs best on.

There have been attempts to develop techniques that are able to circumvent the no free lunch theorem to various degree. Two notable examples are that of hybrid metaheuristics and hyper-heuristics, both of which are beyond the scope of this thesis.

4 of course, this is an over-simplification made for the purposes of illustration

One caveat of the NFL is that, at the moment, there is some contention on whether or not the NFL is only valid for discrete problem domains. For most problems being solved computationally on a Von Neumann architecture this detail should not present a problem due to the fact that we make use of a finite representation with precision bounded by internal representation. For more information and discussion on both sides of the argument about this detail of NFL, readers can refer to both [4, 141].

2.4.5.3 *Comparisons Between Metaheuristic Approaches*

Comparison between metaheuristic techniques has been said to be more difficult than for other types of algorithm [121]. However, comparing metaheuristics to one another is an important step in metaheuristic development in order to determine where a new approach stands in relation to others in terms of various performance measures a suitability to certain situations. Since the NFL theorem prohibits any implication that one approach is in general 'better' than any other, one cannot make such comparisons based solely on solution quality over a set of problem instances. As such, there are several important considerations that should be taken into account in order to effectively compare metaheuristics to one another in a meaningful way - without violating the implications of the NFL. A discussion of these considerations appear in the next few sub-sections based exclusively on the excellent presentation of the topic provided by Silberholz and Golden in [121].

(A) TESTBEDS

(A.i) The Use of Existing Testbeds In the majority of cases, it is of benefit - and actually considered to be the ideal case [121] - to make use of pre-existing testbeds available from the metaheuristic literature [121]. Not only does this allow comparisons with other metaheuristics to be carried out on a "by-instance" basis [121], but where algorithm results are published, it also removes the need to either obtain or reproduce algorithm source code from the literature in order to compare against the same or a different testbed - of which it will likely provide a different set of results. However, these existing testbeds can sometimes be insufficient or their use is not possible [121] - examples include:

- The existing testbeds have not been made available to other researchers or the problem instances have been generated randomly [121], for which either or both the function used to generate the instances or the seeds used for the pseudo-random number generator are not published or available.

- The testbeds are too small either in terms of available problem instances or of problem instance size [121].
- A new problem is being addressed by the metaheuristics being compared, therefore, there will be no existing testbed available for use [121]

When the available testbeds are insufficient for use in comparing metaheuristics available, there may be little option other than to develop new testbeds.

(A.ii) Creating a New Testbed Although making use of existing testbeds is the preferred option when comparing metaheuristics, there are several situations where there is little option except for developing a new testbed - some of which have already been discussed in the previous sub-section. This section will discuss several factor that should be taken into consideration when developing a new testbed. The first goal in developing a new testbed is that of ensuring problem instances included in the suite effectively mimic real-world situations [121]. Although artificial problems have their uses when developing and testing a metaheuristic, being artificial, many if not most of these fail to fully represent the important characteristics of a corresponding real-life problem instance. Since metaheuristics find their utility being applied to real-world problems as opposed to 'toy' problems it is crucial during the analysis of metaheuristics that the behaviour of algorithms on these kinds of problem is represented in results.

Another goal in creating an effective testbed is the provision of problem instances of various types and levels of difficulty [121]. In terms of types of problem, every metaheuristic has a 'niche' set of problems on which it performs very well and others where it does not perform as well. This observation is a consequence of the 'No Free Lunch Theorem' (Section 2.4.5.2) which states that when averaged over all possible problems, every search method, including metaheuristics, show the same performance - this means that there has to be a subset of all problems where a given metaheuristic performs best. It is therefore important when analysing and comparing metaheuristics that a wide range of problem types are represented in the testbed in order to identify those types of problem where a metaheuristic performs best. In terms of the levels of difficulty found among the test instances, several factors are involved in determining what makes a problem difficult to solve, which are discussed in Chapter 3, Section 3.2. When applied to real-world settings, metaheuristics will likely be targeted towards optimising objective functions which exhibit some of these factors - when analysing metaheuristics, representing all of these factors in the testbed ensures a more accurate depiction of overall performance.

Silberholz and Golden also suggest that large problem instances must be represented in the testbed [121]. One reason for this is that exact method are often able to run in a reasonable

length of time whilst also providing the guarantee of returning the optimal solution [121]. Metaheuristics are not required for these types of problem, they are most useful for those problems where the optimal solution cannot be found so readily. As metaheuristics show their utility being applied to these more intractable problems, it is important to ensure that they are represented in any new testbed.

(A.iii) Making the New Testbeds Available After creation of a new testbed, it is important to make sure it is easily accessible. Providing a testbed for others not only ensures it is widely used but also allows other researchers to make their own comparisons easily [121]. According to Silberholz and Golden in [121], one easy way to make a testbed available is to provide a simple generating function for the problem instances in the testbed. Another approach is simply to publish the testbed [121] or make it available through an author provided webpage or web service. Of these last suggestions, publishing will likely be the preferred option as it relies less on other researchers simply 'stumbing' upon the webpage and has the benefits of having the testbed go through the peer review process.

(B) CLASSIFICATION OF THE PROBLEM INSTANCES Proper classification of the problem instances present in the testbed is crucial to the effective analysis of metaheuristic techniques [121]. As discussed previously, since the no free lunch theorem suggests that every search algorithm has its niche set of problems on which it performs well, this niche set can only be discovered and subsequently reported for a new metaheuristic when problems in the testbed are appropriately separated into classes. According to Silberholz and Golden, state that each of these classifications should be:

- Noted prior to any experimentation
- Discussed individually in terms of a metaheuristics performance

The authors note further that the proper classification of real-world instances, and subsequent analyses, is of particular importance as it can help algorithm developers in industry, with a certain type of dataset, to decide on the most suitable approach to use [121].

(C) PARAMETERS In terms of comparing the actual algorithms themselves, it is useful to consider the factors surrounding the parameters included in these algorithms. Here, these are defined as the variables belonging to an algorithm that are involved in modifying the behaviour of the algorithm. For example, common parameters in a GA can include: population size, mutation rate and tournament size (for tournament selection). Silberholz and Golden note that in a metaheuristic, parameters can be set statically, such as using a fixed population

size in a genetic algorithm, or based on the problem instance - a population size of $5\sqrt{n}$ for instance, where n is the number of nodes in the instance [121]. [5] Four factors which will be discussed here are: the relation of the number of parameters to the simplicity of an algorithm, tuning and visualising the parameter space, parameter interactions and fair testing of metaheuristics in the context of parameters.

(C.i) Parameters and Algorithmic Simplicity When making comparisons between different metaheuristics in terms of solution qualities, it is useful to consider the complexity of the approach. For example, given two approaches which produce similar results, the simpler of two approaches is the superior algorithm [121]. A few reasons behind this superiority include: (i) being simple to reimplement in an industrial setting, (ii) being simpler to reimplement by researchers and (iii) being simpler to explain and analyse [121]. A number of reasonable metrics exist including the number of lines of source code required to implement the algorithm or the number of steps of pseudocode used to describe it [121]. However, both of these approaches rely too heavily on the programming language used to implement an algorithm, the individual style of the programmer and the level of detail in terms of pseudocode descriptions [121]. Silberholz and Golden prescribe a more meaningful metric - that of the number of parameters used in an approach [121]. Although this metric does not provide an estimation of programming complexity in terms of actually implementing an algorithm, it does however give a good estimation of how difficult the algorithms behaviour will be to understand and tune according to a given application.

(C.ii) Visualising and Tuning the Parameter Space The consequence of the parameter metric prescribed by Silberholz and Golden in the previous section is that metaheuristics making use of many parameters are considered more complex than those with only a few [121]. The main source of this complexity is that the effort required to understand and tune the parameters becomes far greater as the number of parameters considered increases [121] - the size of the parameter space in fact increases exponentially as more parameters are added. For example, if we consider a brute-force approach to tuning the parameters in a metaheuristic, tuning involves testing the performance of a set of m parameter values for each of the n parameters under consideration [121]. If only three values for each of the parameters were tested i.e. 3^n different configurations, a configuration set would be obtained of size: 9 for a metaheuristic with only 2 parameters, 2187 configurations for a metaheuristic with 7 parameters and an intractable 1,853,020,188,851,841 configurations if 32 parameters are used [121]. It is true, however, that there are more intelligent methods used to search for good

5 Although, other algorithms in the literature can a do adjust parameters dynamically based on the progress of the search

parameter settings - automatic parameter tuners such as SMAC (see Chapter 5 - Section 5.4) and Irace are two such examples - but the exponential growth of the parameter space still poses problems to these methods [121]. In addition to these issues, is that of visualisation of the parameter space as the amount of parameters in an algorithm increases [121]. Summarising an example from Silberholz and Golden [121], for an algorithm with only two - or even three - parameters, the parameter space can be quite easily visualised with many standard 2D visualisation techniques, e.g., a scatter plot where the axes each represent one of the parameters and the colour or size of the points can represent the cost of the parameter setting at a given point. In the same way, a space of three parameters may be viewed on a 3D scatter plot, however there will likely be some difficulty in viewing interior points unless various 'slices' through the visualisation are taken. Although Silberholz and Golden make note of a potential visualisation for 4D parameters spaces; such as in [56], they are quick to point out that these types of approaches in no way makes 4-dimensions as intuitively visualised as the 2D and 3D cases above increasing the visualisation difficulty [121].

(C.iii) Interactions Between Parameters Another complexity caused by large parameter sets in a metaheuristic is that these have a tendency to produce complex parameter interactions [121]. Silberholz and Golden state that from the point of view of optimisation, the interaction of parameters implies that the optimisation of individual parameters or even small subsets of parameters will become increasingly ineffective as the number of parameters considered is increased [121]. The authors however cite an instance where non-trivial parameter interactions were observed in a genetic algorithm containing only 3 parameters. The implication of this is that to an extent it is often difficult to avoid all parameter interactions with the exception of metaheuristics making use of only 1 parameter or no parameters at all [121] - where no interaction can occur.

(C.iv) Fair Testing Concerning Parameters The other concern when comparing metaheuristics, apart from simplicity, is that of fairness when tuning parameters [121]. For example, if one algorithm is over-tuned to the entire set of problem instances of which it is being tested, that can produce unfair comparisons; alternatively, if only a representative subset of the problems are used to tune parameters and the remainder to compare the metaheuristics, this would represent a much fairer comparison strategy [121].

(D) COMPARING SOLUTION QUALITY Since metaheuristics are designed to produce solutions of high quality, comparisons between metaheuristics in the context of solution quality performance is amongst the most important [121].

(D.i) Quality Metrics In order to compare the quality of solutions produced by two or more metaheuristics, a suitable objective metric is required - the best metric being the deviation of the produced solutions from optimal [121]. Comparisons made over separate problem instances are, in general, weaker than those made over the same problem; this is because different instances will have different structures in their search landscapes and more than likely different values for the optimum [121]. This metric is clearly only applicable to those problems of which the optimum value is known. For problems where the optimum is not known, several alternative metrics exist. One popular metric is deviation from best known solution [121], where the best known solution published in literature can be used in place of the true optimum for determining performance. It is also possible to use this metric when no best known solution has been published [121] - possibly through the generation of a solution using an algorithm known to perform well on the particular class of problem being addressed - however, comparisons made without the true optimum will be less meaningful [121].

(D.ii) Fairness of Performance Comparison As mentioned briefly in Section 2.4.5.3, it is usually prudent to make use of existing test beds when comparing metaheuristics. In particular, the fairness of such comparisons against original implementations can be ensured, as no re-implementation of algorithms needs to be carried out (where source code is unavailable and results published) and the inadvertent introduction of implementation errors in the test bed itself is unlikely compared to creating a new test bed e.g., when using re-implementations of some problem instances from other test sets. In the context of this thesis, testbeds which can facilitate the fair comparison between metaheuristics used to solve continuous problems include those provided for black-box optimisation competitions such as the Black-Box Optimisation Benchmarking (BBOB) competition [32, 47], where the test set used is available as the *COmparing Continuous Optimisers (COCO)* platform described in [49] as well as the various benchmark sets provided for the *IEEE Congress on Evolutionary Computations (CEC)* competitions such as CEC-2005 [133].

The usual approach to fair performance comparison in the literature is using rank-based analysis. This is most commonly carried out by:

1. Applying the set of algorithms being compared to a suite of problems to obtain a 'raw' performance measure (whether a fitness or time-based measure)

2. Using the raw performance data to generate a rank ordering (best to worst performing) of the algorithms against each of the problems in the set

3. Aggregating the ranks generated for all problems for each algorithm to arrive at a *consensus* ranking against the set as a whole, as per Mersmann et al. [88]

Typically, these ranks will then be presented in tabular form in order to analyse the rankings, however in [88], Mersmann et al. make use of parallel coordinate plots to improve comprehensibility and ease of analysis. In this thesis, both approaches are used for algorithm comparison, however for our plots the axes are inverted to provide easier visualisation of results.

In terms of fairness of comparison, by normalising the raw performances as ranks any bias in comparison from the nature of the problems, e.g., different perceived difficulty levels, are eliminated as only the ordering of the algorithms in the context of the selected performance measure is considered. Subsequent statistical tests will therefore be far less influenced by an unbalanced weighting given to any one data point. However, since ranking certainly involves loss of information, they are far less sensitive to large variances in the data, in this case the variances between performances obtained by algorithms, and thus do not readily show large differences in performance which may be an important factor in a particular study. One possible benefit to this resistance to noise in the data is that it more clearly highlights the differences of real interest, cutting through the stochastic noise inherent to the data obtained from metaheuristics.

2.5 LARGE-SCALE GLOBAL OPTIMISATION

A Large Scale Optimisation Problem (LSOP) is characterised as having many hundreds or thousands of dimensions. These kinds of problem are considered harder to solve than lower dimensional problems and algorithms that show good performance in lower dimensions can perform poorly when used against a scaled version of the same problem [18]. The nature of problems of this scale differ hugely from lower dimensional problems as each increase of dimensionality causes an exponential increase in domain volume [18] - often referred to as the 'curse of dimensionality' (See 'Dimensionality' - Chapter 3, Section 3.2). For example, a 1-dimensional search space containing 100 possible solutions, where one solution can be considered as being the global minimum, will require 100 function evaluations in the worst-case. Scaling the same problem to 2-dimensions has $100^2 = 10000$ possible solutions but extending this even to a 50-dimensional problem would place the global minimum as a single point in $100^{50} = 1 \times 10^{100}$ (a googol) feasible solutions [18]. Further to the scaling of the search domain, dimensionality can also affect structural features of the search space itself. An example from Caraffini et al. in [18] states the fact that a unit sphere in 3-dimensions has a surface area of $S_2 = 4\pi$ and a volume of $V_3 = \frac{4}{3}\pi$. If we were then to extend this unit sphere into n-dimensional space, it can be proved that the ratio between surface area and volume

is $\frac{1}{n}$; meaning that as the dimensionality increases, more and more points can be found on the surface as the volume becomes comparatively smaller [18]. This greatly increases the size of the neighbourhoods around the optimal solutions, in turn greatly increasing the time for conventional metaheuristics to solve the problem.

Computationally, the effective solution of LSOPs can increase the cost significantly; due not only to the large increase in the search space, but also by the large function evaluation budgets required to find high performing solutions [18]. For example, an optimisation algorithm that is capable of solving a 10-dimensional problem within 50,000 function evaluations would require $50,000^{40} = 9.1 \times 10^{187}$ [18] which far outweighs not only the number of atoms estimated in the observable universe (10^{78} to 10^{82}) but also the number of seconds since the universe began $4.361170769 \times 10^{17}$. Clearly, maintaining search coverage in this way simply isn't tractable.

An often forgotten consideration is that of the parameter configurations of the algorithms being used against LSOPs. For example, Caraffini et al. [18] describe a common population scaling strategy whereby the population size should increase exponentially with dimensionality in order to maintain search coverage; however, they quickly point out that, as with function evaluations budgets, this strategy quickly becomes impractical [18]. Using the same example as before, if a population-based optimisation algorithm with a population size of 30 was successful against the 10-dimensional problem, to maintain a consistent solution density in 50-dimensional space, the population size would have to contain $30^{40} \approx 1.22 \times 10^{59}$ individuals [18]. Metaheuristic approaches are therefore only able to cover a small portion of large-scale search spaces since population sizes of this magnitude are impractical, and often impossible, to actually use [18].

From a similar standpoint, consideration can also me given towards the computational budget afforded to the optimisation algorithm i.e., commonly the number of objective function evaluations. Using a similar example to the above, an algorithm which is originally tasked with optimising a 10-dimensional problem using a budget of 50,000 function evaluations, would require - for the same problem in only 50-dimensions - $50000^{40} \approx 9.1 \times 10^{187}$ function evaluations in order to cover the same proportion of the search landscape [18]. Therefore, although it is usual practice when optimising problems with relatively few dimensions (certainly < 50) that the budget be scaled proportionally to the dimensionality, this strategy simply won't scale to LSOPs. However, real-world LSOPs exist and as with smaller real-world problems, often need to be optimised effectively and efficiently in order to produce useful solutions, so research has been focussed on the development of algorithms and approaches

aimed specifically at solving LSOPs whilst at the same time trying to overcome the various difficulties described above [18]. Caraffini et al. roughly categorise the methods for tackling LSOPs as follows [18]:

- Methods which exploit promising search trajectories intensively
  - Perhaps quite unintuitively, these methods effectively abandon the search for global optimiality and instead intensively exploit promising regions already explored in order to find high performing solutions. Perhaps just as unintuitively, one popular method implementing this strategy involves the use of only a small population or by using a population which diminishes during the search [18]. Doing this allows one to control the amount of exploration carried out as there is less diversification present in the population, this in turn allows the population to begin converging towards one of the promising regions discovered early on in the search - with minimum exploration beyond this region. One other way to implement this approach is to combine highly exploitative local search algorithms with other algorithms and population-based structures [18]. Specifically, the local search algorithms used tend to exploit promising regions by perturbing a solution in an axis-parallel manner i.e., only one parameter of the solution is modified at any given time. Indeed, we found that the neighbourhood function utilised in the local search algorithms implemented for this study and making use of this axis-parallel strategy (see Chapter 4, Section 4.1.3) produced far better performance (in terms of solution quality) than allowing steps in any and all axes in a single iteration. Further, and echoed by Caraffini et al. [18], the modification of the Co-variance Matrix Adaption Evolution Strategy (CMA-ES) for separable problems *sep-CMA-ES* (detailed in Chapter 4, Section 4.7) implements this kind of axis-parallel trajectory through use of a diagonal rather than a full co-variance matrix in its updates. This variation of CMA-ES has been shown to be particularly promising within high-dimensional spaces, not only due to its use of an axis-parallel trajectory, but also, it is more suitable to large-scale problems as the calculation of a diagonal co-variance matrix requires far fewer computational resources and thus can be calculated far more quickly than a full matrix.
- Methods which use some mechanism for decomposing the search space
  - An implementation of this strategy is realised in an approach known as Cooperative Co-evolution, where LSOPs are decomposed into several smaller sub-problems which can each be solved independently of each other by a set of co-evolving

populations (one for each sub-problem). A more detailed discussion of Cooperative Co-evolution is presented in the next section (Section 2.5.1).

A common link between these two strategies for tackling large-scale problems is that each, in its own way, is achieving improvement by the exploitation of search directions [18]. Caraffini et al. go further, with the assertion that this means that "every modern metaheuristic for LSOPs gives up the search for the global optimum and simply tries to enhance as much as possible upon an initial sampling" [18]. However, it could be argued that an approach such as cooperative co-evolution does not in fact abandon the search for optimality, but instead is actively seeking it through the combination of simpler (and hopefully optimal) sub-solutions. Further, Ma et al. [83] point out that having multiple co-evolving populations helps the search as a whole maintain diversity; a property widely held to facilitate an effective global search. Also, one could state this same assertion about any metaheuristic algorithm, including those not specifically targeted at LSOPs, where metaheuristics as a general concept each work to enhance the performance of one or more initial states. Such algorithms, with the exception of simple local search algorithms, although not guaranteed to discover the optimal solution, do appear to have this goal in mind; which is particularly true of population-based approaches where there is a general aim to explore as much of the space as possible over the course of the search.

I do however agree with the authors of [83] that this certainly true of approaches categorised by the first strategy; local search algorithms - as their name suggests - do not have a strong focus towards global optimality and small populations will reduce the amount of exploration of the space, which can more easily be interpreted as abandoning the idea of achieving global optimality than for approaches such as cooperative co-evolution.

2.5.1 *Cooperative Co-Evolutionary Algorithm*

Similarly to other evolutionary approaches, a Cooperative Co-Evolution Algorithm (CCEA) finds its inspiration from theories of natural evolution. The predominant evolutionary concept adopted by CCEA is that of *mutualism*, a facet of co-evolution, where the success of two or more species or sub-populations is partially dependent on shared mutually beneficial relationships [83]; in essence, the species can be said to be evolving together (co-evolution) with the mutual goal of continued survival. Three interactions between species have been observed to occur in nature [83]: (i) Resource-Resource Interactions - where each population/species trade

consumable products necessary for each of their survival, (ii) Resource-Service Interactions - where a consumable product is provided in turn for conducting a useful function e.g., bees consume pollen in trade for dispersing pollen to other places, and (iii) Service-Service Interactions - when each species provides a function useful to the other, e.g., a pro-biotic bacterial species provides services to the immune system of humans (as well as having other functions) and the human provides a safe, warm and nutrient-rich environment for the bacteria to live and procreate.

Looking closely at the history of our own species, it is easy to see that humans have themselves benefit from mutualistic relationships with other species as well as between other human groups. Consider our relationships with domesticated animals, particularly dogs, whereby humans benefit from positive emotional attachment as well as work provided by dogs e.g., herding of cattle and sheep, hunting, pest control, protection etc.; and humans reciprocate by providing a home, protection, food, warmth etc. Amongst ourselves, we have, and do, benefit from trade and cooperation between different countries and cultures as well as many other beneficial interations. Henrik Valeur, a Danish architect who introduced the concept of co-evolution to the architectural world considers that; "As we become more and more interconnected and interdependent, human development is no longer a matter of the evolution of individual groups of people but rather a matter of the co-evolution of all people"[137].

CCEA, then, works by applying a simplified version of this natural concept in terms of co-evolving sub-populations of solutions to a given problem. Particularly useful against large-scale problems, each independent sub-population evolves towards the solution to a non-overlapping sub-problem of lower dimensionality. An overall solution can then be obtained through the combination of sub-solutions found by the separately evolving sub-populations [18, 83]. As such, the evaluation of any given individual in any given sub-population requires cooperation with all other sub-populations [83]. Specifically, the fitness of an individual from a sub-population is calculated in terms of complete solutions in which the individual is participating [83]. It is chiefly this divide-and-conquer decomposition approach that places CCEA at an advantage over more traditional evolutionary algorithms [83].

According to Ma et al. [83], CCEA can be said to have four main advantages stemming from its decomposition strategy:

1. Decomposition of the problem into sub-problems allows CCEA to be readily parallelised in order to speed up the optimisation process

2. Since each sub-problem is tackled by a separate sub-population, CCEA can maintain good solution diversity - a difficulty often plaguing other evolutionary algorithms

3. By decomposing a system into "sub-modules", the system as a whole can be made more robust against the errors and failings of any one module; increasing reusability in dynamic environments

4. If decomposition is carried out effectively, the 'curse of dimensionality' (Chapter 3, Section 3.2) can be somewhat alleviated allowing for generally better performance when solving LSGO problems when compared to more traditional EA

Despite the power that can be displayed by CCEA, a difficulty often faced when using this approach is deciding how to go about decomposing a problem into its various sub-problems. Clearly, this problem is trivial if the problem is known to be fully additively or multiplicatively separable (see Chapter 3, Section 3.2) where there is minimal interaction between variables; problems can then be naively separated out as $k = \frac{d}{n}$ instances of the problem where d is the problem dimensionality and n is the number of dimensions per sub-problem. [6]. It is also possible, mostly for separable problems, to have $k = 1$, where each of the d sub-populations evolve solutions for sub-problems of a single dimension. In fact, this was the strategy employed in the first CCEA, Cooperative Co-evolutionary Genetic Algorithm (CCGA), by Potter and De Jong [107]. Complete solutions to separable problems can then be combined through the summation, multiplication or some other reconstitution of sub-solutions. However, a significant disadvantage of CCEA is that as sub-populations are added, the total number of evaluations of the objective function available to CCEA also needs to increase proportionally, since each population conducts its own set of function evaluations in determining the fitness of solutions to its sub-problem. Therefore, depending on the scale of the problem to be solved, it may be wise to consider larger sub-problems in order to reduce computation time.

Several 'flavours' of CCEA can be found in the literature, which include: the first cooperative co-evolutionary approach - CCGA - proposed by Potter and De Jong in [107], Cooperative Co-evolution with Differential Evolution (DECC) by Yang et al. [150] and Cooperative Co-evolution based on PSO (CCPSO) by Li and Yao [75] to name but a few. However, for the remainder of this section we will not focus on these variations on an individual basis, but instead form our discussions around one or another of the two basic CCEA algorithmic

6 Of course, this is a simplified view of problem decomposition for CCEA in general, sub-problems do not have to be of the same dimensionality as one another and can thus be of variable size. Indeed, for problems that are only partially-separable, one decomposition method may be to evolve smaller separable parts of the problem independently from a comparatively larger non-separable part(s)

frameworks outlined by Popovici et al. in [106] as many CCEAs are a variation on one of these frameworks or the other. Before discussing these however, I present a brief discussion of problem decomposition for CCEA.

For a more detailed discourse of cooperative co-evolutionary algorithms, three of the main sources cited in this section can be recommended: The survey paper by Ma et al. [83], focussing specifically of function optimisation [107] by Potter and De Jong and in particular [106] by Popovici et al. which provides a relatively thorough treatment of the subject matter.

2.5.1.1 *Problem Decomposition*

Many decomposition strategies have been developed since the inception of CCEA, including the simpler of those as mentioned previously, however, an overaching issue that arises when performing problem decomposition using any such strategy, is how to decompose problems where variable interactions are present [83]. Indeed, it is exactly this issue that motivates the search for more effective decomposition strategies - since without such interactions i.e., if every problem was fully separable, then a simple strategy using a sub-problem size of $k = 1$ would not have a more effective substitute.

In the ideal case then, the decomposition should be conducted with a focus on any possible variable interactions with a view to minimise such interactions; for example, a problem that can arise if interacting variables do not form part of the same sub-problem, is that the CCEA can easily become trapped in a pseudo-minimum [83] [7]. By keeping all interacting components together, these sub-problems can be solved separately without negatively affecting the solution to the other (separable) sub-problems.

Decomposition methods range from those which make use of Static Variable Grouping or Random Variable Grouping, to somewhat more complex methods such as linkage learning-based variable grouping, domain knowledge-based variable grouping and overlap and hierarchical grouping [83]. However for brevity, we will only discuss the first two of these here.

(A) STATIC VARIABLE GROUPING Static variable grouping methods make no attempt to discern any interdependence amongst variables, but instead simply decompose high-

7 Ma et al. note that a pseudo-minimum does not refer to a local minimum of the original (full) problem, but is in fact a minimum that is created by incorrect decomposition. A difficulty that likely plagued Potter and De Jongs early CCGA which indeed performed much better against separable problems than against non-separable problems

dimensional problems into a series of low-dimensional sub-problems where the grouping remains fixed throughout the execution of the algorithm [83]. The original CCEA by Potter and De Jong (CCGA) [107] is an example of static variable grouping, where the grouping is held as n 1-dimensional sub-problems and each searched by a separate sub-population [83]. As mentioned previously, this algorithm did not perform well when faced with non-separable problems. To overcome this problem, several other static grouping methods were proposed. First, in [124], a pool of static decompositions, where each could potentially vary in size from a single variable sub-problem to a full length problem, were used to produce new solutions through random combination. Another decomposition method which statically decomposed a problem into m s-dimensional sub-problems, where $n = m \times s$, was introduced by van den Bergh and Engelbrecht in [138] which was shown to perform better than Potter and De Jongs 1-dimensional decomposition on several benchmark functions [83]. A later method, very similar to this, by Cao et al. [152], made use of a sequential sliding window to decompose problems and was compared to several random variable grouping methods, however, it was not compared against the method of van den Bergh and Engelbrecht or any other static variable grouping method for that matter.

(B) RANDOM VARIABLE GROUPING Random Variable Grouping (RG) is often employed to partially alleviate the problem faced by static variable grouping, whereby since groups are fixed throughout the optimisation process, if two interacting problem variables are placed into different groups on initialisation of the algorithm, there can be no way of them to be part of the same group at all during execution [83]. It does this by randomly selecting variable for each sub-problem [83], that is, the grouping of variables is changed throughout the course of optimisation. Two schemes for selecting the size of the sub-problems in RG are: (i) using a *fixed* size or (ii) *dynamically* changing the size throughout the optimisation process [83]; each of these will be discussed separately.

(A.i) Random Variable Grouping with a Fixed Sub-problem Size Here the number of sub-problems m is fixed and in each co-evolutionary cycle, randomly selects a set of variables s for each sub-problem such that $n = m \times s$. In this way, RG looks to improve the probability of combining any interacting variables into the same sub-problems [83]. A proof presented in [151], concludes that there is a relatively high probability that two variables which interact will be part of the same sub-problem during at least two co-evolutionary cycles, and thus further conclude that with little domain knowledge, RG is quite effective at capturing variable interactions. On the other hand, if the number of interacting variables is large, the probability of finding these variables together in the same sub-problem, for at least one co-evolutionary

cycle, remains very low [99]. The first CCEA using random grouping, and using fixed sub-problem sizes - was proposed by Yang et al in [150] implemented as part of their cooperative co-evolution with differential evolution (DECC-I) approach, where results showed that, particularly for non-separable problems, it outperformed CCEA using a static grouping approach. Later, the authors improved upon this algorithm with the development of DECC-G which makes use of adaptive weighting strategy during its co-adaptation process; it was found that DECC-G outperformed standard DE as well as DECC using a static grouping scheme [151].

(A.ii) Random Variable Grouping with Dynamic Sub-problem Size According to Ma et al. [83] one of the main disadvantages faced when using static grouping and random grouping with a fixed sub-problem size is that in order to correctly set the sub-problem size s, one must have some prior knowledge of the problem domain in question. For example, a small s is suitable if the problems being addressed are separable in nature, where a larger s is more suitable for use against non-separable problems so as to improve the probability that interacting variables will be grouped together [83]. Therefore, having an approach which dynamically tunes the value of s to the domain is desirable [83]. In [150], additionally to DECC-I, Yang et al. DECC-II in which given a predefined range, randomly tuned the value of s on each cycle [83]. The probability of selecting s from $S = \{s_1, \ldots, s_t\}$ is calculated based on its recent performance [83]. An approach presented in [57] (CCDECD) shows the applicability of the use of heuristic rules based on knowledge of the domain for the modification of s [83]. Several studies including [99, 100, 76] make use of a fixed s until no further improvements to solution fitness can be achieved at which time a new value of s is selected uniformly at random from $S = \{s_1, \ldots, s_t\}$ [83]. Additionally to this, in [103] the value for s is decreased gradually as the search progresses.

2.5.1.2 *Basic Algorithmic Frameworks for CCEA*

In this section we describe two general frameworks which underpin most CCEA implementations, namely; the single population CCEA and the multi-population CCEA. In the single population scheme, individuals interact with other individuals from the same population; whereas, for the multiple population scheme, individuals can interact with individuals from one or more different populations. Pseudocode for each of these schemes, derived from [106], is given in Algorithm 1 and 2 respectively.

Algorithm 1 Single Population CCEA

1: **procedure** SP-CCEA
2: *population* ← INITIALISE
3: $\mathrm{evaluators}$ ← select *evaluators* from the *population*
4: Evaluate each *individual* ∈ *population* by interacting with *evaluators*
5: **while** terminate == false **do**
6: $\mathrm{parents}$ ← select *parents* from *population*
7: $\mathrm{children}$ ← produce *children* from *parents* using variational operators (x-over/mutation)
8: $\mathrm{evaluators}$ ← select *evaluators* from *parents* + *children*
9: Evaluate each *individual* ∈ *children* by interacting with *evaluators*
10: Select survivors to be passed on to the next generation
11: **end while**
12: **return** solution
13: **end procedure**

Algorithm 2 Multiple Population CCEA

```
1:  procedure MP-CCEA
2:      for each (pop ∈ populations) do
3:          pop ← INITIALISE
4:          evaluators ← select evaluators from (populations - pop)
5:          Evaluate each individual ∈ pop by interacting with evaluators
6:      end for
7:      while terminate == false do
8:          for each (pop ∈ populations) do
9:              parents ← select parents from pop
10:             children ← produce children from parents using variational operators (x-
    over/mutation)
11:             evaluators ← select evaluators from (populations - pop)
12:             Evaluate each individual ∈ children by interacting with evaluators
13:             Select survivors to be passed on to the next generation
14:         end for
15:     end while
16:     return solution

17: end procedure
```

A few details have been omitted from both algorithms shown here. Firstly, it is common to find CCEA implementations which make use of a so called *archive*, a form of search memory maintained over generations i.e., throughout the whole evolutionary process [106]. Also, there is no indication about which specific interactions should be evaluated, how the results of any interactions can be combined in order to provide individuals a measure of fitness or how communication between populations is achieved when multiple populations are considered [106].

3

CHAPTER 3: CONTINUOUS OPTIMISATION BENCHMARK FUNCTIONS

3.1 INTRODUCTION

Firstly in this chapter, some of the main factors affecting continuous problem difficulty are discussed. In the next section, a short discussion of the decision to construct a new benchmark suite is provided. Finally in this chapter, we present a summary of the functions included in the new suite. Full descriptions of all functions included in the suite can be found in Appendix E.

The primary source used for the selection of benchmark functions in this suite was the highly cited review by Jamil and Yang [63]; cross-checked for the commonality of selected functions with other continuous benchmark function sets (see Chapter 3 Section 3.3). Others (i.e., Bent Cigar, Deflected Corrugated Spring, Inverted Cosine Wave, Levy, Rastrigin, Schwefel and Sum of Different Powers Functions) were found in various secondary sources [73, 93, 134, 36, 51]. One other reason for using secondary sources was simply to ensure that the function definitions and generated plots (see Appendix E) were correct and that corresponding definitions and plots were consistent between sources. However, another reason was that in order to construct a balanced suite of generally well known functions, the sets of functions such as those found in other benchmark suites, e.g., [133, 32, 47], could not be referenced in their entirety due to over/under representation of problems with certain features (see section 3.3 for discussion). Therefore, other - perhaps less well known functions - had to be selected where their definitions and outputs could be easily and reliably validated. Functions with discrepancies between their descriptions and output from several sources were therefore not used in order to help ease the replicability of the study being undertaken.

3.2 WHAT MAKES A FUNCTION DIFFICULT TO OPTIMISE

Several factors influence the difficulty in optimising an objective function [63]:

1. **Ruggedness** of the fitness landscape

2. **Dimensionality** of the objective function
3. **Separability** of the objective function
4. **Global Basin Structure**
5. **Variable Scaling**
6. **Search Space Homogeneity**
7. **Basin Size Homogeneity**
8. **Contrast Between Local and Global Optima**
9. **Plateau Sizes**

Each of these will be discussed in turn over the following few paragraphs.

RUGGEDNESS The ruggedness, or modality, of a function can either refer to: (i) the number of local optima, present in the landscape, or (ii) the number of global optima present in the landscape. Weise defines ruggedness simply as "From a simplified point of view, ruggedness is multi-modality plus steep ascends or descends in the fitness landscape". We focus on multi-modality in the local sense so we use the term multi-modality to this effect. If a search algorithm encounters these optima in the search for global optimality, and becomes trapped in (unable to escape) that region of space, this can have a negative impact on search progress [63]. Therefore, functions with a large number of optima (multi-modal functions) tend to be more difficult to search than functions with few optima (also multi-modal) or a single optima (Uni-modal functions). Local search strategies will tend to perform poorly on multi-modal functions, demanding the use of a global strategy.

DIMENSIONALITY When considering objective functions of multiple dimensions, optimisation practitioners often run up against the so-called 'curse of dimensionality', an idea first highlighted by Bellman in 1961 in [8]. The curse of dimensionality refers to the phenomena where an exponential increase of volume in a function space occurs as new dimensions are considered. If for instance a 1D function was to be sampled evenly 20 times in the range $[0, 1]$, we would of course have a sample of size 20^1. However to retain the same density of sample points in the 2D case, in terms of distance between consecutive points, we would require a sample of size $20^2 = 400$, $20^3 = 8000$ for the 3D case and a massive $20^4 = 160,000$ for a 4D sample. It is clear to see that for dimensions even higher than this, and likely including a smaller granularity of samples per dimension, that adequately sampling the available space quickly becomes intractable. The obvious consequence for search and optimisation is that

algorithms need to search over increasing numbers of possible solutions in the search for optimality - causing a rapid deterioration of solution quality obtained within a given time budget and the speed of convergence to the optimum solution when no budget is used. However, another less obvious consequence is that a search methodology that is found to be useful for problems in smaller dimensions may turn out to be almost useless when considering the same problems in higher dimensions. Therefore, given that real-world optimisation problems tend to be defined in highly n-dimensional space, optimisation methods that work well within some reasonably high range of dimensions are incredibly valuable.

SEPARABILITY A separable function is defined as a function of n parameters that can be rewritten as the product of n functions of one parameter each. That is, they can be easily partitioned into several sub-problems each with lower dimensionality and greater ease of searching [63, 12]. Combining the solutions to each sub-problem constructs a final solution.

There are two types of separable function related to the nature of this final solution construction: (i) additively separable functions and (ii) multiplicatively separable functions. Given a function F of n parameters $\{x_i, ..., x_n\}$, a function can be described as additively separable if there exist a set of functions $\{f_1, ..., f_n\}$ of one parameter, such that: $F(x_1, ..., x_n) = f(x_1) + \ldots + f(x_n)$. An example of such a function is that of the Rastrigin function (see section E.12). Similarly, given a function F of n parameters $\{x_i, ..., x_n\}$, a function can be described as multiplicatively separable if there exist a set of functions $\{f_1, ..., f_n\}$ of one parameter, such that: $F(x_1, ..., x_n) = f(x_1)f(x_2)\ldots f(x_n)$. The Alpine Function no.2 [63] (not included in our benchmark suite) can be described as being multiplicatively separable. In general non-separable functions tend to be harder to optimise than separable ones. The main reason for this is due to the fact that a separable function of n independent parameters can be optimised by n independent optimisation processes on each parameter [63].
Optimisation methods that can take advantage of the separable nature of functions tend to perform well. In order to increase the difficulty of functions for the purposes of benchmarking new and existing optimisation methods, and to consequently drive the development of ever more effective techniques, researchers have been making use of testbeds where the normally separable problem instances have been made non-separable through coordinate rotation by use of a rotation matrix [12, 114]. We do not consider functions that have undergone coordinate rotation in this thesis since (i) there are many inherently non-separable benchmark functions available, and (ii) for our purposes, it was useful to look at performances of algorithms against separable problems as well as non-separable as no comparative studies exist, to our knowledge, that show the behaviour of different metaheuristic algorithms for large-scale separable problems. Further, multiplicatively separable functions are currently beyond the

scope of the present study and so only functions that are additively separable are included in our benchmark suite. The reason for this is that due to the $\prod$ term present in these functions, some solution values returned would eventually exceed the maximum value of the internal representation when searching above some threshold dimensionality. For fairness and in order to avoid the introduction of bugs in our algorithms - if modifying the internal representation - the decision was made to avoid the use of this type of function.

GLOBAL BASIN STRUCTURE For a local optimum $\hat{x}$, a *basin* or *basin of attraction* consists of the set of configurations from which a gradient walk, or steepest descent, will reach $\hat{x}$ [12]. Therefore, since every basin contains at least the local optimum itself then every basin can be considered as being non-empty [12]. The global basin is a similar structure to local basins, however at a larger granularity instead considering the set local basins able to reach the global optimum by means of a suitable basin-hopping algorithm. For example, a highly multi-modal problem such as Rastrigin's Function (Section E.12) forms a clearly defined parabola of local minima and maxima when viewed from a distance tending towards the global optimum [12]; thus giving the impression that the non-linear Rastrigin function is convex. Problems without a global basin structure are considered harder to solve as the global optima can appear at any point in the search space with no overall guidance as to where it is located [12]. Some algorithms such as ILS and other basin-hopping approaches can take advantage of global basin structures in order to converge on the global optimum.

VARIABLE SCALING Variable scaling refers to the difference in scale with respect to the search space dimensional bounds. This means that it would be necessary to perform smaller steps in some dimensions and larger steps in others [12]. Even when the bounds on the search space are uniform across all dimensions, the problem may still behave very differently in certain dimensions than for others [12]. Algorithms such as CMA-ES (section 4.7) are particularly suited to this type of problem [12].

SEARCH SPACE HOMOGENEITY Many of the standard benchmark functions developed for testing the effectiveness of optimisation approaches are often designed with a homogeneous search space structure - often defined by an individual simple formula [12]. However, in the real world it is rare to find problems with such stuctures; instead, problems tend to have far more complex structure [12]. To combat this issue, many benchmark sets consist of functions that are composed of several functions blended into one such that the search space of the resulting function has different characteristics in different areas [12] - much more like how real-world problems might appear.

basin size homogeneity The size of global basin is known to play a role in determining the hardness of a problem [12]. Torn et al. in [136] emphasise this fact by stating that if the basin is large the global optimum is easier to detect and the problem is easier to solve that for those where the global basin is far smaller. However, problems can also be caused in cases where many local optima basins exist. For example, many algorithms for use with multi-modal functions, such as niching methods, make the general assumption that basin sizes are mostly similar and as a result use appropriate distances to differentiate between the basins; of course, this means that if the variance of basin sizes is large then these techniques will perform poorly [136].

contrast between local and global optima This problem characteristic refers to the quality, or height, differences between the local and global optima when compared to the average quality of the search space as a whole. If this contrast is high then good locations in the search space are more easily detected [136].

plateau sizes Plateaus make optimisation problems harder as they do not provide any directional information required by trajectory-oriented approaches - large plateaus effectively partitioning the search space making it difficult to transition from one area to another [136].

3.3 constructing a custom function suite versus using existing benchmark function suites

As there is a need for a balanced suite of benchmark functions for this study, i.e., a balance of the features under investigation, one was constructed by sampling from the large quantity of existing - and widely used - functions. Previous attempts have been made to construct similar standard benchmark suites, such as: for continuous optimisation competitions like CEC-2005, CEC-2010 and CEC-2013. As stated in Chapter 2, Section 2.4.5.3, it is recommended that one should try to make use of existing and widely used benchmark sets, such as these, rather than constructing a separate suite.

However, the need for a balanced suite of functions restricted the direct use of the existing test suites; although many of the functions present in these suites have been included. There were several reasons for this. Firstly, the current focus of the continuous global optimisation community is to find approaches capable of solving problems which are resistant to current methods [21]. As such, and since effective algorithms currently exist for solving classes of

problem with uni-modal, low dimensional and separable characteristics, these are rather under-represented in these suites compared to problems with more resistant characteristics, such as: multi-modal, high-dimensional and non-separable [21]. Table 3.1 provides a summary of the proportion of problems with certain features present in these suites as compared to the suite constructed for this study. The proportions are also presented graphically in Fig. 3.1 for convenience. The Black-Box Optimisation Benchmark (BBOB) benchmark suite is not presented, due to the difficulty in verifying the problem features possessed by the benchmark functions used; specifically, the features are not always explicitly stated in the published suite descriptors [32, 47]. However, Clerc in [21], from which I also verify the correctness of some of the feature proportions found in these other suites for Table 3.1, provides data suggesting that this set is also too unbalanced for our purposes - in favour of multi-modal and non-separable functions.

Table 3.1: Summary of the Proportion of Problems With Certain Features Present in Various Existing Test Suites Versus the Test Suite Constructed for this Study

Feature / Suite	**CEC-2005**	**CEC-2010 (LSGO)**	**CEC-2013**	**CEC-2013 (LSGO)**	**Bespoke Suite**
Uni-modal	5/25 (20%)	8/20 (40%)	5/28 (18%)	7/15 (47%)	10/17 (59%)
Multi-modal	20/25 (80%)	12/20 (60%)	22/28 (82%)	8/15 (53%)	7/17 (41%)
Separable	2/25 (8%)	3/20 (15%)	4/28 (15%)	4/15 (27%)	8/17 (47%)
Non-separable	22/25 (88%)	2/20 (10%)	24/28 (85%)	3/15 (20%)	9/17 (53%)
Partially-separable	1/25 (4%)	15/20 (75%)	0/28 (0%)	8/15 (53%)	0/17 (0%)
Less than 10 Dimensions	0/25 (0%)	0/20 (0%)*	0/28 (0%)	0/20 (0%)*	17/17 (100%)

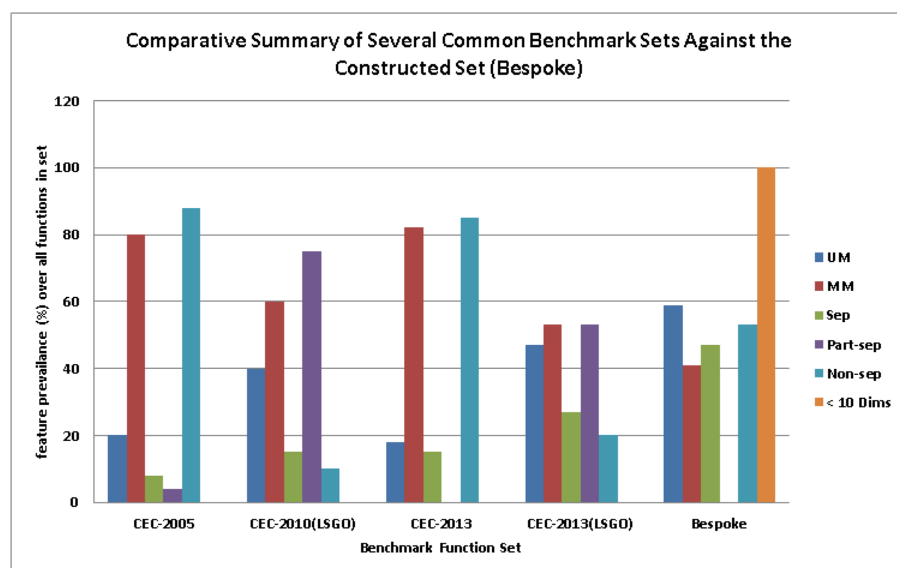


Figure 3.1: Barchart Summarising Prevalence of Problems With Certain Features Present in Each Benchmark Test Suite

From Table 3.1 and Fig. 3.1, we can see that the bespoke function suite used for this thesis is indeed more evenly distributed in terms of the problem featured of interest (i.e., separable/non-separable and uni-modal/multi-modal). Additionally, as we explore how different metaheuristics behave at different problem scales, it is required that low dimensional versions of our benchmark functions can be evaluated during our investigations. It can be clearly seen that every one of the published standard benchmark suites fails to meet this requirement, where for the 'bespoke' suite, any scale of instance for each function is possible.

3.4 FUNCTIONS

Appendix E provides details and definitions of the standard continuous benchmark functions we implemented and used in this thesis, where all included 2D function plots were generated using statistics package *R* through sampling the functions by a uniform step size in each dimension.

In order to eliminate positional bias in terms of the selected initial solutions when benchmarking algorithms, existing suites are commonly constructed from functions in which the coordinate system has been shifted with respect to the base function definition (e.g. 'CEC-2005 Special Session on Real-Parameter Optimisation' function suite [133]). In this way, the location of optima (including the global optima) can be modified whilst preserving the features and structure of the function. With respect to the experiments in this thesis, the issue of positional bias is addressed by selecting an initial solution in the search space uniformly at random for each individual run of an algorithm. Therefore, the use of shifted functions for the purposes of preventing positional bias is unnecessary.

The following section provides a table summarising key details of the functions included in the suite, including: domain of each function, location of the global minimum, value of the global minimum and a note of the features of interest (separability and modality) provided by each function.

3.4.1 *Summary of Functions*

Table 3.2: Summary of n-Dimensional Benchmark Function Suite

Function Name	**Domain**	**Location of Global Minima/ Minimum**	**Global Minima Value(s)**
Ackley Function	$-32.768 \leqslant x_i \leqslant 32.768$	$x_i^* = 0$	$f(x^*) = 0$
Alpine Function no.1	$-10 \leqslant x_i \leqslant 10$	$x_i^* = 0$	$f(x^*) = 0$
Bent Cigar Function	$-100 \leqslant x_i \leqslant 100$	$x_i^* = 0$	$f(x^*) = 0$
Brown Function	$-1 \leqslant x_i \leqslant 4$	$x_i^* = 0$	$f(x^*) = 0$
Chung-Reynolds Function	$-100 \leqslant x_i \leqslant 100$	$x_i^* = 0$	$f(x^*) = 0$
Deflected Corrugated Spring Function	$0 \leqslant x_i \leqslant 2\alpha^*$	$x_i^* = \alpha$	$f(x^*) = -1$
Exponential Function	$-1 \leqslant x_i \leqslant 1$	$x_i^* = 0$	$f(x^*) = -1$
Griewank Function	$-600 \leqslant x_i \leqslant 600$	$x_i^* = 0$	$f(x^*) = 0$
Inverted Cosine Wave Function	$-5 \leqslant x_i \leqslant 5$	$x_i^* = 0$	$f(x^*) = -n + 1$
Levy Function	$-10 \leqslant x_i \leqslant 10$	$x_i^* = 1$	$f(x^*) = 0$
Qing Function	$-500 \leqslant x_i \leqslant 500$	$x_i^* = \pm\sqrt{i}$	$f(x^*) = 0$
Rastrigin Function	$-5.12 \leqslant x_i \leqslant 5.12$	$x_i^* = 0$	$f(x^*) = 0$
Rosenbrock Function	$-30 \leqslant x_i \leqslant 30$	$x_i^* = 1$	$f(x^*) = 0$
Schwefel Function	$-500 \leqslant x_i \leqslant 500$	$x_i^* = 420.9687$	$f(x^*) = 0$
Sphere Function	$-5.12 \leqslant x_i \leqslant 5.12$	$x_i^* = 0$	$f(x^*) = 0$
Sum of Different Powers Function	$-1 \leqslant x_i \leqslant 1$	$x_i^* = 0$	$f(x^*) = 0$
Sum Squares Function	$-10 \leqslant x_i \leqslant 10$	$x_i^* = 0$	$f(x^*) = 0$

4

CHAPTER 4: METAHEURISTIC APPROACHES USED IN THIS THESIS

4.1 INTRODUCTION

In this chapter we give some background to the metaheuristic approaches implemented for this study. We cover the essentials of each technique and include descriptions of common operators and algorithmic variations found in the literature. When discussing several different operator implementations for a metaheuristic, the operator described last in the corresponding section was used in our implementation of the metaheuristic. For clarity, the operators for each algorithm are given as follows:

- Random Mutation Hill Climb (RMHC)
  - Neighbourhood function (described in Section 4.1.3)
- Steady-state Genetic Algorithm (SSGA)
  - Selection: Tournament Selection
  - Crossover: 1-Point Crossover
  - Mutation: Single Gene Mutation (as described in Section 4.3.2.3)
  - Replacement: Replace-Worst

Following the background of each metaheuristic algorithm, we outline the metaheuristic implementations used throughout our experiments along with a brief outline of the implementation specifics - highlighting important design decisions. We also provide pseudocode representing the actual Java™ source code we produced or sourced externally. Finally, we describe the 'tunable parameters' for each metaheuristic optimised in the automatic parameter tuning phase (Chapter 6) along with their value ranges - as used by SMAC - and justification for their selection.

In the remainder of this section, we must provide some general details that affect multiple implementations, namely: the solution representation used consistently throughout all implementations and the neighbourhood function utilised by our single solution methods - RMHC and SA.

4.1.1 *Exposing Algorithm Parameters*

The selection of the parameters to be tuned by the selected automatic parameter tuning method - SMAC (see Chapter 5 Section 5.4) - for each of the algorithms used in this study were selected as being the most common parameters being tuned by practitioners as described in published works. Of course, more parameters for some of the algorithms could have been exposed beyond those commonly found in the literature; for example, the selection of the most appropriate boundary handling schemes for PSO or the crossover or mutation operator used by the GA. However, there are a couple of reasons for not including these kinds of parameters in the tuning procedures.

Firstly, parameters of the type given in the example above go beyond simply modifying the behaviour of the algorithms through the changing of values; selecting one setting over another would change the internal mechanics of the algorithm itself, in effect producing a different algorithm altogether. As tuning takes place for this study for each problem-dimension pairing and any comparison between algorithms in also in this context, the returned results and analyses could be considered invalid as essentially different algorithms are being compared - even when considering plots of a single problem with dimensionality along the x-axis.

The second reason exposing more parameters was problematic is that of the scale of the data generation that was to be conducted; specifically, the correspondingly large amount of time and computational cost required. As will be discussed later in Chapter 6 Section 6.1, the number of independent tuning procedures for each algorithm-problem-dimension triplet amounted to 1700. With the worst-case maximum number of runs of the target algorithm being carried out by SMAC during these processes set to 1000 - the recommended default number of runs; the wall clock time required to complete even a single set of parameter configurations for one algorithm (i.e., each problem-dimension pairing) grew enough on a single machine to warrant parallelisation of these independent tuning processes on our departments computer cluster. Even then, finding a full set of configurations of an algorithm would take upwards of a day and worst of all, tuning sep-CMA-ES with SMAC - considering its higher computational cost (due to calculating the diagonal covariance matrices) - would take almost 3 days to find a complete set of configurations. With these numbers in mind, since SMAC - like all automated parameter configuration tools - is still bound by the 'curse of dimensionality', a parameter space volume increasing exponentially with each new parameter considered would call for a larger maximum number of target algorithm runs to ensure SMAC is still able to find effective configurations. Further, since SMAC is probabilistic in nature, more repeats of the entire tuning process would have to carried out to improve the likelihood

and therefore confidence that SMAC has - to the best of its ability - found the most effective configurations for each algorithm.

4.1.2 *Solution Representation*

For each of the implemented metaheuristics, a fixed double solution representation was used. Each parameter p in a solution of size d, where d is the dimensionality of the problem, can take on a value from the interval $[0.0, 1.0] \subset \mathcal{R}$ which is then later scaled by the objective function i.e., the benchmark problem, to fall within the interval $[UB, LB] \subset \mathcal{R}$ where LB and UB are the lower and upper bounds for the given problem dimension respectively. There is a couple of reasons to having the solutions encoded in this way. Firstly, the implementations are independent from the problem being solved; allowing the implementations to be reused on different problems - should time allow - e.g., real-world problems, without requiring significant modification therefore avoiding the possibility of programming error that may introduce noise in further comparisons. Secondly, the search space of the encoded representation and the decoded solution are equivalent, that is, the representation does not change the nature of the search landscape and its associated features. The encoded solutions are decoded within the objective functions using the following:

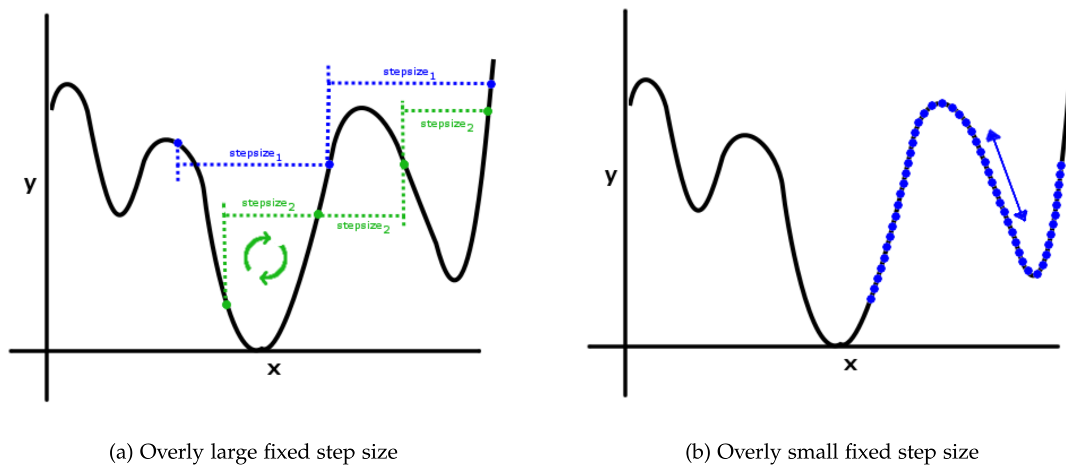
$$x'_i = LB + x_i \times (UB - LB) \tag{4.1}$$

Where x_i represents an encoded parameter i of an encoded solution representation x and x'_i is the decoded (scaled) parameter. Once scaled, the solution can then be used directly in evaluation of the objective function.

4.1.3 *Neighbourhood Function*

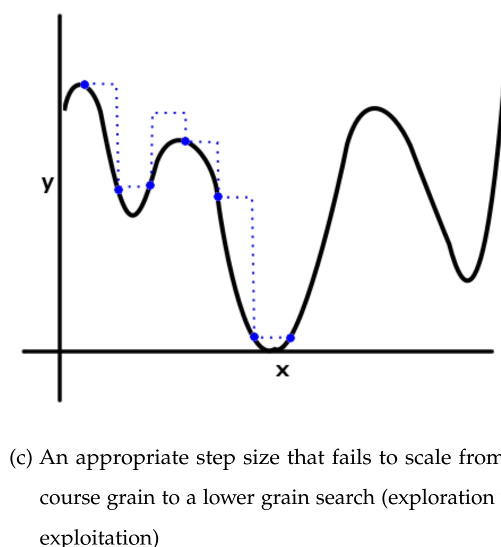
For those algorithms that require the use of a neighbourhood for generating new solutions in the search space from a given point, namely: random mutation hill climbing and simulated annealing, a common neighbourhood function is used. If working within discrete spaces - e.g., if the problems were combinatorial in nature - a neighbourhood can be quite easily defined as being a move to the next (or previous) discrete state in one or more dimensions. However, given that optimisation takes place in a continuous context in these studies - i.e., the notion of a next discrete step in any given direction is meaningless due to continuous nature of values in any given direction - steps within the space have to be defined in terms of a 'step size' variable. It is common to use a fixed step size tuned to a given optimisation problem, however this strategy has several disadvantages (see Fig. 4.1) and so, dynamic step sizes that scale with

search progress and/or solution state are sometimes used. When using a fixed step size, it is critical to the performance of the search algorithm that an appropriate value be selected as it imposes a structure (as viewed by the algorithm) to the underlying search landscape. A value that is too large can make the landscape appear flatter to the algorithm making it harder to traverse the landscape as well as hampering progress towards local optima - including global optima, and in general moving the algorithms behaviour closer to that of random search. On the other hand, if the step size value is too small, it may be more difficult to escape from local optima and by utilising small moves the region of global optimality may never be reached with an allocated computational budget, i.e., the algorithm is too slow to converge. These issues with fixed step sizes can be better illustrated through example as shown in Fig. 4.1 where we assume an approach that can accept worsening solutions such as SA.



(a) Overly large fixed step size

(b) Overly small fixed step size



(c) An appropriate step size that fails to scale from a course grain to a lower grain search (exploration vs. exploitation)

Figure 4.1: Effects of inappropriate fixed step sizes

In Fig.4.1a(a), an inappropriately large value for the step size can cause the search to 'skip over' local optima as with $stepsize_1$ or end up becoming trapped within the basin of a local optimum unable to reach the minimum like $stepsize_2$. In Fig. 4.1b(b) on the other hand, a step size which is too small makes very slow progress towards the minimum. Even with a suitably appropriate step size for the problem being solved it can still affect the quality of the final solution returned - even if the global basin has been discovered - this can be seen in Fig.4.1c(c). Although this is less likely to cause problems similar to the step size in Fig.4.1b(b) - due to the scale of the basin of attraction in relation to the step size - this example shows that there is still a possibility of this occurring as the search nears optimality.

Several methods exist for dynamically selecting step sizes and have been found to be quite successful albeit far more computationally expensive; however, for brevity we will not discuss them further.

However, another method exists, and is the one we use for our algorithms in this study, namely, the use of stochastic step sizes. We use this here as it combines the benefit of low computational cost when using fixed step sizes with some ability to scale appropriately as the search progresses, as found in dynamic step methods. In order to avoid the problems of overly large steps sizes - as described above - a step size is not simply selected before execution, bounded only by the boundary constraints of the problem, but rather a step size is selected uniformly at randomly on each iteration of the algorithm from a neighbourhood defined by a user supplied parameter and centred at the current solution. If the neighbourhood exceeds any bounds of the problem then the violating bound of the neighbourhood is set to that of the violated bound. All of this means that the neighbourhood can be larger than allowable when compared to one when using a strictly fixed step size, giving some of the benefits of a higher grained exploration of the search space, but since the defined neighbourhood essentially contain all possible neighbours perpendicular to the current solution within its bounds, a stochastic selection from among these neighbours can allow for a finer-grained search.

4.2 HILL CLIMBING ALGORITHMS

Hill climbing algorithms are a family of local search metaheuristics originally developed for searching within discrete combinatorial search spaces - although the algorithm can be adapted to work with a variety of different solution encodings [16]. As with many local search methods, hill climbing algorithms tend to maintain only a single individual throughout the entire search

procedure [115]. Although there are many varieties, many of which were developed to solve issues found with other hill climbing variations, the general functioning of these algorithms are quite similar and tend to differ mainly in how candidate solutions are sampled [96]. Hill climbing, or descent if dealing with minimisation, therefore attempts to discover solutions of better quality by iteratively selecting a member of the immediate local neighbourhood and testing whether or not there is an improvement - accepting an improving solution as the new current solution and rejecting worsening solutions [96]. Given this description, hill climbing methodologies can be said to be composed of two main phases: (i) a step function, that defines the neighbourhood of the current solution to be searched and (ii) an acceptance strategy determining what solutions are to be accepted as a new current solution. Although we discuss hill climbing algorithms here, these phases also apply to most other local search methods.

4.2.1 *Common Variations*

4.2.1.1 *Simple Hill Climbing Algorithm*

Simple Hill Climbing, also referred to as Next-best Neighbour and Next-descent/ascent Hill Climbing, is a deterministic search algorithm which involves the generation of neighbours sequentially one at a time, by the neighbour operator(s), and evaluating their objective value. If any improvement is found over the current solution, the neighbour becomes the new current solution without evaluating any more solutions of the previous current solution [35]. Pseudocode of simple hill climbing is shown in Algorithm 3 derived from the description of the approach found in [35].

Algorithm 3 Pseudocode of Simple Hill Climbing

```
1:  procedure HillClimb
2:      currentSolution ← RandomSolution
3:      terminategetsfalse

4:      while terminate == false do
5:          candidateSolution ← NextNeighbour(currentSolution)   ▷ Get next reachable neighbour
6:          if (Score(candidateSolution) ⩽ Score(currentSolution)) then
7:              currentSolution ← candidateSolution
8:          end if
9:          if (currentSolution.score is good enough) then
10:             terminate = true
11:         end if
12:     end while
13:     return currentSolution                     ▷ return best solution discovered
14: end procedure
```

4.2.1.2 *Steepest Descent Hill Climbing Algorithm*

In this variation on simple hill climbing, all possible neighbours of the current solution are considered - at least when considering discrete representations - and the best neighbour solution is selected to compete with the current solution [89]. If the best neighbour is found to be better than the current solution, then neighbour replaces it and the whole process begins anew [89]. If no neighbour is found that is better than the current solution then the algorithm has converged on a local optimum and the algorithm can either terminate here or the search can be restarted from a random position in the search space - keeping note, but making no further use of, the best solution discovered during the previous search [89].

Steepest descent can be extended to continuous domains, as with all hill climbing approaches, by introducing the notion of a step size parameter which defines a discrete neighbourhood for the current solution. For course, one must be careful of the value selected for the step size. If the size is too small the neighbourhoods can become very large - increasing computational cost, the speed of algorithm will be bounded by the small step size and once a local optimum is discovered the step size will be too small to jump over to other ‘hill/troughs’ [82]. If on the other hand the step size is too large, the algorithm may skip over local optima

and lose all information related to promising search trajectory; moreover, the algorithm will have a difficult time converging to a local optimum [82]. The consequences of the selected step sizes are best illustrated by Fig. 4.1 in Chapter 6.

Pseudocode of steepest descent hill climbing is shown in Algorithm 4 derived from the pseudocode found in [89]. This pseudocode includes random restarts which makes it more akin to a global optimiser such as those found in section 4.2.2.

Algorithm 4 Pseudocode of Steepest Descent Hill Climbing

```
1:  procedure SD-HillClimb
2:      t ← 0
3:      overallBest ← initialise

4:      while (t ≠ t.MAX) do
5:          foundLocal ← FALSE
6:          currentSolution ← RandomSolution
7:          while (!foundLocal) do
8:              neighbours ← getAllNeighbours(currentSolution)
9:              candidateSolution ← getBest(neighbours)
10:             if (Score(candidateSolution) ⩽ Score(currentSolution)) then
11:                 currentSolution ← candidateSolution
12:             else
13:                 foundLocal ← TRUE
14:             end if
15:         end while
16:         t ← t + 1
17:         if (currentSolution ⩽ overallBest) then
18:             overallBest ← currentSolution
19:         end if
20:     end while
21:     return overallBest                          ▷ return best solution discovered
22: end procedure
```

4.2.1.3 *Stochastic Hill Climbing Algorithm*

Stochastic Hill Climbing SHC is a local optimisation algorithm and is an extension of deterministic hill climbing approaches such as simple hill climbing (first-best neighbour) and steepest

ascent/descent hill climbing [16]. One of the most popular implementations of SHC, and the one utilised for this study, is that of Random Mutation Hill Climbing (RMHC) [16] introduced by Forrest and Mitchell in [35] where it was used in conjunction with other hill climbing approaches to investigate the behaviour of genetic algorithms (section 4.3). Pseudocode of a stochastic hill climbing approach - specifically the RMHC algorithm by Forrest and Mitchell as derived from [16] - is given in Algorithm 5.

Algorithm 5 Pseudocode of Stochastic Hill Climbing (RMHC)

```
 1: procedure RMHC
 2:     currentSolution ← RandomSolution

 3:     while terminate == false do
 4:         candidateSolution ← RandomNeighbour(currentSolution)
 5:         if (Score(candidateSolution) ⩽ Score(currentSolution)) then
 6:             currentSolution ← candidateSolution
 7:         end if
 8:     end while
 9:     return currentSolution              ▷ return best solution discovered
10: end procedure
```

For RMHC, the call to RandomNeighbour mutates a single locus of the current solution at random [16], in effect taking a single dimensional step. This contrasts with simple hill climbing, and is in fact the only difference, which generates each neighbour of the current solution in turn until one is found that is improving.

4.2.2 *Global Optimisation with Hill Climbing*

As hill climbing algorithms are local methods that maintain only a single solution at a time, when performing global optimisation, they can easily become trapped in local optima. Extension of the basic algorithms to search within multiple local neighbourhoods provides the needed diversification to avoid this problem. Approaches such as: random-restart hill climbing, local beam search and stochastic beam search can provide diversification through repetition of the search process - accomplished by various means each with different properties. Simulated annealing can also be considered as a global variation of hill climbing utilising non-deterministic search, however, this is covered separately in section 4.5. The next few subsections will cover these other global hill climbing approaches in more detail.

4.2.2.1 *Random-restart Hill Climbing*

Another approach to improving on the quality of solution discovered by local search techniques such as hill climbing is to simply restart the search from a different randomly selected initial point in the search space. This is the basis of random restart hill climbing, the motivation being the hope that the global optimum will eventually be found through trial and error. Again there are several variations, each determined either by how a restart of the algorithm is triggered - after a given length of time or until the algorithm terminates naturally - or whether a fixed number of restarts are used or not. These variations are not mutually exclusive and can be combined. An obvious disadvantage of this approach is that it requires more time and computational resources than a single repeat of the standard hill climbing algorithm used, increasing linearly with the number of restarts required. Another problem with this approach is that as the problem instance size increases, random sampling has a much reduced chance of finding a point in the search space that would lead the algorithm to converge on a solution of higher quality [80] - this chance reducing further as the current solution quality approaches optimality. Therefore, reaching these higher quality solutions requires more of a biased sampling which can be facilitated by stochastic search [80]. Despite its issues, random-restart hill climbing is fairly easy to implement, requiring only one or two additional parameters, an additional loop surrounding the main hill climbing algorithm and some very minor modifications e.g. keeping track of the overall best solution globally rather than returning the best overall solution from a single local repeat.

4.2.2.2 *Local Beam Search*

Instead of making use of only a single solution as in local hill climbing algorithms, local beam search maintains a population of k solutions. Here, k solutions are randomly generated over the search space and all members of their neighbourhoods are generated [127]. The k members over all neighbourhoods are selected and the algorithm begins a new iteration. Although this might appear as though several repeats of hill climbing are being performed concurrently, the repeats run independently and no information is shared between search processes; on the other hand, local beam search does share information between localities by considering all neighbourhood members to become part of the newly maintained set of size k [127]. In this way, areas of the search space that do not hold much promise can be rejected in order to redirect search efforts to more promising regions. This however leads to the disadvantage that over time the maintained set of k solutions can still converge to a local optimum as weaker but more diverse solutions and regions are quickly rejected.

4.2.2.3 *Stochastic Beam Search*

To address the disadvantage of local beam search eventually becoming stuck in local optima, stochastic beam search does not select the best k members from the whole list of k neighbourhood members, but instead selects k members randomly with a probability of selection given by an increasing function of a members quality [127].

4.2.2.4 *Iterated Local Search (ILS)*

The main basis of iterated local search (ILS) is that the search should not focus on the entire set of all possible feasible solutions S, but instead, the resulting solutions returned from some underlying heuristic - usually local search - in other words, the far smaller set S^* of locally optimal solutions s^* [81]. The search behaviour can therefore be categorised as a markov chain of locally optimal solutions returned from this underlying heuristic, leading to solutions better than those that can be found through random repeated trials - as with random restarts [80]. In practice, local search methods such as hill climbing have been the most commonly used search heuristic in ILS, but in fact, any optimisation algorithm can be used even if this is deterministic rather than stochastic [80, 81].

Lourenco, Martin and Stützl in [80] and [81] state two factors that determine whether an algorithm is an instance of ILS:

1. There must only be a single chain of solutions being followed - this then excludes the use of any population-based heuristics
2. The search for improving solutions must be taking place within a reduced search space defined by the output of the underlying search heuristic

ILS requires a mechanism for moving within the space S^* of local optima. Unlike random restart this mechanism - a perturbation of the current s^* - provides a sampling biased by increasingly better local optima. However, as Blum and Roli point out in [11], it is - in most cases - infeasible to construct such a neighbourhood structure as S^*; therefore the trajectory between local optima has to be performed without introducing the notion of a neighbourhood structure explicitly [11].

However, and as discussed in section 4.2.2.1, the bias provided by ILS will still help alleviate the issue faced by the random restart method where random sampling has a decreasing chance of discovering new points in the space that would lead to convergence on a new s^*. The perturbation of the current local optima should neither be too large or too small

[80]. Too large and the new intermediate solution s' will be random, we lose the useful bias and the algorithm will degenerate into a random restart-like approach [80, 11]. If conversely, the perturbabtion is too small, the search will often return back to s^* meaning that little of the space of S^* will be investigated [80, 11].

4.2.3 *Random Mutation Hill Climb: Implementation*

4.2.3.1 *Details and Description*

As detailed above, a random mutation hill climbing algorithm does not generate all improving neighbourhood solutions, as with steepest ascent/descent hill climbing - but rather 'mutates' a single solution parameter selected uniformly at random and also mutating the selected parameter by uniform mutation (Section 4.3.2.3) - and accepts the solution as the new current solution if and only if the new solution is improving. Furthermore, the improving neighbour is not accepted by a probability based on the magnitude of the improvement but is instead accepted immediately as the 'first choice' generated. As with the simulated annealing implementation (Chapter 4 - Section 4.5) this implementation makes use of the neighbourhood function described in Section 4.1.3. As such, the implementation only uses a single tunable parameter - *stepSize* - that defines the neighbourhood of a solution.

4.2.3.2 *Pseudocode*

Algorithm 6 Pseudocode of RMHC Implementation

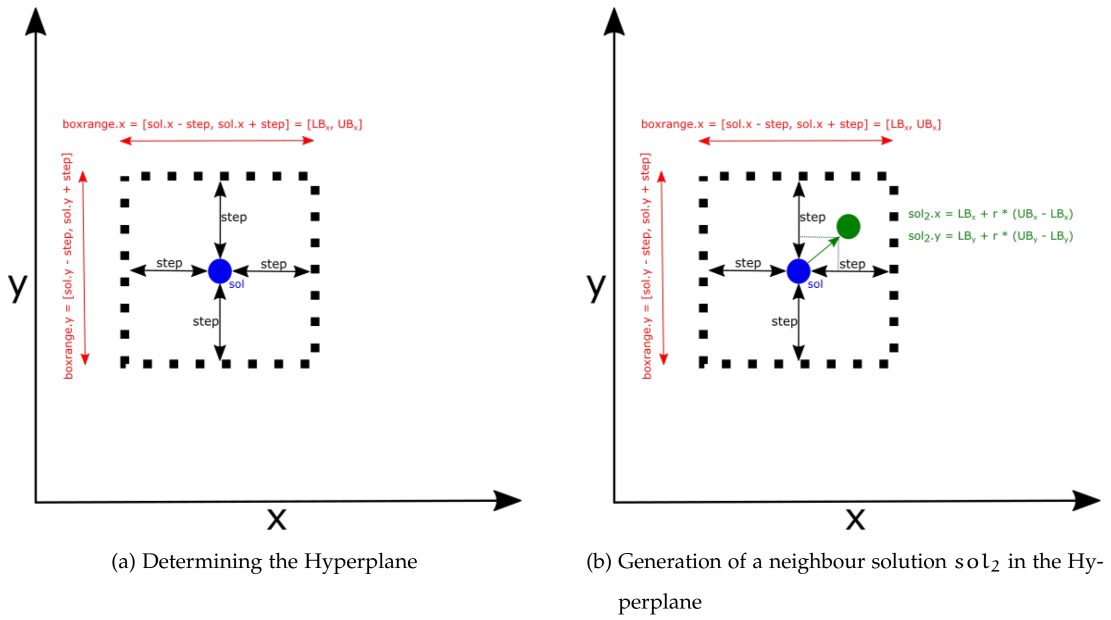
```
1:  procedure RMHC(stepValue, pSize)                              ▷ Main loop procedure
2:      problemSize ← pSize
3:      stepSize ← stepValue
4:      S ← generateInitialSolution(problemSize)
5:      S_score ← evaluateSolution(S)

6:      while (terminate == false) do
7:          N ← generateNeighbour(S, problemSize, stepSize)  ▷ get random neighbour of
    S
8:          N_score ← evaluateSolution(N)

9:          if (N_score < S_score) then
10:             S ← N
11:             S_score ← N_score
12:         end if
13:     end while

14:     return S                                       ▷ return best solution discovered
15: end procedure

16: procedure GenerateInitialSolution(problemSize)
17:     solution ← init
18:     for (i = 0 to problemSize-1 by 1) do
19:         solution[i] ← uniform random float ∈ [0.0, 1.0]
20:     end for
21:     return solution
22: end procedure

23: procedure GenerateNeighbour(solution, problemSize, stepSize)
24:     neighbour ← solution
25:     r ← uniform random float ∈ [0.0, 1.0]
```

```
26:     boundMin ← neighbour[i] − stepSize
27:     boundMax ← neighbour[i] + stepSize

28:     if (boundMin < 0.0) then                              ▷ Boundary handling
29:         boundMin ← 0.0
30:     else
31:         if (boundMin > 1.0) then
32:             boundMin ← 1.0
33:         end if
34:     end if

35:     boundRange ← boundMax − boundMin
36:     r ← uniform random float ∈ [0.0, 1.0]
37:     step ← boundRange ∗ r
38:     neighbour[i] ← boundMin + step

39:     return neighbour
40: end procedure
```

In this implementation, as with all the implementations described in this section, the procedure *evaluateSolution* is provided by the benchmark problem suite implementation presented in Chapter 3 of which all implementations make use. Therefore, *evaluateSolution* is not expanded in the pseudocode; however, the solution is decoded and evaluated by the benchmark suite in the manner described in section 4.1.2.

4.2.3.3 *Tunable Parameters*

Only one parameter is used that can affect the performance of the algorithm, specifically, *step size*. As discussed in section 4.1.3, this parameter defines the maximum radius of an axis-aligned hyperplane along each dimension of the search space centred on the current solution location. A neighbouring point - perpendicular to the current solution - from within the hyperplane is then generated stochastically by uniformly selecting a new value for one of the solution parameters. The result is a stochastically variable neighbourhood with the ability to scale with search granularity.



(a) Determining the Hyperplane

(b) Generation of a neighbour solution sol_2 in the Hyperplane

Figure 4.2: Definition of a neighbourhood of a solution sol using the stepsize parameter and the stochastic generation of a neighbouring solution sol_2

In the example shown in Fig. 4.1.3, a neighbourhood is defined by firstly determining its lower and upper bounds in each dimension i of the problem (Fig. 4.2a) - where $i \in [1, ..., d]$ and d is the problem dimensionality. Since the box is to be centred on the current solution S, the lower bound LB_i and upper bound UB_i of each box dimension is calculated by:

$$\begin{aligned} LB_i &= S_i - stepSize \\ UB_i &= S_i + stepSize \end{aligned} \tag{4.2}$$

where S_i is the position of the solution S in the search space in the ith dimension. To then generate a neighbouring solution N, a different uniform pseudorandom value $r_i \in [0.0, 1.0]$ is generated in a single random dimension i and is applied to the current solution in the range of LB_i and UB_i as follows:

$$N_i = LB_i + r_i \times (UB_i - LB_i) \tag{4.3}$$

Through initial experimentation using the neighbourhood function as shown in Fig. 4.1.3, generating neighbours along a single dimension only i.e., directly perpendicular, was found to be a more effective strategy than those generated through perturbing along several dimensions. Thus, we opted to use the single dimensional version for each of the algorithms making use of a neighbourhood function.

4.3 GENETIC ALGORITHMS (GA)

Genetic algorithms are a population-based evolutionary algorithm, developed and introduced by Holland in the 1960's. Genetic algorithms (GA) are inspired by natural evolution and genetics [17]. In fact, Holland specifies in [54] that genetic algorithms are based on the classic view of chromosomes as a sequence of genes; a view that eventually led to the development of mathematical genetics founded by Fischer in [34] [1], which provide formulae that describes the rate of spread of given genes throughout a population. A GA is therefore a more generalised and computer executable equivalent of the formulation used by Fischer in mathematical genetics [54]. A typical implementation of a genetic algorithm might consist of the following:

1. A population
   - A set of candidate solutions to a given problem being solved
2. An Objective Function or "Fitness"
   - A measure of quality used to determine good solutions from not so good solutions in the population
3. A Selection Operator
   - A strategy of determining which individuals in a population, at a given time, are allowed to perform crossover and which ones can be carried across to the next generation. There are many such strategies.
   - Solutions which are among the fittest in a given population are able to bypass the selection procedure employed, whereby a small portion of the very best solutions in the population are brought brought across to the next generation, typically without undergoing any changes - mimicking the natural concept of 'survival of the fittest'. This is known as elitism or elitist selection.
4. A Crossover Operator

1 First published in 1930

- Also referred to as a recombination operator, this is a method of combining the existing genetic information from two 'parent' solutions together in order to create one or more 'child' solutions. In search terms, crossover represents the main source of 'exploitation' during a search i.e., along with selection operators, it facilitates the convergence of the population towards a local optimum
- Typical methods of combination include: uniform crossover, one-point crossover and two-point crossover. There are however a wide selection of possible alternatives.

5. A Mutation Operator
   - A means of ensuring that the population contains enough diversity so as to be able to better cover the space of possible solutions. Mutation in a search is the main source of 'exploration' i.e., it is considered a divergence operator which works against selection and crossover to help prevent premature convergence by allowing the search to escape local optima and explore the global search space.
   - Some selection strategies, such as elitism, help create less diversity in a population by only accepting better solutions as parents and replacing the worst solutions with their children - meaning that the population will eventually converge to sharing a common set of traits with the side-effect of causing the crossover operator to become ineffective.

The main generalisations of a GA from mathematical genetics, as discussed by Holland in [55] and [54], consist of:

1. A concern with the interaction of genes on a chromosome rather than assume that genetic alleles can act independently from the others [55, 54]
2. An enlargement of the set of available genetic operators that can be used, over and above the mutation operator used almost exclusively in mathematical genetics. [55, 54]

From the first generalisation, the fitness function becomes complex and non-linear which quite often cannot be approximated usefully through the summation of the effects from different genes [55]. The second generalisation places more emphasis on the genetic operators, in particular on crossover [55]. Holland considers crossover to be essential to the success of a GA since crossover is an operation that is carried out in all mating organisms in nature as opposed to mutation whereby the mutation of a given gene takes place at a far lower frequency (less than 1 in a million individuals) [55]. The effect of crossover within a GA will be discussed more fully in section 4.3.2.2.

4.3.1 *The Basic GA Procedure*

A basic instance of GA is illustrated by pseudocode given in Algorithm 7 and the process flowchart shown in Fig. 4.3 and can be implemented as follows:

1. A population of candidate solutions is maintained by the algorithm, each one representing a potential solution to the given problem. Each individual solution is encoded as a number of variables, each representing some feature of the problem being optimised, whose values are taken from some predetermined alphabet - for example, binary, integer and real numbers - (Alg. 7: line 2)[2]

2. Each individual in the population is assigned a fitness score representing the quality of the solution, i.e., how well it solves the problem according to the objective function, which ultimately determines whether or not the solution will be selected for crossover - (Alg. 7: line 4)

3. A number of solutions are selected according to some selection strategy. These strategies can often be categorised as being 'elitist' or 'non-elitist', however, in practice selection strategies tend to fall within a spectrum with selection strategies being described as more or less elitist than others. A common strategy that can be said to fall at the centre of this spectrum is tournament selection described in section 4.3.2.1 - (Alg. 7: line 6)

4. The selected solutions are recombined, by application of a crossover operator, to create one or more candidate (child) solutions that somehow inherit components from multiple parents - (Alg. 7: line 7)

5. Given some probability, typically very small, each generated child solution may undergo mutation whereby one or more of its genes is modified in some way - (Alg 7: line 8)

6. Child solutions are then replaced into the population either by replacing other - often less fit - individuals or by generating a new population using the child solutions - (Alg 7: line 9)

In this procedure, steps 2 to 5 represent the main GA loop (Alg 7: line 3) and are therefore repeated until a user specified termination condition is satisfied.

2 The individual solutions in a GA are designed to be analogous to natural genetic structures; therefore, chromosomes (solutions) are composed of a number of genes (representing problem variables/parameters. This terminology amongst others from evolutionary genetics is commonplace in the study of genetic algorithms and some other evolutionary algorithms.

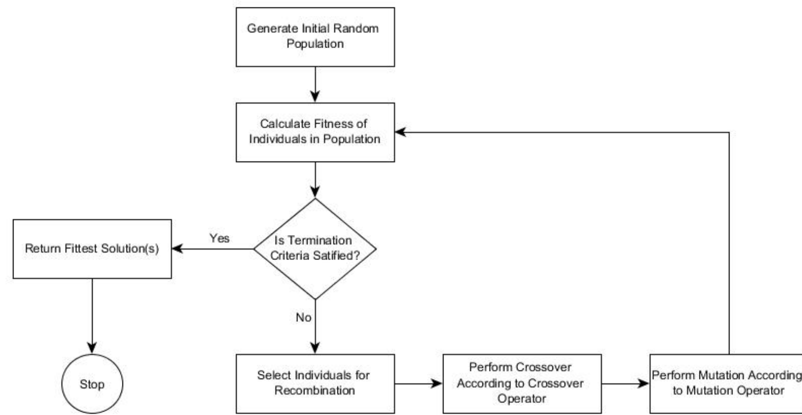


Figure 4.3: Basic operation of a typical genetic algorithm implementation

Algorithm 7 Pseudocode of a Basic Genetic Algorithm

```
1:  procedure GA
2:      population ← initialisePopulation
3:      S_best ← ∅
4:      terminate ← false

5:      while terminate == false do
6:          evaluatePopulation(population)
7:          S_best ← getBest(population)

8:          parents ← selectParents(population, numberOfParents)
9:          children ← crossover(parents, numberOfChildren)
10:         children ← mutation(children)
11:         population ← replace(population, children)
12:         if (S_best.score is good enough) then
13:             terminate ← true
14:         end if
15:     end while
16:     return S_best                          ▷ return best solution discovered

17: end procedure
```

4.3.2 *Genetic Operators*

GAs work on three simple rules of genomic evolution analogous to those observed in natural evolutionary genetics; referred to as the genetic operators of the algorithm. Each operator, namely: selection, crossover and mutation [29], has a specific role to play within the evolutionary process and work in conjunction to maintain the course of improvement within the population over time as well as maintaining the stability of the system. Since we focus on continuous (real-valued) problems in our study, all genetic operators will be discussed in terms of a real-valued encoding scheme.

4.3.2.1 *Selection*

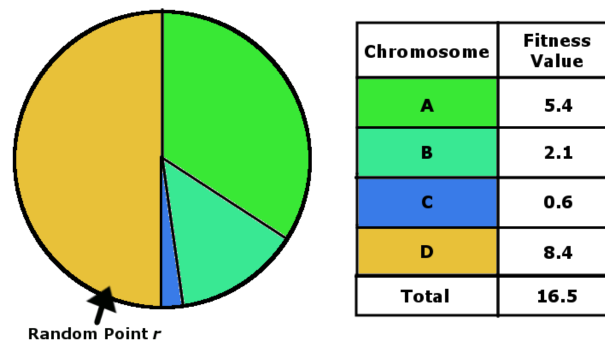
The responsibility of the selection operator in a GA is to choose a number of solutions from the population to become parents in order to produce offspring that will make up the next generation [29]. In this way, it acts as the force that increases the mean quality of the solutions in the population [29]. Parent selection in evolutionary algorithms such as GA is typically a probabilistic procedure - one in which the solutions of higher quality have more chance of being selected as parents [29]. However due to this probabilistic nature of selection, lower quality solutions typically have a small - but positive - opportunity of being selected; this helps to maintain diversity in the population otherwise the search can quickly become trapped in local optima [29]. Over the following paragraphs, we briefly describe the selection operator used in the GA studied for this thesis - namely, Tournament Selection - however, firstly, a couple of other common selection operators found in the literature are presented for comparison purposes: Fitness Proportionate Selection and Rank-based Selection.

(A) FITNESS PROPORTIONATE SELECTION Each solution in the population becomes a parent with a probability proportionate to its fitness [77]. Higher quality solutions have more chance of being selected to propagate their information to the next generation than those of lower quality. Therefore this strategy applies a selection pressure to those individuals in the population with higher fitness - to drive further progress. Two fitness proportional selection methods are given in the literature, namely: Roulette Wheel Selection and Stochastic Universal Sampling (SUS) - with the latter being developed to solve limitations found for the former - however, for brevity, we will only be discussing Roulette Wheel Selection here.

(A.i) Roulette Wheel Selection In Roulette Wheel Selection RWS, a probability of selection, proportionate to fitness value, is first determined for each solution in the population given by the equation [77]:

$$p_i = \frac{f(x_i)}{\sum_{j=1}^{n} f(x_j)} \tag{4.4}$$

where, $f(x_i)$ is the fitness of the solution x_i and n is the total number of solutions in the population. This equation provides, for each solution, a real valued probability in the range $[0, 1]$. Considering each of these probabilities as corresponding to a portion of a roulette wheel, a random number from the range $[0, 1]$ is generated representing a roulette ball landing in a portion of our hypothetical roulette wheel, selecting an individual for inclusion into the mating pool [22]. To gain an intuition of roulette wheel selection, an example is given in Fig. 4.4.



Chromosome	Fitness Value
A	5.4
B	2.1
C	0.6
D	8.4
Total	16.5

Figure 4.4: Roulette Wheel Selection example showing a single selection over a population of four individuals. Each individual, A,B,C and D represent a 32.7%, 12.7%, 3.6% and 51% portion of the roulette wheel respectively. Random point r selects the portion representing chromosome D similar to a hypothetical roulette ball

In this example, solution D has the highest probability of being selected, with solution C being the lowest. This is a useful feature of roulette wheel selection; although the solutions of higher fitness are more likely to be selected, it does not exclude the possibility that some solutions of lower fitness may be selected to contribute to the next generation, thus preserving some diversity in the population. The sum total of all the fitnesses t is calculated and a random point r is selected in the range $[0, t]$. The fitness of each individual in the population is subtracted from the value of r until $r < 0$. The individual whose fitness causes r to become negative is selected and the process is repeated for the total number of selections to be made. There is also nothing preventing the same individual to be chosen more than once and this can indeed happen. The repeated subtraction step in the above example, equivalent to a linear search over the population elements gives this implementation of roulette wheel selection a time complexity of $O(n)$; if, however, an implementation were to make use of binary search,

this can be reduced to $O(\log n)$ [77]. Another possibility is to make use of the implementation outlined in [77] whereby "stochastic acceptance" is used to select an individual with a typical time complexity of $O(1)$ depending on the distribution of fitness weights.

Although a popular selection operator, roulette wheel selection carries some limitations. Firstly, the operator in its current form is not suitable for minimisation problems or problems where fitnesses can take on negative values - although, both issues may be overcome with appropriate fitness scaling or transformation [5]. Another limitation is that since the likely variance of fitnesses in the population is initially high, the fittest solution can quickly take over the population leading to premature convergence [38, 5].

(B) TOURNAMENT SELECTION In Tournament Selection, a competition, or tournament, is held between between n solutions chosen randomly from the current population. The winner of each tournament is allowed to pass through to the intermediate population - either with or without replacement [3][38]. Usually the size of each tournament is kept small, this is due to the increasing percentage of diversity invariably lost as the size of tournament groups increase; likely resulting in premature convergence [38]. An example of tournament selection is shown in Fig. 4.5.

Tournament selection is popular amongst GA practitioners and has become the main selection method used in GA for several reasons [82]:

1. Unlike fitness proportionate selection, tournament selection is readily compatible with minimisation problems, fitness functions that may return negative values and generally any other particulars of the fitness function [82]
2. The selection pressure is easily controlled by changing the size of the tournament [82]; larger sizes make it harder for weaker solutions to compete as the chance of much better solutions being included in a tournament increases with group size
3. It provides a means of automatic fitness rescaling in that solutions of similar fitness near optimality are not considered as equal - as would be the case with fitness proportionate selection and SUS [82]

In terms of tournament sizes, a typical tournament size is $t = 2$ [82] however for certain representations it is common to be a bit more selective by using a larger tournament size [82]. It is also worth noting that $t = 1$ is equivalent to random selection, however, if $t \geqslant p$, where p

3 With replacement means that a selected individual can be selected more than once from the population meaning that subsequent tournaments have the possibility of containing a given individual from a previous tournament. Conversely, without replacement means that once an individual has taken part in a tournament it is no longer allowed to take part in subsequent ones.

is the population size, the probability of the fittest solution in the population being part of the tournament approaches 1.0 and so effectively selects the fittest solution on each tournament [82].

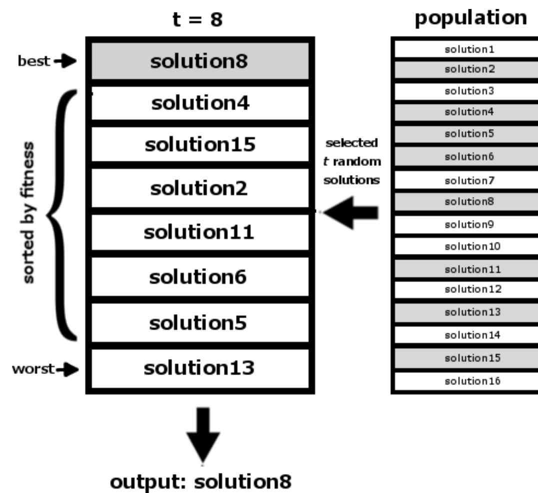


Figure 4.5: Tournament selection, where t = 8 (tournament size)

4.3.2.2 *Crossover (Recombination)*

Crossover is used to combine existing solutions into new, hopefully better, candidate solutions [29] and represents the main source of exploitation in the algorithm [104] - as such, they encourage the eventual convergence of the GA to a optimum solution. Many techniques exist for recombining parent solutions whose use is heavily reliant on the properties of the solution representation used; e.g., the specific encoding of the solution chromosomes and whether such solutions are of fixed or variable length. Several of the more common crossover operators will now be described.

(a) uniform crossover Here, each parent solution is able to contribute genes to child solutions at the gene level rather than at the segment level - as is the case with one and two-point crossover [29]. For certain problems, this added flexibility can far outweigh any disadvantages stemming from the destruction or dismantling of whole building blocks. A probability is defined, often referred to as the mixing ratio (usually set to 0.5), which determines which parent will contribute each corresponding gene value to the child solution [29] - and by doing so determining an approximate ratio of genetic material taken from each parent. For example, supposing the crossover ratio of Fig. 4.6 below is 0.5, then it follows that roughly half of the genes from each parent will contribute to the resulting child solution.

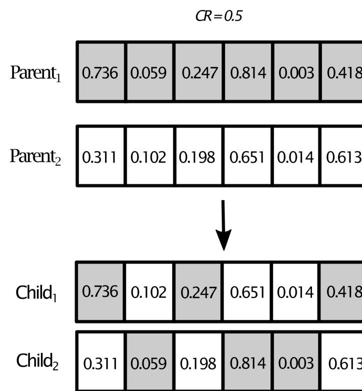
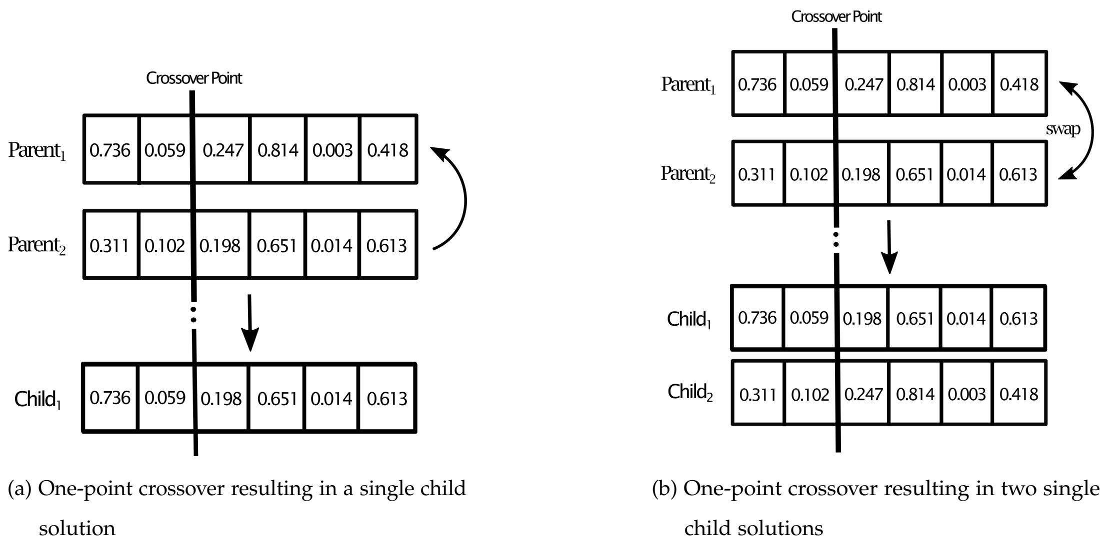


Figure 4.6: Uniform Crossover

(B) ONE-POINT CROSSOVER A single crossover point within a solution chromosome is selected at random [116]. When solutions in a GA are of a fixed length, a single crossover point common to both parents is used [65, 92]. Usually, two child solutions are produced from two parent solutions [116] however, one child can also be produced (see below). Given a fixed-length scheme, two variations of crossover can occur and are illustrated in Figs. 4.7a and 4.7b:

1. All genes after the crossover point in both parent solutions are swapped over between the two, resulting in two child solutions
2. All genes before the crossover point are copied from the first parent and all genes above the crossover points are copied from the second parent resulting in a single child solution



(a) One-point crossover resulting in a single child solution

(b) One-point crossover resulting in two single child solutions

Figure 4.7: One-point Crossover

4.3.2.3 *Mutation*

Although the two genetic operators discussed so far serve an important purpose in genetic algorithms, an algorithm implemented with only these two operators would simply perform an organisation of the genes of various solutions in order to construct, hopefully, better solutions [82] . This can place an upper limit to the quality of the solutions produced, as only gene values available to the population on initialisation could be arranged to make solutions. The problem here is that the population lacks diversity, without this the population would simply converge upon a local optimum with no means of escape [82]. A solution, and the solution adopted in genetic algorithms, is to allow the application of a third operator, mutation, which also mimics the corresponding process found in nature. Mutations are often found to be unhelpful, such as, the mutations of one or more genes causing anything from colour blindness to Turner syndrome. The crux of the matter however is that without this process in our own genes, although able to mix between individuals, human beings would be unchanged from our first appearance (aside from cross breeding with other primate species) and so would not have been able to evolve into the species we are today. Mutation is utilised in genetic algorithms for the same reason; without mutation new gene values are not created and, as mentioned previously, genes are simply swapped between the populous to create new children - whose possible optimisation potential is highly constrained to the available genes [82].

Mutation makes use of user-defined parameter - the mutation rate P_m - that determines which gene(s) will undergo mutation. As far back as 1989, a parametric study was carried out suggesting that a rate calculated as: $P_m = \frac{1}{n}$ represents an optimal mutation rate for many cases [111, 26]. Although this finding is in reference to binary-coded GA, our own studies show that improved performance can be found using this scheme with real-valued GA. Typically, mutation is carried out over the entire solution string, i.e., every gene has a chance to be mutated, however schemes where only one gene is mutated each time also exist such as the one described in [26].

A few common mutation operators common to real-parameter genetic algorithms will now be discussed along with a short note about mutation operators for binary representations.

(A) UNIFORM MUTATION This mutation operator, generates a single child solution given a single parent solution. It selects a gene at random, replacing its value with one selected uniformly at random between the upper and lower bounds specified for that particular gene

[128]. More formally; given a solution vector x, the operator generates a child x' by selecting a gene $i \in \{1, ..., n\}$ at random from the vector $x = \{x_1, \ldots, x_i, \ldots, x_n\}$ in order to generate a mutated vector $x' = \{x_1, \ldots, x'_i, \ldots, x_n\}$, where x'_i is a uniformly distributed random value selected from the range $(l(x_i), u(x_i))$ where $l(x_i)$ and $u(x_i)$ represent the lower and upper bound of x_i. The operator can play an importnt role in the beginning stages of the search process, where solutions should be allowed to explore more freely [91].

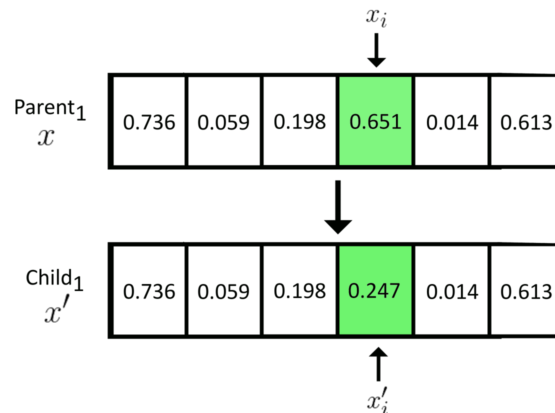


Figure 4.8: Uniform Mutation: where x is transformed to x'

(B) SINGLE GENE MUTATION In [26], the authors describe the use of a mutation operator for real-valued GA where a single gene in a solution is selected and mutated using polynomial mutation. In this thesis, we present - and in fact make use of - an alternative to this strategy where like the operator described above, the gene to be mutated is selected randomly (actually, uniform random) however, an additional random decision is made before this step that determines whether to mutate the solution at all. This additional random element of the operator is added by generating a uniform random number $r \in [0.0, 1.0]$ and comparing with the user supplied mutation rate P_m. If $r <= P_m$, mutation is carried out on the solution as described previously and the solution is left in its un-mutated state otherwise. The actual mutation is carried out using uniform mutation. Over a number of trials with various mutation operators and replacement strategies, this operator coupled with worst-replacement (see Section 4.3.2.4) was found to be greatly superior to the other variants trialled over the majority of our selected benchmark suite (Chapter 3) and comparable over the remaining functions.

4.3.2.4 *Replacement*

The replacement process is somewhat similar to that of parent/survivor selection, where selected solutions are distinguished based on quality [29], but is instead concerned with

the selection of replacement slots for child solutions to enter into the population - and thus, is used at a different stage in evolution. Where they differ however, is that, in practice, selection tends to be stochastic whereas replacement is usually deterministic [29] - although, replacement schemes such as 'reverse tournament selection' and random replacement with an element of stochasticity show that replacement strategies can exist on a spectrum between deterministic and non-deterministic approaches. Replacement is a process that is mainly relevant in steady-state genetic algorithms (section 4.3.3) where the entire parent population is not being replaced by the child population, but some replacement strategies can be used with generational genetic algorithms such as: age-based replacement where $\mu = \lambda$, $(\mu + \lambda)$ selection and (μ, λ) selection. Given that the size of GA populations, steady state or otherwise, tend to remain constant, selections therefore have to be made as to which existing solutions will be discarded to make room for newly generated solutions. Replacement selection, as mentioned, is often based on fitness values - with bias often given to those of higher quality - however, the concept of solution age is also frequently used [29]. Therefore, replacement strategies can broadly be categorised as: (i) fitness-based replacement and (ii) age-based replacement [29]; it is worth noting however that some fitness-based replacement strategies can also take age into account as well. Since replacement selection is so similar to parent selection, in theory any of the selection operators discussed in section 4.3.2.1 can be used for replacement [29], however, there have been several special replacement operators developed and now commonly in use by practitioners [29]. The following paragraphs outline some of the more commonly used fitness-based and age-based replacement schemes.

(A) Fitness-based replacement strategies

(A.i) Replace Worst Originally used in the GENITOR steady-state GA [144], this replacement strategy selects the λ worst solutions in the population to be replaced [29] - where λ refers to the number of child solutions that have been produced. Removing the worst solutions from the population in this way will lead to fast improvements to the mean fitness of the population but can also quickly lead to premature convergence [29] as much needed diversity is purged from the population, reducing the takeover time of the best solution and significantly increasing innovation time. Therefore, this replacement strategy is mostly used in situations where the population is large and/or the detection and pruning of duplicate solutions is being carried out [29].

(A.ii) $(\mu + \lambda)$ Selection This replacement strategy comes from evolution strategies and in general refers to the merging of the parent population μ with the child population λ which

is then ranked according to fitness. The best μ individuals are then chosen to form the next generation [29].

(A.iii) (μ, λ) **Selection** Also originating from evolution strategies, this replacement strategy is made up of both an age component and a fitness component where typically, $\lambda > \mu$ children are generated from a set of μ parents [29]. The age component of the strategy means that the entire parent population will be deleted so that no individual lasts beyond a single generation [29]. The remaining λ population are then ranked according to their fitness and the best μ solutions go on to form the next generation [29]. Eiben and Smith note that in evolution strategies, this replacement operator is often preferred over $(\mu + \lambda)$ as: (i) since all parents are discarded in (μ, λ) it could allow escape from small local optima and can therefore be useful in multimodal search spaces containing many local optima [29], and (ii) if the fitness function is non-stationary i.e., it changes over time along with the location of the global optimum, $(\mu + \lambda)$ selection preserves those solutions relevant to an outdated optima and so is not able to track the new optima as well [29].

(B) AGE-BASED REPLACEMENT STRATEGIES The general concept of age-based replacement is that the fitnesses of solutions are not used in determining which solutions will be replaced and instead designed in such a way as to have each solution exist for the same number of algorithm iterations [29]. Eiben and Smith point out that since age-based replacement is not reliant on fitness values, there is the possibility that the mean or best fitness of a given iteration will be lower than the preceding iteration [29], but point out that, although counter-intuitive, this may actually be of some benefit should the population be centred near to a local optimum and if a decrease in the mean or best fitness between iterations does not happen too often [29].

(B.i) Random Replacement The naive Random replacement involves selecting, uniformly at random, a solution to be deleted from the population. At first glance it may not appear as if random replacement belongs to the class of age-based replacement strategies, or fitness based strategies for that matter, however, Eiben and Smith [29] suggest that this strategy has the same mean effect, meaning that on average individual solutions live for μ iterations - where μ is the size of the parent population - giving us some concept, however probabilistic, of age. Eiben and Smith further state that the random replacement strategy is not recommended since experiments by De Jong and Sarma in [24] between a steady-state GA using random replacement and a comparable generational GA showed that the steady state GA exhibited more variance in performance [29]. A reason for this was given by Smith and Vavak in [125]

stating that random replacement is far more likely to lose the best solution in the population than 'delete-oldest' replacement [29] - a comparable age-based replacement strategy.

(B.ii) Delete-oldest Replacement This operator represents the steady state GA equivalent of the simple genetic algorithm (SGA) age-based replacement strategy - where the number of offspring equals the number of parents, i.e., $(\mu = \lambda)$ and so the child population entirely replaces the parent population so that no one individual survives for more than a single generation [29]. In a steady state GA where the child population is smaller than the parent population $(\lambda < \mu)$ age-based replacement instead forms a first-in-first-out (FIFO) queue [29]. As such, the first λ solutions at the head of the queue will then be replaced on each generation.

4.3.3 *Steady State GA vs. Generational GA*

The first occurrence of a steady state genetic algorithm was that of the GENITOR algorithm [144] developed by Whitley and Kauth in 1988. The main difference between a generational GA (GGA) and a Steady State GA (SSGA) such as GENITOR is how the offspring in each iteration enter the population. In generational GA, new offspring are created from a subset of the individuals in a population through the mechanisms of crossover and mutation and form a new population which replaces the old population [140]. In SSGA, typically a single child solution is generated which replaces a single individual in the current population chosen by a replacement strategy. SSGA is computationally equivalent to GGA but since only one individual is replaced in each iteration, it typically takes more iterations (generations) for the algorithm to converge; however, the computational costs associated with genetic operations in SSGA are far lower and SSGA itself is much simplified over GGA [113].

In GGA, if the new population is to include some members of the previous population this is determined by a variable G often referred to as the 'generation gap' [24] as originally defined from a set of empirical studies of overlapping generations by De Jong in [25]. However, around the same time, Holland was performing analyses on two reproductive schemes: (i) R_1, replaced a single individual in the current population at random with the child of a single parent using fitness-proportionate selection - a likely precursor to SSGA developed and popularised by Whitley in [144] - and (ii) R_d in which the expected number of offspring from parents selected deterministically were produced which replaced the entire current population [24].

Since a member of the population is to be replaced by a child solution on each iteration, another step is required in each iteration; namely, the selection of a solution to be replaced. No such step is required in GGA as the whole population is typically replaced, at least for

those implementations with no overlapping generations. As with other genetic operators there are many strategies for selecting a replacement slot. Common methods include: random replacement and worst replacement. These will be given more attention in section 4.3.2.4.

The SSGA model has been widely studied and applied in various contexts [29]. In [140] an SSGA and GGA were compared in terms of their ability to work in non-stationary environments, that is, their ability to adapt to the position or a dynamic optimum. An SSGA using a "delete the oldest" replacement strategy was shown to outperform an equivalent GGA in terms of both online and offline optimisation. The authors attribute the success of SSGAs for these applications to the fact that unlike GGAs, since an SSGA is incremental and a newly generated child solution is immediately considered as part of the mating pool from the next iteration, the algorithm is more able to more quickly move towards the optimum at a much earlier stage in optimisation [140].

4.3.4 *Steady State Genetic Algorithm: Implementation*

4.3.4.1 *Details and Description*

The genetic algorithm implementation chosen is that of an SSGA (Section 4.3.3) that replaces only a single solution in the population with a single child solution generated from the recombination of two parents in any given iteration. The selection of parents in this implementation is achieved by application of 'tournament selection' which are subsequently recombined using one-point crossover to form the child solution. Mutation is accomplished through the use of a single-gene uniform crossover (Chapter 4 - Section 4.3.2.3); as the solution representation used (as in section 4.1.2) bounds each gene by the interval $[0.0, 1.0]$. If mutation of a solution is to be performed, a randomly selected gene from a mutation operation is set to a uniform random value selected from this interval. This type of mutation was decided upon empirically through performance comparison with some more typical operators; such as those which are applied to the entire chromosome, and was found to perform best on average across the full benchmark suite [4].

A steady-state GA was selected due to its reduced computational complexity - both in terms of runtime and memory. It is well known that one of the most expensive steps in generational GA is that of population generation; ***therefore,*** if generational GA was tuned

4 The algorithm implementation was identical for each variant; the only variable being modified in the experiment was the choice of mutation operator

using our parameter tuning method (Chapter 7), the generation of parameter configuration sets at high dimensionalities by SMAC would likely take far longer than it did for an SSGA.

4.3.4.2 *Pseudocode*

Algorithm 8 Pseudocode of SSGA Implementation

```
1:  procedure GA(pSize, mutationRate, tSizeProportion)
2:      popSize ← pSize
3:      mutation ← mutationRate
4:      population ← initPopulation(popSize)
5:      tSizeProp ← tSizeProportion        ▷ Proportion of population to use in tournaments
6:      tSize ← 2 + floor(popSize ∗ tSizeProp)
7:      S_best ← getBest(population)
8:      while terminate == false do                          ▷ Main GA loop procedure
9:          evaluatePopulation(population)
10:         parent_1 ← TournamentSelect(tSize)
11:         parent_2 ← TournamentSelect(tSize)
12:         child ← OnePointcrossover(parent_1, parent2)
13:         child ← UniformMutation(child, mutationRate)

14:         replace ← getWorst(pop)
15:         population[replace] ← child

16:         S_best ← getBest(population)
17:     end while
18:     return S_best                              ▷ return best solution discovered
19: end procedure
```

```
20: procedure TournamentSelect(tournamentSize)
21:     tournamentSet ← random set of tournamentSize solutions from pop
22:     best ← tournamentSet[0]
23:     for (i=1 to tournamentSize-1 by 1) do
24:         if (tournamentSet[i] < best) then
25:             best ← tournamentSet[i]
26:         end if
27:     end for
28:     return best
29: end procedure

30: procedure onePointCrossover(parent1, parent2)
31:     r ← uniform random float ∈ [0.0, 1.0]
32:     chromosomeIndex ← ⌊r ∗ parent1.length⌋
33:     for (i = 0 to parent1.length-1 by 1) do
34:         if (i < chromosomeIndex) then
35:             child[i] ← parent1[i]
36:         else
37:             child[i] ← parent2[i]
38:         end if
39:     end for
40:     return child
41: end procedure

42: procedure uniformMutation(child, mutationRate)
43:     r ← uniform random float ∈ [0.0, 1.0]
44:     if (r <= mutationRate) then
45:         r ← next uniform random float ∈ [0.0, 1.0]
46:         geneIndex ← ⌊r ∗ child.length⌋
47:         child [geneIndex] ← r
48:     end if
49:     return child
50: end procedure
```

```
51: procedure getWorst
52:     worst ← 0
53:     for (i = 1 to popSize by 1) do
54:         if (population[i].score > population[worst].score) then
55:             worst ← i
56:         end if
57:     end for
58:     return worst
59: end procedure
```

```
60: procedure getBest
61:     best ← 0
62:     for (i = 1 to popSize by 1) do
63:         if (population[i].score < population[worst].score) then
64:             best ← i
65:         end if
66:     end for
67:     return best
68: end procedure
69: procedure evaluatePopulation(population)
70: end procedure
```

4.3.4.3 *Tunable Parameters*

The parameters used to control the performance of the algorithm in this implementation are as follows:

- Population Size
  - Type:- integer $\in [2, 250]$
  - Used by the initialization procedure to define the size of the population that will remain fixed throughout execution of the algorithm. A lower bound for this parameter of 2 is the minimum population size that would allow the genetic algorithm to be referred to as such; specifically, a population size of only one would essentially represent regular stochastic hill climbing (as crossover would not be possible). The upper bound was set as to fall in line with some of the other population based approaches represented here, as there seemed to be no relatively

general agreement to a reasonable range for population size in genetic algorithms. The default value for this parameter as used initially in SMAC is 100.

- Tournament Size Percent
  - Type:- float $\in [0.01, 0.7]$
  - Defines the size of the tournament groups used for parent selection as a proportion of the main population i.e., how many randomly selected solutions from the population are chosen to take part in the tournament. This is equivalent to setting an explicit number of individuals from the population - as is more typical - but provides prevention against the tournament size becoming larger than the main population when SMAC is actively tuning both tournament size and population size independently. In this implementation of the operator, n tournaments are run in order to select n parent solutions; thus, two tournaments are carried out to produce two parents - one from each tournament. This parameter also allows for the adjustment of selection pressure; a large group size will reduce the chances of weaker solutions being selected for recombination with the inverse being true of a smaller group.

    The upper bound on the value, of 0.7, was selected arbitrarily for the purposes of the automatic parameter tuning performed by SMAC. This value was judged to be a reasonable upper limit in order to prevent premature convergence resulting from the repeated selection of the very best solutions. Additionally, should an upper bound of 1.0 have been allowed, any parameter tuning process that produced a value for this parameter that lay very close to this bound would have caused the tournament selection procedure to be roughly equivalent (depending on the actual proportion of the population involved) to selecting the best solution in the entire population. Since our tournament selection creates tournaments with replacement and separate tournaments are carried out to select each parent, not only will the selection result in the same solution being selected far more often, but also that - without the action of mutation - the entire population would be quickly replaced with exact copies of a single 'super-individual'. Mutation would indeed continue to be applied however, becoming the primary means of search progress, making the GA process equivalent to a random search. It is clear that avoiding such problems makes comparison of this SSGA implementation with other approaches used in this study more straightforward and meaningful.

- Mutation Rate

– Type:- float $\in [0.01, 1.0]$

– Defines the rate, and therefore the probability, of a randomly selected gene being mutated or not. In other words, if a uniform random number generated for each mutation procedure is less than the mutation rate, then the gene is mutated. Given our implementation makes use of 'single gene' uniform mutation operator (Chapter 4 - Section 4.3.2.3); the effect on mutation if different than for other more common mutation schemes. Usually, lower values of this parameter results in less frequent mutation operations and in turn, results in slower overall convergence towards the global optimum with more risk of premature convergence to local optima - due to imbalance with crossover operations. Higher values on the other hand will result in faster convergence but will invariably produce too many perturbations to high quality solutions to progress further towards optimality. For this operator, better performance was found over more common operators where the rate of mutation is much higher - typically closer to 1.0. I hypothesise that the operator attempts to behave more like other typical operators - where the mutation rate give each gene in a solution the chance to mutate - but there appears to be some benefit to having no mutation at all. There is credence to this hypothesis in that early Evolution Strategy (EA) implementations made use of only a mutation operator - where crossover was omitted [89]. A mutation only GA variant was used successfully in [72] and was selected as it was found to be fast and reliable; however this was in the context of binary search spaces. We intend to investigate this further in terms of continuous search spaces in future work.

4.4 PARTICLE SWARM OPTIMISATION (PSO)

Particle swarm optimisation (PSO) is a population based metaheuristic introduced by Kennedy and Eberhart in [67] and is heavily influence by the computational models and rules for coordination developed at the time that simulated the flocking behaviour of birds and the schooling of fish [67]. The PSO algorithm maintains a population of solutions (particles), known as the swarm, each representing a point in n-dimensional space [7]. Initialising at random locations in the search space, the particles traverse the landscape in order to discover the global optimum of the given objective function [105]. This is in a way similar to a swarm of organisms in nature foraging for food or a new nesting location, the particle swarm in PSO makes use of the objective function - akin to biological feedback from an environment - to determine the quality or the amount of food at a given location; the swarm searches the space for the location containing the best solution - or most amount of food [105].

In models of flocking birds, the movement of each individual is determined both by (i) the influence of it's own perception of the environment and satisfaction of its goals and (ii) the influence of social pressures such as: separation from other individuals in the swarm, alignment to the average velocity of the flock and cohesion where the individual moves towards the average position of its neighbours. Similarly, in PSO the movement of a particle depends on its own velocity, good solutions or locations previous discovered by the particle and the good solutions discovered by its neighbouring particles [33]. This particle behaviour is modelled in PSO by having each particle maintain both the location of the best solution discovered so far by the particle itself - along with a corresponding objective value - and the location of the best solution discovered by all particles in its immediate neighbourhood [105].

4.4.1 *Neighbourhood Topologies*

The choice of neighbourhood topology used by a PSO algorithm can play a significant role in its overall behaviour. Investigations, such as in [66] by Kennedy, show that a particles neighbourhood topology interacts directly with the objective function of the problem being optimised, and further, that it significantly affects the performance of the swarm where the particular effect it has is dependent on the problem [87]. Therefore, where some topologies are well suited to certain functions, other functions can present problems to these same topologies [66, 87].

Several topologies common to PSO implementations include [87]:

1. Ring Topology
   - Also referred to as *lbest* (or local best) topology, here each particle in the swarm is only affected by the best performance of its k immediately adjacent neighbours. A common case is to have lbest $= 2$.
2. Star Topology
   - In this topology, social influence passes through only one central individual/hub, commonly selected at random, and is itself influenced by the rest of the population [68, 87]. On each iteration, every particle in the swarm moves towards only the central particle, and the central particle directs itself towards the best performing particle in the neighbourhood. This topology can be used to control the possibility of premature convergence, as since the population remains isolated from each other directly, the central particle - constantly adjusting its trajectory towards the best particle - effectively slows the transmission of good solutions throughout the population and thus maintaining diversity.
3. Von Neumann / Toroidal Topology
   - Here, each particle is connected to several others, specifically to the particles immediately above, below and on each side of it and any edges are wrapped [68].
4. Fully Connected Topology
   - Also referred to as a *gbest* (or global best) topology or *full* topology, this neighbourhood scheme prescribes the sharing of information amongst all particles in a population; that is, each particle is directly connected to one another. Here, all particles in the population are influenced and directed by the best solution found by any particle in the whole population. Kennedy and Mendes in [68] observed that fully connected (gbest) populations had a tendancy of converging more quickly that ring (lbest) populations and further, are more prone to local optima convergence. This however does not prevent it from being one of the most widely used topologies [87].

In the PSO algorithm used for this thesis, a fully connected topology is implemented since: (i) no additional computation or neighbourhood link management needs to be performed to make use of the topology, which would otherwise extend the required runtime of our already very expensive parameter tuning processes, and (ii) it allows the PSO implementation

to more closely resemble the neighbourhood structures of the other population-based metaheuristics being investigated; where operators typically choose members of the population at random (i.e., fully connected). This allows for a fairer comparison to be conducted between these approaches based on the merits of algorithmic approach rather than on the choice of neighbourhood structure.

In the PSO literature, such as in [48], it is usual to find the term *informants* used in place of 'neighbour' as used throughout the rest of the thesis as it more clearly represents the role of individuals and their neighbours 'informing' the direction of search. However, for consistency with the other metaheuristics described, the terms neighbour and neighbourhood will be used.

4.4.2 *Parameters of PSO*

4.4.2.1 *Particle Position, Velocity and Related Parameters*

Every particle in the swarm maintains for each dimension of the problem both its position within the search space and its velocity [105]. Velocity is added to the current position of the particle to move it to a new position of the search space - representing a particle step [105]. One of the first identified issues related to PSO was that of swarm explosion, where the swarm would gradually begin to drift apart and more regularly violate bounds on the search space [13]. This goes against the desire for a convergent algorithm. The solution proposed by researchers was to 'clamp' the velocity to a lower and upper bound specified by a range $[-V_{max}, \dots, V_{max}]$ [105] where V_{max} is the maximum velocity threshold. By clamping velocity, uncontrolled increases in magnitude from a particles current location can be avoided. Whenever the velocity of a particle in any dimension exceeds $\pm V_{max}$ it is set to the value of the violated bound as in[30]:

$$v_{id}(t+1) = \begin{cases} V_{max} & \text{if } v_{id}(t+1) > V_{max} \\ -V_{max} & \text{if } v_{id}(t+1) < -V_{max} \end{cases} \quad (4.5)$$

As with all tunable parameters in metaheuristics, and as per NFL (Chapter 2 Section 2.4.5.2), there is not a single optimum value of V_{max} to maximise performance in all conceivable problem space and so time should be taken to find a suitable value for each problem space being addressed [105].

The velocity is updated on each iteration of the algorithm based on the three components: (i) inertia weight ω, (ii) the cognitive component c_1 and (iii) the social component c_2 [10].

The inertia weight is used to specify how the current velocity influences the new velocity and also controls the scope of the search [10]. A higher value of ω favours exploration and lower values favour exploitation [10]; it is important to determine an appropriate balance, as with all metaheuristics, between these two actions and so an inertia weight should be careful selected to avoid either slow convergence or premature convergence. In a similar way, the cognitive and social parameters specify how much social and cognitive influence contributes to the new velocity [30]. When $c_1 < c_2$ the search is more biased towards global influences and biased towards personal influence when $c_1 > c_2$ [30]. Also, and as with ω, higher and lower values of c_1 and c_2 favour more exploration and exploitation respectively [30] and is determined by the velocity update formula [10]:

$$v_{id}(t+1) = \omega \times v_{id}(t) + c_1 \times r_1 \times (p_{id} - x_{id}) + c_2 \times r_2 \times (p_{gd} - x_{id}) \tag{4.6}$$

where v_{id} is the velocity of a particle at iteration t of the algorithm, c_1 and c_2 represent the cognitive and social components respectively and ω is the inertia weight. Two random parameters r_1 and r_2 are also used to randomly modify the influence of the social and cognitive parameters have on any given iteration [30, 10]. An illustration of a single velocity and position update is shown in Fig. 4.9.

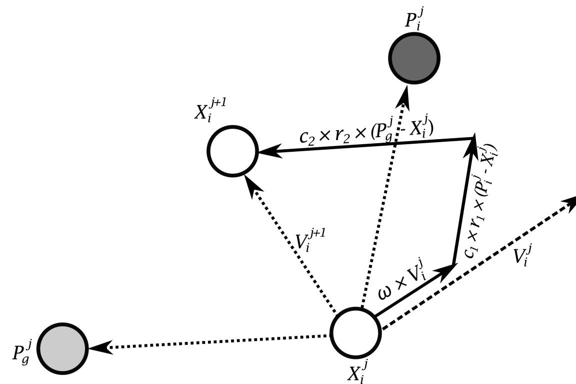


Figure 4.9: Velocity and Position Update in PSO

4.4.2.2 *Inertia Weight*

The motivation for introducing an inertia weight ω into the velocity equation was to gain more control over exploration vs. exploitation which would eliminate the need for the V_{max} parameter [7]. The concept of inertia weight was first proposed by Shi and Eberhart in [119] when they introduced a fixed inertia weight [7]. They stated that inertia weight plays a role in balancing global vs. local search (exploration vs. exploitation); a low value of inertia weight encourages a more local search of the available space where larger values would facilitate a more global search [119, 7]. As well as simply selecting a fixed inertia weight for a given

problem, as above, many other methods that adaptively control the value of ω have been proposed in the literature; some of which will be discussed here.

A simple adaptive method is to linearly decrease value of ω so as to shift the mode of the search from being more exploratory early on in the search process to more exploitative in later stages [119, 120]. Xin et al. in [148] calculate the value of ω as follows [148]:

$$\omega = (\omega_s - \omega_e)(t_{max} - t)/t_{max} + \omega_e \quad (4.7)$$

where t_{max} represents the total number of algorithm iterations allowed, t is the current iteration count and ω_s and ω_e represent the initial and final values for the inertia weight respectively [148] - bounding the weight to the range $[\omega_e, \omega_s]$. A weight decreasing linearly from $\omega_s = 0.9$ to $\omega_e = 0.4$ is shown to improve performance [120, 148].

Eberhart and Shi proposed a random inertia in [28], where a value of ω is selected randomly from the range $[0.5, 1.0]$ and by the equation [28]:

$$\omega = 0.5 + \frac{rand[0, 1]}{2} \quad (4.8)$$

where $rand[0, 1]$ is a uniformly distributed random number between 0 and 1. Thus, the mean weight will be ≈ 0.75. It was used to address the issue with using linear decreasing inertia weight when tracking non-linear dynamic systems, where it is not predictable whether more exploration or more exploitation would be of benefit at any given time during execution [28]. They report that convergence rate is increased in early the stages of execution [28], although, it is not reported whether this strategy is of any use for static systems.

Based on linear decreasing inertial weight and random inertia weight, Feng et al. presented two strategies that used a chaotic mapping to set the inertia weight referred to as the Chaotic Descending Inertia Weight (CDIW) and the Chaotic Random Inertia Weight (CRIW) [31]. They report "outstanding" performance of CRIW in comparison to random inertial weight due to the alternation between rough search and "minute" search throughout the evolutionary process [31].

Al-Hassan et al. in [2] developed an optimised PSO algorithm known as PSOSA that makes use of simulated annealing for optimising the inertia weight and was tested against an urban planning problem [7].

4.4.2.3 *The Cognitive and Social Parameters*

The cognitive and social parameters also referred to as the acceleration coefficients, are represented by c_1 and c_2 respectively and as mentioned earlier, each determines how much either cognitive (personal performance) influence or social (global performance) influence contributes to the velocity update equation. It is often the case that these parameters are fixed throughout execution of the algorithm with typical, and widely used, values of $c_1 = c_2 = 2$ [132]. However, a couple of adaptive variants have been identified.

Ratnaweera et al. in [110] presented a time-varying acceleration coefficient scheme (PSO-TVAC) with the goal if enhancing global search at the beginning of the search process and improving convergence to the global optima near the end of the search by adjusting the values of c_1 and c_2 over time - specifically, by reducing the cognitive component and increasing the social component [110]. With a large value of the cognitive component and small value of the social component at the beginning of the search, the search will favour global exploration of the available space. Conversely, a small value of the cognitive component and a large value of the social component will better facilitate local exploitation of the global best near the end of the search [110].

4.4.3 *Boundary-Handling in PSO*

Due to the nature of the displacement update of individual particles in PSO, there is always the possibility that particles can overshoot any bounds placed on the search space and end up in infeasible regions if not adequately caught and remedied. Left unchecked, individual particles and indeed most of the population (the global search itself) can become lost outside the boundaries resulting either with: (i) infeasible solutions to the given problem, if the search converges on a solution in the infeasible regions and violating particles are allowed to be evaluated, or (ii) potential for sub-optimal solutions to be returned, if the search eventually returns to converge on a solution within bounds or if out-of-bound solutions are not evaluated. Also, real-world applications often search in spaces that are high-dimensional and bounded in order to retain the semantics of the parameters [19]. Therefore, detecting and repairing boundary violations during the search process becomes an essential consideration. Additional to this, the method chosen to deal with boundary violations has also been shown to be crucial to the performance of PSO - where some methods may "paralyze" a PSO searching in high-dimensional and complex search spaces [19].

Despite this being an important consideration for constrained optimisation problems, particularly in the case of real-world high-dimensional spaces, how a PSO should handle boundary violations has been given little mention in the literature. A mathematical proof, and empirical results using common velocity initialisation techniques, by Helwig and Wanka [52] showed that for a PSO algorithm in high-dimensional spaces:

> "Uniform velocity initialization causes all particles to leave the search space with overwhelming probability (w.o.p.) with respect to the search space dimensionality n... Examples demonstrated that this probability rapidly approaches 1 if the search space dimensionality is increased. [52]"

It is further noted that none of the initialisation approaches used could fully avoid this eventuality, the results suggest that half-diff velocity initialisation has the fewest drawbacks - thus demonstrating the importance of boundary handling in influencing search performance [52].

Chu et al. later provided further empirical results to strengthen this proof in [19] where the average fraction of particles "flying outside" the boundary in the first iteration, from 100 independent runs of PSO against two benchmark functions at increasing dimensionality, were recorded. The results showed a clear trend that this fraction increased significantly as the dimensionality of each benchmark function increased. Also in this same study, Chu et al. compared the ability of three basic boundary-handling schemes: random, reflecting and absorbing, in handing and preventing boundary violations and how this translates to algorithm performance over the two original benchmark functions. These results showed that both the random and absorbing schemes had the ability to hamper the searches performed with PSO when compared with the reflecting approach.

4.4.3.1 *Boundary-Handling Approaches*

This section covers several common boundary-handling schemes in the literature, including those studied by Chu et al. in [19].

(1) RANDOM In this scheme, if a particle violates the boundary of one of its parameters, a value chosen randomly from a uniform distribution, between the lower and upper bounds of the offending parameter, is assigned as the new value [19]. That is, if X_c represents the the location of the current infeasible particle variable and LB and UB denote the lower and upper bounds of the variable respectively, then a new feasible location can be chosen as in $X_c \in [LB, UB]$ [101].

(ii) REFLECTING In this boundary-handling scheme, any violation to a parameter boundary is 'reflected' back inside the feasible region as if having ricocheted off a hard surface [149]. This is done by relocating the particle at the violated boundary and the sign of the velocity component in the offending dimension is flipped [149]

(iii) DAMPING The scheme works similarly to reflection in that the particle rebounds back into the feasible region; however, a randomly generated damping factor from the interval $[0, 1]$ is used to reduce the velocity component in that dimension which, as with reflecting, has its sign changed [149]. Also as with the reflecting scheme, this reflection is carried out after having been absorbed to the violated boundary [149].

(iv) ABSORBING Here, a particle violating a parameter boundary is set to the boundary value it violated. For example, if a particle would pass across the upper bound for a given parameter, it would instead be placed at the upper boundary and its velocity at that dimension zeroed [149]. The name of the approach stems from the metaphor of a particle leaving a parameter boundary and being 'absorbed' back. As before, if we consider X_c to be the location of the current infeasible particle variable, then X_c is set to the violated boundary as follows [101]:

$$X_c = \begin{cases} LB & \text{if } X_c < LB \\ UB & \text{if } X_c > UB \end{cases} \tag{4.9}$$

Even though the velocity is zeroed, the particles memory of personal best and global best locations will reorient the particle inside the feasible search space [149].

4.4.4 *Particle Swarm Optimisation (PSO): Implementation*

4.4.4.1 *Details and Description*

This implementation was derived from code sourced from [85] and [86] by Gandhi Manalu and was modified in order to work within our experimental framework. The main omission from this code however, which had to be implemented, was that of handling boundary violations (see Section4.4.3) so particles were simply allowed to move beyond the bounds on the search space and therefore assuming that enough particles would remain within bounds in order to find the global optimum. This approach is suitable if it is known that there are no points or regions in the infeasible space with a better objective value than that of the feasible global optima - as is the case with the example benchmark function provided with the source code

cited above - as any bound violating particles would eventually, and likely rather quickly, be drawn back within bounds through the effect of the cognitive and social components of the velocity formula (See section 4.4.2.1). Otherwise, any infeasible solution measuring better performance than the true global optima will likely cause the search to converge somewhere in infeasible space.

Since we do not know the nature of the search spaces beyond the defined bounds on our benchmark function set, it was necessary to use a boundary handling strategy to avoid any possibility of objective performances being reported from infeasible regions. For this we opted to use a simple absorbing scheme (Section 4.4.3) - one in which the velocity of the violating parameter is also set to zero. During our studies, we found that this scheme outperformed other variants tested, namely: (i) absorbing and truncating the velocity to 30% of its original, (ii) Random replacement within the bounds and a random update to the velocity and (iii) a dampened reflect scheme - where the solution was reflected back into the bounds at 20% of the distance violated and the velocity was set to zero. A 'full' reflecting scheme was tried initially - where 100% of the violated distance was used and velocity zeroed - however this lead to particles which were reflected outside the opposing boundary. Although we could have dealt with this with multiple reflects (within a while loop) until the particle lay within the bounds, it was felt that since there was no mention found in the literature relating to this issue with the reflecting scheme that it would not be conducive to a 'common' implementation of the algorithm.

4.4.4.2 *Pseudocode*

Algorithm 9 Pseudocode of PSO Implementation

```
1:  procedure PSO(iWeight, cogCoeff, socialCoeff, probSize, popSize)        ▷ Main loop
2:      problemSize ← probSize
3:      swarm ← ∅
4:      swarmSize ← popSize
5:      w ← iWeight                                                          ▷ Inertia Weight
6:      c1 ← cogCoeff                                  ▷ Cognitive Acceleration Coefficient
7:      c2 ← socialCoeff                                  ▷ Social Acceleration Coefficient
8:      Vmax ← maximum allowed value
9:      fitnessValueList ← INSTANTIATE(swarmSize)  ▷ Current list of all particle fitnesses
10:     pBestLocations ← INSTANTIATE(swarmSize)  ▷ List of particles best locations so far
11:     pBestScores ← ∅                          ▷ Best objective scores of all particles so far
12:     gBestScore ← ∅                                      ▷ Objective score of global best
13:     gBestLocation ← ∅                                         ▷ Location of global best

14:     INITIALISESWARM
15:     UPDATEFITNESSLIST

16:     while (terminate == false) do                                         ▷ Main Loop
17:         for (i = 0 to swarmSize − 1 by 1) do                       ▷ Update pBest Lists
18:             pBestScores[i] ← fitnessValueList[i]
19:             pBestLocations[i] ← swarm[i].location
20:         end for

21:         bestParticleIndex ← GETMININDEX(fitnessValueList)
22:         if (fitnessValueList[bestParticleIndex] < gBestScore) then          ▷ Update
    gBestScore
23:             gBestScore ← fitnessValueList[bestParticleIndex]
24:             gBestLocation = swarm[bestParticleIndex].location
25:         end if
26:         for (i = 0 to swarmSize − 1 by 1) do
27:             r1 ← randomfloat ∈ [0.0, 1.0]
28:             r2 ← randomfloat ∈ [0.0, 1.0]
29:             particle ← swarm[i]
```

```
30:         newVel ← INITIALISE()                          ▷ New list of velocities for the particle
31:         newLoc ← INITIALISE()                          ▷ New location of the particle
32:         for (j = 0 to problemSize by 1) do
33:             */Update the velocity and position of the current particle */
34:             newVel[j] ← (w ∗ particle.velocity[j]) + (r1) ∗ c1) ∗
    (pBestLocations[i].location[j] − particle.location[j]) + (r2 ∗ c2) ∗ (gBestLocation −
    particle.location[j])
35:             newLoc[j] ← particle.location[j] + newVel[j]
36:             (*/Boundary Handling: Absorbing and Zero Velocity */)
37:             if (newLoc[j] < LB) then
38:                 newLoc[j] ← LB
39:                 newVel[j] ← 0.0
40:             else
41:                 if (newLoc[j] > UB) then
42:                     newLoc[j] ← UB
43:                     newVel[j] ← 0.0
44:                 end if
45:             end if
46:         end for
47:         particle.velocity ← newVel
48:         particle.location ← newLoc
49:     end for
50:     UPDATEFITNESSLIST
51:   end while
52:   return gBestLocation and gBestScore
53: end procedure

54: procedure INITIALISESWARM
55:   for (i = 0 in swarmSize − 1 by 1) do
56:       newParticle ← INITIALISE
57:       /* Randomise the location in the space defined by the representation */
58:       newLoc ← INSTANTIATE()
59:       newVel ← INSTANTIATE()
60:       for (j = 0 in problemSize − 1) do
61:           newLoc[j] ← uniform random float ∈ [LB, UB]
62:           newVel[j] ← uniform random float ∈ [−V_max, V_max]
63:       end for
```

```
64:         newParticle.location ← newLoc
65:         newParticle.velocity ← newVel
66:         swarm[i] ← newParticle
67:     end for
68: end procedure

69: procedure updateFitnessList
70:     for i = 0 in swarmSize − 1 by 1 do
71:         fitnessValueList[i] ← scoreSolution(swarm[i])
72:     end for
73: end procedure

74: procedure getMinIndex(fitnessList)
75:     index ← 0
76:     minValue ← fitnessList[0]

77:     for (i = 0 in fitnessList.length by 1) do
78:         if (fitnessList[i] < minValue) then        ▷ For a minimisation problem
79:             index ← i
80:             minValue ← fitnessList[i]
81:         end if
82:     end for
83:     return index
84: end procedure
```

4.4.4.3 *Tunable Parameters*

The parameters used to control the performance of the algorithm in this implementation are the following:

- Population Size
  - Type:- integer $\in [2, 250]$
  - Used by the initialization procedure to define the size of the population that will remain fixed throughout execution of the algorithm. The default population size initially trialled by SMAC is set to 50
- Inertia Weight

- Type:- float $\in [0.4, 1.0]$
- The inertia weight is used to scale the effect of the previous velocity of a particle in a given dimension. A lower and upper bound of 0.5 and 1.0 respectively was suggested by Eberhart and Shi in [28] and so we allowed SMAC to explore a little below this value. Also as per [28] the default value used by SMAC in the tuning process was 0.5.

* Cognitive and Social Acceleration Coefficients (c_1 and c_2)
  - Type:- float $\in [1.0, 2.0]$
  - These parameters are responsible for scaling the influence of the location of a particles 'best ever' solution and the location of the global best solution in the velocity update formula respectively. The upper bound for these parameters was determined by Clerc [20], where $c_1 = c_2 = 2.05$ and rounded to 2. Suganathan in [132] also used a fixed value of 2 for both coefficients. The default value is also 2.0.

4.5 SIMULATED ANNEALING (SA)

Simulated Annealing (SA), developed by Kirkpatrick et al., is a general purpose algorithm for the global optimisation of continuous objective functions[27] [5] which takes its inspiration from the process of metallurgical annealing, and is implemented as an extension of the Metropolis-Hastings algorithm - a Monte-Carlo algorithm originally used to draw sample states from the probability distributions of thermodynamic systems. SA is essentially a descent method for global optimisation - modified by random ascent moves [27] - in which the search for solutions mimics the controlled heating and cooling of annealing in order to find the global optima of an objective function. The slow cooling, often referred to as the cooling schedule, is used to decrease the probability of accepting worse solutions nearer the end of the search process, and conversely, allowing for greater exploration of the solution space in the early stages. SA has been shown to produce solutions that lie near to the global optimum in a computation time within a polynomial upper bound; these findings have further been shown to be independent of initial conditions [27].

The basic idea behind SA, is to have the heuristic generate, uniformly at random, a move from a set of neighbouring states N and probabilistically decide on whether or not to accept the new move - determined by the acceptance probability function. By making use of the temperature, updated by the cooling schedule as the search progresses, the acceptance probability function will accept more worsening moves at the beginning of the search - maximising exploration of the space - and gradually reducing to stochastic descent as the temperature approaches zero. Therefore, SA is composed of two stochastic processes: one for the generation of candidate moves and another of the acceptance of candidate moves [27].

In the next few sections, the underlying concepts behind simulated annealing will be discussed. Firstly, we gain some intuition into annealing within the field of condensed matter physics and how this motivated the development of the Metropolis-Hastings algorithm - the backbone of the complete SA algorithm. Next, SA itself is described in comparison to physical annealing and how the Metropolis-Hastings algorithm is used in SA to imitate annealing for solving optimisation problems.

5 It is also a popular monte-carlo approach for any kind of optimisation including discrete optimisation such as combinatorial optimisation [27]

4.5.1 *Annealing in Condensed Matter Physics*

Although largely beyond the scope of this section, a short description of the physical annealing process will aid further discussion and understanding of the simulated annealing algorithm. In condensed matter physics, annealing is a thermal process for allowing solids in a heat bath to obtain low energy states [1]. This process can be stated in the following two steps [1]:

1. Melt the solid in a heat bath by increasing the temperature to a maximum value
2. Slowly and carefully decrease the temperature of the heat bath until the particles of the solid rearrange themselves at it's ground state

As described by Aarts et al. in [1], in the liquid phase of the material, that is, when the atoms are in an excited state, the atoms arrange themselves randomly. Conversely, at the ground state of the material the atoms form in a highly structured lattice in which the energy value is minimal [1]. The authors continue to explain that the ground state of the material can only be reached if the maximum temperature is high enough and the cooling sufficiently slow.

4.5.2 *Acceptance Probability*

When the neighbouring solution selected is a worsening move, the probability P of accepting the solution is calculated from a function of the difference between the solution scores ΔE (between the current solution and the neighbour solution) and the current value of the temperature parameter T, as in [82]:

$$P = e^{\frac{-\Delta E}{T}} \tag{4.10}$$

The difference ΔE is calculated as the fitness of the neighbour score minus the fitness of the current solution score [82]:

$$\Delta E = f(x') - f(x) \tag{4.11}$$

From equation 4.10, the probability that a worsening move is accepted is larger at a higher value of T and smaller at lower values of T. At very low values of T, the simulated annealing algorithm behaves more like a hill climbing algorithm [117] - accepting only those moves which improve on the current solution. Also, as the value of ΔE increases, i.e., the difference between the current and neighbouring solution scores, the probability of acceptance at a given T decreases exponentially [117].

4.5.3 *Cooling Schedules*

The cooling schedule, also known as the annealing schedule, in SA is used to determine a finite sequence of value for T over time as the algorithm progresses and a finite number of steps taken at each value of T [1]. A cooling schedule is fully defined by [1]:

- An *initial value* of T
- A *minimum value* of T
- A *decrement function* used to reduce the value of T over time
- The *number of steps* to be taken at each temperature

There are many cooling strategies proposed in the current SA literature however a common method, first proposed by Kirkpatrick in [69] is simply to decrease the temperature geometrically by a fixed constant $\alpha \in [0.0, ..., 1.0]$ [1] and is thus known as the geometric cooling schedule. This scheme falls under the category of static cooling schedules [1], where the rate of cooling is not influenced by the state of the search; thus, the strategy remains fixed throughout the search process and must be specified before execution of the algorithm[53]. Conversely, adaptive (or dynamic) cooling schedules adjust the rate of temperature decrement from information obtained during execution of the algorithm [53].

In [130], Strenski and Kirkpatrick attempted to categorise optimal cooling schedules by studying SA over a set of very small problem instances over a finite number of iterations using three common schedules: geometric, linear and inverse-logarithmic [130, 53]. Their results suggested that optimal schedules are not those which decrease monotonically [130, 53] - that is, they do not only decrease with time but may remain stable or increase. The experiments also show that there is very little difference, in terms of performance, between linear and geometric schedules [130, 53]. Furthermore, the authors observe that "excessively high" initial temperatures do not affect geometric schedules [130, 53].

For our experiments, a static schedule implementation (Linear Cooling) is used with the purpose of producing performance results closer to that of a baseline typical SA implementation. Dynamic schedules have been shown to improve the performance (in terms of solution quality) over some static schedules - therefore, their use would present an atypical, and thus unfair, viewpoint of SA when compared to typical implementations of other metaheuristic approaches. The following subsections outline a few of the more common static schedules; as we make no use of dynamic schedules, these are not covered.

4.5.3.1 *Geometric Cooling Schedule*

Also referred to as the exponential cooling schedule and first proposed by Kirkpatrick at al. in [69] during their early work on cooling schedules [1], the geometric cooling schedule is one of the simplest of the static schedules but is still often in use in many practical applications [1]. It is categorised by the use of a positive real-valued constant α with a value less than but close to 1.0; typically set between 0.8 and 0.99 [1]. The temperature decrement function is then given by [98]:

$$T_t = T_0 \alpha^t \tag{4.12}$$

where T_t represents the temperature at time/cycle t and T_0 is the starting temperature.

4.5.3.2 *Logarithmic Cooling Schedule*

Of some special theoretical importance, the logarithmic cooling schedule first proposed by Geman and Geman in [37], makes use of the following decrement function [98]:

$$T_t = \frac{c}{\log(c + d)} \tag{4.13}$$

where d is typically set to a value of one. It has been shown theoretically in [43], "that when c is set greater than or equal to the largest energy barrier in the problem, this schedule will lead the system to the global minimum state in the limit of infinite time"[98]. However its use is impractical due to its extremely slow rate of temperature decrease [98].

4.5.3.3 *Linear Cooling Schedule*

In the linear schedule, another commonly used static schedule [98], the value of the temperature parameter is decreased by a constant amount by the decrement function [98]:

$$T_t = T_0 - \eta t \tag{4.14}$$

4.5.4 *Simulated Annealing (SA): Implementation*

4.5.4.1 *Details and Description*

The main detail defining this implementation of SA is that it makes use of a linear cooling schedule as opposed to the more common geometric schedule. This decision was made to account for the automatic tuning of the algorithm and the fact that comparisons would be limited to a maximum number of function evaluations. When tuning SA making use of a geometric schedule, the automatic parameter tuner would often produce a value for α

that reduced the temperature too quickly resulting in a large portion of the search being carried out at the minimum temperature - often very close to zero. Since these portions of the search processes are almost equivalent to a hill climb (descent) algorithm, it was felt that the results from the algorithm would not fully represent the performance achievable by SA when compared to hill climbing. As such, a linear schedule was selected that would not use parameters tuned by our parameter tuner but instead used a constant decrement calculated as a function of: (i) the maximum number of function evaluations E_{max} to be carried out before termination, (ii) the number of iterations to be carried out at each temperature T_{iters} and (iii) the initial and minimum values of T - T_{init} and T_{min}. Specifically, the decrement d was calculated as follows:

$$d = \frac{T_{diff}}{T_{states}} \tag{4.15}$$

where: $T_{diff} = T_{init} - T_{min}$

and: $T_{states} = E_{max}/T_{iters}$

T_{diff} represents the difference between the initial temperature and the minimum temperature and T_{states} is the total number of temperature states. This function results in a value for d that adapts to the other parameters tuned automatically by our selected tuner so as to reach T_{min} on the final round of iterations. The linear cooling schedule is also used during the tuning of the remaining parameters, so although we have removed the ability to tune the cooling schedule itself, parameter configurations will be sought that maximise the performance of the algorithm when using this scheme.

4.5.4.2 *Pseudocode*

Algorithm 10 Pseudocode of SA Implementation

```
1:  procedure SA(stepValue, pSize, startT, minT, iterT α)          ▷ Main loop procedure
2:      stepSize ← stepValue
3:      problemSize ← pSize
4:      T ← startT
5:      T_min ← minT
6:      Tdec ← getTDecrement
7:      iterPerT ← iterT

8:      S_best ← null                                    ▷ Overall best solution discovered
9:      S_best.score ← null

10:     S ← generateInitialSolution(problemSize)
11:     S_score ← evaluateSolution(S)

12:     while (terminate == false) do
13:         for (i = 0 to iterPerT-1 by 1) do
14:             N ← generateNeighbour(S, problemSize, stepSize)   ▷ random neighbour of
    S
15:             N_score ← evaluateSolution(N)

16:             if (N_score < S_score) then
17:                 S ← N
18:                 S_score ← N_score
19:                 if (S_score < S_best.score) then
20:                     S_best ← S
21:                     S_best.score ← S_score
22:                 end if
23:             else
24:                 r ← uniform random float ∈ [0.0, 1.0]
25:                 if (calculateAcceptance(S_score, N_score, T) < r) then
26:                     S ← N
27:                     S_score ← N_score
28:                     if (S_score < S_best.score) then
29:                         S_best ← S
```

```
30:                 S_best.score ← S_score
31:             end if
32:         end if
33:     end if
34:     end for
35:     T ← T − Tdec
36:     if (T < Tmin) then
37:         T ← Tmin
38:     end if
39: end while

40: return S                                    ▷ return best solution discovered
41: end procedure

42: procedure GenerateInitialSolution(problemSize)
43:     solution ← init
44:     for (i = 0 to problemSize-1 by 1) do
45:         solution[i] ← uniform random float ∈ [0.0, 1.0]
46:     end for
47:     return solution
48: end procedure

49: procedure GenerateNeighbour(solution, problemSize, stepSize)
50:     neighbour ← solution
51:     r ← uniform random float ∈ [0.0, 1.0]

52:     boundMin ← neighbour[i] − stepSize
53:     boundMax ← neighbour[i] + stepSize

54:     if (boundMin < 0.0) then                           ▷ Boundary handling
55:         boundMin ← 0.0
56:     else
57:         if (boundMin > 1.0) then
58:             boundMin ← 1.0
59:         end if
60:     end if
```

```
61:     boundRange ← boundMax − boundMin
62:     r ← uniform random float ∈ [0.0, 1.0]
63:     step ← boundRange ∗ r
64:     neighbour[i] ← boundMin + step

65:     return neighbour
66: end procedure

67: procedure CalculateAcceptance(S_score, N_score, T)
68:     return exp((S_score − N_score)/T)
69: end procedure

70: procedure getTDecrement
71:     Tdiff ← T − Tmin
72:     totalT − iters ← maxEvals/iterPerT
73:     return Tdiff/totalT − iters
74: end procedure
```

4.5.4.3 *Tunable Parameters*

The parameters used to control the performance of the algorithm in this implementation are as follows:

- Step Size
  - See Sections 4.1.3 and 4.2.3.3 for a complete description
- Initial T
  - Type:- float $\in [0.5, 200.0]$
  - Defines the initial temperature from which to start execution of the algorithm. The default value used initially by SMAC is 1.0
- Min T
  - Type:- float $\in [0.00001, 0.45]$
  - Defines the minimum temperature of the algorithm where the annealing schedule should stop reducing the temperature. In regular implementations, the minimum temperature would represent a stopping criterion. However, since we require each algorithm to run for a set number of objective function evaluations, if the minimum

temperature is reached before the evaluation budget is exhausted, the algorithm will perform at that temperature for the remainder of the allocated budget

- Iterations Per T
  - Type :- integer $\in [10, 500]$
  - Defines the number of iterations performed by the algorithm at each temperature step. The default value used initially by SMAC during tuning is 100

4.6 Differential Evolution (DE)

Differential Evolution (DE) is a population based metaheuristic from the class of evolutionary algorithms first introduced by Storn and Price in [129] and was designed specifically for the optimisation of real-valued, real parameter functions. In a similar way to other evolutionary algorithms such as GA, DE generates new solutions through mutation and crossover. Mutation in DE creates a new solution by adding a scaled difference between two randomly chosen solutions from the population to a third random solution[95, 129]. The resulting solution is referred to as the donor solution and will be crossed over with another randomly preselected (parent) solution referred to as the target solution to produce a new candidate solution referred to as the trial vector [129, 23]. If the trial solution possesses a better objective score than the target solution the target solution will be replaced by the trial solution in the next generation - this final step is referred to as selection [129].

This solution generation scheme differentiates DE from other evolutionary algorithms that typically generate new solutions in a more probabilistic manner through the action of probabilistic crossover and mutation operators [95]. Another differentiation is that DE prescribes the replacement of child solutions only if that solution is better than it's parent [95]; GA, for example, instead replaces each child solution, even if that child creates a worsening move in respect to the population.

A typical DE implementation can now be said to consist of four fundamental steps: (i) population initialisation, (ii) mutation, (iii) crossover and (iv) selection, each of these will be covered in more detail in the next sub section.

4.6.1 *The Differential Evolution Process*

In this section each of the four fundamental steps in differential evolution will be discussed in order of occurrence in the evolutionary process.

4.6.1.1 *Initialisation of the Population*

DE begins with a randomly initialised population of NP n-dimensional real-valued solution vectors which represents a candidate solution to the n-dimensional problem being tackled [23]. For each parameter in a solution vector, it is often the case that it is to be bound within a given range beyond which the solution will become infeasible. Storn and Price [129] note

that random population generation should cover the whole search space and so prescribe the use of a uniform probability distribution for all random decisions in DE, including those involved in solution generation. This is also noted by Das and Suganthan [23]. Therefore, a given solution will be constrained by the minimum and maximum bounds of its parameters, as in [23]:

$$\vec{X} = \{[x_{1,min}, x_{1,max}], [\ldots], [x_{n,min}, x_{n,max}]\} \tag{4.16}$$

So then the jth parameter of the ith solution vector may be instantiated as follows [23]:

$$x_{j,i} = x_{j,min} + r_{i,j} \in [0,1] \cdot (x_{j,max} - x_{j,min}) \tag{4.17}$$

where $r_{i,j}$ is a uniformly distributed real value from the interval $[0,1]$ and is generated individually for each parameter j of solution vector i [23].

4.6.1.2 *Mutation*

As mentioned briefly earlier, for each target vector of index i, where $i = \{1, 2, \ldots, NP\}$ and NP is the population size, the simplest form of mutation in DE is to create a donor vector from three randomly selected individuals $\vec{X}_{r_1^i}$, $\vec{X}_{r_2^i}$ and $\vec{X}_{r_3^i}$ in the population [23]. Where r_1^i, r_2^i and r_3^i are indices of the population in the range $[1, NP]$ which are to be mutually exclusive as well as being distinct from the the target vector index i [129, 23]. This condition clearly holds only if the population size NP is $\geqslant 4$ [129]. A constant $F \in [0,2]$ [129], which we refer to as the *differential weight*, scales the difference between any two of the three vectors and this scaled difference is added to the third [129, 23]. Thus, F is used to control the magnitude of the differential variation [129]. The mutation function for generating a mutant vector $\vec{V}_i$ is as follows:

$$\vec{V}_i = \vec{X}_{r_1} + F \cdot (\vec{X}_{r_2} - \vec{X}_{r_3}) \tag{4.18}$$

A graphical example of the production of a mutant vector using differential mutation is shown in Fig. 4.10 below.

As shown in this example, the mutation step begins with the selection of three random position vectors (indicated in black) are first selected from the population. Next, the difference vector is calculated between the two vectors $\vec{X}_{r_2}$ and $\vec{X}_{r_3}$ (represented by the longest red arrow); this difference is calculated by the bracketed sub-term in the second term of equation 4.18. Finally, the difference vector is weighted by $F \in [0,2]$ and added to remaining random vector $\vec{X}_{r_1}$ resulting in a new mutant position vector (the shorter red arrow) to be recombined with the parent vector in the crossover step. It is worth briefly mentioning that the labelling of

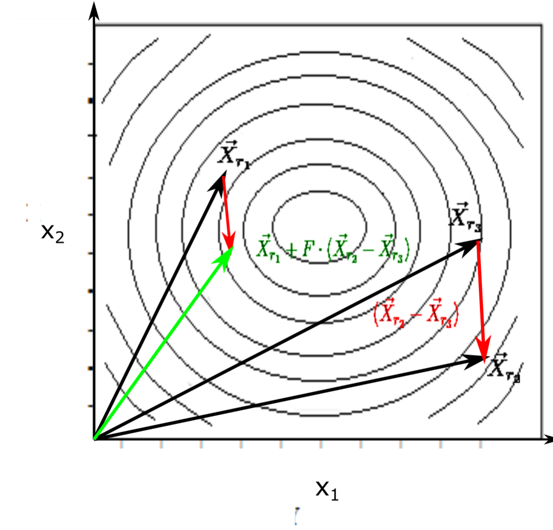


Figure 4.10: Production of a mutant vector $\vec{V}_i$ from the scaled difference vector $(\vec{X}_{r_2} - \vec{X}_{r_3})$

the three vectors $X_{r_1^i}$, $X_{r_2^i}$ and $X_{r_3^i}$ is down to the order in which they have been selected from the population, thus, the choice of vectors to be used for the different terms of the mutation equation are arbitrary and so any other ordering of vectors in the equation should not impact the performance of the algorithm.

4.6.1.3 *Recombination*

A recombination step is introduce that is designed to improve the diversity in the resulting trial vectors, where a trial vector [129]:

$$\vec{U}_i = [u_{1,i}, u_{2,i}, \ldots, u_{n,i}] \tag{4.19}$$

is generated according to [129]:

$$u_{ji} = \begin{cases} v_{ji,G+1} & \text{if } (rand_u \in [0,1] \leqslant CR) \vee j = rand_n \in [0,n] \\ x_{ji,G} & \text{if } (rand_u \in [0,1] > CR) \wedge j \neq rand_n \in [0,n] \end{cases} \tag{4.20}$$

$$where, j \in [0,n]$$

where, $CR \in [0,1]$ is a user supplied constant referred to as the *crossover weight* which determines an approximate rate in which parameters will be donated from the mutant vector, $rand_u$ is a uniformly random number - regenerated for every j^{th} parameter of the ith vector [23]. $rand_n$ is randomly chosen parameter index that ensured that at least one parameter value in $\vec{V}_i$ is recombined with $\vec{U}_i$ and is generated once for each vector in each generation [23]. This type of crossover represents the binomial (uniform) crossover described in Section 4.6.2 below [23].

4.6.1.4 *Selection*

In order to maintain the size of the population, a selection operator is applied to decide which of the target vector $\vec{X}_i$ or $\vec{U}_i$ is carried over to the next generation G, i.e., $G = G + 1$ [23]. For a minimisation problem, this selection is determined by:

$$\vec{X}_{i,G+1} = \begin{cases} \vec{U}_{i,G} & \text{if } f(\vec{U}_{i,G}) \leqslant f(\vec{X}_{i,G}) \\ \vec{X}_{i,G} & \text{if } f(\vec{U}_{i,G}) > f(\vec{X}_{i,G}) \end{cases} \tag{4.21}$$

where, $f(\vec{X})$ is the objective function being minimised [23]. In short, this means that the trial vector will replace the target vector in the subsequent population iff returns an equal or better objective function value and the target vector is kept otherwise [23]. As noted in Section 4.6 this is one way DE differs from other evolutionary algorithms, where the replacement of only solutions of greater or equal value means that the mean objective value of the population can only increase or stay the same but never degenerates [23]

4.6.2 *Variants of DE*

Apart from the commonly used DE scheme described thus far, there are as many as 10 different variants proposed by Price et al. in [129, 108]. Different DE strategies are represented by the notation *DE/x/y/z* where [129, 108]:

1. *x* represents the base vector that is to be mutated
2. *y* denotes the number of difference vectors to be used, and
3. *z* specifies the crossover method to be used either *bin* (independent binomial) or *exp* (exponential)

The strategy presented above is described as *DE/rand/1/bin*, as: (i) the base vector is randomly selected, (ii) 1 vector difference is added to this base vector and (iii) the number of parameters provided by the mutant vector closely follows a binomial distribution [129, 108].

4.6.3 *Differential Evolution (DE): Implementation*

4.6.3.1 *Details and Description*

This implementation was based loosely on the differential evolution implementation source code found at [84] and pseudocode found in [129] and represents a *DE/rand/1/bin* approach as described in Section 4.6.2. As such the implementation randomly selects a base vector (the

vector to be mutated), one difference vector is added to the base vector and the number of mutant parameters recombined with the base vector is roughly binomial.

4.6.3.2 *Pseudocode*

Algorithm 11 Pseudocode of DE Implementation

```
1: procedure DifferentialEvolution(dWeight, xProb, popSize, pSize)    ▷ Main loop
2:     problemSize ← pSize
3:     populationSize ← popSize
4:     diffWeight ← dWeight
5:     xoverProb ← xProb
6:     population ← init()
7:     S ← generatePopulation(problemSize)
8:     while (terminate == false) do
9:         xIndex ← uniform random integer ∈ [0, populationSize − 1]
10:        aIndex, bIndex, cIndex ← null
11:        do
12:            aIndex ← uniform random integer ∈ [0, populationSize − 1]
13:        while (aIndex == xIndex)
14:        do
15:            bIndex ← uniform random integer ∈ [0, populationSize − 1]
16:        while (bIndex == xIndex || bIndex == aIndex)
17:        do
18:            cIndex ← uniform random integer ∈ [0, populationSize − 1]
19:        while (cIndex == xIndex || cIndex == bIndex || cIndex == aIndex)
20:        randIndex ← uniform random integer ∈ [0, pSize − 1]
21:        parent ← population[xIndex]
22:        candidate ← parent                                          ▷ clone

23:        individualA ← population[aIndex]
24:        individualB ← population[bIndex]
25:        individualC ← population[cIndex]
26:        for (i = 0 to pSize by 1) do
27:            r ← uniform random float ∈ [0.0, 1.0]
28:            if (i == randIndex || r < xoverProb) then
29:                candidate[i] ← individualA[i] + diffWeight × (individualB[i] −
   individualC[i]))
30:            end if
31:        end for
```

```
32:         for (j = 0 to pSize by 1) do              ▷ Boundary handling: absorbing to LB or UB
33:             if (candidate[j] < 0) then
34:                 candidate[j] = 0.0
35:             else
36:                 if (candidate[j] > 1) then
37:                     candidate[j] = 1.0
38:                 end if
39:             end if
40:         end for

41:         parent.score ← evaluateSolution(parent)
42:         candidate.score ← evaluateSolution(candidate)
43:         if (candidate.score < parent.score) then
44:             population[xIndex] ← candidate
45:         end if
46:     end while
47:     return getBest(population)
48: end procedure
```

4.6.3.3 *Tunable Parameters*

The parameters used to control the performance of the algorithm in this implementation are as follows:

- Population Size
  - Type:- integer $\in [2, 250]$
  - Used by the initialization procedure to define the size of the population that will remain fixed throughout execution of the algorithm. The default population size initially trialled by SMAC is 20
- Differential Weight
  - Type:- float $\in [0.01, 1.0]$
  - Used in the mutation step to scale the difference between two of the three randomly selected mutation vectors which are to be added to the third random vector. The default value of this parameter and therefore the first one attempted by SMAC is 0.5. In [23], the authors state that this parameter has a value that typically lies in the range $[0.4, 1.0]$ so this default seemed an appropriate starting point.

- Crossover Rate
  - Type: float $\in [0.01, 1.0]$
  - Used in the binomial crossover procedure, this parameter represents the probability (or rate) that parameters from the mutant vector is passed on to the trial vector. Further, Das and Suganthan [23] note that the use of crossover rate only aids in approximating the true probability of a crossover occurring.

4.7 COVARIENCE MATRIX ADAPTION EVOLUTIONARY STRATEGY (CMA-ES)

CMA-ES is an extension of the well-known Evolution Strategies, and was mostly developed by Nikolaus Hansen and Ann Auger [82]. It targets difficult black-box optimisation problems, which may be non-linear and non-convex [45].

The framework is highly complex so we focus here only on the high-level aspects and parameters we consider for variation. Additionally, whilst variants intended for multi-objective optimisation and elitist variants have been proposed [45] this section will only deal with single objective optimisation and non-elitist selection. A more in depth treatment of this approach can be found in the excellent tutorials by Hansen [45] and Luke [82].

4.7.1 *Algorithm Overview*

The classic version of CMA-ES, more formally referred to as $(\mu/\mu_w, \lambda)$CMA-ES, contains several components intended to improve its performance. Further from [82], we will begin with a description of a basic implementation without inclusion of these components before discussing these components further. CMA-ES generates new solutions by sampling a multivariate Normal distribution, specified by a mean vector $\vec{m}$ of length n and an $n \times n$ covariance matrix C [82].

During execution of the basic CMA-ES loop, the algorithm samples the μ highest quality solutions from its current distribution, evaulates these solutions and uses these to update the distribution [82]. Sampling and updating the covariance matrix to reflect these solution vectors C is achieved more easily through eigendecomposition of C into $BDDB^T$, which effectively splits the matrix into two separate matrices B and D [82].

In CMA-ES, new solutions are sampled randomly from the current distribution - where the actual sampling is scaled by a value σ, the step size parameter, which for all intents and purposes acts as the algorithms mutation rate; the larger the value of σ, the further spread out the samples are relative to the current distribution [82].

Given the components: C, $\vec{m}$ and σ, a point P_i in the solution space is represented by three different but equivalent forms, aimed at making the mathematics easier to perform[82]. Specifically, a solution is given by: $P_i = \{\vec{x}^{(i)}, \vec{y}^{(i)}, \vec{z}^{(i)}\}$; these are defined as:

1. $\vec{z}^{(i)}$: an n-dimensional real-valued solution which has been sampled at the multivariate Normal distribution $N(\vec{0}, I)$ where the mean $\vec{0}$ is the origin (at zero) with I - the identity matrix - as the $n \times n$ covariance matrix [82]

2. $\vec{y}^{(i)}$: is the point $\vec{z}^{(i)}$ that has been transformed into the space where the distribution currently is, i.e., $N(\vec{0}, I)$. This is achieved through the use of the equation $\vec{y}^{(i)} = C^{\frac{1}{2}}\vec{z}^{(i)} = BDB^T\vec{z}^{(i)} = BD^T\vec{z}^{(i)}$ [82]

3. $\vec{x}^{(i)}$: this is the point $\vec{y}^{(i)}$ scaled by the step size σ and translated by $\vec{m}$ i.e., $\vec{x}^{(i)} = \vec{m} + \sigma\vec{y}^{(i)}$ [82]

Remaining in the context of this simple implementation, the distribution can be updated by completely rebuilding the distribution to fit with the new samples, for example by firstly updating the mean as in [82]:

$$\vec{m} \leftarrow \frac{1}{\mu}\sum_{i=1}^{\mu}\vec{x}^{(i)} \tag{4.22}$$

and update the covariance matrix by [82]:

$$C \leftarrow \frac{1}{\mu - 1}\sum_{i=1}^{\mu}(\vec{x}^{(i)} - \vec{m})(\vec{x}^{(i)} - \vec{m})^T \tag{4.23}$$

However, given that the fitnesses for our set of $\vec{x}$ have already been calculated so we could use their rank order instead of simply relying on truncation selection. This is done by sorting the truncated population such that fitter solutions appear first [82]. Weights are then calculated for each member of the ranked truncated population to give each more or less weight that could impact the outcome of $\vec{m}$ and C. These should be assigned in such a way as to have all weights sum to 1.0 [82]. Using these weights the distribution mean $\vec{m}$ and the covariance matrix C can be updated by the following [82]:

$$\vec{m} \leftarrow \frac{1}{\mu}\sum_{i=1}^{\mu}\omega_i\vec{x}^{(i)} \tag{4.24}$$

and update the covariance matrix by [82]:

$$C \leftarrow \frac{1}{\mu - 1}\sum_{i=1}^{\mu}\omega_i(\vec{x}^{(i)} - \vec{m})(\vec{x}^{(i)} - \vec{m})^T \tag{4.25}$$

However, and as mentioned previously, CMA-ES makes use of several components in order to improve the performance of the algorithm [82].

4.7.2 *CMA-ES for Large-scale Optimisation*

It is well documented that due to the high computational cost associated with calculation of the co-variance matrices in particular, that CMA-ES in its original form is unsuitable for use on

large-scale optimisation problems [114]. Indeed, from our own experiences with the algorithm, by the time we attempted to solve for 1000D problems, runtime was already exceeding 2-3 hours for a single repetition. Given that there was also a need to tune CMA-ES as we did for our other approaches (See Chapter 6) where up to 1000 repeats of the algorithm would be performed, using the full (original) approach was not feasible.

A partial solution to the above problem comes from a small modification to the approach *sep-CMA-ES* - as introduced by Rose and Hansen in [114] and provided with our implementation (see Section 4.7.3) - where instead of calculating full co-variance matrices, only diagonal matrices are computed; reducing the time and space complexity from quadratic (as we ourselves were observing) to linear - so effectively reducing model complexity and increasing the learning rate [114]. This modification however comes with reduced generality to the types of problem of which it is suitable; specifically, by reducing the co-variance matrices to diagonal (identity) matrices the degrees of freedom possessed by the original implementation that provided rotational invariance - allowing it to successfully solve many non-separable problems - is much reduced meaning that dependencies between solution variables are not captured and coordinates are sampled independently [114]. Consequently, some decreased performance should be expected from this variant [114]. Ros and Hansen report that for separable problems, sep-CMA-ES "significantly outperforms" regular CMA-ES with the added benefit that the time and space scale-up is linear [114]. Further, for low to moderate levels of non-separability sep-CMA-ES still outperforms CMA-ES for all dimensionalities by almost a factor of $\frac{n}{10} + 1$ [114]. However, for larger levels of non-separability, the authors report an advance to sep-CMA-ES only on problem dimensionalities greater than 100 [114]. Lastly, for a single fully non-separable problem instance (the block-ellipsoid function) regular CMA-ES was always found to be significantly superior [114].

However despite the possible loss of performance over our benchmark suite, of which there are several non-separable problems, we continue to use sep-CMA-ES for our experiments for its significant - and in practice, noticeable - reduction in time complexity.

4.7.3 *Covariance Matrix Adaption Evolution Strategy (CMA-ES): Implementation*

4.7.3.1 *Details and Description*

The CMA-ES implementation is that of a Java library sourced from [44] to work within our experimental framework. This implementation of the algorithm, using its default settings,

utilises a full covariance matrix for its updates, which for all benchmark functions at much higher dimensions, ($\geqslant 1000$), had a runtime in the region of 2-3+ hours for a single repeat of the algorithm. As such, this setup was unsuitable for our purposes where dimensions greater than 1000 were being probed and 20 repeats at each dimension of a problem were necessary. Furthermore, when tuning the algorithm, up to 1000 repetitions of the algorithm would be carried out by SMAC in search for parameter configurations, so this represented a greater obstacle to the approach.

However, the developers site, from which the source code was found, noted an inbuilt setting which if toggled would instruct CMA-ES to use only diagonal covariance matrices and speed up the approach from quadratic to linear time as described in the relevant article pertaining to this development, sep-CMA-ES, by Ros and Hansen [114]. They state however that this modification to the original approach makes sep-CMA-ES less effective on most non-separable objective functions, however it was found to vastly outperform the original implementation on separable functions. We found that sep-CMA-ES was still competitive with our other implementations and so still included in our algorithm set.

4.7.3.2 *Tunable Parameters*

The parameters used to control the performance of the algorithm in this implementation are as follows:

- Population Size
  - Type:- integer $\in [6, 150]$
  - As with the other population based approaches described, this parameter determines the number of solutions maintained by the algorithm at any one time. The lower bound on this parameter of 6 was determined to be suitable since the implementation as written defaulted to a population size λ of 6 for a 2-dimensional problem through the function: $\lambda = (4+3) \times \log_e(n)$, where n is the problem dimensionality. The upper bound was determined to be lower than that of the other population based approaches to further reduce the overall runtime since this placed a smaller upper bound on the number of diagonal covariance matrices that would need to be calculated.
- Initial Sigma (Step-size)
  - Type:- float $\in [0.2, 30.0]$
  - Represents the initial standard deviation used in the parameterised Normal distribution modelled by the algorithm and thus can be regarded as the initial step

size. Again, the default σ used by the algorithm was 0.2 so therefore we made use of it also as the parameters lower bound. The upper bound was determined by studying an article [46] by Hansen and Kern, which investigated the modification of this parameter in the context of several standard continuous benchmarks - some of which are included in our own suite. The authors concluded that a value of half the range between the lower and upper bound of the search space represented a reasonable setting; stating that a "considerable impact" on the performance against multi-modal functions is observed if initial step size is set too low [46]. However, since we are investigating with several benchmark functions a reasonable intermediate upper bound had to be selected. Given that one of our functions (specifically, the Griewank function) had parameter bounds of $[-600, 600]$, half of this would present too many possible combinations for SMAC to enable it to find an effective set of parameter configurations.

5

CHAPTER 5: THE PROBLEMS OF ALGORITHM SELECTION AND CONFIGURATION

5.1 INTRODUCTION

The algorithm selection problem (ASP) first identified and discussed by Rice [112] is the problem of selecting the most appropriate algorithm for solving a given problem and has become increasing relevant to optimisation researchers due to the realisation that no one algorithm can be developed that can work effectively against all problems - as per the NFL (Chapter 2 - section 2.4.5.2). Rice in [112] presented a formal abstract model that aimed to help explore this problem [139]:

1. The space of problems P [139]
2. The feature space F of all measurable characteristics of the problems contained in p as calculated by a feature extraction procedure f [139]
3. The space of algorithms A [139] that can be used against problems in P
4. The performance space Y containing a mapping between each algorithm $a \in A$ to a set of performance metrics [139]

Following Rice [112] and Vanschoren [139] the ASP can then be defined as follows:

> "For a given problem instance $x \in P$ with features $f(x) \in F$, find a mapping $S(f(x))$ into algorithm space A, such that the selected algorithm $\alpha \in A$ maximises the performance mapping $y(\alpha(x)) \in Y$"

Where S is a selection mapping method responsible for selecting the appropriate algorithm α given features $f(x)$ [139].

Simply put, the algorithm selection problem involves selecting the most appropriate algorithm from a suite of algorithms that best solves a given problem instance.

The various algorithms used to solve optimisation problems, in terms of both deterministic and stochastic algorithms, are often highly parameterised, sometimes involving tens or

hundreds of parameters [1]. Finding an optimal set of values for these settings, hereafter referred to as a ***parameter configuration***, can greatly improve the performance of a target algorithm against a given set of problem instances but this can often be an intractable goal - especially when large numbers of parameters and many possible values for each parameter are concerned.

The rationale for tuning the metaheuristic algorithms used for our purposes is to provide an appropriate set of parameter configurations for each such that all later experiments comparing the results of the metaheuristics can be made in the context of well performing algorithms for each problem instance at each given dimension. By tuning individually for each dimensionality of a problem we can obtain an appropriately 'good' parameter configuration for a given dimensionality - meaning that comparisons between other algorithms for the same problem instance at the same dimensionality can be made more meaningfully. All experimentation with the metaheuristic approaches in this thesis requires that each approach gives the best possible performance, in terms of average solution quality, for each given problem instance. This is very different from requiring that all algorithms perform equally well on all problem instances as prohibited by the no-free-lunch theorem. When comparing metaheuristic approaches, the no-free-lunch theorem falsifies any claim that one approach is better than any other when averaged over all problems, this also carries the implication that for some problems an approach may simply perform badly when compared to others - regardless of the settings of any tunable parameters. However, there will exist, for all approaches with at least one tunable parameter, one or more parameter configurations that facilitate the best performance on a given problem in relation to all other configurations.

5.2 RELATIONSHIPS BETWEEN ALGORITHM SELECTION AND OTHER AREAS OF RESEARCH

5.2.1 *Meta-learning*

Most related to the problem of algorithm selection is that of *meta-learning*, a term originating from the field of educational psychology. Meta-learning has been described as "*being in aware and taking control of one's own learning*" [9] or alternatively, *learning to learn*. More concisely, Lemke et al. considers meta-learning as being an understanding and subsequent adaptation of learning itself - on a higher level - rather than simply about gaining "subject knowledge". An individual can then introspect about their approach to leaning and adapt it depending on

1 a parameter here refers to a variable that is part of an algorithm that can be tuned to alter the behaviour and/or performance of the algorithm

the given task requirements [74]. As used in the context of machine learning, the concept of meta-learning is still closely based on this same description. However, here the term *subject knowledge* will translate to base learning, which refers to the aquisition of experience of a specific learning task [74]. In [14], Brazdil et al. explain the distinction between base-learning and meta-learning, as being categorised mainly by the scope of the level of their adaptations; specifically, base-learning is concerned only with the accumulation of experience on a specified task, where meta-learning on the other hand focusses on gathering experience over a number of applications of a learning system [14].

In this section we will follow a brief outline of meta-learning; specifically on its relationship to algorithm selection in particular but also introducing specific methods widely held to be instances of meta-learning as well as a comparison in terms of the how the constituent components of meta-learning are represented, or not represented, in each.

5.2.1.1 *Definition*

So close is the relationship between meta-learning and algorithm selection, meta-learning as a concept can be traced back directly to the formalisation of the algorithm selection problem by Rice [112]. However, the actual use of the term only appeared in the literature for machine learning in the 1990s [74]. Since then, many definitions of meta-learning have appeared in the literature, most sharing the common understanding - as noted by Lemke et al. [74] - that *"meta-learning becomes meta-learning by looking at different problems, domains, tasks or contexts or simply past experience"*. This understanding is most easily seen in the definition presented by Brazdil et al. in [14] stating that; *"meta-learning is the study of principled methods that exploit metaknowledge to obtain efficient models and solutions by adapting machine learning and data mining processes"*. The authors explain that the term *metaknowledge* refers to the knowledge obtained from past learning experiences, either from past experiences on the same data or using data obtained from another problem domain [14]. Therefore, it would be reasonable to conclude from this definition that the main prerequisite for any meta-learning approach is the use of at least one form of metaknowledge. From the common understanding found in the various definitions, Lemke et al. propose their own more formalised definition - provided here verbatim [74]:

1. A meta-learning system must include a learning sub-system, which adapts with experience
2. Experience is gained by extracting metaknowledge extracted
    a) ...in a previous learning episode on a single dataset, and/or

b) ...from different domains or problems

A further concept in meta-learning, also noted by Lemke et al. [74] is that of bias, here, referring to a set of assumptions which influence the choice of hypotheses that could explain the data. Brazdil et al. [14] refer to two types of bias that could impart influence in a meta-learning system. Firstly, *declarative bias* which relates to the representation given to the space of hypotheses, for example, where hypotheses are represented exclusively by neural networks; and *procedural bias*, affecting the precedence of hypotheses, for example, preferring hypotheses providing shorter runtimes over others [74, 14]. The assumption given this notion is that the bias in the base-learners remains fixed, where for meta-learning the system attempts to select the most appropriate bias dynamically [74].

5.2.1.2 *Algorithm Recommendation / Selection*

Out of the many forms of 'meta-learning' discussed in the literature, including but not limited to ensemble methods and dynamic bias selection, algorithm recommendation is the only form which is applicable to each point from the definition outlined above according to Lemke et al.[74]. Here the interest to the metalearner is the relationship between the characteristics of the data and the algorithm performance with the end goal of being able to select one or more algorithms from a set of algorithms suitable for a given problem [2] [74]. Since it is generally infeasible to select amongst all the possible alternative algorithms available for a given problem in a trial and error manner, meta-learning can therefore prove useful in recommending an algorithm to an end-user or in automatically weighting algorithms according to their suitability [74].

Vanschoren [139] also points out that it is not simply the algorithms themselves that allows the performance of the same algorithm to vary over different datasets, but also that these algorithms can have different parameter configurations. Therefore it would be reasonable to suggest that an algorithm with various different parameter settings can be considered as representing different algorithms altogether [139]. The author however suggests that the study of this subject and its effects should remain separate.

In machine learning the traditional application of algorithm selection is classification; however, efforts have been made to generalise the concepts to other areas of application such as: regression, sorting, constraint satisfaction and optimisation [126, 74]. For the latter,

2 Note that this description of algorithm selection in the meta-learning context is equivalent to that defined by Rice as described in Section 5.1

optimisation, one such application example by Pavelski et al. [102] is that of meta-learning optimisation for the flowshop problem through the use of decision trees. Specifically, the work proposed a meta-learning approach for knowledge discovery that operated on the performance data from four different metaheuristic algorithms whilst they solved several flowshop problem instances. As well as being able to recommend the most suitable metaheuristic for each problem instance, it is also capable of suggesting well-suited parameter configurations [102]. Although the authors suggest that for algorithm recommendation in this particular case study, there is "a lot of room for improvement", their results were deemed promising as the rules induced indicated that some metaheuristic parameters were more preferrable than others [102].

5.2.1.3 *Considerations for Using a Meta-learning Approach*

This section focusses on several practical decisions which have to be made before applying meta-learning to a problem, which can include: the selection of a meta-learning algorithm, the choice of appropriate metaknowledge and the issue of implementing and maintaining *metadatabases* [74].

A) PREREQUISITES It is usually beneficial to consider whether or not a particular methodology is appropriate for use at all against a given problem or applicaton before effort is expended. Meta-learning is no different, and as such can not be considered as a "magic cure" to any and all machine learning problems [74]. As Brazdil points out in [14], it is important that the metafeatures extracted from a problem domain are representative of that domain; a failure to ensure this will prevent meta-learning from identifying whether or not other domains are similar [74]. Similarly, atempting to apply meta-learning to new problems which have never been tackled in the past (or rarely so) will not be able to take advantage of past experiences which would improve its predictions [14, 74]. Other factors influencing the applicability of meta-learning also include: (i) the possibility that performance estimations - being reliant on the accuracy and effectiveness of being able to estimate true performance - may be too unreliable and (ii) different metafeatures might be applicable to each dataset [74].

B) META-LEARNING ALGORITHMS The selection of a particular meta-learning algorithm is very problem specific, in general, Lemke et al. [74] suggest that traditional algorithms used in classification can be very successful for meta-learning algorithm selection and could include: meta-decision trees, neural networks or support vector machines (SVM) to name but a few. The application of regression algorithms however remains less popular [74] .

C) META-KNOWLEDGE As discussed in section 5.2.1.1, Brazdil et al. [14] define meta-knowledge as being derived from the application of a learning system. A common type of metaknowledge is the performance of algorithms in the context of certain problem domains, which is then linked to characteristics of the task [74].

In terms of extracting metaknowledge from the data, the most intuitive include statistical or information-theoretic features e.g., according to Brazdil [14], for classification problems we could use the number of classes or features, ratio of examples to features and the average entropy of a class/classes [74]. For different areas of application however, features can be very different [74].

As an alternative to focussing on the data only for extracting metaknowledge, information about individual algorithms and how they produced a solution to the problem could also be used such as their predicted confidence [74]. One method of achieving this is by building a model that is both easy to generate and train and interrogating its properties. Another is an approach known as landmarking and roughly involves using the performance of simple algorithms in order to describe a problem and then correlating this information with the performance of far more complex algorthms [74].

It is also worth noting that as with all learning problems, meta-learning is also subject to the curse-of-dimensionality (Chapter 3, Section 3.2), which is 'traditionally' solved by selecting only a subset of the most relevent features - as would be done with regular feature selection of which meta-feature selection is not foundationally different [74].

5.3 AUTOMATIC PARAMETER TUNING

The problem can be laid out more formally as the *algorithm configuration problem* (ACP) [59, 58], where, given a target algorithm with tunable parameters A, a set or distribution of problem instances I and a cost metric c the goal is to find a parameter configuration(s) of A that minimises the cost c on I [59, 58].

If we were to choose to tune exhaustively on an approach with p tunable parameter settings each with j discrete values, a total number of dimensions d (according to the geometric progression) over a set of benchmark functions $\mathcal{F}$ with cardinality $|\mathcal{F}|$, we would need to test $j^p \times d \times |\mathcal{F}|$ individual parameter combinations. For example, an approach with only 2 tunable parameters with 10 discrete values each, considering 10 different dimensions and a

set of benchmark functions with cardinality 10 then $10^2 \times 10 \times 10 = 10,000$ separate tunings would need to be carried out to ensure the best parameter combination for each scenario. It follows that with more tunable parameters, an increased number of parameter values, more dimensions and over a larger set of benchmark functions, this approach quickly becomes intractable.

For this reason many automatic parameter tuners (APTs) have been developed that can produce a reasonable, albeit approximate, set of parameter configurations without the need to test every possible configuration of the target algorithm. [3] Several notable APTs are described and made use of in the literature, including model-free methods such as: F-Race, iRace, ParamILS and ROAR along with model-based techniques such as SMAC; each with their own set of benefits and drawbacks. In terms of optimising the parameters of the algorithms used in this study we opted to make use of the SMAC automatic configuration tool described in [59, 58].

5.3.1 *Model-free vs. Model-based Methods*

The existing APTs differ in whether or not explicit models are used to represent the dependence of the performance of a target algorithm on particular parameter configurations [59]. Model-free approaches, studied as far back as the 1990s, have the advantages of being able to be used 'out-of-the-box' and are relatively simple when compared to model-based methods [59, 58]. The various model-free approaches can be separated into those which focus only on optimising numerical parameters (i.e., integer or real-valued) such as F-race and those which can also be used within categorical domains (i.e., discrete and unordered) such as ParamILS [59, 58].

Model-based methods, on the other hand, has been cited as a promising path towards the next generation of automatic algorithm configuration methods [59, 58]. A well known branch of this class of methods is the Sequential Model-Based Optimisation (SMBO) framework, with benefits ranging from:

3 However, in our case, automatic parameter tuning only improves the value of j^p meaning that, given the same example as above, the total number of individual tuning operations, if the APT only tested half of the possible configurations, would be reduced from 10,000 to $0.5(10^2) \times 10 \times 10 = 5,000$ tuning operations - the number of dimensions to tune against as well as the number of benchmark functions is not affected by the use of an APT.

1. Being able to interpolate performances between between parameter settings that have been observed [59, 58] [4]

2. Able to extrapolate from what is known to unseen regions of the parameter space [59, 58]

3. And, can be used to provide quantitative data regarding the importance of both individual parameters and interactions between parameters [59, 58]

SMBO has however been shown to have a few limitations - on account of being derived from the statistics literature on black-box optimisation - that make it an inappropriate choice for automated algorithm configuration. Amongst other limitations, these include: a reliance of computationally expensive models and being focussed heavily on deterministic algorithms [59, 58]. However, work continues to overcome these and other limitations of the framework - such examples include the development of methods such as Random Online Aggressive Racing (ROAR) and Sequential Model-based Algorithm Configuration (SMAC).

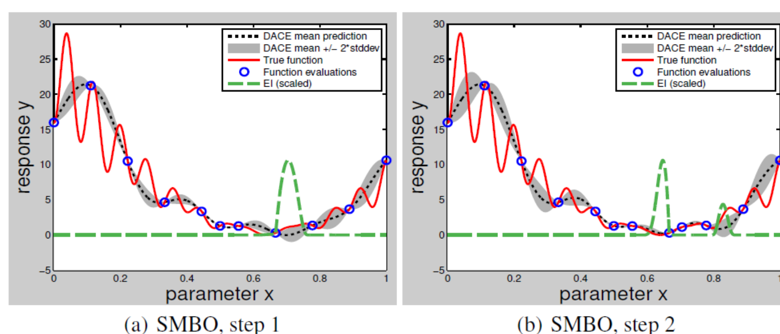
In simple terms, SMBO works by iterating between fitting models based on observed configurations and using these to select promising configurations [59]. Specifically, SMBO methods construct regression models from sampled configurations that are used to predict performance and then use these model in the optimisation process. According to Hutter at al. [59, 58]:

> "In the context of parameter configuration, the model is fitted to training set $\{(\theta_1, o_1), \ldots, (\theta_n, o_n)\}$ where parameter setting $\theta_i = (\theta_i, 1, \ldots, \theta_i, d)$ is a complete instantiation of the target algorithm's d parameters and o_i is the target algorithm's observed performance when run with configuration θ_i. Given a new configuration $\theta_n + 1$, the model tries to predict its performance $o_n + 1$."

As mentioned above, SMBO was derived from the statistics literature on global black-box function optimisation. As such, a simple example of its basic workflow as provided by Hutter et al. in [59, 58], considers a rough approximation of the algorithm of the most notable example in this literature [59, 58], Efficient Global Optimisation or (EGO) [64]. Here, they consider a deterministic algorithm A with a single continuous parameter x and runtime as a 1-dimensional function of x represented by the solid line shown in Fig.5.1a, SMBO looks to optimise (minimise) the runtime of algorithm A by searching for a suitable value for x. Also note in Fig.5.1a that SMBO is first initialised by running algorithm A with the parameter

4 interpolate here is used in the mathematical sense, where performances of new parameter settings within the range of a set of known parameter setting performances (data points) can be constructed.

values indicated by circles [59, 58]. Next, a Gaussian process (GP) regression model[5] [109] is fit to the sampled parameter value data [59, 58]. The black dotted line in Fig.5.1a and b shows the predictive mean of the GP regression model that has been trained on the parameter value samples given by the circles; where the shaded region surrounding this line represents the uncertainty in prediction - growing as distance increases from the training data [59]. SMBO then uses this performance data in order to predict and then select a promising configuration for the next run of algorithm A [59, 58]. Promising parameter configurations are predicted to perform well and/or be found in regions where the model is still uncertain [59]. These expectations are both encapsulated by the Expectated Improvement (EI), the dashed lines in Fig.5.1, which has a larger value in areas of low predictive mean and high predictive variance [59, 58] (representing high performing configurations and found in uncertain areas respectively). Expected Improvement (EI) therefore provides a automatic tradeoff between focussing on promising parts of the parameter space (exploitation) and finding out more information about more unknown regions (exploration) [59, 58].



(a) SMBO, step 1 (b) SMBO, step 2

Figure 5.1: Taken from [58], This figure shows two steps of SMBO (EGO) for the optimization of a 1D function. The true function is shown as a solid line, and the circles denote our observations. The dotted line represents the mean prediction of a noise-free Gaussian process model (the DACE model), with the grey area representing its uncertainty. Expected improvement, EI, (scaled by the authors for visualisation) is shown by a dashed line.

An exact formula for EI, is introduced in [61] as $E[I_{exp}]$ for log-transformed costs and used for this example by Hutter et al. [59, 58], is defined as:

$$EI(\theta) := E[I_e xp(\theta)] = f_{min}\Phi(v) - e^{\frac{1}{2}\sigma_\theta^2 + \mu_\theta} \cdot \Phi(v - \sigma_\theta), \tag{5.1}$$

where, μ_θ and σ_θ^2 represent the predictive mean and variance of the log-transformed cost obtained by a configuration θ respectively. Further, Φ denotes the cumulative distribution

5 A generalisation of a gaussian distribution, a gaussian process is defined by its mean function and covariance function (a gaussian distribution is defined by its mean vector and covariance matrix). Therefore, a Gaussian distribution is over vectors, where a Gaussian process is over functions. [109, 71, 40]

function (CDF) of a standard Normal distribution and f_{min} is the empirical mean performance of the incumbent configuration θ_{inc} [59, 58].

5.4 SMAC: SEQUENTIAL MODEL-BASED ALGORITHM CONFIGURATION

SMAC is an instantiation of the general Sequential Model-Based Optimization (SMBO) framework and an extension of a previous approach known as Randomised Online Aggressive Racing (ROAR) which is also an instantiation of the general framework. This section discusses what makes SMAC different from ROAR and the basic iterative procedure by which it constructs and samples from its models in order to make predictions about suitable parameter configurations for a given target algorithm.

5.4.1 *The SMAC Tuning Procedure*

5.4.1.1 *Random Online Aggressive Racing (ROAR)*

Before continuing on to discuss SMAC specifically, it is first necessary to briefly cover its precursor, Random Online Aggressive Racing (ROAR), as SMAC makes use of the aggressive racing element of this approach. ROAR is a model-free instantiation of the general SMBO framework and works by firstly selecting new parameter configurations of the target algorithm uniformly at random which it iteratively compares to the current best configuration, known as the *incumbent*, using a form of aggressive racing.

This racing is applied by a generalisation of the SMBO frameworks *intensify* procedure, adapted by the SMAC developers to be able to configure algorithms to multiple problem instances, so as to solve the additional problem - introduced by multiple instances - of deciding which instances to use on each run. The original *intensify* procedure as defined in the SMBO framework is only responsible for: (i) determining how many evaluations with each candidate configuration are to be performed, and crucially, (ii) when confidence in a configuration is strong enough to accept it as the new incumbent configuration.

5.4.1.2 *Basic SMAC Procedure*

As hinted in the previous section, SMAC makes several generalisations which solve some of the problems relating to ROAR, namely: (i) use of models that allow for categorical parameters and (ii) use of models that allow for sets of problem instances to be used [59, 58]. The most commonly used model in SMAC is a random forest model, which supports these

generalisations, and are covered in section 5.4.1.3. Along with the updated *intensify* procedure from ROAR, a summary of the basic iterative SMAC procedure can then be defined as follows [60]:

1. Construct a random forest model to predict algorithm performance over instances
2. Use the constructed model to select new promising configurations
   a) Uses the expected improvement criterion (EI) which combines predicted mean with prediction uncertainty (section 5.3.1)
   b) Finds configuration with the highest expected improvement via local search
3. Compare each new configuration to the current 'best' configuration (the incumbent)
   a) Uses the same aggressive racing strategy as for ROAR
   b) Save all data generated from runs to construct a model in the next iteration

5.4.1.3 *Random Forest Models*

The default model used within SMAC is Random Forests, an ensemble learning method used as a standard model in machine learning for regression and classification [59]. From a basic standpoint, random forests are essentially collections of regression trees. Similar in nature to decision trees, regression trees make use of real values at their leaves instead of class labels - in the context of SMAC these real values represent the performance of the target algorithm [59]. Regression trees have been used in the past to model heuristic performance both in terms of runtime and quality of solutions and are known to perform well for categorical input data [59]. Random forests also share the benefit of being able to model performance as runtime and solution quality, but additionally, allowing the uncertainty of given predictions to be quantified [59]. In addition to these benefits, further benefits stated by Breiman and Cutler [15] make it clear why the developers of SMAC would choose random forests as their default model:

- They can be run efficiently on large data sets
- They can handle thousands of input variables without variable deletion
- They are able to give estimates of what variables are important in the classification
- They create internal and unbiased estimates of the generalisation error as the building of the forest progresses
- They possess an effective approach to estimating missing data and can additionally maintain accuracy even when a large proportion of the data is missing

- They have methods for balancing error in class population unbalanced datasets
- The forests generated can be saved and reused on other data
- Prototypes can be developed which give information regarding the relationship between variables and a generated classification
- They are able to compute proximities between pairs of cases to be used in the contexts of clustering, outlier detection or giving more interesting views of the data through scaling
- The capabilities of the previous point can be extended to unlabelled data
- They offer an experimental approach to detecting variable interactions

In terms of computational costs, random forests are also fast, Breiman and Cutler claim that with a dataset containing 500 cases and 100 variables, an implementation of a random forest managed to produce 100 trees in 11 minutes using an 800MHz processor [15]. They also state that the main memory requirements come from the storage of the datasets themselves along with 3 arrays each with the same dimensionality as the data but if proximities are calculated the storage requirements grow as the number of cases multiplied by the number of trees in the forest [15].

Part III

METHOD

6

CHAPTER 6: ALGORITHM PARAMETER CONFIGURATION

6.1 Introduction

Before conducting any experimentation using the metaheuristic approaches outlined in Chapter 4, it was first necessary to conduct parameter tuning processes on each approach. It is considered standard practice when using stochastic approaches such as metaheuristics to tune the parameters of the utilised approach in order to produce the best possible performance against a particular problem instance or set of instances being addressed. This tuning aimed to obtain configurations that provided the best possible performance for each algorithm - based on a measure of solution quality - against a specific problem at a specific dimensionality.

From the point of view of comparing the performances of the algorithms in our experiments, it would be unwise to leave our implementations in an untuned state as any findings would carry lower confidence since there would be know way of knowing if the implementations could have performed better. For example, if one approach was found to perform better on some problem at higher dimensionalities than some other approach we could not state this with high confidence since we do not know how much better each approach can become with appropriate tuning. In this scenario, the better approach may already be close to an optimally performing state whereas the other approach may have much performance to be gained if tuned. If both approaches are tuned, comparisons like this can more accurately reflect the full potential of each approach and as such lead to more trustworthy conclusions.

It is sometimes the case that a practitioner will tune their approaches against several different problem instances (possibly of different sizes) in order to generate an average set of parameter configurations that, it is hoped, will provide good performance against all similar instances. For our purposes, this type of tuning of our implementations would cause inconsistent performance across problems and their various dimensionalities. Similarly to the disadvantages of having the implementations remain untuned, here we would not know with any confidence to which problems and to which dimensionalities the approach has been tuned to address better than others.

To eliminate the potential issues discussed above we tuned our implementations on a per-instance-per-dimension basis; that is, the implementations would be tuned against each benchmark problem at each dimensionality independently so as to obtain the best performance possible from an approach against a single problem at one specific dimensionality. This would result in far fairer comparisons; if one approach is compared with another and is found to perform better at higher dimensionalities for a given problem, this result carries a higher confidence level since there is no tuning that would allow the worse approach to perform better. Furthermore, this approach to tuning allows us to more easily and more justifiably generalise any conclusions to higher levels such as, to other approaches of the same class or to problem types.

To reduce the computational cost of tuning each metaheuristic approach when considering each subsequent dimension x_i as a linear progression (e.g., $x_i \in \{2, 3, 4, ..., n\}$) the dimensions to be considered instead follow a geometric progression, specifically $x_i = \lfloor x_i = x_{i-1}\sqrt{2} \rceil$, where $x_i \in \mathbb{N} : \mathbb{N} \leqslant n$. To illustrate this point, considering we make use of our 17 standard benchmark functions $\mathcal{F}$ with cardinality $|\mathcal{F}|$ (see Chapter 3, Section 3.4) up to a maximum dimensionality of 1500, we would need to carry out $d - 1 \times |\mathcal{F}| = 25,483$ separate tuning procedures if we were to consider a linear increment of 1 to the dimensionality of each problem [1]. This problem would be compounded further when we consider that multiple repeats of each of these would have to be performed to offset the stochasticity inherent in our selected APT (see Section 6.2 and Chapter 5, Section 5.4 for details) - this would require 127,415 tuning procedures if we conduct each tuning procedure five times. Conversely, using a geometric progression of dimensionality results in a set of dimensions d_{gp} with cardinality $|d_{gp}| = 20$ meaning we only need to perform a more manageable $|d_{gp}| \times |\mathcal{F}| = 340$ tuning procedures, equating to 1700 when conducting each procedure five times.

6.2 PARAMETER TUNING METHOD

We tuned our metaheuristic implementations using the state-of-the-art automatic parameter tuner SMAC. For each metaheuristic approach, the tuning process consisted of the following steps:

1. Identified tunable parameters and appropriate value ranges. These were specified in SMAC parameter files to be used by the SMAC tuning procedures.

1 a dimensionality of 1 is not considered so we only consider 1499 out of the 5000 dimensionalities

2. Compiled a set of SMAC instance files each containing an algorithm recognisable string identifying a benchmark function and a specific dimensionality to tune against.
3. Compiled a set of scenario files to be used by SMAC which among other things addresses the instance file to be used and the path to an algorithms executable code.
4. Conducted five independent tuning procedures for each scenario file in order to offset the possibility of obtaining less optimal parameter configurations due to the stochastic noise that is invariably part of random processes such as those used by SMAC
5. Obtained tuned approach performance data by conducting a series of repeated runs using each of the five independent configuration sets against their corresponding problem-dimension pairing.
6. Using the tuned performance data above, produced a 'final' parameter configuration set containing the best configurations from each of the five independent problem-dimension configurations

Each of these steps with be covered in more detail in the following subsections.

6.2.1 *Parameter Identification and Range Selection*

For each metaheuristic approach implemented, a suitable set of SMAC parameter files needed to be produced. Identification of the 'tunable' parameters was straightforward as any parameter of an approach can be considered tunable when its value can affect the behaviour of the algorithm [121] and thus its performance outcome. Since all user specified variable parameters used during an algorithms execution have an affect on algorithm behaviour these are the ones tuned to obtain better performance. The parameter file content for each of our implementations showing the identified parameters and ranges will be made available via the university's digital repository in due course.

On the other hand, selection of an appropriate range for these parameters was not as straightforward. Care needed to be taken when selecting the ranges of the parameters used in conjunction within the algorithm; as SMAC, having no sense of correspondence between parameters, can potentially return an infeasible value for one parameter in respect to the other. One such example is that of the parameters used in the GA tournament selection, *population size* and *tournament size*. When originally tuned by SMAC, both parameters were stated as integer types each representing a number of individuals. Since SMAC has no knowledge of each use within the algorithm and it will always return the best configuration found

during tuning - even if that means the operator in question become equivalent to another, we found in early tuning attempts that tournament size (of type integer) would often return with a larger value than population size suggesting a tournament consisting of the whole population. Of course, given a GA tournament with replacement consisting of the whole population, it becomes increasingly likely that the best solution in the population will take part in all tournaments - although small there is still a non-zero probability that this will not be the case on every iteration. This was overcome by making tournament size a parameter representing a proportion of the main population, which allowed us to prevent this problem for any population size SMAC produced.

6.2.2 *Conducting the Tuning Procedures*

It is suggested that when using SMAC that multiple independent tuning procedures are carried out to better ensure a high quality parameter configuration output due to the inherent stochasticity used within the SMAC implementation. This being considered, five independent tuning procedures were carried out per SMAC scenario/instance file (problem-dimension pair) for each metaheuristic approach implemented. We make use of 17 standard benchmark functions (Chapter 3) and a rounded geometric progression of problem dimensionality - resulting in 20 different dimensionalities ranging from $[0, 1448]$. Across all problem-dimension pairs for each approach, we carried out a total of $17 \times 20 \times 5 = 1700$ independent SMAC tuning processes run by utilising the departments CPU cluster. It was determined that the dimensionality increase of each problem being addressed by our approaches was to follow a geometric progression. The reason for this was mainly to reduce computational cost, since even tuning with a linear progression using dimensionality increases of size 10 in the range $[0, 1500]$ would need 12,750 independent tunings.

6.2.3 *Generation of Tuned Performance Data*

Having obtained a set of independent parameter configurations, five for each problem-dimension pairing, the algorithms were initialised with each configuration in order to generate the required performance data. Each algorithm was repeated for 20 independent runs with each parameter configuration relating to the current benchmark problem with a fixed maximum function evaluation budget of 50,000. In practice, it was far simpler to compile the tuning configurations obtained from SMAC into five comma separated value (CSV) files per problem with each line representing one of the five tuned parameter configurations at a given

dimensionality. These files would then be passed to the algorithm on execution which would carry out 20 independent repeats for each configuration. Fig. 6.1 shows an example of one of these compiled CSV files.

```
dim=2,  c1='2.0', c2='2.0', inertiaRate='0.7', popSize='50'
dim=3,  c1='2.0', c2='2.0', inertiaRate='0.7', popSize='50'
dim=4,  c1='1.1278175019901626', c2='1.4833886999205852', inertiaRate='0.4401729629396143', popSize='165'
dim=6,  c1='1.126243469518207 8', c2='1.2571467889597245', inertiaRate='0.7353011476808933', popSize='146'
dim=8,  c1='1.005954467241296', c2='1.0160879984589266', inertiaRate='0.7582652478158005', popSize='108'
dim=11,  c1='1.2062130052663433', c2='1.1213648961513178', inertiaRate='0.7171198456046894', popSize='221'
dim=16,  c1='1.0489475842909564', c2='1.2809857205555781', inertiaRate='0.6766207861718936', popSize='182'
dim=23,  c1='1.1062310957599955', c2='1.4050099636591442', inertiaRate='0.6677417805705328', popSize='236'
dim=32,  c1='1.1678393383411656', c2='1.0314616634985572', inertiaRate='0.72912131489805', popSize='223'
dim=45,  c1='1.9267431996625533', c2='1.1399060916013448', inertiaRate='0.6763259760109199', popSize='182'
dim=64,  c1='1.1768606688287502', c2='1.0669707080618969', inertiaRate='0.7029910688358594', popSize='220'
dim=91,  c1='1.8342116750479402', c2='1.0010597879025378', inertiaRate='0.6846094516554043', popSize='207'
dim=128,  c1='1.4988992860205215', c2='1.0018270758408594', inertiaRate='0.6769109230043653', popSize='239'
dim=181,  c1='1.5236922574245115', c2='1.0028616429469575', inertiaRate='0.5961094011782737', popSize='236'
dim=256,  c1='1.6325529288481204', c2='1.0430337303902295', inertiaRate='0.672410269291469', popSize='222'
dim=362,  c1='1.6168078198323692', c2='1.0191106976778248', inertiaRate='0.5904451836113005', popSize='248'
dim=512,  c1='1.7186775640269385', c2='1.1847251742530893', inertiaRate='0.6013429922694741', popSize='227'
dim=724,  c1='1.691634360228445', c2='1.0156918360808054', inertiaRate='0.6243383375260789', popSize='250'
dim=1024,  c1='1.9002598650777534', c2='1.0130073290318302', inertiaRate='0.5759667214321772', popSize='245'
dim=1448,  c1='1.9049669817841566', c2='1.0584854069921459', inertiaRate='0.5782790957502374', popSize='220'
```

Figure 6.1: Example of a parameter configuration file compiled from the individual problem-dimension configurations obtained from SMAC

For each dimension of the current problem, a separate CSV file was produced as output at different stages of the search process - in 5000 evaluation increments - containing the objective value of the best solution at that stage of the search for each of the 20 independent repeats carried out. Therefore, the output CSV files for 50,000 function evaluations would represent final solutions to the problem instances given our budget.

6.2.4 *Obtaining a Final Parameter Configuration Set*

Given the output CSV files at 50,000 function evaluations from our implementations described above, the median of the 20 independent repeats was used from each to determine the best parameter configuration for each dimension-problem pairing. The median was used in this case as the distribution of the data contained in most of the output CSV files was not Normally distributed; that is, it tended to be either very left or right skewed. Therefore, in the majority of cases the mean of the repeat data would not accurately convey the central tendency of the performance data from an algorithm for a given dimension-problem pair. This however is not unusual for data obtained from stochastic approaches such as metaheuristics.

To calculate the median from our algorithm outputs and subsequently compile a 'final' set of parameter configuration files for use in our experiments an R script was written. This would read in the set of five CSV parameter configuration files for each problem - from our five independent SMAC tuning procedures - and for each dimension of the problem, generate the median solution performance from the output CSV files associated with each of the five SMAC tuning procedures. Where a median was found to be lowest out of the five for a given

dimension, the corresponding line of the associated parameter configuration CSV file is copied into a new file.

6.3 IMPROVING THE GENERALITY OF EMPIRICAL RESULTS AND OBSERVATIONS

In order to better generalise my observations, results and conclusions derived from our tuned metaheuristic algorithms as well as to provide stronger evidence that observed relationships are not simply produced from biases that might be inherent to SMAC; a repetition of the tuning procedure was carried out using another popular automatic parameter tuning approach. The selected alternative, Irace [79], was used to tune our algorithms on a representative subset (7 functions) of the benchmark function suite. Although other widely used APTs exist, such as ParamILS, Irace is capable of handling real-valued parameters as well as discrete parameters, whereas ParamILS is designed only for use with discrete parameters.

Irace, developed by López-Ibánez et al. [79] makes use of an iterated racing approach to algorithm configuration and consists of three distinct steps: (i) sampling of new configurations based on a particular distribution, (ii) selects the most promising configurations from the samples through the use of racing and (iii) updating the distribution used for sampling in order to bias further sampling towards promising configurations [79]. A racing procedure typically starts with a finite number of candidate configurations. At each step of a race, the candidate configurations are evaluated on a single problem instance. After a number of steps, those configurations which are shown to perform statistically worse are rejected and the race continues with those configurations which survive [79].

The Irace implementation used (Version 3.3) was produced by the developers as a CRAN library for the statistics package *R*. An R script, all required input files and command line scripts (.bat files for Microsoft Windows command line and a shell script (.sh) for UNIX-based systems) were constructed using templates provided as part of the source files of same CRAN library - available from [78].

As with SMAC, the maximum number of single algorithm executions allowed during tuning was set to 1000, in keeping with the execution budget afforded for SMAC tuning processes, and all other required settings were kept as default. Also, as Irace also has a stochastic element, five independent tuning procedures were carried out for each Algorithm-

Problem-Dimension triplet in order to better insure against unsuccessful tuning procedures (see Section 6.2.2).

In an effort to make the comparison between the two tuning approaches as fair as possible, the allowable parameters ranges used in tuning each algorithm were identical to those used by SMAC. This removes the potential of any difference in tuning performance observed between the two approaches being down simply to one tuning method having to search over larger parameter spaces than the other, i.e., if the difference between the lower and upper bounds are larger.

6.4 RESULT OF PARAMETER TUNING WITH SMAC

For each of the algorithms tuned with SMAC, the median performance over all function-dimension pairings (compared to the untuned median performance) was improved significantly; particularly when searching at higher dimensionalities. The line plot in Fig. 6.2 shows the median performance results - aggregated from 20 repeats - of our DE implementation against the Ackley function (Chapter 3) over all dimensionalities. The red line depicts the median performances of the untuned algorithm and the black line is the median performances of the algorithm using the final configuration set. For this particular result, no significant difference in the performance between the tuned and untuned versions was observed up to 91 dimensions, however, from the 128 dimensional problem instance, the median performance difference increases noticeably. Similar trends were found true for each of the algorithms implemented, where the tuned version was found to perform significantly better on all benchmark functions.

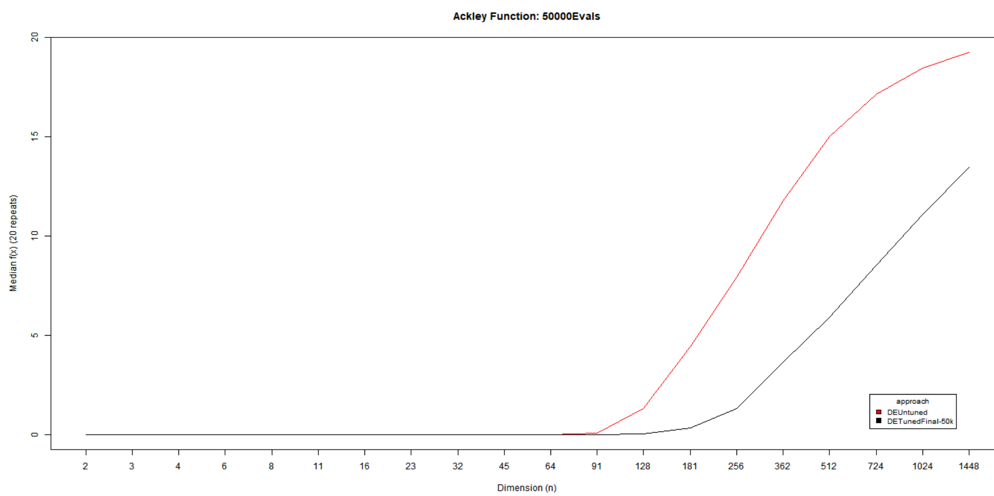


Figure 6.2: Result Obtained from DE Implementation (tuning at 50,000 function evaluations) vs. Untuned Version

Part IV

RESULTS AND CONCLUSIONS

7

CHAPTER 7: EMPIRICAL RESULTS

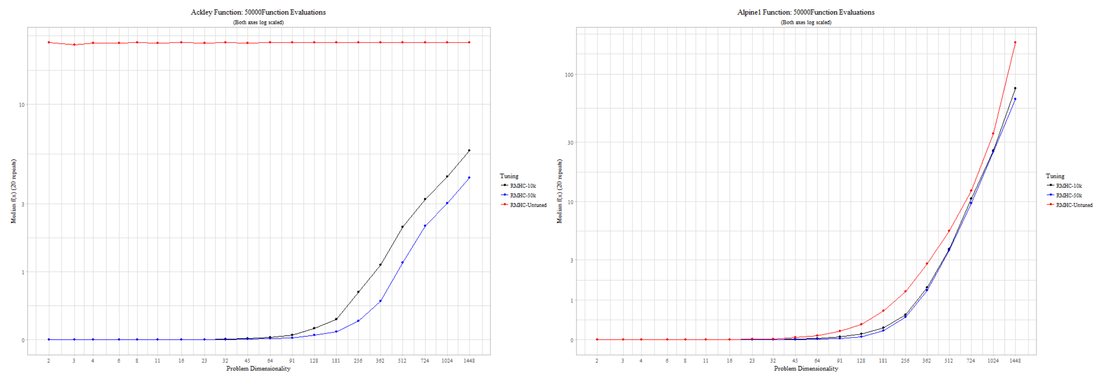
7.1 ALGORITHM TUNING RESULTS

In this section we outline the results obtained through our selected tuning procedure as compared to the same algorithms in their untuned states as well as present findings from our analysis related to parameter tuning at differing function evaluation budgets.

This first set of results represents a 'sanity-check' for further experimentation, specifically, I compare the metaheuristic approaches in the context of the best tuning found by the selected parameter tuner (SMAC), therefore, it was sensible to ensure that the performance of the algorithms was indeed being improved beyond the baseline 'untuned' equivalent algorithms. Non-improvement may have indicated a problem/bug in the tuner which would have to be remedied before any data generation and comparison could be performed.

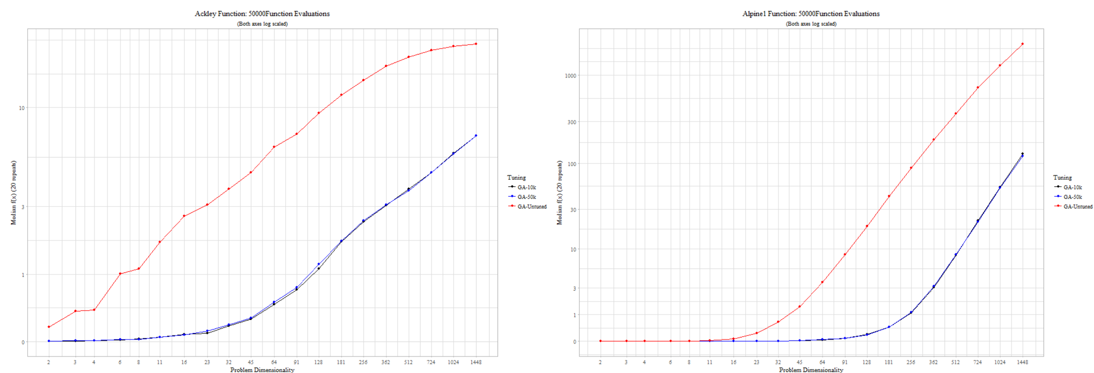
Before continuing on to presenting the tuning results, I must clarify what is meant by an 'untuned' algorithm in this context. This is where an algorithm has been initialised with a parameter configuration obtained either from: (i) common settings found in literature sources or (ii) where no suitable settings could be found in the literature based on specific design choices - such as, when using a different parameter representation to work better with SMAC e.g., GA Tournament Size vs Tournament Size Percent (See Chapter 4 Section 4.3.4.3) - a sensible value for the parameter was used instead. In short, 'untuned' here refers to an algorithm that has a single set of parameter settings that have not been selected in the context of our benchmark suite and therefore, are not necessarily appropriate or effective at obtaining high performance on this problem set and at larger problem scales.

Here we present several plots showing that SMAC indeed effectively tuned all algorithms often well beyond their baseline untuned performance - however, the full plot listing showing the tuning results for all problems and algorithms can be found in Appendix D section D.1.



(a) RMHC Tuned to Ackley Function at 50,000 GA Evaluations

(b) RMHC Tuned to Alpine n.1 Function at 50,000 GA Evaluations



(c) SSGA Tuned to Ackley Function at 50,000 GA Evaluations

(d) SSGA Tuned to Alpine n.1 Function at 50,000 GA Evaluations

Figure 7.1: Examples of Successful Tunings of RMHC and SSGA Using SMAC on Ackleys Function and Alpine n.1

7.1.1 *Comparing SMAC and Irace Tuning Performance*

To better generalise my observations, results and conclusions and generally provide stronger evidence that observed relationships are not simply produced from biases present in SMAC; a repetition of the tuning procedure, over a representative subset of my benchmark set(7 functions), was carried out using the automatic parameter tuning approach, Irace (See Chapter 6, Section 6.3 for a discussion on this method). To provide a fair comparison, Irace was provided with the same initial conditions as were true for SMAC where possible[1], specifically these include:

1 Where these conditions were not individual to a particular approach and implementation thereof

- The tuning algorithms were both afforded a maximum budget of 1000 algorithm executions to use during the tuning procedures. This was kept constant as the problem size was increased [2]
- The ranges for each algorithms parameter set were consistent between tuning methods
- The exact same target algorithm files and benchmark function suite file (utilised as executable Java .jar files) were used by both SMAC and Irace
- As with SMAC, the final set of configurations for each algorithm were taken as the best of five independent tuning procedures (determined individually for each problem-dimension tuple)

The results of tuning each algorithm using Irace in comparison to tuning with SMAC are presented in Appendix D, section D.2.

From the results, it can be easily verified that, in general, Irace found parameter configurations that produce results which follow a similar trend to those of the algorithms tuned with SMAC. However, the similarities between the two tuning methods is limited only to this single observation. Over the representative subset of functions utilised by the Irace experiments, SMAC was shown consistently to have produced far better parameter configurations which again, in general, produced a far more stable set of performance values (i.e., little variance between independent repeats of an algorithm, illustrated by the interquartile range bars on the plots in Appendix D.2) than the corresponding performances obtained with Irace configurations. Given that this is the case, and also that the initial conditions were kept consistent between SMAC and Irace where possible, then SMAC has shown itself to be the more reliable, consistent and robust to noise of the two approaches, at least against this particular suite of problem instances. Additionally, these results also show SMAC as being less prone to converging to locally optimal parameter settings - prematurely or otherwise - as a result of its inherent stochasticity.

2 In general for SMAC, the search for configurations converges before the maximum number of algorithm executions is reached - even for the maximum problem size of 1448 - so not increasing this budget as problem size increases should not present a problem in terms of observing how an algorithms real performance decreases as problem size increases. Further, for SMAC at least, the configurations returned should therefore be reasonably close to optimal for that tuning method.

Here we presents several findings related to the tuning of our metaheuristic implementations over two distinct evaluation budgets.

Plots showing the median objective value (raw) returned from 20 algorithm repeats vs. problem dimensionality for each of the benchmark problems and metaheuristics is provided in Appendix A. For each of these plots, each line represents an approach either tuned using a 10,000 or 50,000 function evaluation budget and error bars represent the lower and upper quartiles of the data at each point - Q1 and Q3 respectively. Additionally, the plots have been generated at 5000 evaluation increments for the same set of algorithm repeats, i.e., output was generated for each algorithm repeat when the number of evaluations: $(eval \bmod 5000) == 0$. For brevity, we will form our discussions here in terms of only one or two Functions from our suite (Chapter 3) and present select plots from Appendix A as appropriate.

Additionally, Appendix B provides supplimentary descriptive statistics tables for each of the plots in Appendix A.

7.2.1 *Effects Observed when Tuning with 10,000 and 50,000 Function Evaluation Budgets*

We compared our implementations on their ability to produce quality solutions when tuned with two different function evaluation budgets. Each set of resulting parameter configurations generated using these two budget schemes were then applied against our benchmark function suite - running with a budget of 50,000 objective function evaluations. The goal of this experiment was to see whether algorithms should be tuned at the same number of evaluations of which it would be expected to perform - hereafter referred to as the *target evaluations* - or if, in some instances, it were possible to obtain better or equal performance with a smaller evaluation budget.

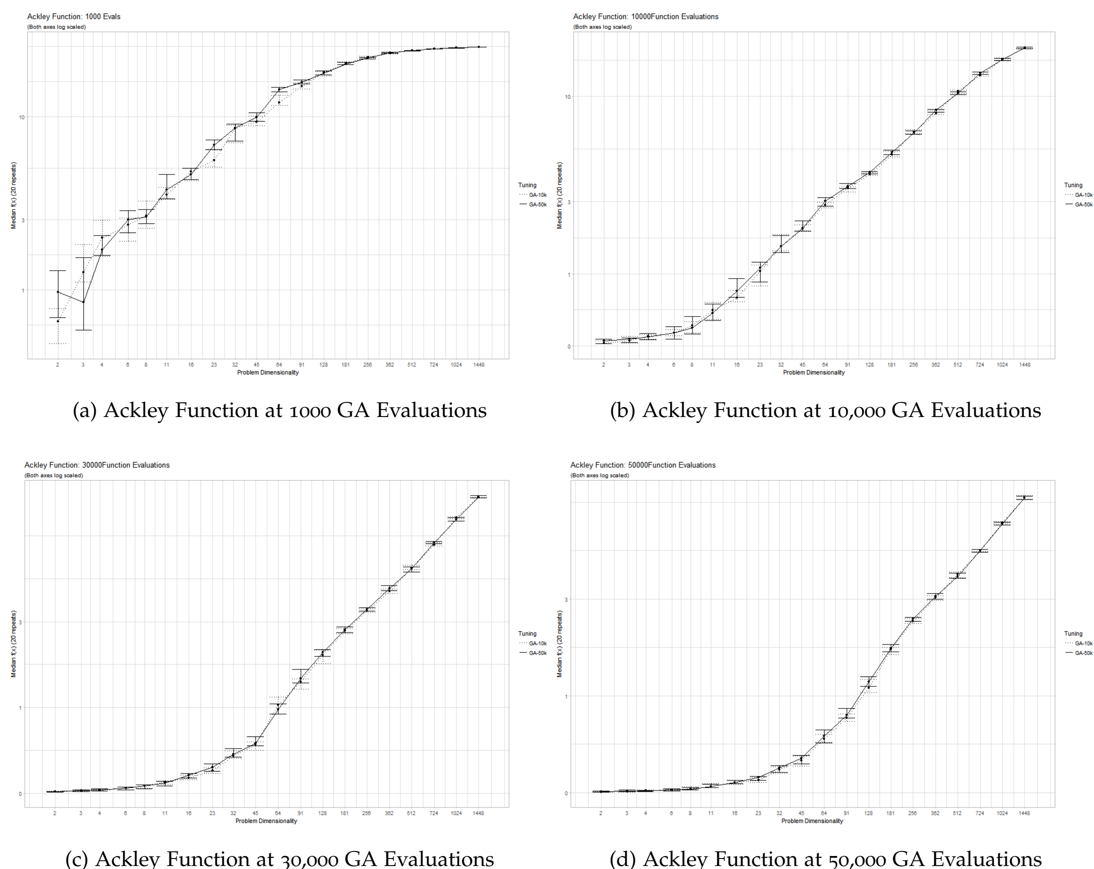
In general, we found that, when configuring the parameters for the metaheuristics, it is better to use an evaluation budget at least equal to the target evaluations when solving problems of higher dimensionality. Specifically, at lower dimensionalities (typically $< 91D$), using either tuning evaluation budget will produce comparable median solution quality; and at higher dimensionalities (typically $> 91D$), the performance obtained from each scheme begins to diverge in favour of tuning at the higher evaluation budget - here 50,000 evaluations.

However, in a relatively high number of cases no such divergence occurs, i.e., there is no difference when tuning algorithms at the target evaluation budget or a much lower budget. Table 7.1 summarises this divergent behaviour as observed in the comparison plots in Appendix A. In this table, *no-div* indicates where very little or no divergence of solution quality performance is found between tuning evaluation budget schemes, *UM* and *MM* indicates whether a function is uni-modal or multi-modal respectively and *S* and *NS* indicate whether or not a function is separable (either multiplicatively or additively). Additionally, PSO is intentionally omitted from the table as this will be treated separately.

Table 7.1: Plot summary of problem dimensionalities where algorithm performance between 10k and 50k evaluations diverge in favour of tuning at 50k function evaluations

Function/Alg.	**CMA-ES**	**DE**	**GA**	**RMHC**	**SA**
Ackley (MM,NS)	no-div	91	no-div	91	no-div
Alpine n.1 (MM,S)	181	91	no-div	1448	no-div
Bent Cigar (UM,S)	no-div	91	no-div	32	45
Brown (UM,NS)	no-div	1448	no-div	256	128
Chung-Reynolds (UM,PS)	1448	128	no-div	181	91
Deflected Corrugated Spring (MM,NS)	512	91	no-div	no-div	1448
Exponential (MM,NS)	1448	181	1448	512	no-div
Griewank (MM,NS)	no-div	91	no-div	181	64
Inverted Cosine Wave (MM,NS)	no-div	91	no-div	no-div	no-div
Levy (MM,NS)	181	128	no-div	362	128
Qing (MM,S)	no-div	64	no-div	64	64
Rastrigin (MM,NS)	181	16	no-div	181	181
Rosenbrock (UM,NS)	no-div	64	no-div	362	128
Schwefel (MM,S)	n/a	16	no-div	no-div	no-div
Sphere (UM,S)	1448	128	no-div	362	91
Sum of Different Powers (UM,S)	no-div	no-div	no-div	724	no-div
Sum Squares (UM,S)	no-div	128	no-div	128	64
Mean	**771**	**178**	**1448**	**349**	**221**
No Divergence Count	**9 of 17**	**1 of 17**	**16 of 17**	**3 of 17**	**6 of 17**

The first thing to notice in Table 7.1 is that for 16 out of the 17 benchmark functions, when using our GA implementation no significant difference can be observed between tuning at 10,000 or 50,000 evaluations. This finding suggests that when using our GA (steady-state) for continuous function optimisation that the approach may be tuned at a significantly lower number of evaluations than the required target evaluation budget. Since automatic parameter tuning, despite not being an exhaustive process, can still be a computationally costly operation - and will likely require a lot of wall clock time to complete - this finding potentially eases this restriction that may prohibit APT for other approaches, allowing GA to be tuned faster. However, one caveat is that this in no way means that the target evaluation budget can also be lowered; looking at the the plots in Appendix A it is clear to see that the algorithm has not yet converged by 10,000 evaluations for the vast majority of our benchmark suite, as progress towards better solutions continues up until our 50,000 target evaluation budget - and in some cases appears as though this progress would continue given a larger budget. Fig. 7.2 shows our GAs performance on Ackley's function for 1000, 10000, 30000 and 50000 evaluations illustrating this observation more clearly.



(a) Ackley Function at 1000 GA Evaluations

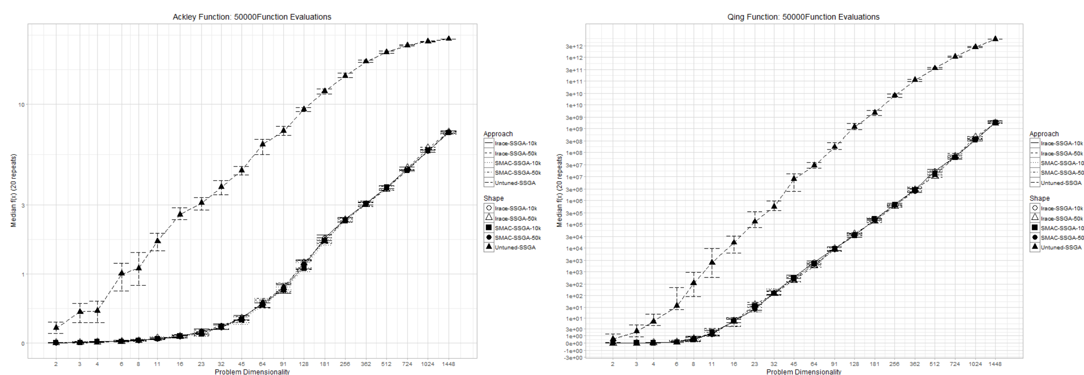
(b) Ackley Function at 10,000 GA Evaluations

(c) Ackley Function at 30,000 GA Evaluations

(d) Ackley Function at 50,000 GA Evaluations

Figure 7.2: Example of GA Continuing to Progress Beyond 10,000 Function Evaluations for Ackley's Function

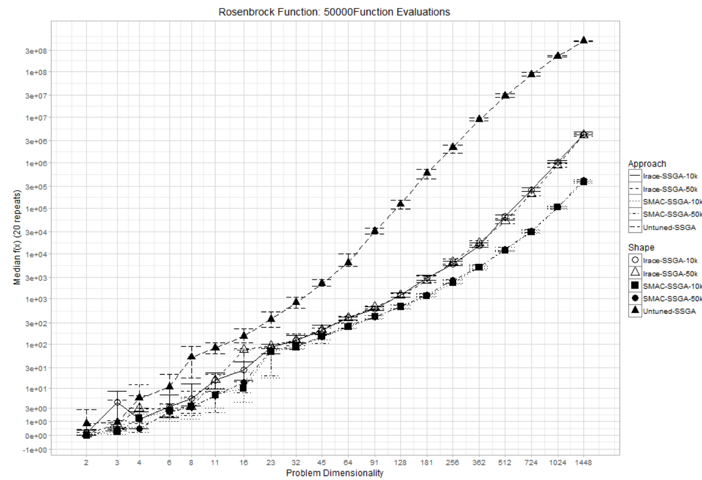
The performance results for SSGA tuned by Irace, across a representative subset of the function suite, provides further evidence for the generality of this result. As with the algorithm tuned by SMAC, the performances for both the 10,000 evaluation and 50,000 evaluation tuning budget strategies are for all intents and purposes identical, with no significant divergence between the results. Fig. 7.4 illustrates this observation for both Ackley and Qing functions.



(a) Ackley Function at 50,000 SSGA Function Evaluations (b) Qing Function at 50,000 SSGA Function Evaluations

Figure 7.3: SSGA Performance Results Comparison (Irace Tuned) Example - 10,000 and 50,000 Tuning Evaluation Budgets

Moreover, with the exception of the Rosenbrock function, the SSGA tuned with Irace produces performances that are almost identical to those of the SSGA algorithm tuned using SMAC across the representative function subset used for the Irace experiments. For Rosenbrock, we find that the configurations found by SMAC slightly outperform those of Irace. Despite this performance improvement only being ‘slight’, the interquartile ranges for each tuning method (SMAC vs. Irace) are very small and diverge cleanly with no overlap.



(a) Rosenbrock Function at 50,000 SSGA Function Evaluations

Figure 7.4: SSGA: Divergence of Performance between 10,000 and 50,000 Tuning Evaluation Budgets of SMAC and Irace

The Irace results for the remainder of the benchmark function subset can be found in Appendix D Section D.2.

The next thing we can observe from Table 7.1 is that DE appears to be the most sensitive - of all the implementations - to the number of evaluations used to tune the approach. Further, when compared with the other approaches not yet discussed, DE requires to be tuned at the target evaluation budget of 50,000 evaluations far sooner, in terms of problem size - with a mean point of divergence occurring at 178 (if we consider browns function an outlier this is decreased further to 93).

As mentioned previously, we deal with PSO here separately as it behaves quite differently from the other approaches discussed so far. What can be observed in general here is not simply a divergence of performance at a certain dimensionality when using different tuning evaluation budgets or even observing no such divergence at all. For around $\frac{2}{3}$ of our function suite, it can be seen that no performance deviation occurs between budgets at lower dimensions, but after a significant divergence of performance at mid ranged dimensions when using different budgets, a re-convergence occurs in higher dimensions that is here interpreted as there being no difference between using lower or higher tuning budgets. Also, against many of these functions, this divergence and re-convergence of performance can occur several times throughout an experiment. This effect can be seen clearly for Alpine no.1 function shown in Fig. 7.5.

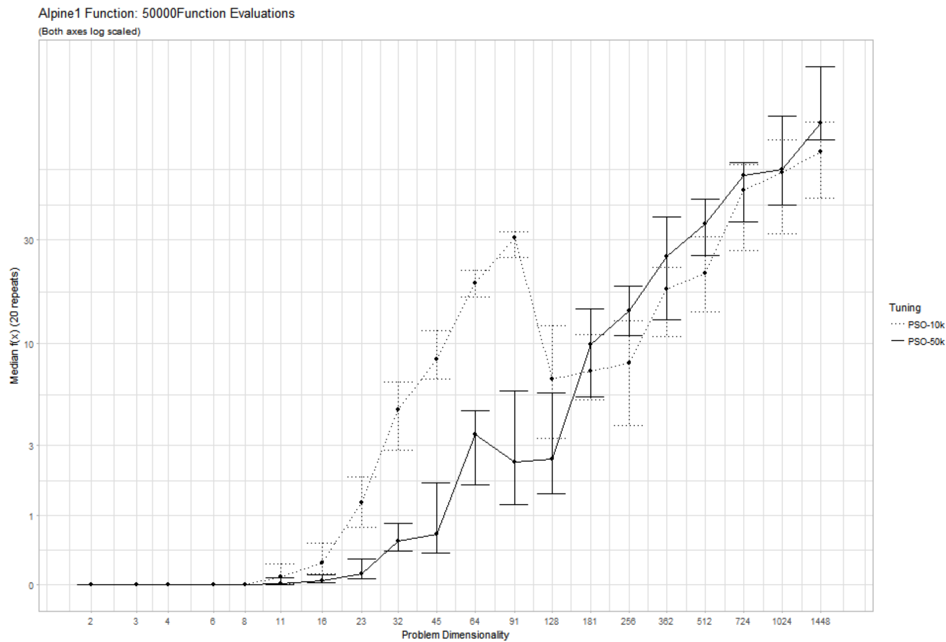
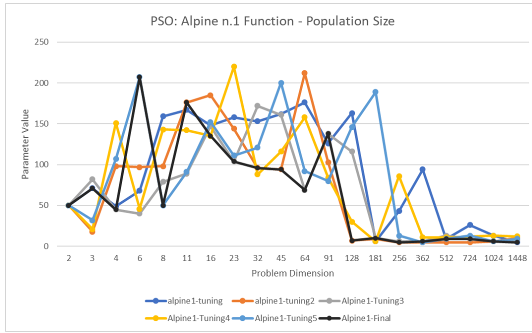
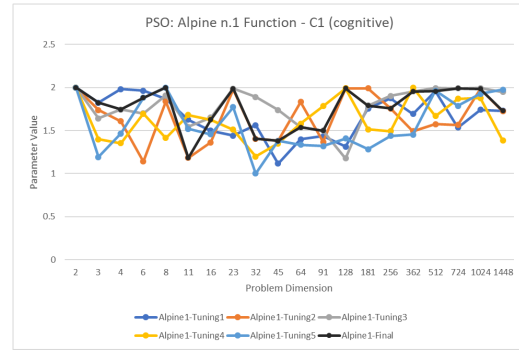


Figure 7.5: Example showing divergence between parameter tuning evaluation budgets for PSO (Alpine n.1 Function) at dimensions 11-16 and re-convergence occuring at dimension 128

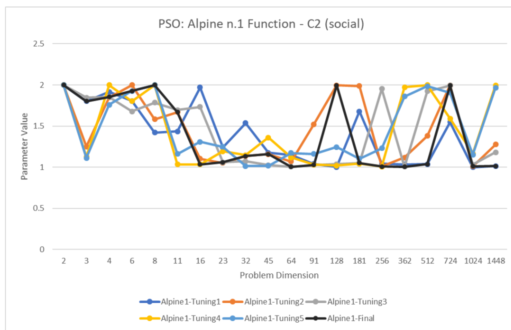
Through some analysis of the original five parameter configuration set returned for PSO, we feel that this behaviour can be attributed to a dramatic change in population size: (i) a large increase from 10s of individuals to 100s when the lines representing tuned evaluation budgets diverge, and (ii) a large decrease in population size causing re-convergence of the budget strategies. Fig. 7.6, shows plots of each parameter used to tune our PSO implementation, where each line represents the parameter values from one of our original five configuration sets tuned with a 10,000 evaluation budget (from which we selected our final configuration). We have also included in each a line representing the parameter values of the final parameter configuration. We make use of the 10,000 budget configuration files as in Fig. 7.5 the 50,000 evaluation budget line is far more stable and so is less likely to include any 'strange' parameter settings around within the dimensionalities of interest.



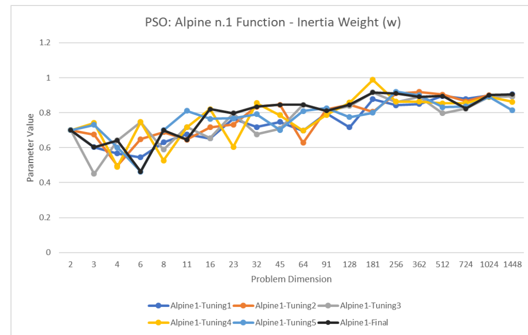
(a) PSO: Plot of Population Sizes from the Five Original Parameter Configuration Sets and the Final Configuration Set



(b) PSO: Plot of c_1 (Cognitive Component) from the Five Original Parameter Configuration Sets and the Final Configuration Set



(c) PSO: Plot of c_2 (Social Component) from the Five Original Parameter Configuration Sets and the Final Configuration Set



(d) PSO: Plot of Inertia Weight ω from the Five Original Parameter Configuration Sets and the Final Configuration Set

Figure 7.6: Plots showing individual values for each tuned parameter from our original 5 parameter configurations sets obtained from SMAC

From the plots in Fig. 7.6, it is clear to see in the population size plot that a dramatic increase of population size corresponds to to a similarly dramatic decrease in solution quality obtained from PSO at the same dimensionality (Fig. 7.5). The reverse is also true when the lines in Fig. 7.5 re-converge at a problem dimensionality of 128, in that a corresponding increase of the solution quality occurs. This effect is not only limited to the Alpine n.1 function, the remaining function plots in Appendix A in which we observe this effect[3] also show similar reliance on the population size parameter. Clearly, there is a threshold above which the presence of a large population presents a problem to our PSO implementation and conversely, below this threshold there appears to be no ideal value for population size at all.

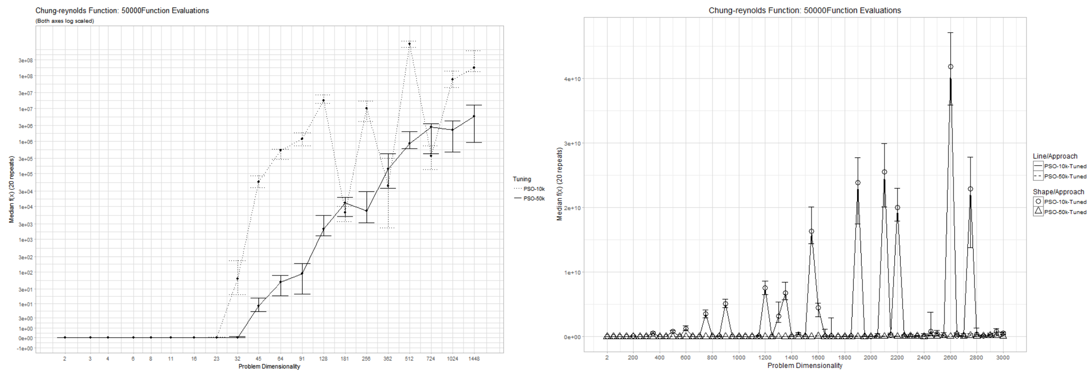
3 Ackley, Bent Cigar, Brown, Chung-Reynolds, Exponential (however less pronounced), Griewank, Inverted Cosine Wave, Levy, Qing, Rastrigin, Sphere (less pronounced) and Sum Squares

Also from the plots in Fig. 7.6, it can be seen that an apparent negative correlation between population size and the social acceleration coefficient (c_2) exists. A suggested explanation of this effect could be due to the sharing of components between particles i.e., in cases where the population size is smaller, the social coefficient may have to be larger in order to maintain a consistent level of exploitation.

I felt that it would be worth exploring this behaviour further through an extension of the tuning experiment at a larger scale, specifically by: (i) doubling the maximum number of dimensions of each problem to 3000 dimensions in order to see if this behaviour continues beyond the original 1500-D limit, and, (ii) increasing the dimensionality of the problem linearly - by increments of 50 - in place of the usual geometric progression - to observe how this behaviour presents itself at smaller dimensional granularities. The results of this experiment can be found in Appendix F Section F.1.

The results show that the effect observed for the functions in my original PSO experiment is also represented in this extended version for the same set of functions, with the same $\frac{1}{3}$ of functions exhibiting normal behaviour i.e., performance divergence with very little to no re-convergence. Further, the effect does present itself as being cyclic in nature - even at these extended problem scales and smaller granularity. As an example, in Fig. 7.7, we see that the performances between tuning budget strategies diverge and re-converge multiple times, and that each consecutive divergence increases in magnitude as the dimensionality increases. [4]

4 For some of the other functions included in Appendix F that display this behaviour, such as: Ackleys Function and Alpine Function no.1, the high frequency of the divergences coupled with the similar baseline performances for both tuning budget schemes does not facilitate clear comparison, however, on close examination the same behaviour can also be observed for these functions. The majority of the comparisons however provide clear indications of where divergence is occurring.



(a) PSO Performance Against Chung-Reynolds Function Over a Geometric Progression of Dimensionalities from 2 to 1448 Dimensions (Original Experiment) after 50,000 Evaluations

(b) PSO Performance Against Chung-Reynolds Function Over a Linear Progression of Dimensionalities in Increments of 50 Dimensions (Extended Experiment) after 50,000 Evaluations

Figure 7.7: Comparison of Performance (Median Solution Quality of 20 Independent Repeats) Against the Chung-Reynolds Function Between Original and Extended Scale PSO Experiments

Of course, this exponential decrease in performance could easily be attributed to the increased difficulty of finding good solutions within exponentially increasing search space volumes (curse-of-dimensionality). However, the root cause for the occurrence of this cycle of behaviour is still largely unclear. One possibility is that where the performance divergence occurs, the underlying structure of the function at that particular dimensionality becomes such that the automatic parameter tuner finds it more difficult in earlier stages of the search to identify a parameter configuration providing a significant improvement in performance in the shorter search scale of 10,000 function evaluations. As suggested of the original PSO experiment, from SMACs perspective, no configuration is found to be more effective than any other. In effect, this would mean that the same behaviour is largely unseen for the 50,000 budget strategy, as we observe, where PSO is given enough time during tuning to explore the potential of the various configurations provided to it.

Finally, the performance results of PSO when using the parameter configurations obtained through Irace do not clearly show that the observed effect can be further generalised to alternative tuning methods other than SMAC; the corresponding PSO performances when tuned using Irace (Appendix D, Section D.2) are too erratic to reliably differentiate between noise produced by more poorly performing parameter configurations and actual instances of the same effect discussed thus far. However, comparing the noise between different scales of the same problem for PSO to the other algorithms configured using Irace, we see that the configurations for PSO produces the most inconsistent performance data. It may therefore

be hypothesised that what we are observing with the Irace results for PSO are exaggerated instances of the original effect, caused by the increased difficulty and inconsistently in finding reliable configurations when compared with SMAC.

7.3 Performance comparison between algorithm implementations

7.3.1 *Random Mutation Hill Climbing (RMHC)*

The first, rather unexpected, observation that can be made from the algorithm comparison plots of Appendix C Sections C.1.4 and C.1.4 is that our Random Mutation Hill Climbing (RMHC) implementation regularly performs competitively with far more complex approaches. It places third - albeit jointly in some cases - in 6 of the 17 benchmark functions, second in 4 of the functions and first position in 7 functions (beating every other approach outright in 3 out of the 7 - including the far more complex sep-CMA-ES).

This observation, although relatively trivial, actually provides some evidence contrary to the widely held belief that hill climbing algorithms, particularly of the most basic stochastic variations, are not viewed as very useful when compared to more powerful evolutionary algorithms [115]. Rosete-Suarez et al. in [115] describe this belief as being a bias likely introduced due to: (i) a lack of rigour in terms of defining the 'true' complexity of problems and (ii) the situation that it is rare to find comparisons against basic stochastic hill climbing variations. In fact, Rosete-Suarez et al. also offer a multitude of examples from the literature where simple stochastic hill climbing variants outperform some Evolutionary Algorithms in various contexts [115]. This, conclusion should not be new to the search and optimisation community - we know from the NFL theorem [146] that all search methods perform equally well when averaged over all possible search spaces; the obvious consequence of which is Hill Climbing Algorithms, as a valid search strategy, must have a niche set of functions for which it is ideally suited [115].

When compared to the Irace performance results, we see that Irace - although being afforded similar initial conditions (e.g. total number of algorithm executions) - was not as successful in tuning RMHC as SMAC. Therefore, depending on whether SMAC or Irace was used for the tuning procedure, the performance results will not likely be consistent with the observations above; especially if such comparisons are made between the remaining metaheuristics that have also been tuned through the use of Irace.

7.3.2 *Changes to Rank Ordering of Performance at Larger Scales*

Here we make use of tables of ranks and ranking plots to investigate the scalability of our metaheuristic implementations given a fixed budget of function evaluations (50,000) as problem scale increases. The full set of results can be found in Appendix C, which includes plots showing the raw performance medians with and without IQR error bars (Sections Sections C.1.4 and C.1.4) as well as providing the rank ordering plots and tables (Sections C.2 and C.3) respectively. However, for the purposes of discussion, selected results will be presented in this section. Additionally, in order to gain some insight into the general robustness and effectiveness of the metaheuristic implementations over our benchmark suite as dimensionality increases, and in effect, an idea of the comparative performance when averaged over functions possessing features of interest for this study i.e., multi-modality, uni-modality, full separability and full non-separability; plots presenting the mean aggregate rank ordering of each algorithm over several subsets of the benchmark suite will be discussed. Due to space restrictions, the tables corresponding to these plots can be found in Appendix C Section C.3.2. Additionally, and in order to more easily discuss the general trends in terms of scalability, results in this section are often presented as pooled mean rankings over dimensionality ranges rather than individual dimensions e.g., for: $\{2, 3, 4, 6, 8\}$D, $\{11, 16, 23, 32, 45\}$D and so on.

For both of these types of plots, care should be taken in the interpretation of the aggregate ranks as there are a couple of reasons an algorithm can appear to improve as the scale of problems increase. Specifically, an improvement of the overall movement of rank over time of a given algorithm can be attributed to: (i) the rank changing in relation to the worsening of other algorithms ranks where the performance at larger scale is only better comparatively, and (ii) a genuine improvement in the performance of the algorithm resulting in better ranking. Depending on the situation however, the interpretation may matter more or less, for example, when simply determining which algorithms scale better than others in comparative terms, whether improved scaling is due to better performance or not matters little. On the other hand, when looking to identify algorithms which are more successful - performance-wise - as the scale increases, interpretation of these plots will be more troublesome[5]. In the plots found in Appendix C Sections C.1.3 and C.1.4, we can observe only a few isolated instances where the median performance is found to be better than that achieved at the previous dimensionality scale, where afterwards the expected trend of decreasing performance recovers. Therefore for our purposes in this section, any perceived improvement to rank order as dimensionality

5 This situation was always very unlikely to transpire due to the effects of the curse of dimensionality. It is mentioned here only to draw attention to the possibility of misinterpreting this kind of rank visualisation

increases can be safely assumed to attributed only to the rank reduction of other algorithms (point (i) above).

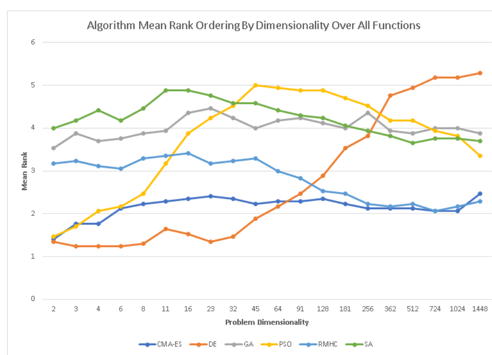
To reiterate the corresponding hypothesis, we expected that the ordering (and therefore the performance) between algorithms would change as problem dimensionality increases. In providing evidence supporting this hypothesis, we hoped to further find out: (i) whether there is a pattern, in terms of the presence of certain problem features, as to what kind of problems and/or at which point an algorithm performs well or ceases to perform well, (ii) the scale of performance degradation for each algorithm in terms of dimensionality increase, and (iii) the comparative scalability of the algorithms given *(ii)*.

7.3.2.1 *Rank Ordering Over the Entire Benchmark Suite*

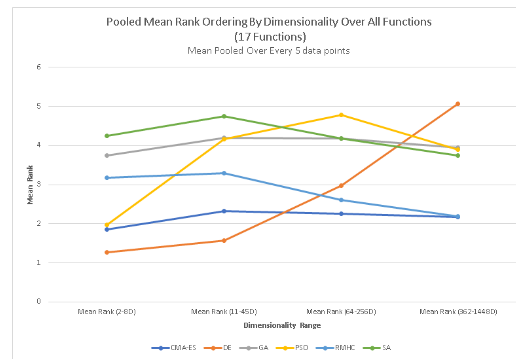
Before exploring the results in the context of problems possessing specific features, it is worth discussing the comparative results in terms of the entire benchmark suite i.e., the general performance of algorithms when all four of the investigated features are present. It is often the case, particularly for real-world problems, that little is known about the underlying structure of a particular problem or problem class; algorithm selection then must be performed from a more general 'higher level' context, such as in terms of a given application domain e.g., the space of symbolic regression problems, of a certain size. Algorithm selection at these higher granularities will almost certainly produce less meaningful recommendations compared to those made in the context of a more defined set of problems, however they can still usefully give some idea as to the likelihood that one algorithm is more suitable for use than another. For example in Fig. 7.8 and Table 7.2, the mean aggregate rank (between approximately 1.5 - 2.5) of CMA-ES over all scales provides some confidence - given no deeper investigation - that most of the comparative rankings are good (around ranks 1 or 2) for functions which can include: unimodal, multimodal, separable and non-separable features, and further that it remains robust at larger scales. On the other hand, at scales of above 128 dimensions for this set of problems, DE would likely be unsuitable. Thus, performing a general comparison such as this allows one to focus in on promising recommendations for further analysis if and when more is known about the structure of the specific problem being tackled, whilst simultaneously being able to discard algorithms which are more unlikely to be useful.

Table 7.2: Pooled Mean Rank Table Over All Functions

Alg./Dim. Range	Mean Rank (2-8D)	Mean Rank (11-45D)	Mean Rank (64-256D)	Mean Rank (362-1448D)
CMA-ES	1.86	2.33	**2.26**	**2.16**
DE	**1.27**	**1.58**	2.98	5.07
GA	3.75	4.20	4.18	3.94
PSO	1.98	4.16	4.79	3.89
RMHC	3.18	3.29	2.61	2.19
SA	4.25	4.74	4.19	3.74



Mean Aggregate Rankings



Pooled Mean Rankings

Figure 7.8: Mean Aggregate Ranking for Each Dimensionality and Pooled Mean Rank Over Every n Data Points Over All Benchmark Functions (where $n = 5$)

From the plots in Fig. 7.8, we can immediately make observations about the behaviour of both PSO and DE in comparison to the other algorithms as the scale of the problems in the benchmark suite increase. Firstly, DE is shown to be the best approach to use on average on our benchmark suite until the problem scale reaches 64 dimensions, after which it sharply decreases in performance in comparison to the other approaches, becoming the worst performing approach at a scale of 362 dimensions of which it does not recover. This observation is also mirrored clearly in the pooled mean plot albeit with less precision. Similarly to the observation made of PSO in section 7.2.1, here PSO performs relatively well at the lower dimensionalities of the benchmark suite becoming significantly worse at the low-mid to mid-high dimensionality ranges, to the point of ranking as the worst approach on average at 45 dimensions. Again, and where this observation is similar to that made in section 7.2.1, is that from 45 dimensions onwards, PSO becomes more efficient in comparison to the other

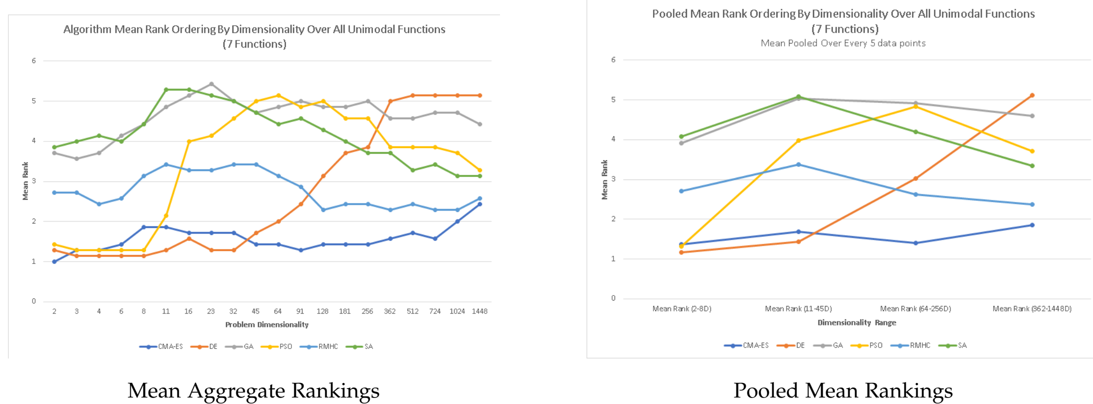
algorithms - settling at an average rank of approximately 3.5 at a problem scale of 1448 dimensions, beaten only by CMA-ES and RMHC. It is unclear if this behaviour is a direct effect of the main behavioural observation discussed in section 7.2.1, as the cyclic and sharp decreases in performance which would readily explain the similarity of behaviour here were related only to parameter configurations from a tuning strategy making use of 10,000 function evaluations, where the ranks discussed in this section are related to the performance of algorithms tuned with a budget of 50,000 evaluations. However, what can be observed from the 50,000 function evaluation plots for PSO in Appendix A is that for each of the benchmark functions where the cyclic behaviour can be observed for PSO tuned to 10,000 evaluations, a corresponding but often far less pronounced decrease - occurring at similar dimensionalities - can also be seen for PSO tuned at 50,000 evaluations which also recovers along with the 10,000 evaluation tuned PSO. Looking at the comparison plots in Appendix C, sections C.1.3 and C.1.4, we can see the effect of these small decreases in performance on rank ordering more clearly - particularly for the plots concerning: Bent Cigar, Chung-Reynolds, Rastrigin and Sum Squares. Here, where there is a small decrease in performance around the mid ranges for PSO, it is often enough to worsen the standing of the approach in comparison with the others - regularly falling into the worst performing algorithm at these points. As dimensionality increases, and since the drop off in performance often presents itself as being more gradual for PSO when compared to other approaches (i.e., is more robust to scale), PSO is eventually ranked better than those approaches which do not scale as well. The effect of these small decreases in performance on the ranking of PSO is also viewed clearly on the rank orderings on individual functions in Appendix C Section C.2.1.

From a more general standpoint, from the plots in Fig. 7.8 we can observe only slight changes in ordering amongst the remaining approaches - at least in the context of aggregated ranks. The ranks of SA and GA switch in favour of SA beyond 181 dimensions, this does not reoccur higher scales. The difference in average ranking between SA and GA, although larger at small to small-medium scales (2-64 dimensions) decreases at higher scales to that of being negligible. Additionally, the gradual improvement to the rank of both these approaches as dimensionality increases beyond 16 dimensions seems likely to be an effect originating mainly from the sharp decline of the ranks of both DE and PSO as discussed previously rather than an improvement in the performance of SA and GA at larger scales. Both RMHC and CMA-ES are seen to be the best algorithms to use on this benchmark suite on average, where the typical mean rank ordering is in the range of $\approx$ 1.6 - 3.5. At larger scales, the mean ranks of both algorithms coalesce at around 2.1 - 2.5, with the only ordering change occurring

at 1448 dimensions in favour of RMHC. Otherwise, CMA-ES maintains its rank as the best algorithm to use on this suite overall.

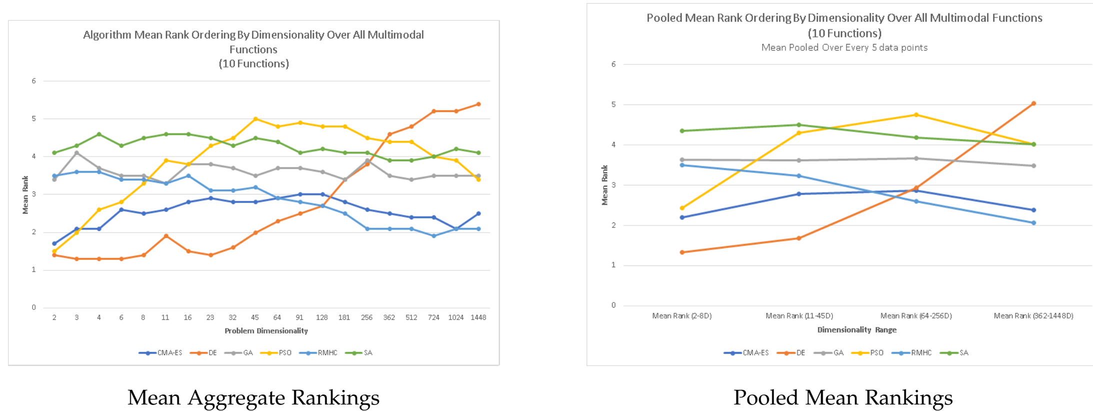
7.3.2.2 *Rank Ordering Over the Set of Uni-modal Versus Multi-modal Functions*

Here and in the next section, the ranking of algorithms against sets of problems possessing complementary pairs of features will be discussed. Fig. 7.9 presents both the mean aggregate ranks and pooled mean ranks over several dimensionality ranges (as in the previous section) over all uni-modal problems in the benchmark suite. Fig. 7.10 similarly presents the rank orderings over all multi-modal problems. The pooled mean ranking tables for uni-modal and multi-modal problems are provided as Tables 7.3 and 7.4 respectively.



Mean Aggregate Rankings | Pooled Mean Rankings

Figure 7.9: Mean Aggregate Ranking for Each Dimensionality and Pooled Mean Rank Over Every n Data Points Over All Uni-modal Functions (where $n = 5$)



Mean Aggregate Rankings | Pooled Mean Rankings

Figure 7.10: Mean Aggregate Ranking for Each Dimensionality and Pooled Mean Rank Over Every n Data Points Over All Multi-modal Functions (where $n = 5$)

Table 7.3: Table of Pooled Mean Ranks Over All Uni-modal Functions

Alg./Dim. Range	**Mean Rank (2-8D)**	**Mean Rank (11-45D)**	**Mean Rank (64-256D)**	**Mean Rank (362-1448D)**
CMA-ES	1.37	1.69	***1.40***	***1.86***
DE	***1.17***	***1.43***	3.03	5.11
GA	3.91	5.03	4.91	4.60
PSO	1.31	3.97	4.83	3.71
RMHC	2.71	3.37	2.63	2.37
SA	4.09	5.09	4.20	3.34

Table 7.4: Table of Pooled Mean Ranks Over All Multi-modal Functions

Alg./Dim. Range	**Mean Rank (2-8D)**	**Mean Rank (11-45D)**	**Mean Rank (64-256D)**	**Mean Rank (362-1448D)**
CMA-ES	2.2	2.78	2.86	2.38
DE	***1.34***	***1.68***	2.94	5.04
GA	3.64	3.62	3.66	3.48
PSO	2.44	4.3	4.76	4.02
RMHC	3.5	3.24	***2.6***	***2.06***
SA	4.36	4.5	4.18	4.02

From this set of results, we again see evidence of the poor comparative scalability of DE and PSO on high dimensional problems previously observed when all benchmark functions were considered in the last section. Over both sets of problems, the mean ranking trends for DE appear very similar showing little discernible difference in rankings on average against each set. However, despite the scalability issues at higher dimensions, DE ranks best overall for both uni-modal and multi-modal problems at lower scales. For uni-modal problems, the comparative effectiveness of DE arises only up until a scale of 32 dimensions after which the rankings of the algorithm rapidly decline placing as the least effective algorithm at a scale of 362 dimensions onwards. DE however does fair better comparatively with others against the multi-modal set, maintaining the best average ranking until a scale of 91 dimensions where, as for its rankings against uni-modal problems, there is a sharp deterioration in rank, again becoming the worst performing algorithm at the 362 dimensional scale. As with DE, there appears to be little difference between the trends of PSO against each problem set, again suggesting that problems possessing either of the two features do not significantly affect the scalability of the approach. Additionally, there is little variation between the PSO ranks which have been mean aggregated across all benchmark functions (Fig. 7.8) and those aggregated by either of the problem features. For uni-model problems however, PSO does maintain a good averaged rank compared to the other algorithms over a larger range of dimensions albeit negligable extending only so far as 8 dimensional problems before seeing a sharp decline in its rankings - again falling in line with those presented for all benchmark problems (Fig. 7.8) and multi-modal problems (Fig. 7.10). The rank ordering between DE and PSO switches in favour of PSO at a scale of 256 dimensions for uni-modal problems and 362 dimensions for multi-modal problems.

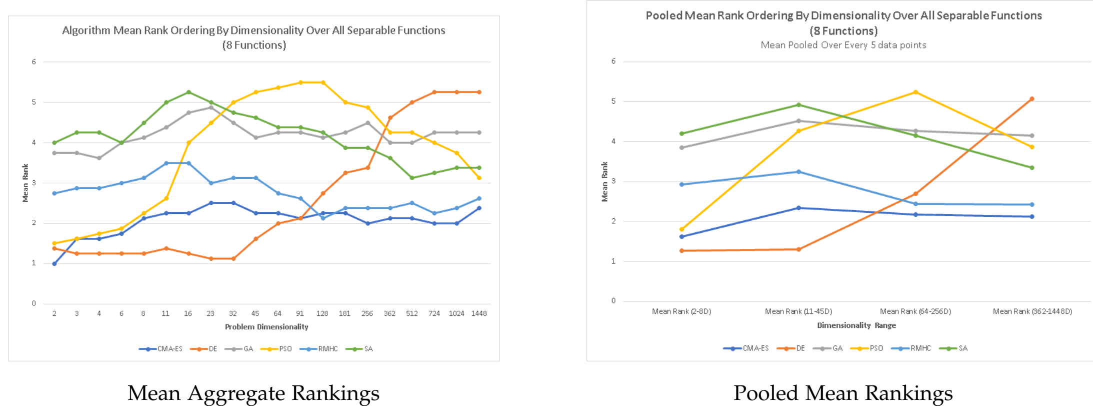
As with the rank results in the context of the entire benchmark suite, the visualisations similarities of mean rankings between distinct pairs of algorithms - where the ranks fall closely to one another but more distant from either algorithm comprising the other pair. Again, this pairing between algorithms with similar scalability relative to the other algorithms can also be observed in the results over the whole suite of problems (Fig. 7.8). However, in the context of the set of multi-modal problems, the distinct pairings become more evident as the scale of the problems increase as at low scales, RMHC and GA rank more similarly to each other than with the remaining algorithms.

Although changes to the rank ordering of the pairing comprised of GA and SA and the pairing of RMHC and CMA-ES are negligable for the set of uni-modal problems, as well as for GA and SA against the set of multi-modal functions, a change of rank order does occur

between CMA-ES and RMHC at a scale of 64 dimensions for the multi-modal set after which the mean rank of RMHC remains better than that of CMA-ES at the remaining scales.

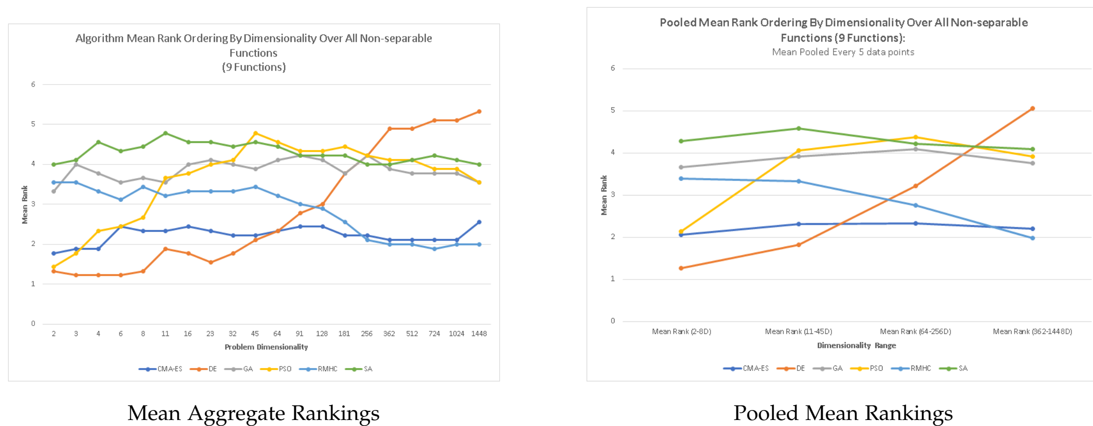
7.3.2.3 *Rank Ordering Over the Set of Separable Versus Non-separable Functions*

In this section, the mean aggregate rankings and pooled mean rankings of the algorithms against the set of separable problems and the set of non-separable problems are presented (Figs. 7.11 and 7.12 respectively). Tables 7.5 and 7.6 additionally provide the pooled mean rankings for both sets of problems.



Mean Aggregate Rankings — Pooled Mean Rankings

Figure 7.11: Mean Aggregate Ranking for Each Dimensionality and Pooled Mean Rank Over Every n Data Points Over All Separable Functions (where $n = 5$)



Mean Aggregate Rankings — Pooled Mean Rankings

Figure 7.12: Mean Aggregate Ranking for Each Dimensionality and Pooled Mean Rank Over Every n Data Points Over All Non-separable Functions (where $n = 5$)

As with the ranking results over uni-modal and multi-modal problems and for those over the benchmark suite as a whole, a similar pattern of rapid decline in scalability relative to the other approaches again emerges for both DE and PSO. Where the ranking trend for DE

Table 7.5: Table of Pooled Mean Ranks Over All Separable Functions

Alg./Dim. Range	Mean Rank (2-8D)	Mean Rank (11-45D)	Mean Rank (64-256D)	Mean Rank (362-1448D)
CMA-ES	1.625	2.35	***2.175***	***2.125***
DE	***1.275***	***1.3***	2.7	5.075
GA	3.85	4.525	4.275	4.15
PSO	1.8	4.275	5.25	3.875
RMHC	2.925	3.25	2.45	2.425
SA	4.2	4.925	4.15	3.35

Table 7.6: Table of Pooled Mean Ranks Over All Non-separable Functions

Alg./Dim. Range	Mean Rank (2-8D)	Mean Rank (11-45D)	Mean Rank (64-256D)	Mean Rank (362-1448D)
CMA-ES	2.07	2.31	**2.33**	2.20
DE	***1.27***	***1.82***	3.22	5.07
GA	3.67	3.91	4.09	3.76
PSO	2.13	4.07	4.38	3.91
RMHC	3.40	3.33	2.76	***1.98***
SA	4.29	4.58	4.22	4.09

over both separable and non-separable problems largely matches those for sets of problems discussed previously; PSO in the context the set of non-separable problems, although showing the same pattern of rapid worsening of rank before improving at larger scales, does not show the same level of decline in ranking as we have seen over other problem subsets. Where previously, in the worst case, PSO achieves a mean aggregate ranking of between 5 - 5.5, this time around the medium scale problems PSO achieves a worst case mean ranking of approximately 4.7. Although this still places PSO as the worst performing at this scale, it is still far closer to the rankings of SA and GA which we have not previously observed over other sets of problems.

The close pairings of the rankings of the remaining algorithms previously discussed in relation to other problem sets remains the same for both separable and non separable problems. In terms of the rank order of the members of these pairings, a change occurs at a problem scale of 128 dimensions in favour of SA when considering only separable problems with no such change in ordering between CMA-ES and RMHC. On the non-separable set however, there is no change to rank order between SA and GA, despite achieving closely related mean rankings, but a change occurs for RMHC and CMA-ES at a scale of 256 dimensions.

7.4 CONCLUSIONS AND DISCUSSION

7.4.1 *Tuning Evaluation Budget Comparisons*

Here we firstly presented a finding that suggests it is generally better, given no other indicators, to use an evaluation budget at least equal to the target evaluations when solving large-scale optimisation problems. This conclusion runs contrary to a current SMAC recommendation that one should use a budget of $10 \times D$ evaluations, where D is the problem dimensionality [62]. For the largest number of dimensions we experimented with (1448D) this equates to a budget recommendation of around 14,480 function evaluations. Further in this study, a suggestion that this should be further extended to $100 \times D$ evaluations [62]. The latter suggestion we feel is 'overkill' and only aids in worsening the situation in relation to the runtime of tuning processes - where common sense would dictate that actually, since an algorithm tuned for this number of evaluations targeting a lower budget would expect more time to explore the available space, that algorithm performance would worsen. We have observed a similar effect from the plots in Appendix A where an algorithm is tuned at a 50,000 evaluation budget, but when the algorithm reaches only 10,000 evaluations during experiments the performance

is shown to be significantly worse (compared to results from the 10,000 evaluation budget) over the various dimension thresholds of all our functions, with the exception of: Deflected Corrugated Spring, Inverted-Cosine-Wave and Rastrigin which showed little or no difference between the two budgets at 10,000 evaluations.

We also found no appreciable difference between tuning at either tuning evaluation budget for our implementation of GA (an SSGA). We conclude from our findings here that when using an SSGA making use of the same operators and solution representation for continuous function optimisation as we have here, the approach can be tuned at a significantly lower number of evaluations than the required target evaluation budget. Automatic parameter tuning can often be computationally expensive so this finding potentially eases this restriction that may prohibit APT for the other approaches we tested, allowing GA to be tuned faster.

Next we presented our findings relating to the initial observation that our implementation of PSO behaves differently from the others when tuned for our two evaluation budgets (10,000 and 50,000). In general, what we observed was that for all but $\approx \frac{1}{4}$ of our function suite, no deviation occurs between budgets at lower dimensions and after a significant divergence of performance in mid ranged dimensions when using different budgets a re-convergence occurs in higher dimensions. We initially interpreted these observations as there being no difference between using lower or higher tuning budgets. After some investigation we concluded further that the effect could be attributed to there being a threshold number of dimensions above which a larger population causes our implementation some difficulty. Similar past results in the field of Genetic Programming in [3], suggested that small population sizes were better at solving some problems. Further, evidence also shows that small population sizes can sometimes lead the search to the optimum solutions in fewer objective function evaluations - for certain problems - when compared to the use of larger sizes [147]. This may explain why a dramatic decrease in population size for the algorithm using the 10,000 evaluation parameter set produces comparable results to the alternate scheme (50,000 evaluation budget); SMAC could have determined that a smaller population size (increasing exploitation effects around any decent local minimum) would be more successful where the scale of the problem instances reached a point where a more equal balance between exploration and exploitation could not do better than a highly exploitative search. Indeed, in the large scale optimisation literature, small populations are used with LSOPs to 'brute-force' promising search directions [18], since full coverage of a very large search space is intractable using reasonable population sizes and/or function evaluation budgets. However, we currently have no explanation or

reason as to why SMAC found a high population size - which invariably produced worse solutions - to be the best configuration.

7.4.2 *Performance Comparison Between Algorithm Implementations*

Here we made the moderately surprising observation that our Random Mutation Hill Climbing (RMHC) algorithm regularly performs competitively with far more complex approaches. We conclude through further analysis of the available literature that results such as those discussed here oppose widely held opinions that simple algorithm like Stochastic Hill Climbing - of which RMHC is a variation - are not very useful when compared to more complex evolutionary algorithms. A contribution here is that although it is rare to find comparisons between more powerful EAs and simple hill climbing algorithms, we have now produced data for this kind of comparison, for large-scale optimisation, which can be made available to the wider research community through the University's Digital Repository.

7.4.3 *Rank Ordering Changes at Larger Problem Scales*

In general we can conclude that when viewed as the aggregate rank over several functions from the benchmark set, much of the noise from small intermittent re-orderings can be removed leaving only the most meaningful changes to rank order thus reflecting a clearer picture of comparative performance between the algorithms covered. Such meaningful transitions between the algorithms are therefore few, but occur around the medium to large problem scales as hypothesised and persist over the remaining (higher) scales. In terms of ranked performance over subsets of the benchmark suite, i.e., subsets delineated by the problem features: modality (uni-modal vs. multi-modal) and separability (separable vs. non-separable, there is not sufficient reason to conclude that any of the problem features considered pose any kind of differing levels of difficulty for any particular one of the metaheuristic implementations compared. Despite generally localised differences to the mean aggregate rankings for different problem subsets, these when compared by problem scale do not show much variation.

7.5 CONTRIBUTIONS

- We have produced a concise review of the subject area for future consumption by other researchers looking to make progress in this area

- We have generated a vast amount of data relating to the large-scale optimisation of several standard continuous function benchmarks found in the literature that can be made available to the wider community through the university's digital repository in due course.
- We have identified several behaviours of common metaheuristics as applied to optimisation problems with many hundreds of dimensions. Filling some of the gaps in our understanding of how the optimisation approaches, we may assume behave in certain ways, actually do behave when presented with much harder problems than have they have typically been investigated against.
- We have identified an effect in PSO in which for the majority of our benchmark set behaves unusually in the presence of large population sizes - an effect we felt to be counter-intuitive. Through further investigation we hope to further explain the causes of this effect and be better able to predict the type of functions where it is likely we will observe this effect occurring, and further, research into the possible benefits or disadvantages such a situation may present
- Results show that the SSGA used in this study can be tuned at a significantly lower evaluation budget than that of the intended target budget without affecting performance (in terms of solution quality). This finding carries the implication that the observed requirement of tuning at the target evaluation budget in other algorithms can be largely alleviated for SSGA - allowing the algorithm to be tuned faster.

7.6 FUTURE WORK

Based on some of the behaviours we are beginning to observe in even the most common of metaheuristic implementations, there is of course scope to continue this line of research - not only to develop more evidence to support our findings here but also to investigate further into the possible reasons - an overlap into the realm of 'Explanatory Metaheuristics' - for behavioural changes observed at the kind of scale we have been dealing with.

In particular, the discovery of a threshold above which large population sizes in PSO are unhelpful warrants much further investigation; as an understanding of how and when this effect will present itself may inspire improvements to the current state of affairs in regards to particle swarm optimisation research and state-of-the-art PSO algorithm performance as well as aiming to generate insight into how parameters interact within metaheuristic algorithms in general - a very interesting prospect quickly gaining momentum in the research community.

BIBLIOGRAPHY

[1] Emile Aarts, Jan Korst, and Wil Michiels. Simulated annealing. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, page 187, 2006.

[2] W. Al-Hassan, M. B. Fayek, and S. I. Shaheen. Psosa: An optimized particle swarm technique for solving the urban planning problem. In *2006 International Conference on Computer Engineering and Systems*, pages 401–405, Nov 2006. doi: 10.1109/ICCES.2006.320481.

[3] D. A. Ashlock, K. M. Bryden, and S. Corns. Small population effects and hybridization. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 2637–2643, June 2008. doi: 10.1109/CEC.2008.4631152.

[4] Anne Auger and Olivier Teytaud. Continuous lunches are free! In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 916–922. ACM, 2007.

[5] Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. *Evolutionary Computation 1: Basic algorithms and Operators*, volume 1. CRC press, 2000.

[6] Sunith Bandaru and Kalyanmoy Deb. Metaheuristic techniques. *Decision Sciences: Theory and Practice*, pages 693–750, 2016.

[7] Jagdish Chand Bansal, PK Singh, Mukesh Saraswat, Abhishek Verma, Shimpi Singh Jadon, and Ajith Abraham. Inertia weight strategies in particle swarm optimization. In *Nature and Biologically Inspired Computing (NaBIC), 2011 Third World Congress on*, pages 633–640. IEEE, 2011.

[8] Richard E Bellman. *Adaptive Control Processes: a Guided Tour*, volume 2045. Princeton University Press, 1961.

[9] John B Biggs. The role of metalearning in study processes. *British journal of educational psychology*, 55(3):185–212, 1985.

[10] James Blondin. Particle swarm optimization: A tutorial. *Availaible from: http://cs. armstrong. edu/saad/csci8100/pso tutorial. pdf*, 2009.

[11] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003.

[12] Yossi Borenstein and Alberto Moraglio. *Theory and Principled Methods for the Design of Metaheuristics*. Springer, 2014.

[13] Daniel Bratton and James Kennedy. Defining a standard for particle swarm optimization. In *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 120–127. IEEE, 2007.

[14] Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to Data Mining*. Springer Science & Business Media, 2008.

[15] Leo Breiman and Adele Cutler. Random forests. URL `https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#remarks`.

[16] Jason Brownlee. *Clever Algorithms: Nature-inspired Programming Recipes*. Jason Brownlee, 2011.

[17] Edmund K Burke, Graham Kendall, et al. *Search methodologies*. Springer, 2005.

[18] Fabio Caraffini, Ferrante Neri, and Giovanni Iacca. Large scale problems in practice: The effect of dimensionality on the interaction among variables. In Giovanni Squillero and Kevin Sim, editors, *Applications of Evolutionary Computation*, pages 636–652. Springer International Publishing, 2017. ISBN 978-3-319-55849-3.

[19] Wei Chu, Xiaogang Gao, and Soroosh Sorooshian. Handling boundary constraints for particle swarm optimization in high-dimensional search space. *Information Sciences*, 181(20):4569–4581, October 2011. ISSN 0020-0255. doi: 10.1016/j.ins.2010.11.030. URL `http://dx.doi.org/10.1016/j.ins.2010.11.030`.

[20] Maurice Clerc. The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages 1951–1957. IEEE, 1999.

[21] Maurice Clerc. *Guided randomness in optimization*, volume 1. John Wiley & Sons, 2015.

[22] David A Coley. *An introduction to genetic algorithms for scientists and engineers*. World Scientific Publishing Company, 1999.

[23] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1):4–31, 2011.

[24] Kenneth A De Jong and Jayshree Sarma. Generation gaps revisited. In *Foundations of Genetic Algorithms*, volume 2, pages 19–28. Elsevier, 1993.

[25] Kenneth Alan De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, MI, USA, 1975. AAI7609381.

[26] Kalyanmoy Deb and Debayan Deb. Analysing mutation schemes for real-parameter genetic algorithms. *International Journal of Artificial Intelligence and Soft Computing*, 4(1): 1–28, 2014.

[27] Ke-Lin Du and MNS Swamy. Simulated annealing. In *Search and Optimization by Metaheuristics*, pages 29–36. Springer, 2016.

[28] Russell C Eberhart and Yuhui Shi. Tracking and optimizing dynamic systems with particle swarms. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 94–100. IEEE, 2001.

[29] Agoston E Eiben, James E Smith, et al. *Introduction to Evolutionary Computing*, volume 53. Springer, 2003.

[30] Andries P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley Publishing, 2nd edition, 2007. ISBN 0470035617.

[31] Yong Feng, Gui-Fa Teng, Ai-Xin Wang, and Yong-Mei Yao. Chaotic inertia weight in particle swarm optimization. In *Innovative Computing, Information and Control, 2007. ICICIC'07. Second International Conference on*, pages 475–475. IEEE, 2007.

[32] Steffen Finck, Nikolaus Hansen, Raymond Ros, and Anne Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical report, Citeseer, 2010.

[33] Len Fisher. *The Perfect Swarm: The Science of Complexity in Everyday Life*. Basic Books, 2009.

[34] Ronald Aylmer Fisher. *The genetical theory of natural selection: a complete variorum edition*. Oxford University Press, 1999.

[35] Stephanie Forrest and Melanie Mitchell. Relative building-block fitness and the building-block hypothesis. In *Foundations of genetic algorithms*, volume 2, pages 109–126. Elsevier, 1993.

[36] Andrea Gavana. Test functions index - ampgo 0.1.0 documentation. URL `http://infinity77.net/global_optimization/test_functions.html#test-functions-index`.

[37] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. In *Readings in Computer Vision*, pages 564–584. Elsevier, 1987.

[38] Robert Ghanea-Hercock. *Applied Evolutionary Algorithms in Java*. Springer Science & Business Media, 2013.

[39] F. Glover and K. Sörensen. Metaheuristics. *Scholarpedia*, 10(4):6532, 2015. doi: 10.4249/scholarpedia.6532. revision #149834.

[40] Jochen Gortler, Rebecca Kehlbeck, and Oliver Deussen. Visual exploration of gaussian processes, 2019. URL `https://distill.pub/2019/visual-exploration-gaussian-processes/`.

[41] Kevin Graham and Leslie Smith. Comparing hyper-heuristics with blackboard systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1141–1145, 2017.

[42] Kevin Graham, Jerry Swan, and Simon Martin. The 'blackboard pattern' for metaheuristics. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1265–1267, 2015.

[43] Bruce Hajek. Cooling schedules for optimal annealing. *Mathematics of operations research*, 13(2):311–329, 1988.

[44] Nikolaus Hansen. Cma evolution strategy source code, 2011. URL `http://cma.gforge.inria.fr/cmaes_sourcecode_page.html`.

[45] Nikolaus Hansen. The cma evolution strategy: A tutorial. 2016.

[46] Nikolaus Hansen and Stefan Kern. Evaluating the cma evolution strategy on multimodal test functions. In *International Conference on Parallel Problem Solving from Nature*, pages 282–291. Springer, 2004.

[47] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical report, 2010.

[48] Nikolaus Hansen, Raymond Ros, Nikolas Mauny, Marc Schoenauer, and Anne Auger. Impacts of invariance in search: When cma-es and pso face ill-conditioned and non-separable problems. *Applied Soft Computing*, 11(8):5755–5769, 2011.

[49] Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tusar, and Dimo Brockhoff. Coco: A platform for comparing continuous optimizers in a black-box setting. *arXiv preprint arXiv:1603.08785*, 2016.

[50] Randy L Haupt and Sue Ellen Haupt. *Practical genetic algorithms*, volume 2. Wiley New York, 2004.

[51] Abdel-Rahman Hedar. Global optimisation test problems. URL `http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm`.

[52] Sabine Helwig and Rolf Wanka. Theoretical analysis of initial particle swarm behavior. In *International conference on parallel problem solving from nature*, pages 889–898. Springer, 2008.

[53] Darrall Henderson, Sheldon H Jacobson, and Alan W Johnson. The theory and practice of simulated annealing. In *Handbook of metaheuristics*, pages 287–319. Springer, 2003.

[54] J. H. Holland. Genetic algorithms. *Scholarpedia*, 7(12):1482, 2012. doi: 10.4249/scholarpedia.1482. revision #128222.

[55] John H. Holland. *Hidden Order: How Adaptation Builds Complexity*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1995. ISBN 0-201-40793-0.

[56] S. Hollasch. *Four-space Visualization of 4D Objects*. PhD thesis, 1991.

[57] Qiang Huang, Thomas White, Guanbo Jia, Mirco Musolesi, Nil Turan, Ke Tang, Shan He, John K Heath, and Xin Yao. Community detection using cooperative co-evolutionary differential evolution. In *International Conference on Parallel Problem Solving from Nature*, pages 235–244, 2012.

[58] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration (extended version). Technical Report TR-2010-10, University of British Columbia, Department of Computer Science, 2010. Available online: http://www.cs.ubc.ca/~hutter/papers/10-TR-SMAC.pdf.

[59] F. Hutter, H.H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, pages 507–523, 2011.

[60] F. Hutter, H.H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration (slide presentation), 2011.

[61] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Kevin P Murphy. An experimental investigation of model-based parameter optimisation: Spo and beyond. In

Proceedings of the 11th Annual conference on Genetic and evolutionary computation, pages 271–278. ACM, 2009.

[62] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. An evaluation of sequential model-based optimiation for expensive blackbox functions. In *GECCO*, 2013.

[63] Momin Jamil and Xin-She Yang. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194, 2013.

[64] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[65] Yılmaz Kaya, Murat Uyar, et al. A novel crossover operator for genetic algorithms: Ring crossover. *arXiv preprint arXiv:1105.0355*, 2011.

[66] J. Kennedy. Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1931–1938 Vol. 3, July 1999. doi: 10.1109/CEC.1999.785509.

[67] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, Nov 1995. doi: 10.1109/ICNN.1995.488968.

[68] James Kennedy and Rui Mendes. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 2, pages 1671–1676. IEEE, 2002.

[69] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[70] Jack PC Kleijnen. Response surface methodology. In *Handbook of simulation optimization*, pages 81–104. Springer, 2015.

[71] Oscar Knagg. An intuitive guide to gaussian processes, 2015. URL `https://towardsdatascience.com/an-intuitive-guide-to-gaussian-processes-ec2f0b45c71d`.

[72] Johannes W Kruisselbrink, Rui Li, Edgar Reehuis, Jeroen Eggermont, and Thomas Bäck. On the log-normal self-adaptation of the mutation rate in binary search spaces. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 893–900. ACM, 2011.

[73] Manuel Laguna and Rafael Martí. Experimental testing of advanced scatter search designs for global optimization of multimodal functions. *Journal of Global Optimization*, 33(2):235–255, 2005.

[74] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. Metalearning: a survey of trends and technologies. *Artificial intelligence review*, 44(1):117–130, 2015.

[75] Xiaodong Li and Xin Yao. Tackling high dimensional nonseparable optimisation problems by cooperatively coevolving particle swarms. In *2009 IEEE Congress on Evolutionary Computation*, pages 1546–1553, 2009.

[76] Xiaodong Li and Xin Yao. Cooperatively coevolving particle swarms for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 16(2):210–224, 2011.

[77] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196, 2012.

[78] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. Iterated racing for automatic algorithm configuration [r package irace version 3.3]. URL `https://cran.r-project.org/package=irace`.

[79] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[80] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of Metaheuristics*, pages 320–353. Springer, 2003.

[81] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search: Framework and applications. In *Handbook of metaheuristics*, pages 363–397. Springer, 2010.

[82] Sean Luke. *Essentials of metaheuristics*, volume 113. Lulu Raleigh, 2009.

[83] X. Ma, X. Li, Q. Zhang, K. Tang, Z. Liang, W. Xie, and Z. Zhu. A survey on cooperative co-evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 23(3):421–441, June 2019. ISSN 1089-778X. doi: 10.1109/TEVC.2018.2868770.

[84] Sarim Mahmood, Scott Rosenquist, and Bryce Benn. skarjoko/differential-evolution, 2013. URL `https://github.com/skarjoko/differential-evolution/blob/master/DifferentialEvolution.java`.

[85] Gandhi Manalu. Particle swarm optimization: Sample code using java, 2010. URL https://gandhim.wordpress.com/2010/04/04/particle-swarm-optimization-pso-sample-code-using-java/.

[86] Gandhi Manalu. Particle swarm optimization: Sample code using java, 2014. URL https://github.com/therealmanalu/pso-example-java.

[87] Angelina Jane Reyes Medina, Gregorio Toscano Pulido, and José Gabriel Ramírez-Torres. A comparative study of neighborhood topologies for particle swarm optimizers. In *IJCCI*, pages 152–159, 2009.

[88] Olaf Mersmann, Mike Preuss, Heike Trautmann, Bernd Bischl, and Claus Weihs. Analyzing the bbob results by means of benchmarking concepts. *Evolutionary computation*, 23(1):161–185, 2015.

[89] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)*. Springer-Verlag, London, UK, UK, 1996. ISBN 3-540-60676-9.

[90] Zbigniew Michalewicz and David B Fogel. *How to solve it: modern heuristics*. Springer Science & Business Media, 2013.

[91] Zbigniew Michalewicz, Thomas Logan, and Swarnalatha Swaminathan. Evolutionary operators for continuous convex parameter spaces. In *Proceedings of the 3rd Annual conference on Evolutionary Programming*, pages 84–97. World Scientific, 1994.

[92] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[93] Marcin Molga and Czesław Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, 101, 2005.

[94] John A Nelder and Roger Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[95] Ferrante Neri and Ville Tirronen. Recent advances in differential evolution: a survey and experimental analysis. *Artificial Intelligence Review*, 33(1-2):61–106, 2010.

[96] Alexander G Nikolaev and Sheldon H Jacobson. Simulated annealing. In *Handbook of Metaheuristics*, pages 1–39. Springer, 2010.

[97] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2nd edition, 2006.

[98] Yaghout Nourani and Bjarne Andresen. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373, 1998.

[99] Mohammad Nabi Omidvar, Xiaodong Li, Zhenyu Yang, and Xin Yao. Cooperative co-evolution for large scale optimization through more frequent random grouping. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[100] Mohammad Nabi Omidvar, Xiaodong Li, and Xin Yao. Cooperative co-evolution with delta grouping for large scale non-separable function optimisation. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[101] Nikhil Padhye, Kalyanmoy Deb, and Pulkit Mittal. Boundary handling approaches in particle swarm optimization. In *Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012)*, pages 287–298. Springer, 2013.

[102] Lucas Pavelski, Myriam Delgado, and Marie-Eleonore Kessaci. Meta-learning for optimization: A case study on the flowshop problem using decision trees. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018.

[103] Peter Frank Perroni, Daniel Weingaertner, and Myriam Regattieri Delgado. Automated iterative partitioning for cooperatively coevolving particle swarms in large scale optimization. In *2015 Brazilian Conference on Intelligent Systems (BRACIS)*, pages 19–24, 2015.

[104] Stjepan Picek, Domagoj Jakobovic, and Marin Golub. On the recombination operator in the real-coded genetic algorithms. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 3103–3110. IEEE, 2013.

[105] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[106] Elena Popovici, Anthony Bucci, R. Paul Wiegand, and Edwin D. De Jong. *Coevolutionary Principles*, pages 987–1033. Springer, Berlin, Heidelberg, 2012. ISBN 978-3-540-92910-9. doi: 10.1007/978-3-540-92910-9_31. URL `https://doi.org/10.1007/978-3-540-92910-9_31`.

[107] Mitchell A Potter and Kenneth A De Jong. A cooperative coevolutionary approach to function optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 249–257, 1994.

[108] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential Evolution: a Practical Approach to Global Optimization*. Springer Science & Business Media, 2006.

[109] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[110] Asanga Ratnaweera, Saman K Halgamuge, and Harry C Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on evolutionary computation*, 8(3):240–255, 2004.

[111] Colin Reeves. Genetic algorithms. In *Handbook of metaheuristics*, pages 55–82. Springer, 2003.

[112] John R Rice. The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier, 1976.

[113] Alex Rogers and Adam Prügel-Bennett. Modelling the dynamics of a steady-state genetic algorithm. *Foundations of genetic algorithms*, 5:57–68, 1999.

[114] Raymond Ros and Nikolaus Hansen. A simple modification in cma-es achieving linear time and space complexity. In *International Conference on Parallel Problem Solving from Nature*, pages 296–305. Springer, 2008.

[115] Alejandro Rosete-Suárez, Alberto Ochoa-Rodríguez, and Michele Sebag. Automatic graph drawing and stochastic hill climbing. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2*, GECCO'99, pages 1699–1706, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-611-4. URL http://dl.acm.org/citation.cfm?id=2934046.2934177.

[116] Franz Rothlauf. *Design of Modern Heuristics: Principles and Application*. Springer Science & Business Media, 2011.

[117] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[118] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981. ISBN 0471099880.

[119] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73. IEEE, 1998.

[120] Yuhui Shi and Russell C Eberhart. Empirical study of particle swarm optimization. In *Evolutionary computation, 1999. CEC 99. Proceedings of the 1999 congress on*, volume 3, pages 1945–1950. IEEE, 1999.

[121] John Silberholz and Bruce Golden. *Comparison of Metaheuristics*, pages 625–640. Springer US, Boston, MA, 2010. ISBN 978-1-4419-1665-5. doi: 10.1007/978-1-4419-1665-5_21. URL https://doi.org/10.1007/978-1-4419-1665-5_21.

[122] S. Singer and J. Nelder. Nelder-Mead algorithm. *Scholarpedia*, 4(7):2928, 2009. doi: 10.4249/scholarpedia.2928. revision #91557.

[123] Sanja Singer and Saša Singer. Complexity analysis of nelder-mead search iterations. In *Proceedings of the 1. Conference on Applied Mathematics and Computation, Dubrovnik, Croatia*, pages 185–196. PMF–Matematički odjel, Zagreb, 1999.

[124] Jim Smith and Terence C. Fogarty. An adaptive poly-parental recombination strategy. In *AISB Workshop on Evolutionary Computing*, pages 48–61, 1995.

[125] Jim Smith and Frank Vavak. Replacement strategies in steady state genetic algorithms: Static environments. *Foundations of genetic algorithms*, 5:219–233, 1999.

[126] Kate A Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6, 2009.

[127] Padhraic Smyth. Local search and optimization, 2007.

[128] Nitasha Soni and Tapas Kumar. Study of various mutation operators in genetic algorithms. *International Journal of Computer Science and Information Technologies (IJCSIT)*, 3, 2014. ISSN 0975-9646.

[129] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4): 341–359, 1997.

[130] Philip N Strenski and Scott Kirkpatrick. Analysis of finite length annealing schedules. *Algorithmica*, 6(1-6):346–366, 1991.

[131] T Stützle. *Local Search Algorithms for Combinatorial Problems*. PhD thesis, 1998.

[132] Ponnuthurai N Suganthan. Particle swarm optimiser with neighbourhood operator. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages 1958–1962. IEEE, 1999.

[133] Ponnuthurai N Suganthan, Nikolaus Hansen, Jing J Liang, Kalyanmoy Deb, Ying-Ping Chen, Anne Auger, and Santosh Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. *KanGAL report*, page 2005, 2005.

[134] Sonja Surjanovic and Derek Bingham. Virtual library of simulation experiments: Test functions and datasets, 2013. URL https://www.sfu.ca/~ssurjano/.

[135] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[136] Aimo Törn, Montaz M Ali, and Sami Viitanen. Stochastic global optimization: Problem classes and solution techniques. *Journal of Global Optimization*, 14(4):437–447, 1999.

[137] Henrik Valeur. *India: the Urban Transition - A Case Study of Development Urbanism*. The Architectural Publisher, 2014.

[138] Frans van den Bergh and Andries Petrus Engelbrecht. A cooperative approach to particle swarm optimization. 8(3):225–239, 2004.

[139] Joaquin Vanschoren. *Understanding Machine Learning Performance With Experiment Databases*. PhD thesis, 2010.

[140] F. Vavak and T. C. Fogarty. A comparative study of steady state and generational genetic algorithms for use in nonstationary environments. In Terence C. Fogarty, editor, *Evolutionary Computing*, pages 297–304, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-70671-7.

[141] Michael D Vose. Continuous lunches are not free. *arXiv preprint arXiv:1507.00581*, 2015.

[142] Ron Wehrens and Lutgarde MC Buydens. Classical and nonclassical optimization methods. *Encyclopedia of Analytical Chemistry*.

[143] Darrell Whitley. Genetic algorithms and evolutionary computing. *Van Nostrand's Scientific Encyclopedia*, 2002.

[144] Darrell Whitley and Joan Kauth. GENITOR: A Different Genetic Algorithm. In *Proceedings of the 1988 Rocky Mountain Conference on Artificial Intelligence*, pages 118–130. Computer Science Department, Colorado State University, 1988.

[145] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[146] David H Wolpert, William G Macready, et al. No free lunch theorems for search. Technical report, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[147] Jonathan Wright and Ali Alajmi. Efficient genetic algorithm sets for optimizing constrained building design problem. *International Journal of Sustainable Built Environment*, 5 (1):123–131, 2016.

[148] Jianbin Xin, Guimin Chen, and Yubao Hai. A particle swarm optimizer with multi-stage linearly-decreasing inertia weight. In *Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on*, volume 1, pages 505–508. IEEE, 2009.

[149] S. Xu and Y. Rahmat-Samii. Boundary conditions in particle swarm optimization revisited. *IEEE Transactions on Antennas and Propagation*, 55(3):760–765, March 2007. ISSN 0018-926X. doi: 10.1109/TAP.2007.891562.

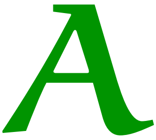
[150] Zhenyu Yang, Ke Tang, and Xin Yao. Differential evolution for high-dimensional function optimisation. In *2007 IEEE Congress on Evolutionary Computation*, pages 3523–3530, 2007.

[151] Zhenyu Yang, Ke Tang, and Xin Yao. Large scale evolutionary optimisation using cooperative coevolution. *Information Sciences*, 178(15):2985–2999, 2008.

[152] z. Cao, L. Wang, Y. Shi, X. Hei, X. Rong, Q. Jiang, and H. Li. An effective cooperative coevolution framework integrating global and local search for large scale optimization problems. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 1986–1993, May 2015. doi: 10.1109/CEC.2015.7257129.

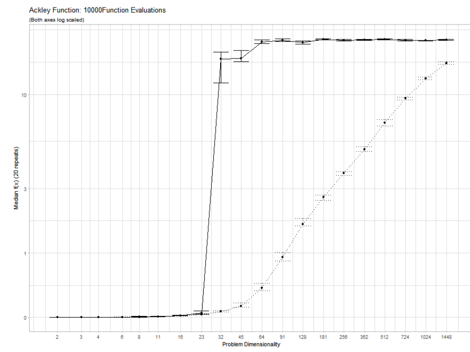
Part V

APPENDICES

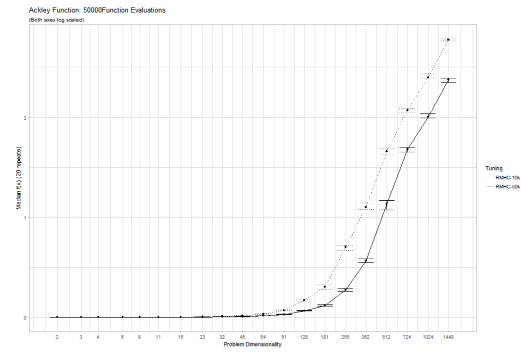


APPENDIX A: 10,000 AND 50,000 EVALUATION COMPARISON PLOTS

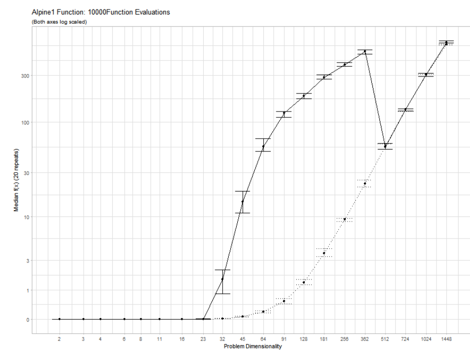
RANDOM MUTATION HILL CLIMBING (RMHC)



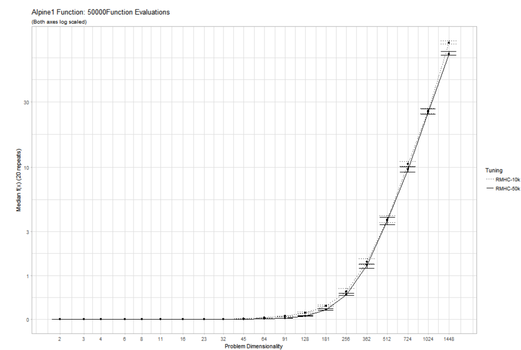
Ackley Function at 10,000 Evaluations



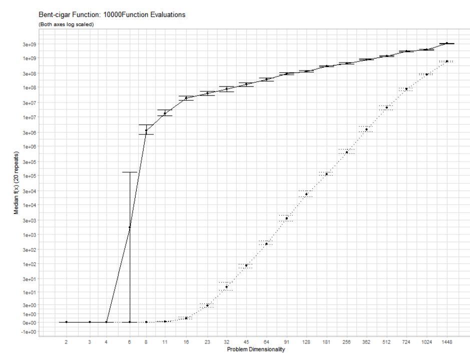
Ackley Function at 50,000 Evaluations



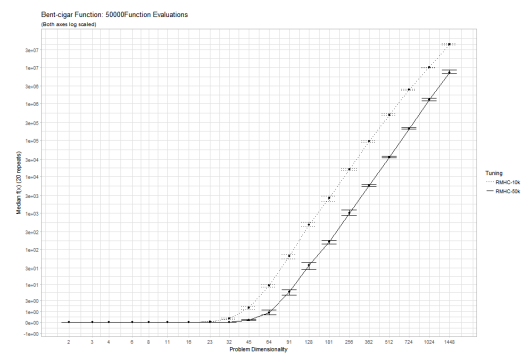
Alpine no.1 Function at 10,000 Evaluations



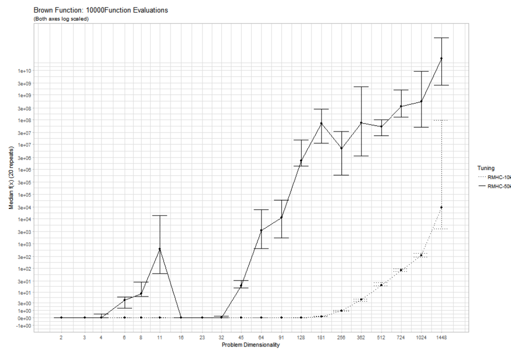
Alpine no.1 Function at 50,000 Evaluations



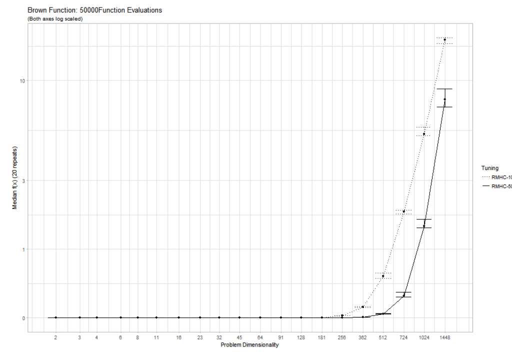
Bent Cigar Function at 10,000 Evaluations



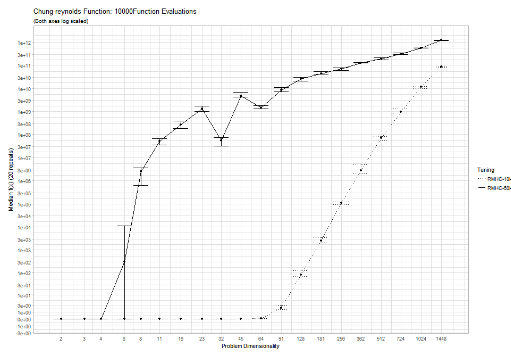
Bent Cigar Function at 50,000 Evaluations



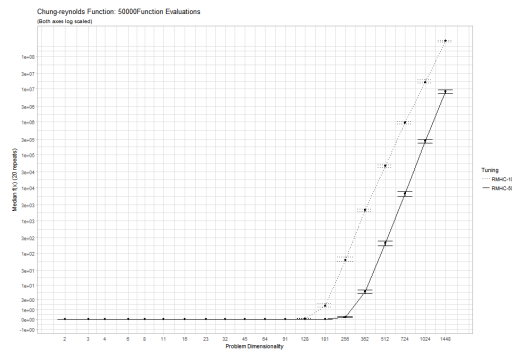
Brown Function at 10,000 Evaluations



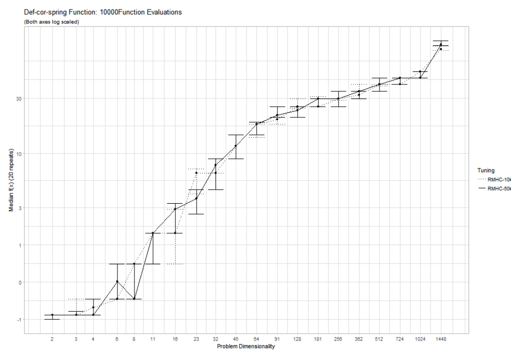
Brown Function at 50,000 Evaluations



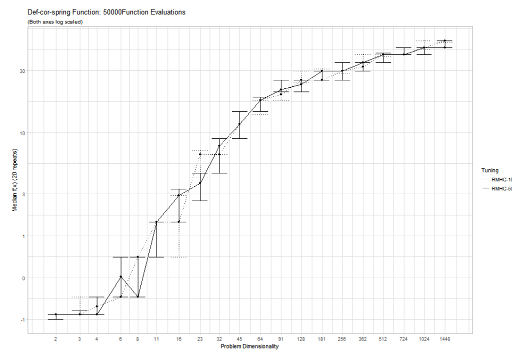
Chung-Reynolds Function at 10,000 Evaluations



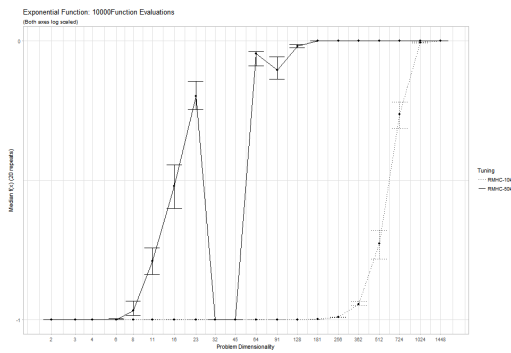
Chung-Reynolds Function at 50,000 Evaluations



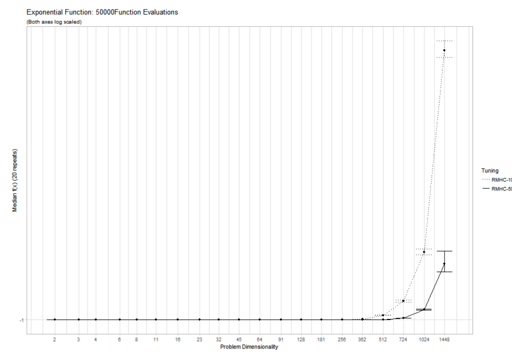
Deflected Corrugated Spring Function at 10,000 Evaluations



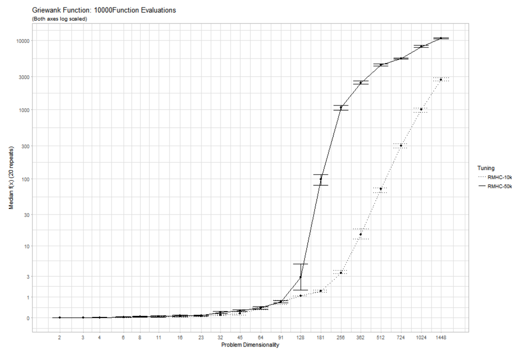
Deflected Corrugated Spring Function at 50,000 Evaluations



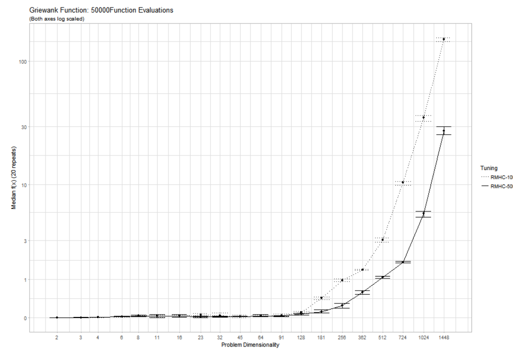
Exponential Function at 10,000 Evaluations



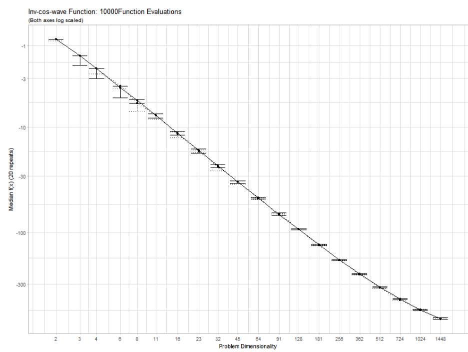
Exponential Function at 50,000 Evaluations



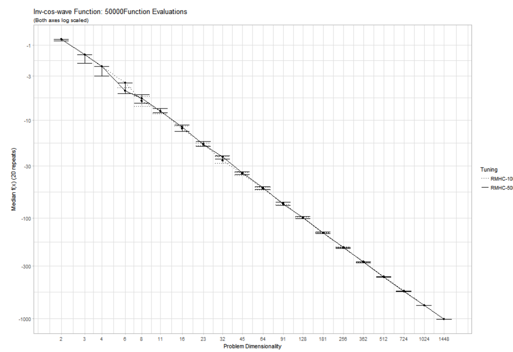
Griewank Function at 10,000 Evaluations



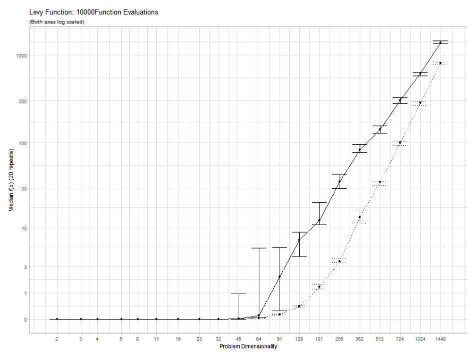
Griewank Function at 50,000 Evaluations



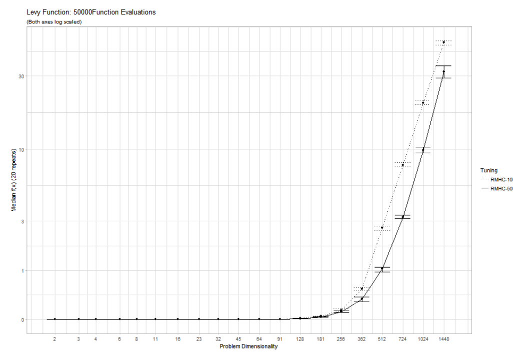
Inverted Cosine Wave Function at 10,000 Evaluations



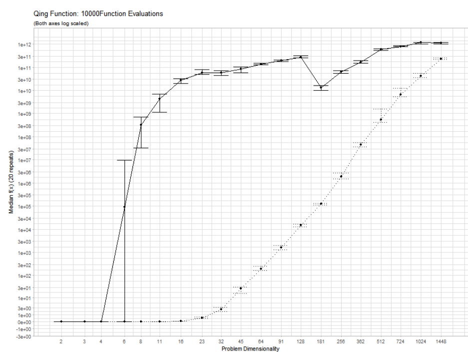
Inverted Cosine Wave Function at 50,000 Evaluations



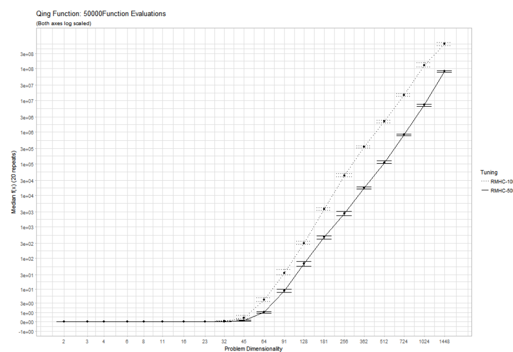
Levy Function at 10,000 Evaluations



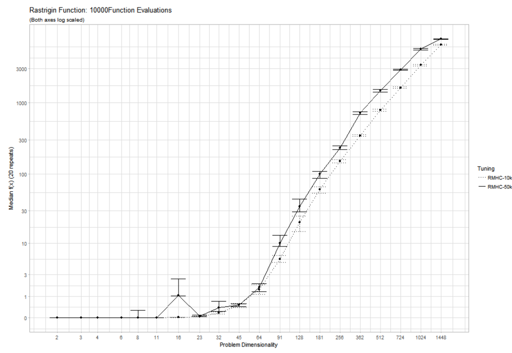
Levy Function at 50,000 Evaluations



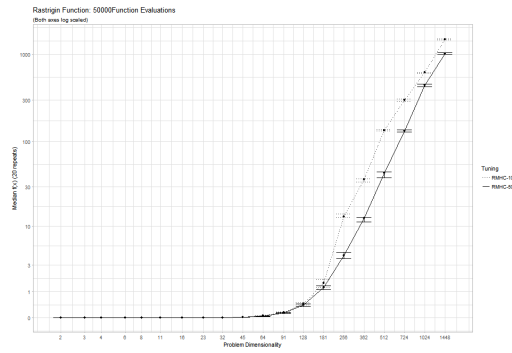
Qing Function at 10,000 Evaluations



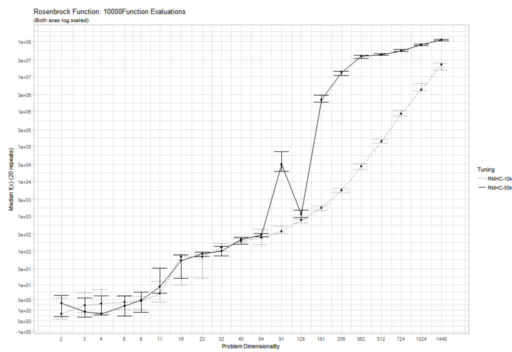
Qing Function at 50,000 Evaluations



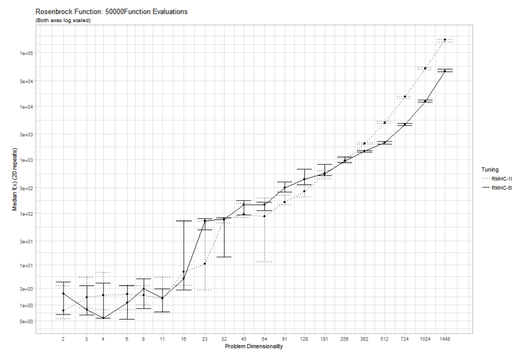
Rastrigin Function at 10,000 Evaluations



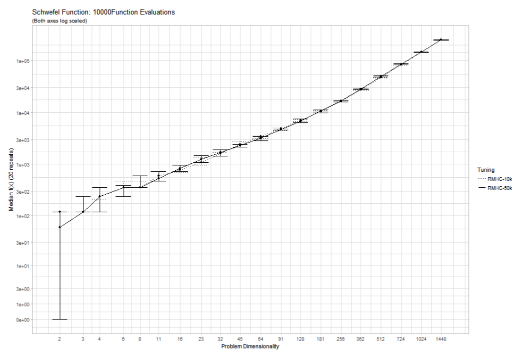
Rastrigin Function at 50,000 Evaluations



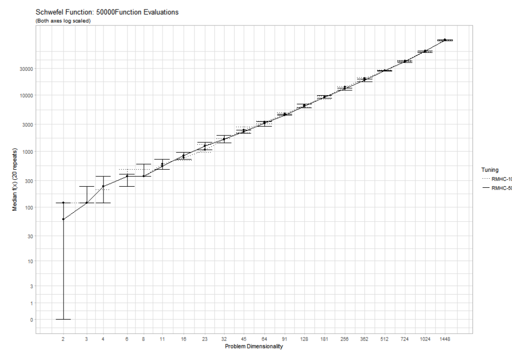
Rosenbrock Function at 10,000 Evaluations



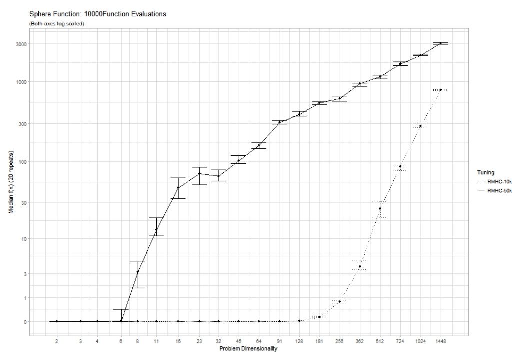
Rosenbrock Function at 50,000 Evaluations



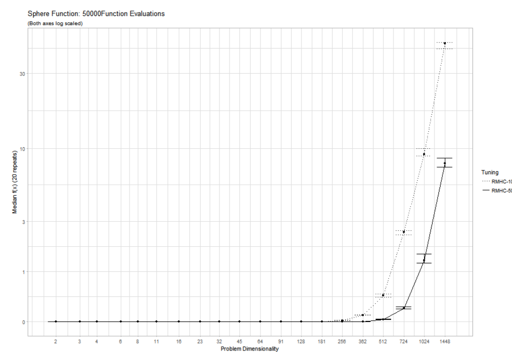
Schwefel Function at 10,000 Evaluations



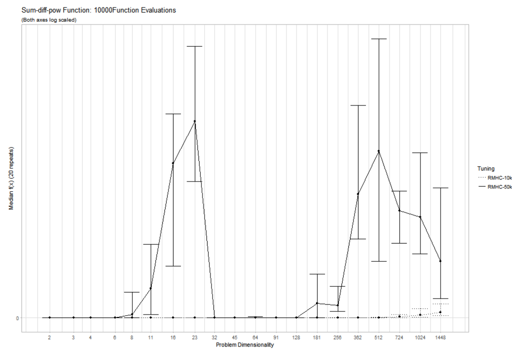
Schwefel Function at 50,000 Evaluations



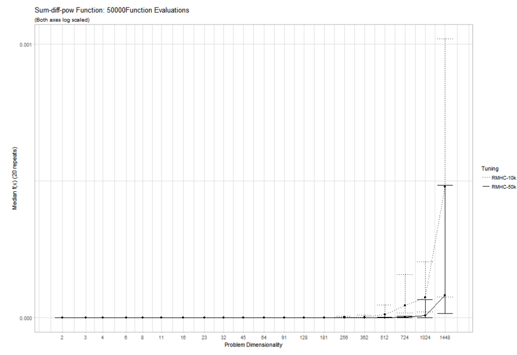
Sphere Function at 10,000 Evaluations



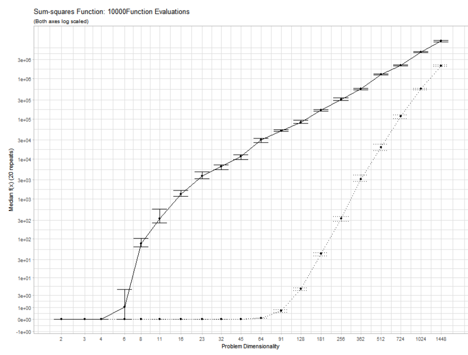
Sphere Function at 50,000 Evaluations



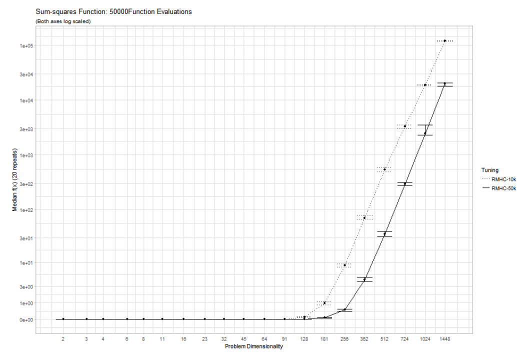
Sum of Different Powers Function at 10,000 Evaluations



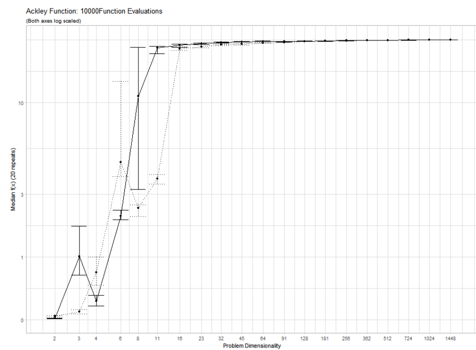
Sum of Different Powers Function at 50,000 Evaluations



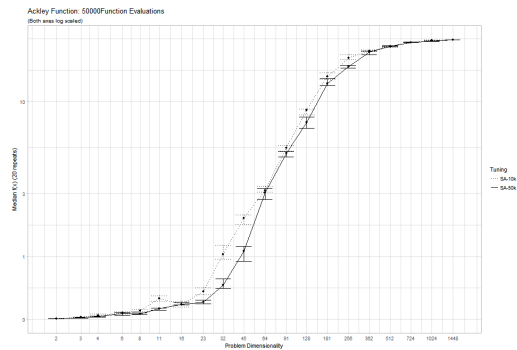
Sum Squares Function at 10,000 Evaluations



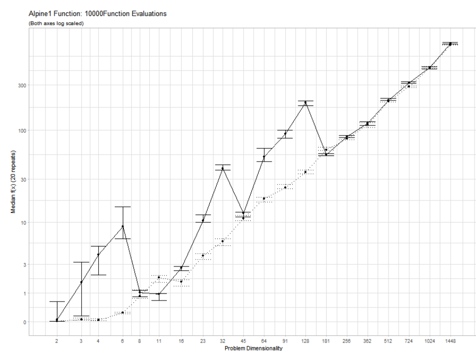
Sum Squares Function at 50,000 Evaluations



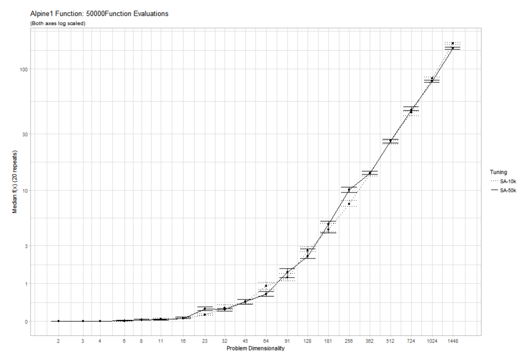
Ackley Function at 10,000 Evaluations



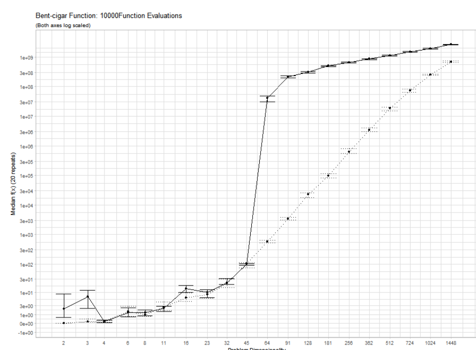
Ackley Function at 50,000 Evaluations



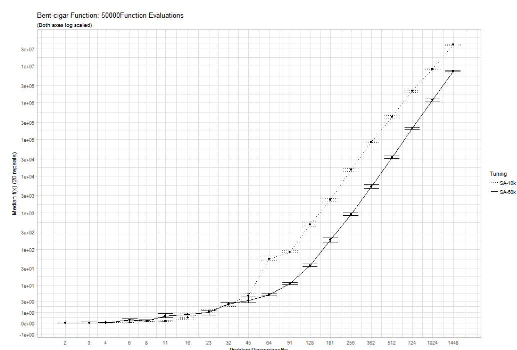
Alpine no.1 Function at 10,000 Evaluations



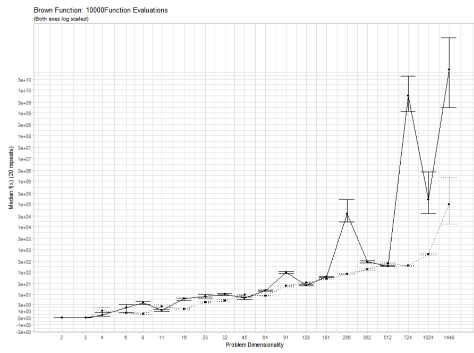
Alpine no.1 Function at 50,000 Evaluations



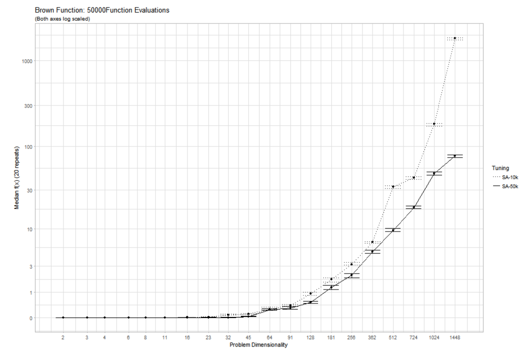
Bent Cigar Function at 10,000 Evaluations



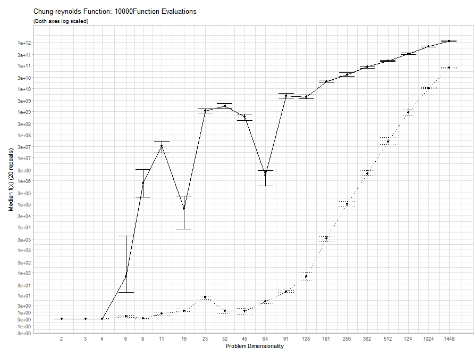
Bent Cigar Function at 50,000 Evaluations



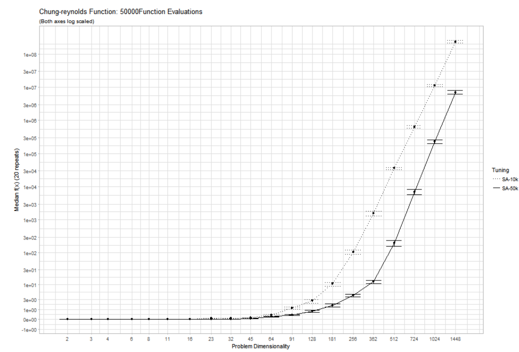
Brown Function at 10,000 Evaluations



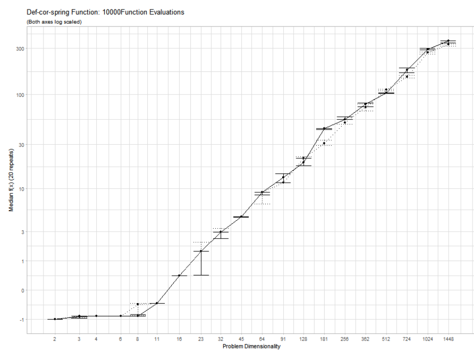
Brown Function at 50,000 Evaluations



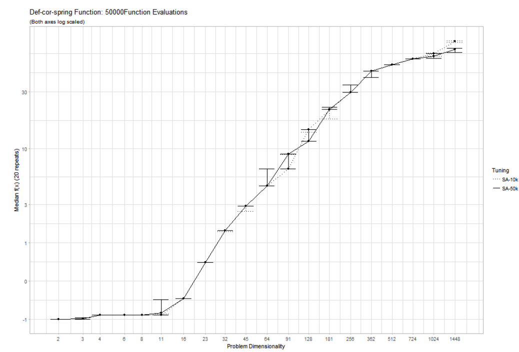
Chung-Reynolds Function at 10,000 Evaluations



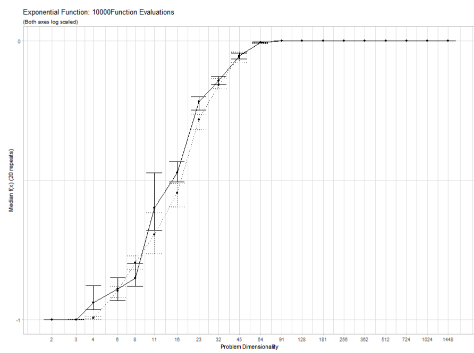
Chung-Reynolds Function at 50,000 Evaluations



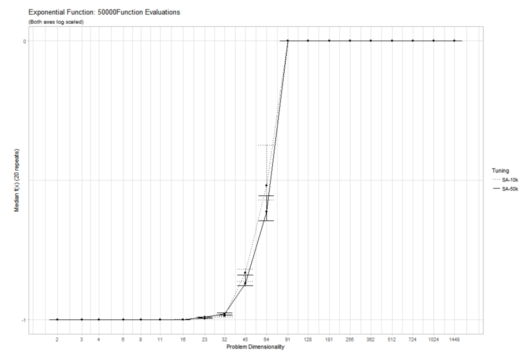
Deflected Corrugated Spring Function at 10,000 Evaluations



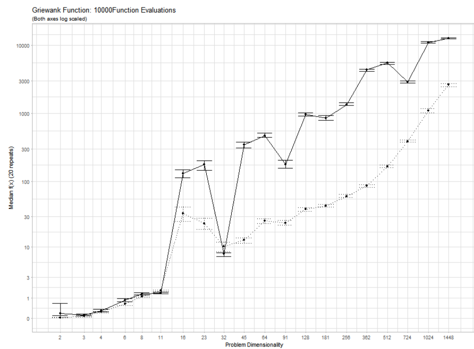
Deflected Corrugated Spring Function at 50,000 Evaluations



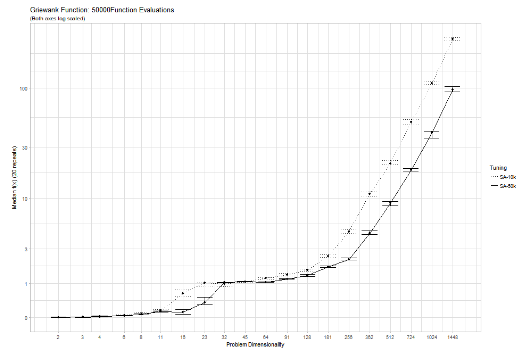
Exponential Function at 10,000 Evaluations



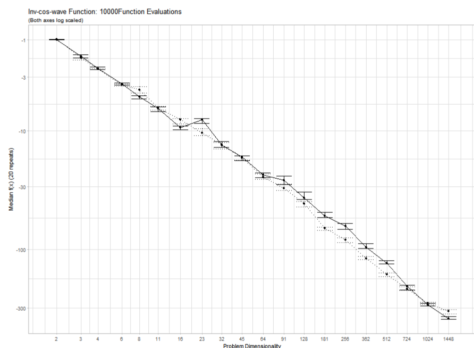
Exponential Function at 50,000 Evaluations



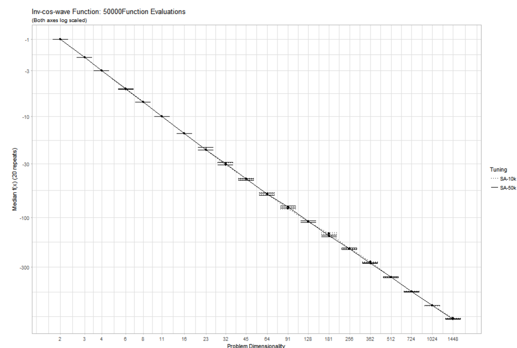
Griewank Function at 10,000 Evaluations



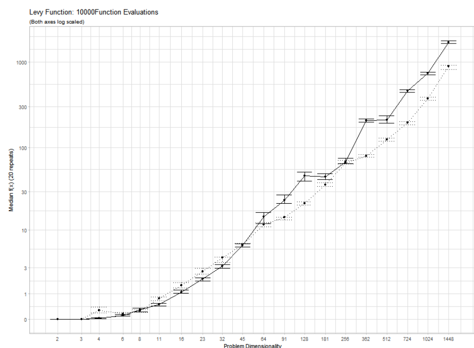
Griewank Function at 50,000 Evaluations



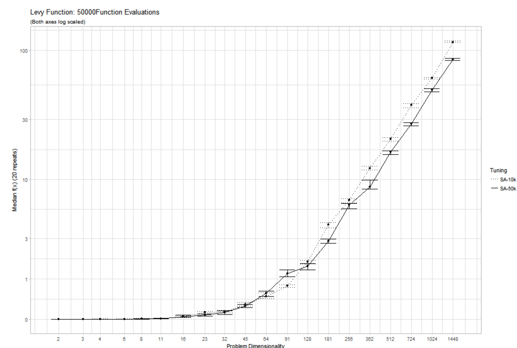
Inverted Cosine Wave Function at 10,000 Evaluations



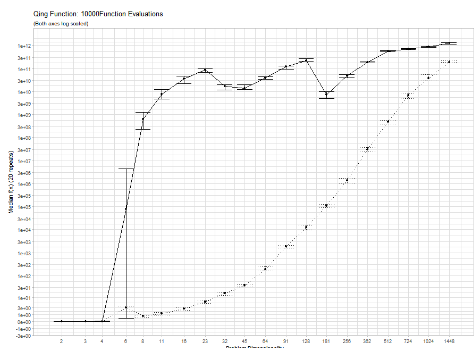
Inverted Cosine Wave Function at 50,000 Evaluations



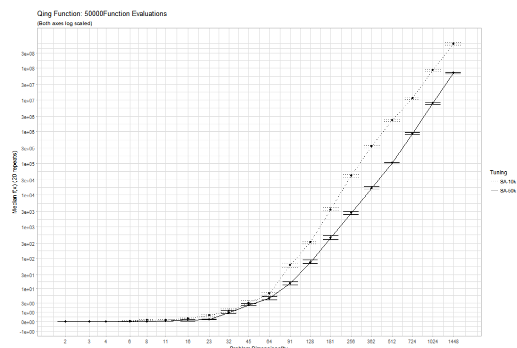
Levy Function at 10,000 Evaluations



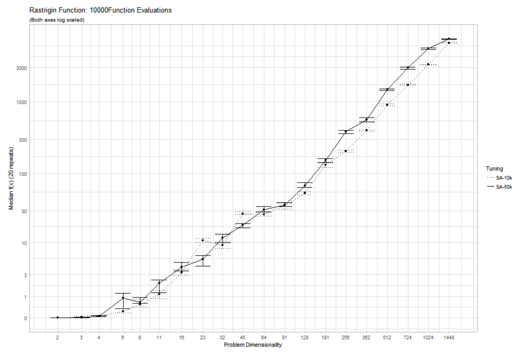
Levy Function at 50,000 Evaluations



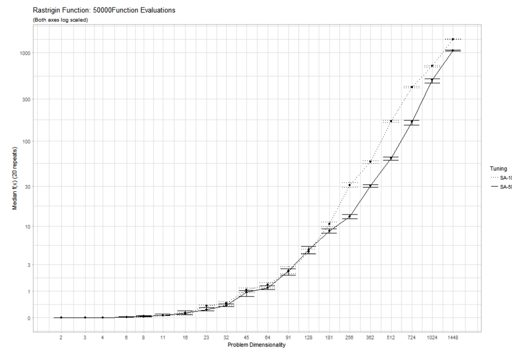
Qing Function at 10,000 Evaluations



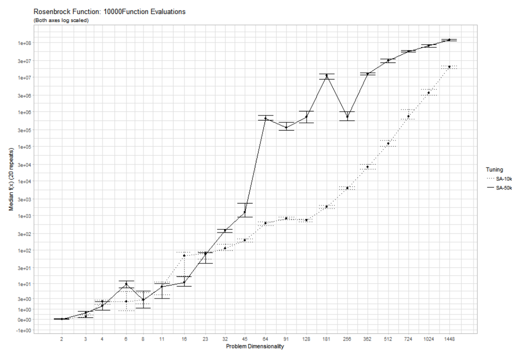
Qing Function at 50,000 Evaluations



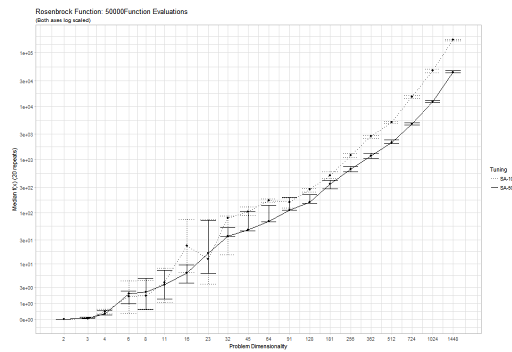
Rastrigin Function at 10,000 Evaluations



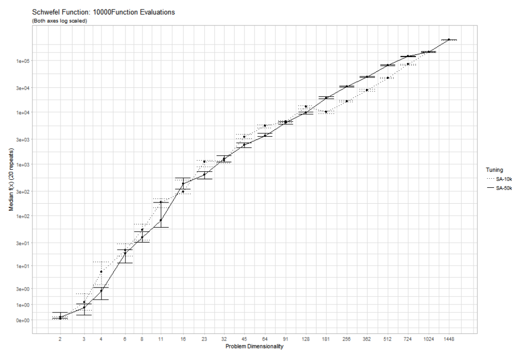
Rastrigin Function at 50,000 Evaluations



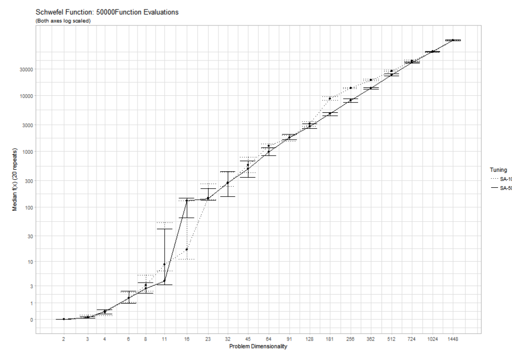
Rosenbrock Function at 10,000 Evaluations



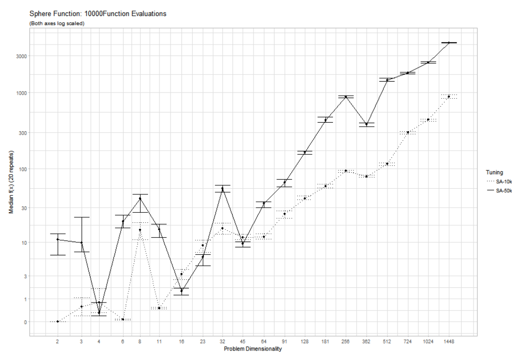
Rosenbrock Function at 50,000 Evaluations



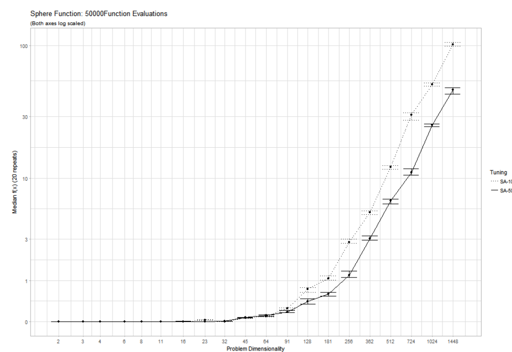
Schwefel Function at 10,000 Evaluations



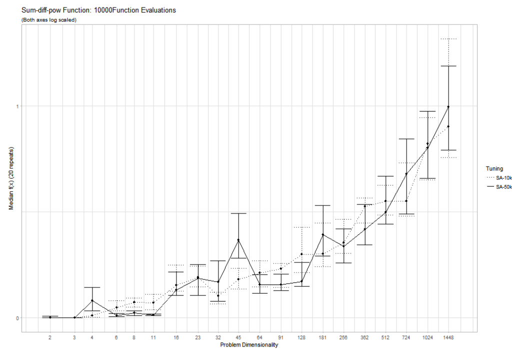
Schwefel Function at 50,000 Evaluations



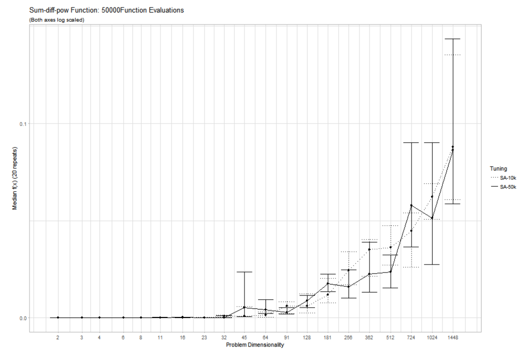
Sphere Function at 10,000 Evaluations



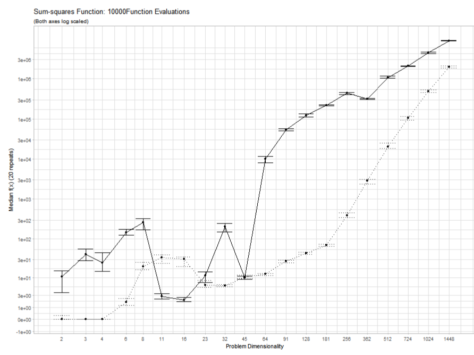
Sphere Function at 50,000 Evaluations



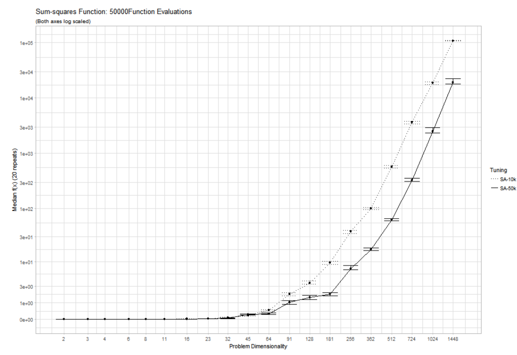
Sum of Different Powers Function at 10,000 Evaluations



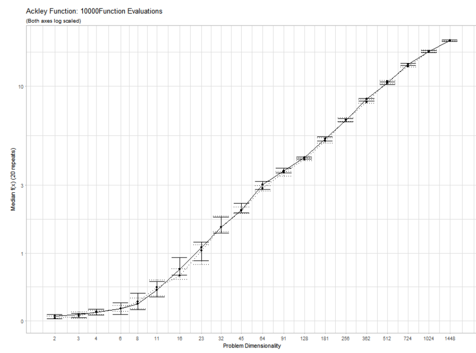
Sum of Different Powers Function at 50,000 Evaluations



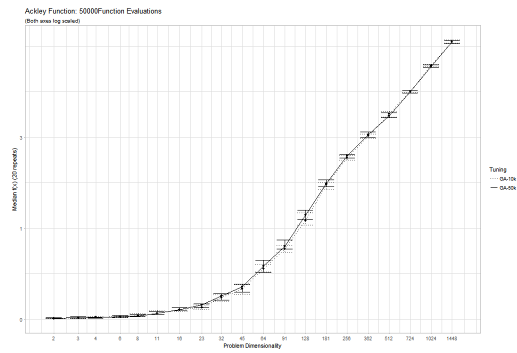
Sum Squares Function at 10,000 Evaluations



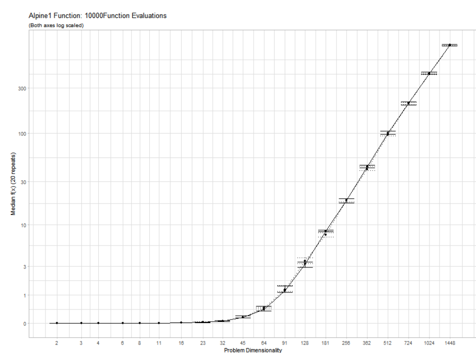
Sum Squares Function at 50,000 Evaluations



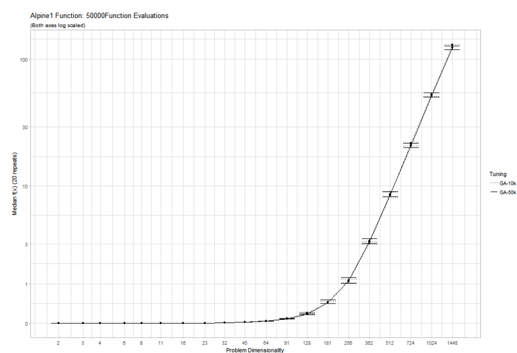
Ackley Function at 10,000 Evaluations



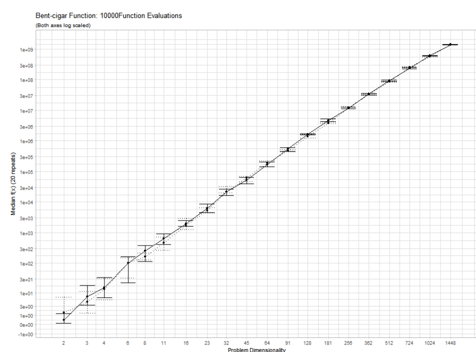
Ackley Function at 50,000 Evaluations



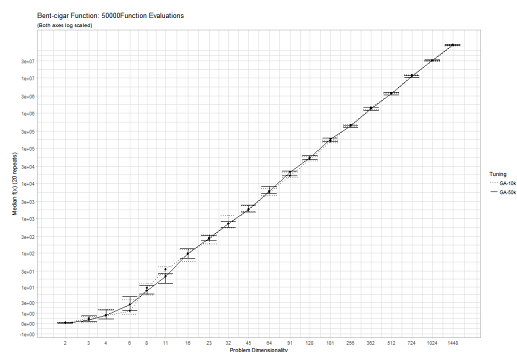
Alpine no.1 Function at 10,000 Evaluations



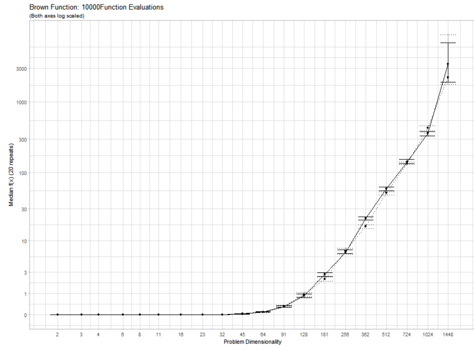
Alpine no.1 Function at 50,000 Evaluations



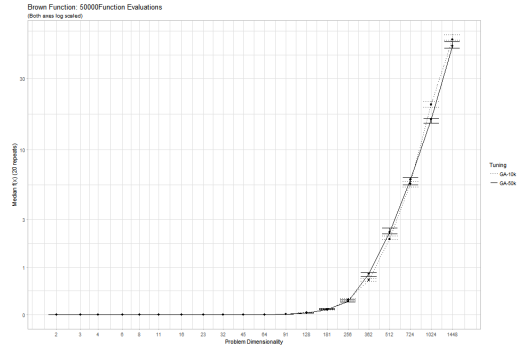
Bent Cigar Function at 10,000 Evaluations



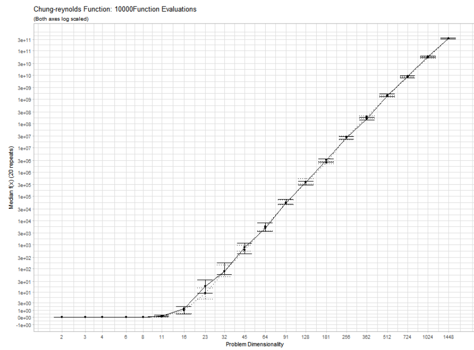
Bent Cigar Function at 50,000 Evaluations



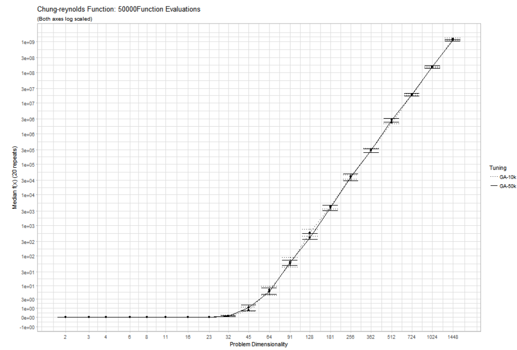
Brown Function at 10,000 Evaluations



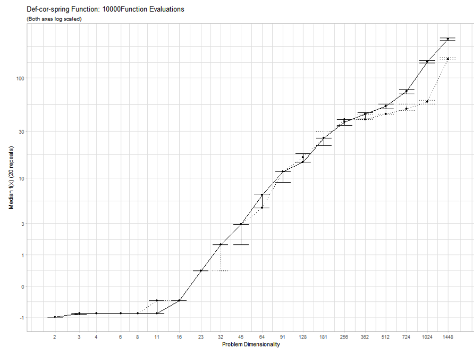
Brown Function at 50,000 Evaluations



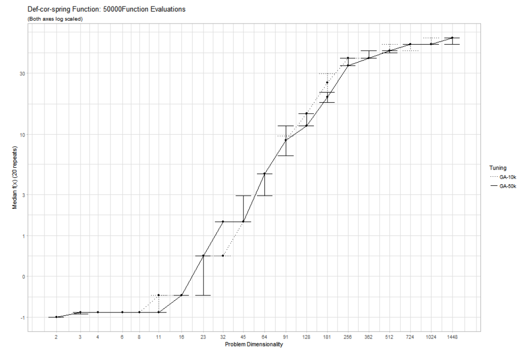
Chung-Reynolds Function at 10,000 Evaluations



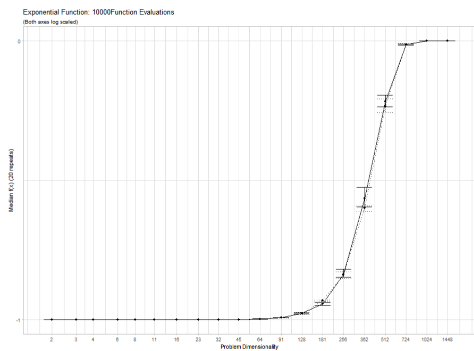
Chung-Reynolds Function at 50,000 Evaluations



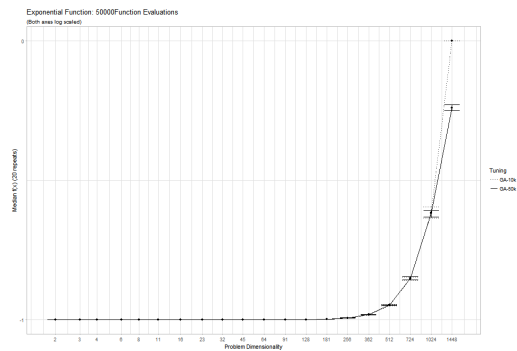
Deflected Corrugated Spring Function at 10,000 Evaluations



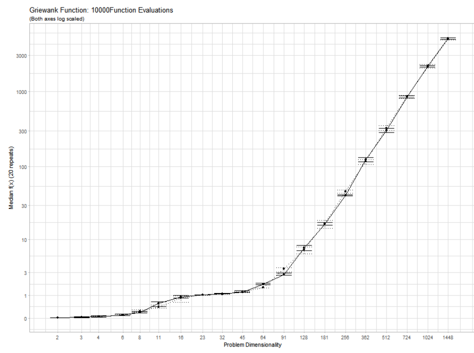
Deflected Corrugated Spring Function at 50,000 Evaluations



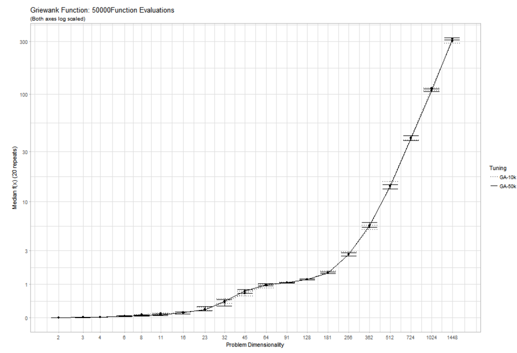
Exponential Function at 10,000 Evaluations



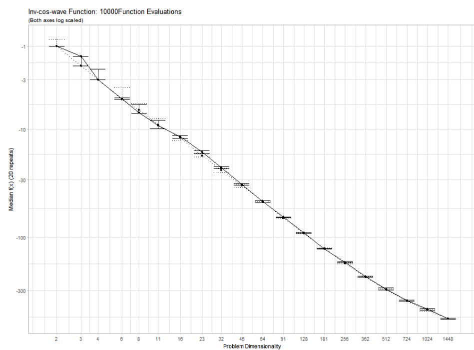
Exponential Function at 50,000 Evaluations



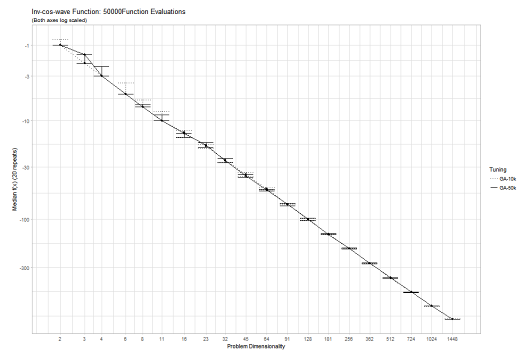
Griewank Function at 10,000 Evaluations



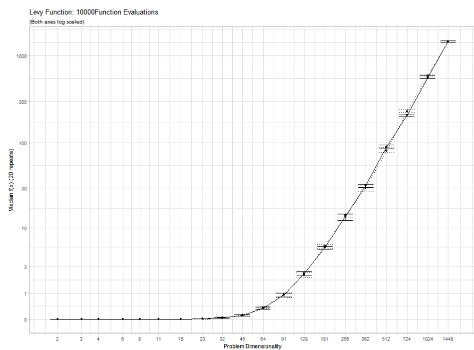
Griewank Function at 50,000 Evaluations



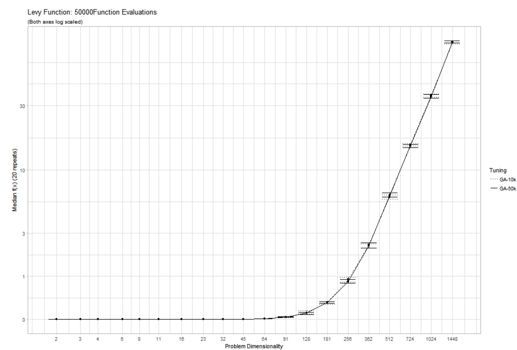
Inverted Cosine Wave Function at 10,000 Evaluations



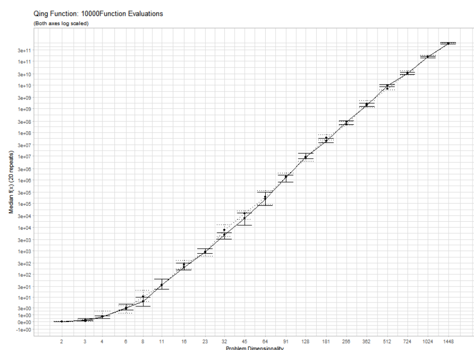
Inverted Cosine Wave Function at 50,000 Evaluations



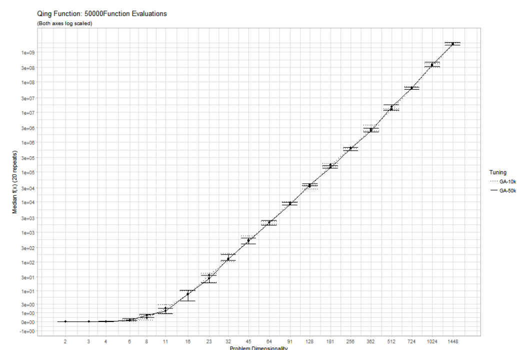
Levy Function at 10,000 Evaluations



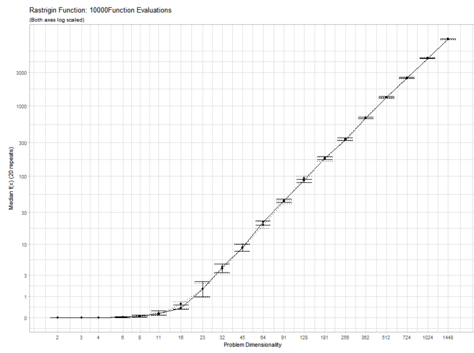
Levy Function at 50,000 Evaluations



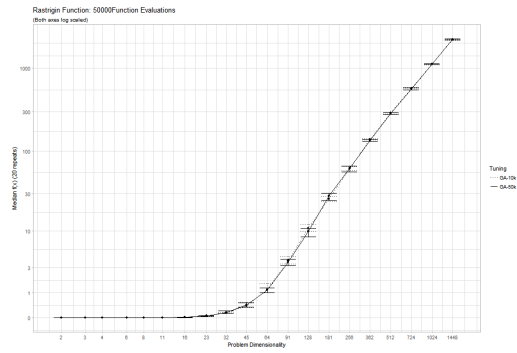
Qing Function at 10,000 Evaluations



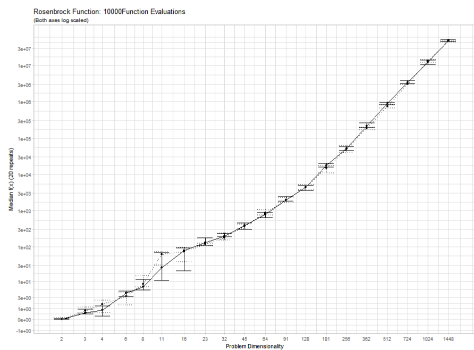
Qing Function at 50,000 Evaluations



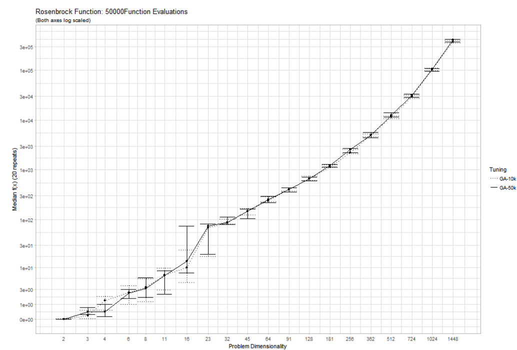
Rastrigin Function at 10,000 Evaluations



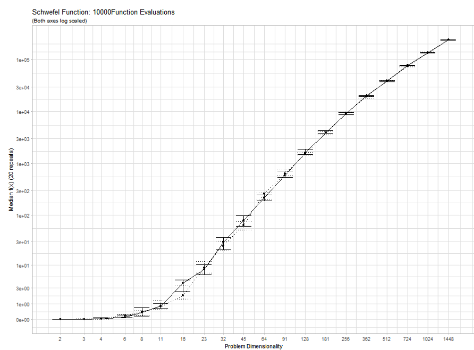
Rastrigin Function at 50,000 Evaluations



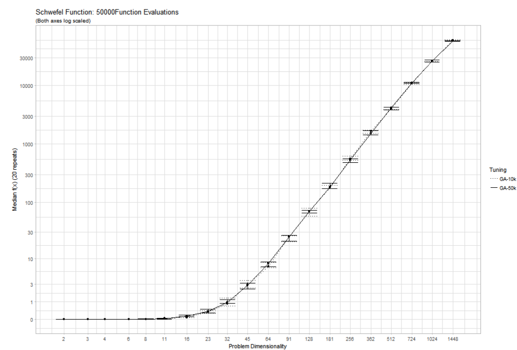
Rosenbrock Function at 10,000 Evaluations



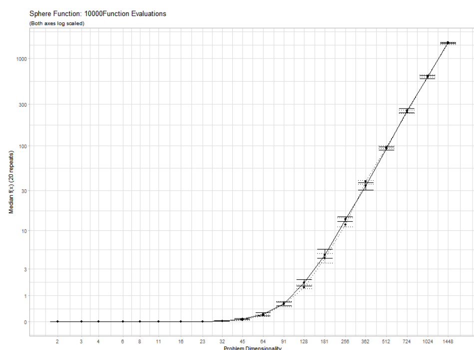
Rosenbrock Function at 50,000 Evaluations



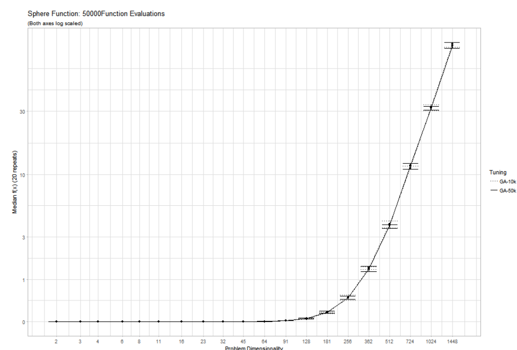
Schwefel Function at 10,000 Evaluations



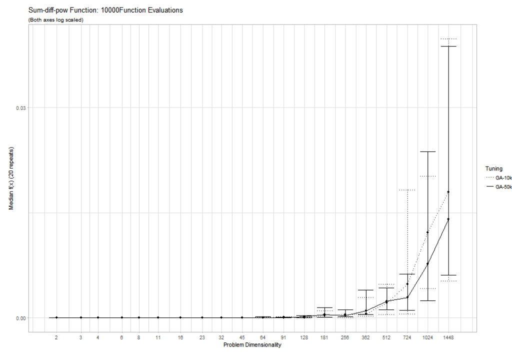
Schwefel Function at 50,000 Evaluations



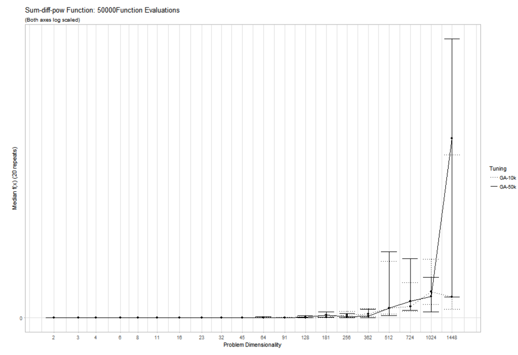
Sphere Function at 10,000 Evaluations



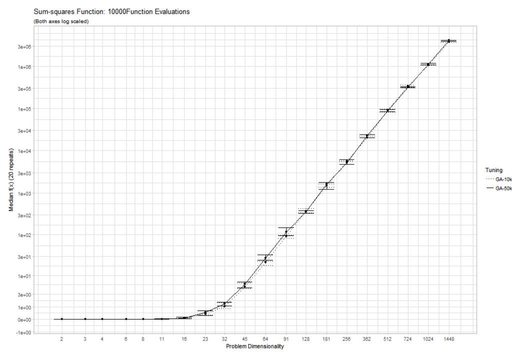
Sphere Function at 50,000 Evaluations



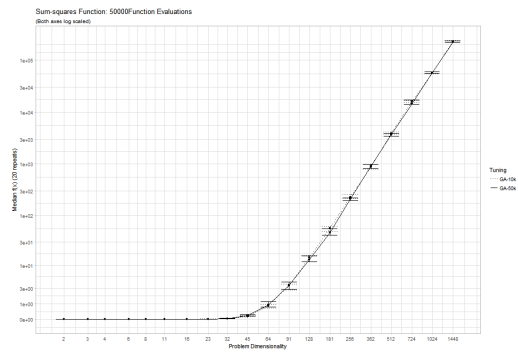
Sum of Different Powers Function at 10,000 Evaluations



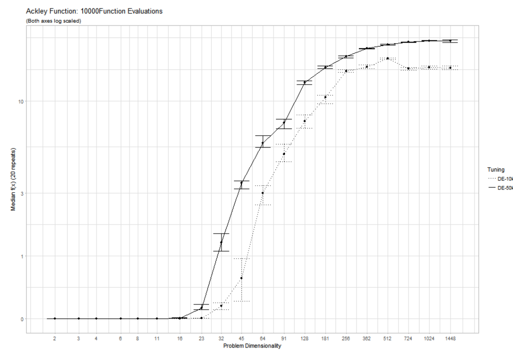
Sum of Different Powers Function at 50,000 Evaluations



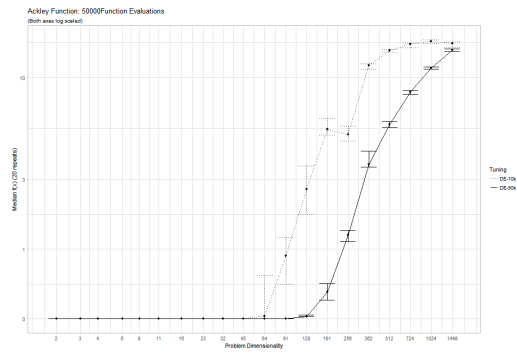
Sum Squares Function at 10,000 Evaluations



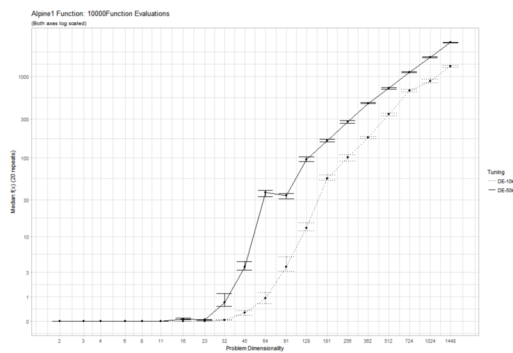
Sum Squares Function at 50,000 Evaluations



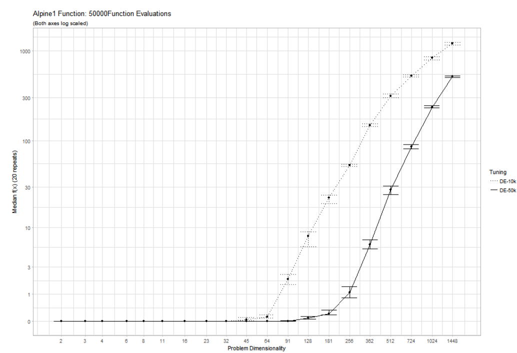
Ackley Function at 10,000 Evaluations



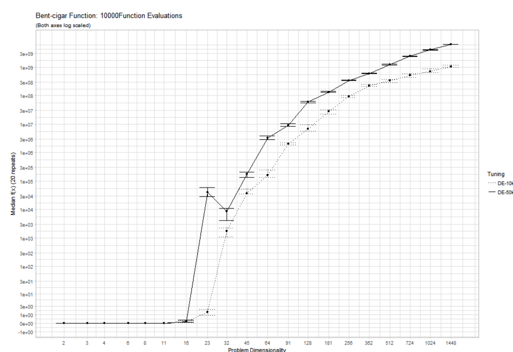
Ackley Function at 50,000 Evaluations



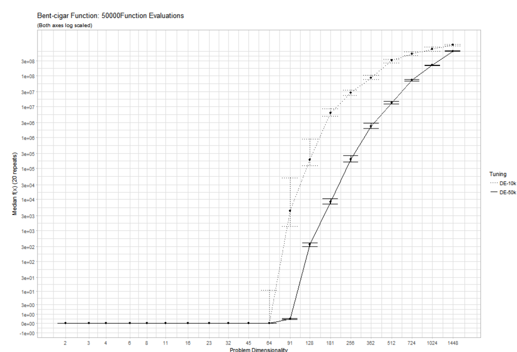
Alpine no.1 Function at 10,000 Evaluations



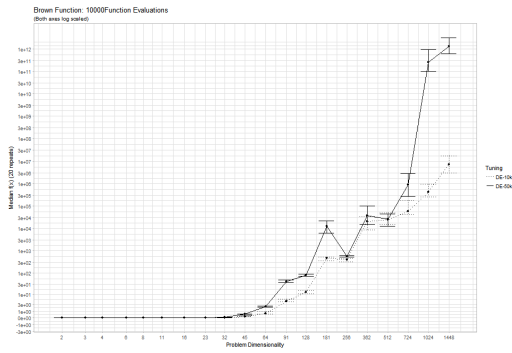
Alpine no.1 Function at 50,000 Evaluations



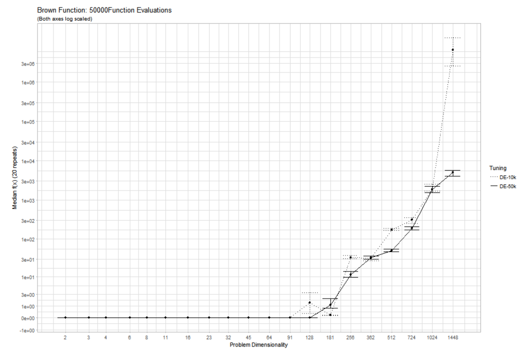
Bent Cigar Function at 10,000 Evaluations



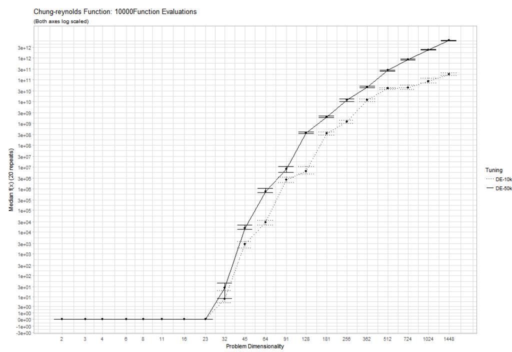
Bent Cigar Function at 50,000 Evaluations



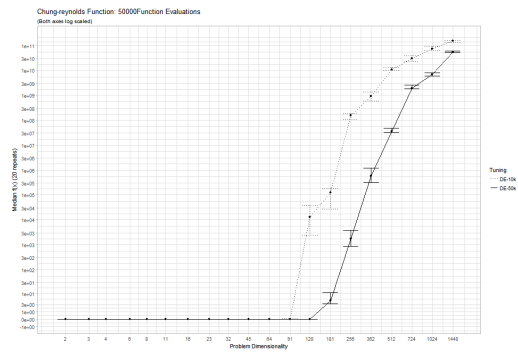
Brown Function at 10,000 Evaluations



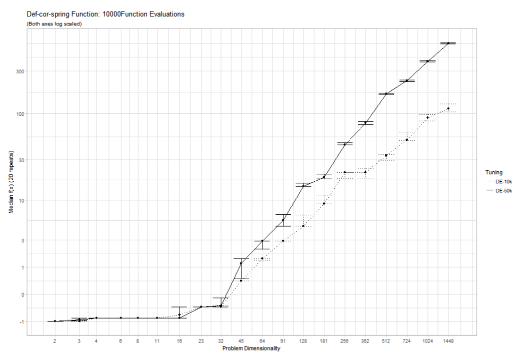
Brown Function at 50,000 Evaluations



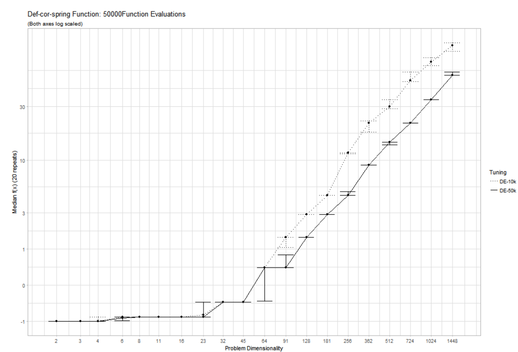
Chung-Reynolds Function at 10,000 Evaluations



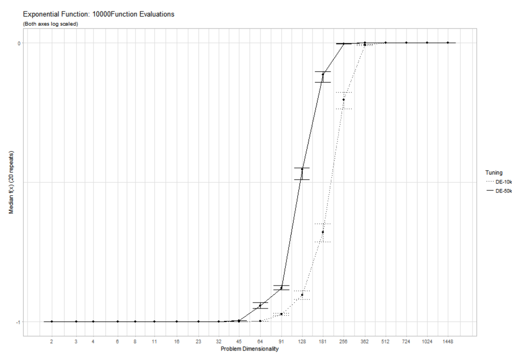
Chung-Reynolds Function at 50,000 Evaluations



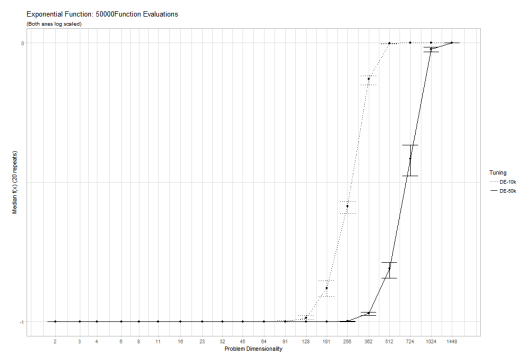
Deflected Corrugated Spring Function at 10,000 Evaluations



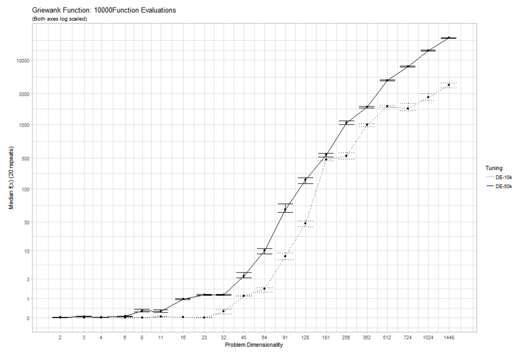
Deflected Corrugated Spring Function at 50,000 Evaluations



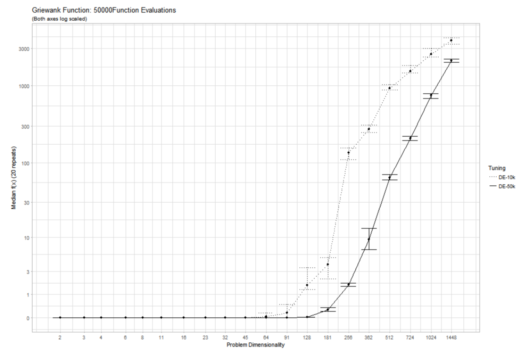
Exponential Function at 10,000 Evaluations



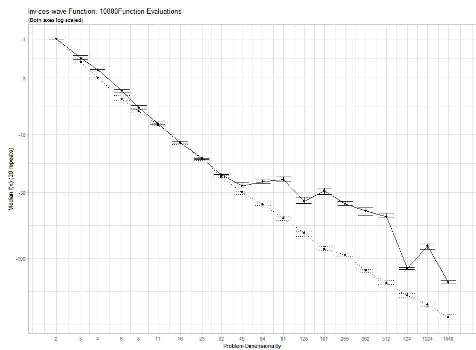
Exponential Function at 50,000 Evaluations



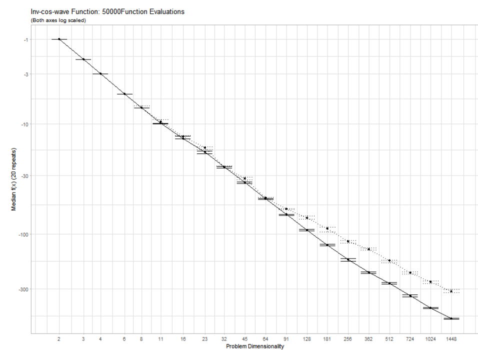
Griewank Function at 10,000 Evaluations



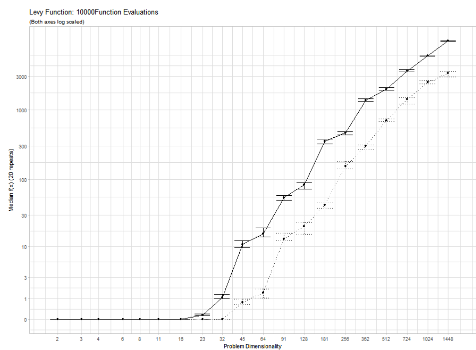
Griewank Function at 50,000 Evaluations



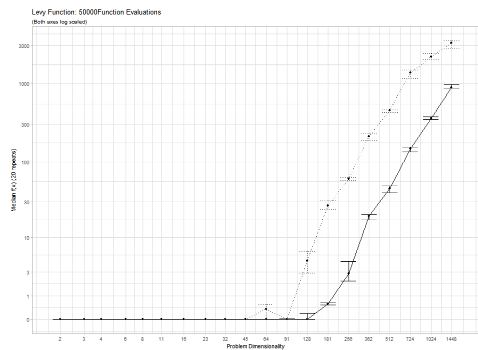
Inverted Cosine Wave Function at 10,000 Evaluations



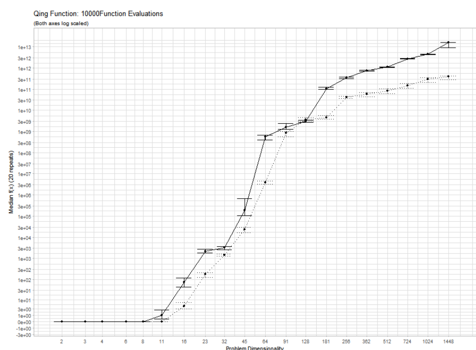
Inverted Cosine Wave Function at 50,000 Evaluations



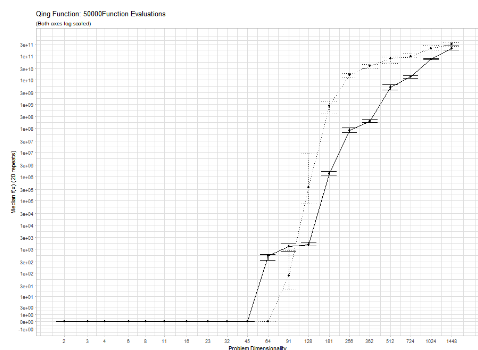
Levy Function at 10,000 Evaluations



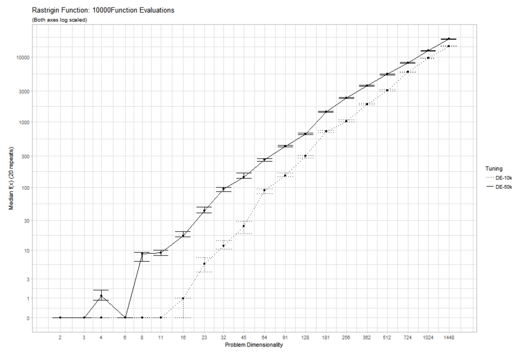
Levy Function at 50,000 Evaluations



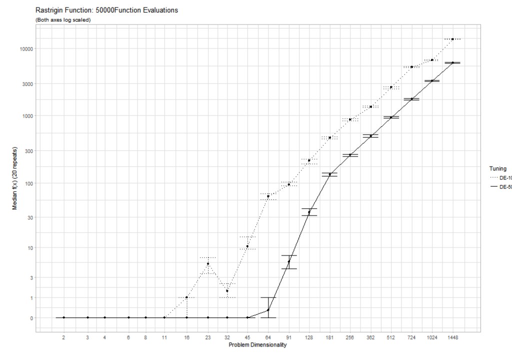
Qing Function at 10,000 Evaluations



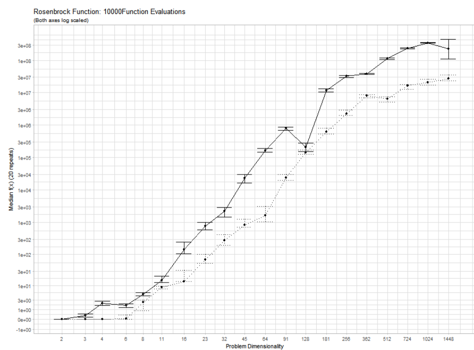
Qing Function at 50,000 Evaluations



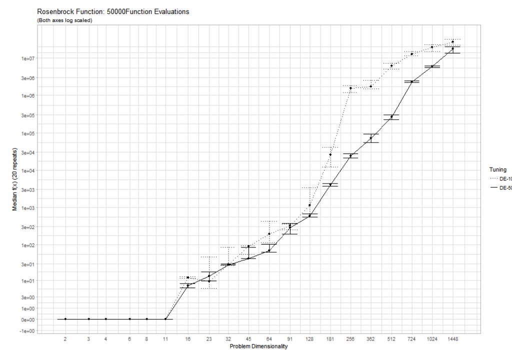
Rastrigin Function at 10,000 Evaluations



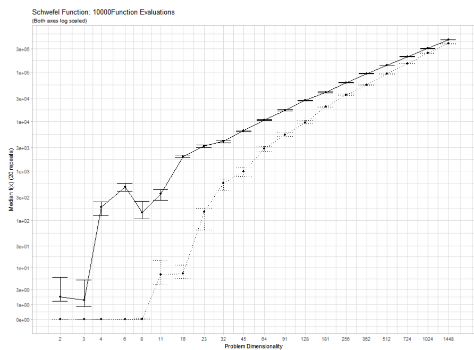
Rastrigin Function at 50,000 Evaluations



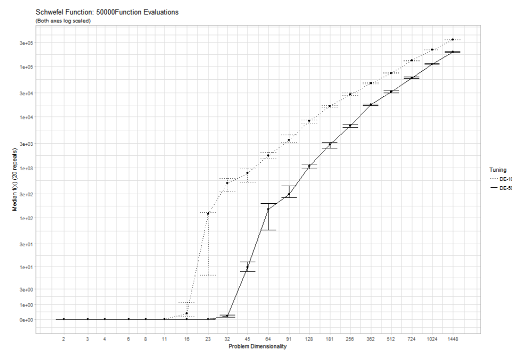
Rosenbrock Function at 10,000 Evaluations



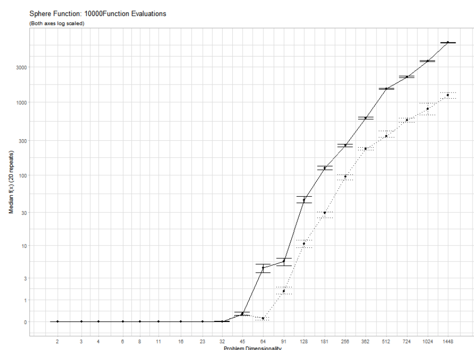
Rosenbrock Function at 50,000 Evaluations



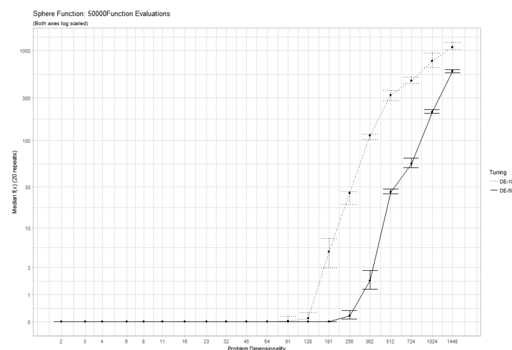
Schwefel Function at 10,000 Evaluations



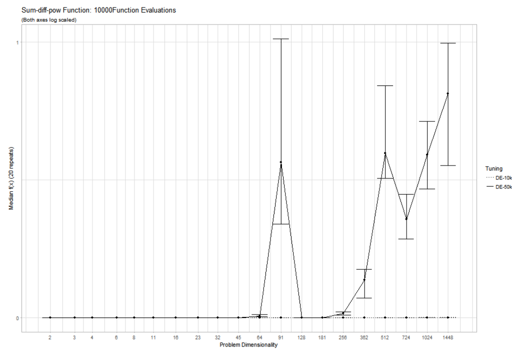
Schwefel Function at 50,000 Evaluations



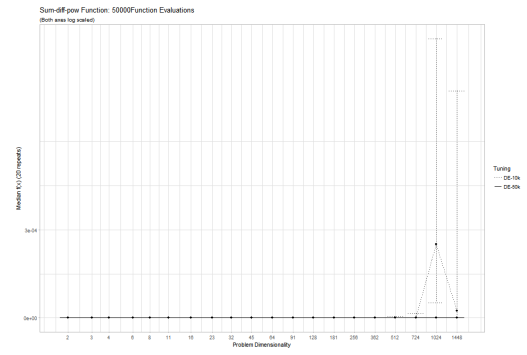
Sphere Function at 10,000 Evaluations



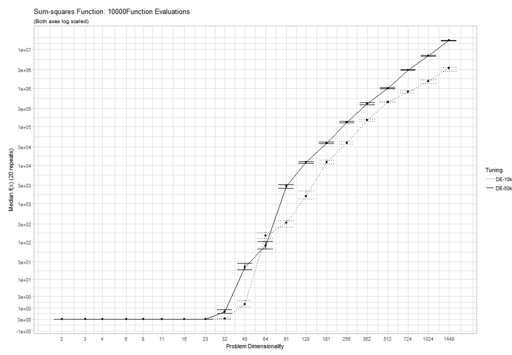
Sphere Function at 50,000 Evaluations



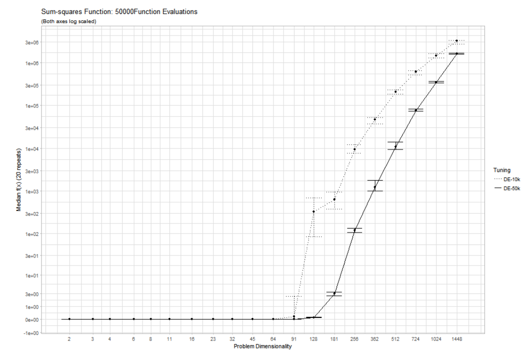
Sum of Different Powers Function at 10,000 Evaluations



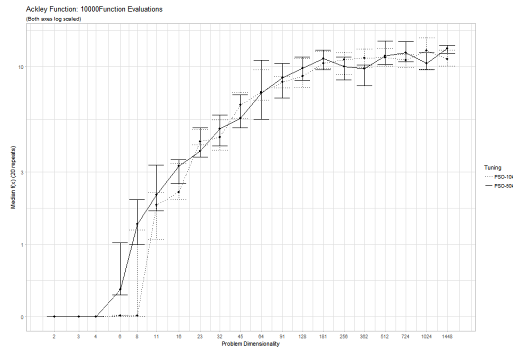
Sum of Different Powers Function at 50,000 Evaluations



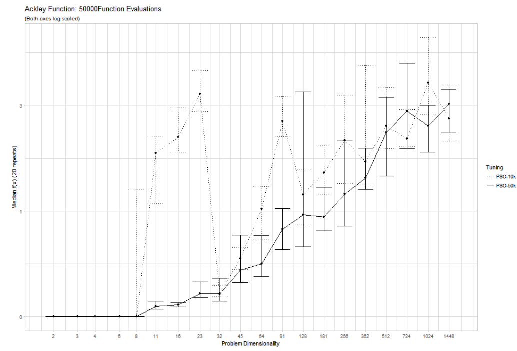
Sum Squares Function at 10,000 Evaluations



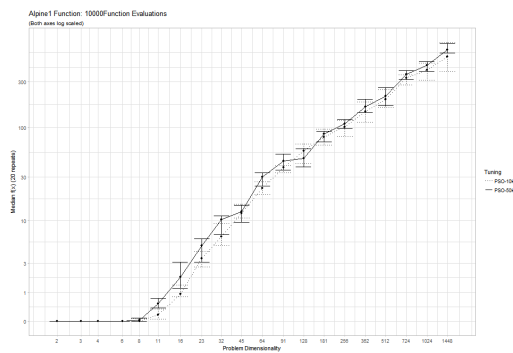
Sum Squares Function at 50,000 Evaluations



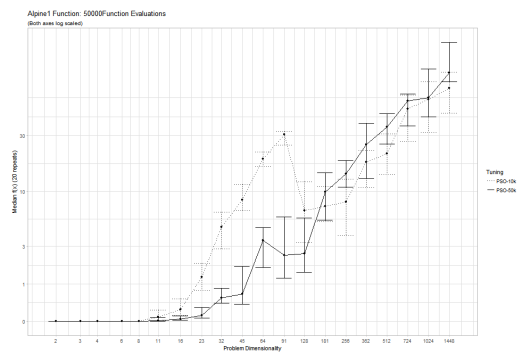
Ackley Function at 10,000 Evaluations



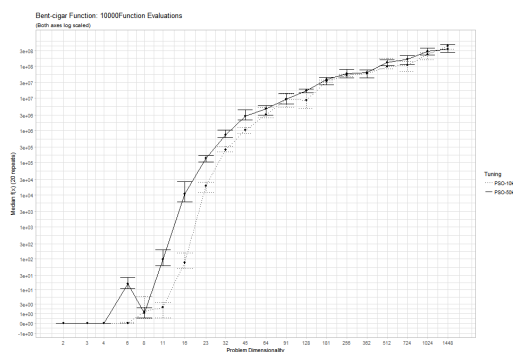
Ackley Function at 50,000 Evaluations



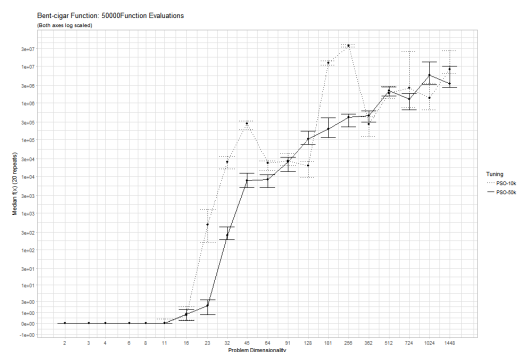
Alpine no.1 Function at 10,000 Evaluations



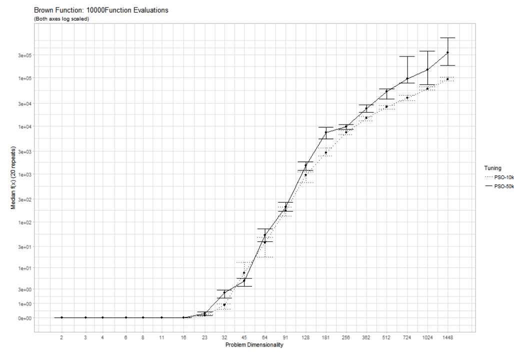
Alpine no.1 Function at 50,000 Evaluations



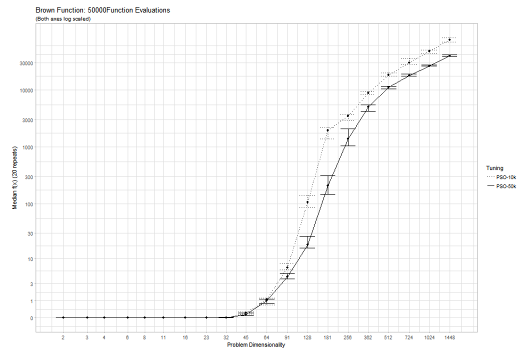
Bent Cigar Function at 10,000 Evaluations



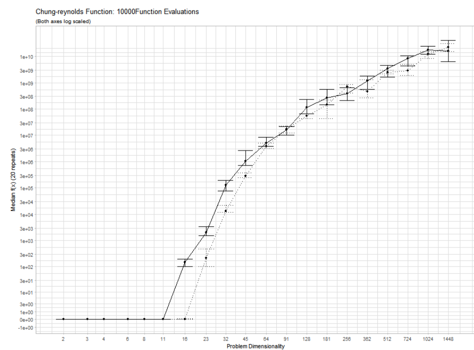
Bent Cigar Function at 50,000 Evaluations



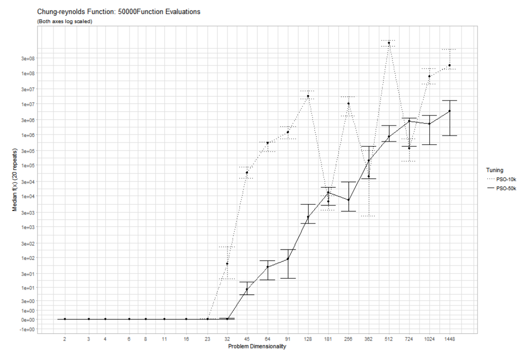
Brown Function at 10,000 Evaluations



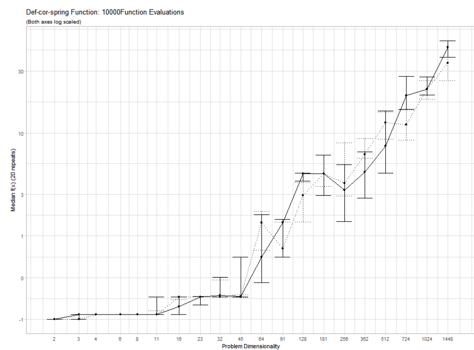
Brown Function at 50,000 Evaluations



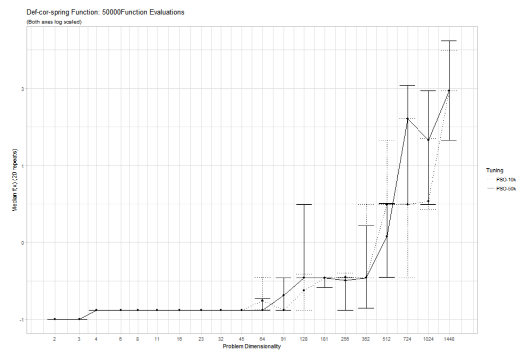
Chung-Reynolds Function at 10,000 Evaluations



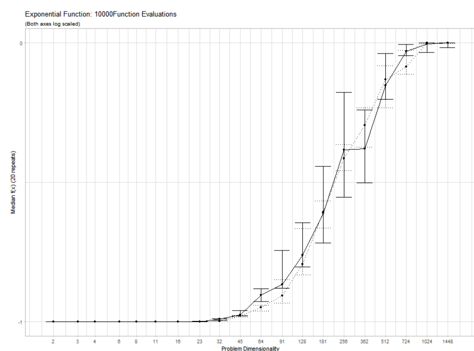
Chung-Reynolds Function at 50,000 Evaluations



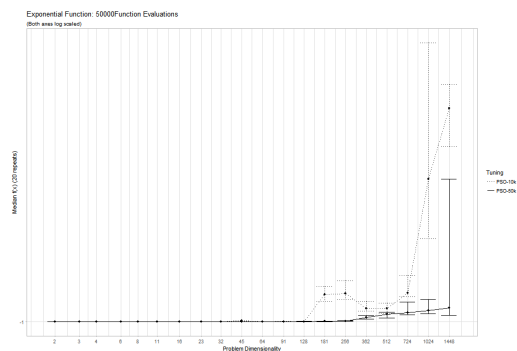
Deflected Corrugated Spring Function at 10,000 Evaluations



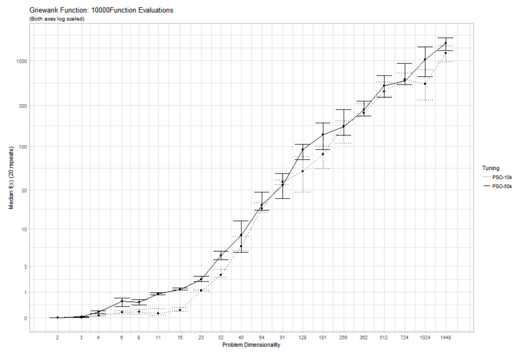
Deflected Corrugated Spring Function at 50,000 Evaluations



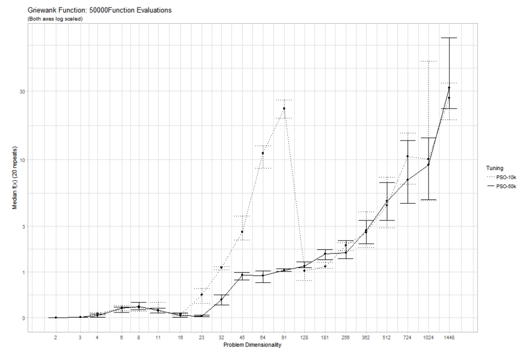
Exponential Function at 10,000 Evaluations



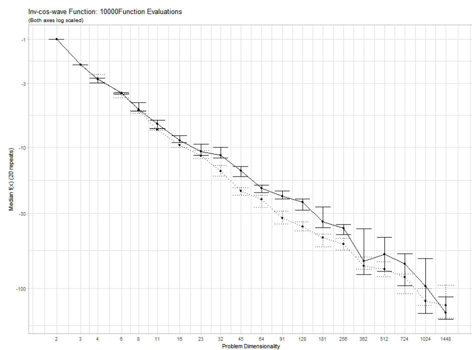
Exponential Function at 50,000 Evaluations



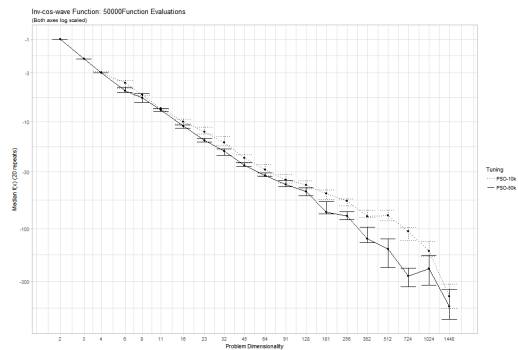
Griewank Function at 10,000 Evaluations



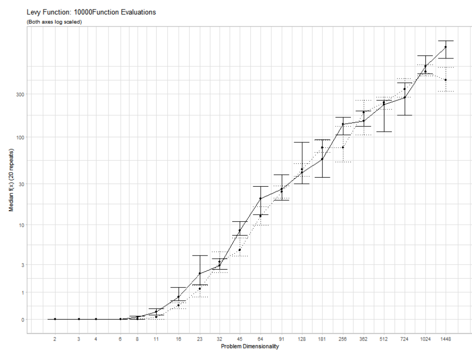
Griewank Function at 50,000 Evaluations



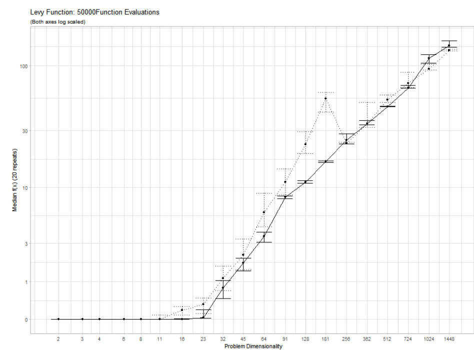
Inverted Cosine Wave Function at 10,000 Evaluations



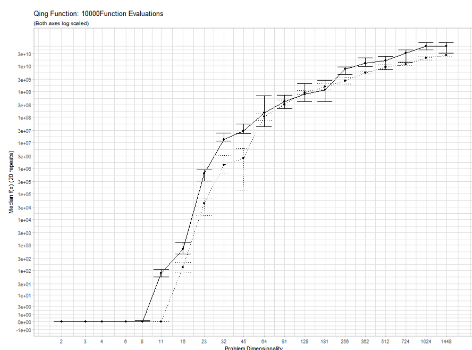
Inverted Cosine Wave Function at 50,000 Evaluations



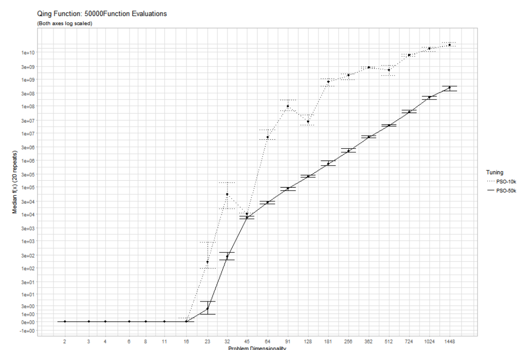
Levy Function at 10,000 Evaluations



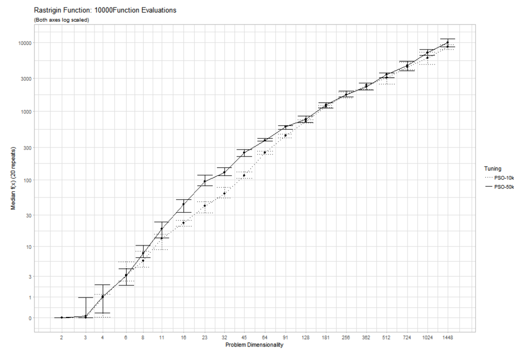
Levy Function at 50,000 Evaluations



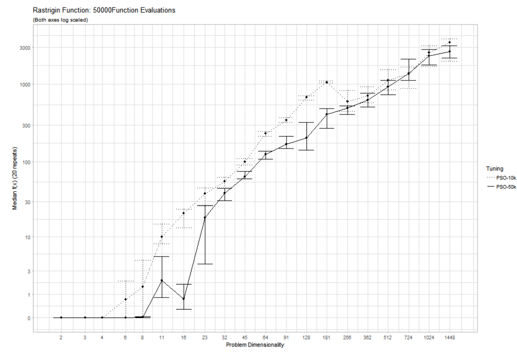
Qing Function at 10,000 Evaluations



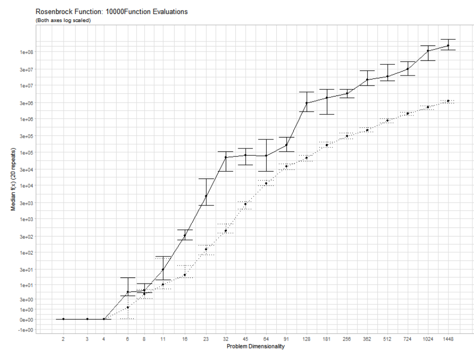
Qing Function at 50,000 Evaluations



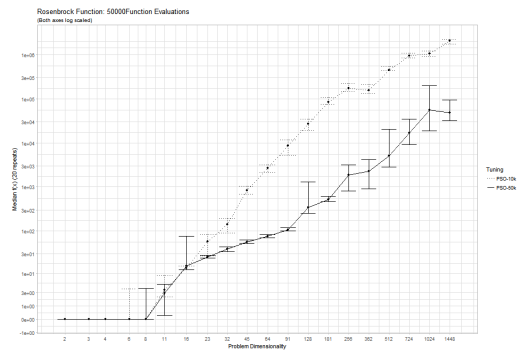
Rastrigin Function at 10,000 Evaluations



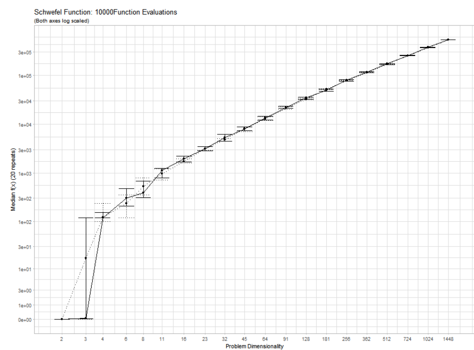
Rastrigin Function at 50,000 Evaluations



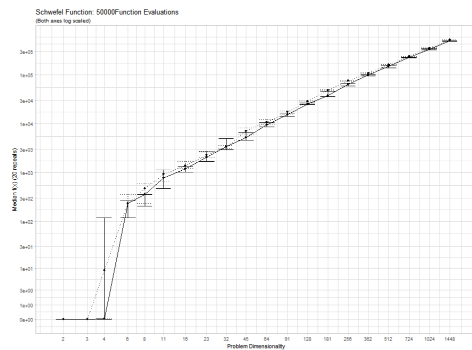
Rosenbrock Function at 10,000 Evaluations



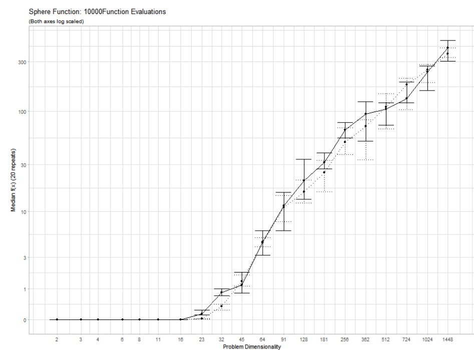
Rosenbrock Function at 50,000 Evaluations



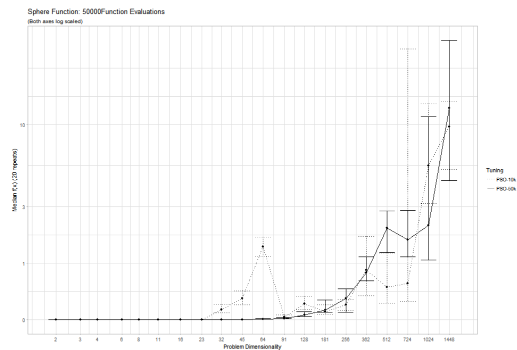
Schwefel Function at 10,000 Evaluations



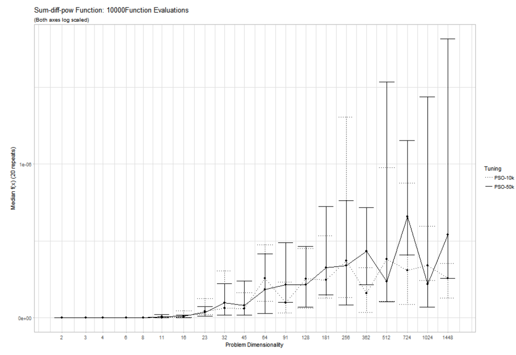
Schwefel Function at 50,000 Evaluations



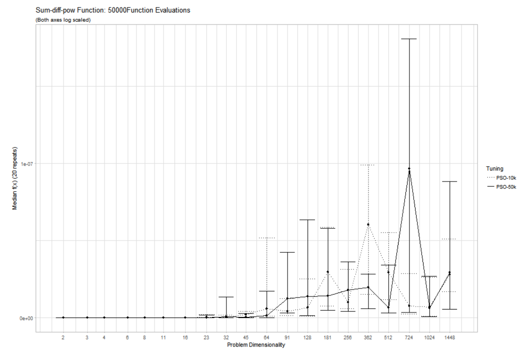
Sphere Function at 10,000 Evaluations



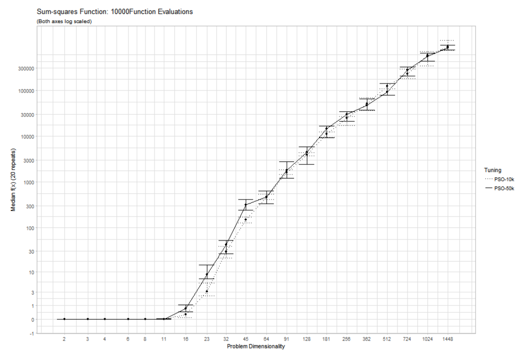
Sphere Function at 50,000 Evaluations



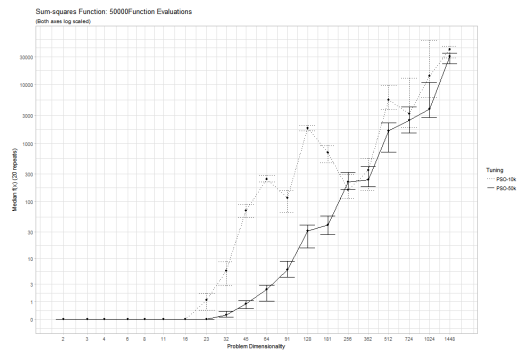
Sum of Different Powers Function at 10,000 Evaluations



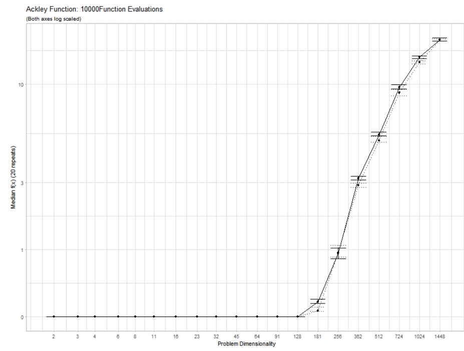
Sum of Different Powers Function at 50,000 Evaluations



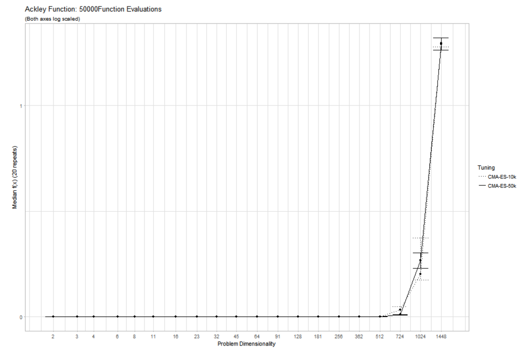
Sum Squares Function at 10,000 Evaluations



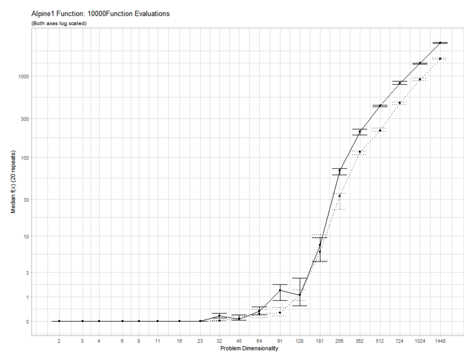
Sum Squares Function at 50,000 Evaluations



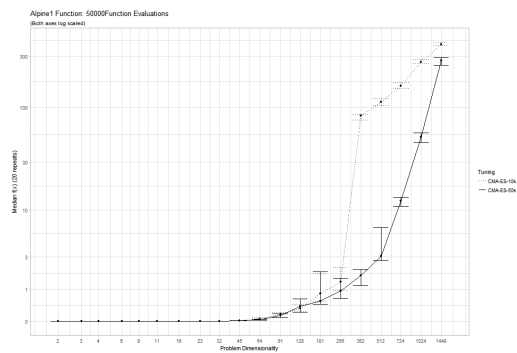
Ackley Function at 10,000 Evaluations



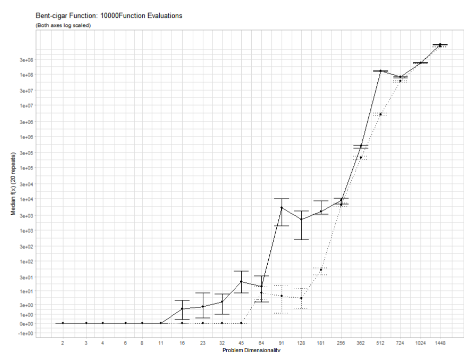
Ackley Function at 50,000 Evaluations



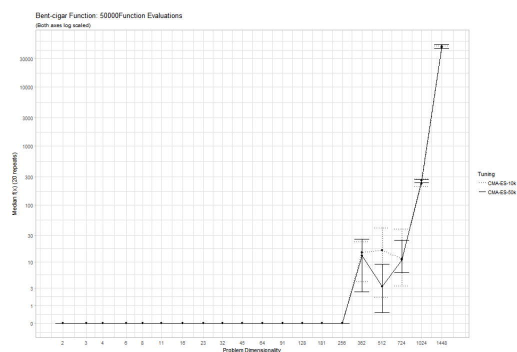
Alpine no.1 Function at 10,000 Evaluations



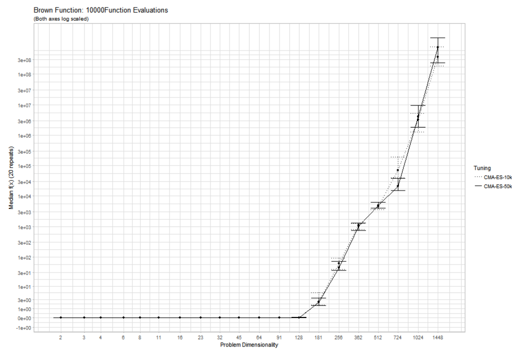
Alpine no.1 Function at 50,000 Evaluations



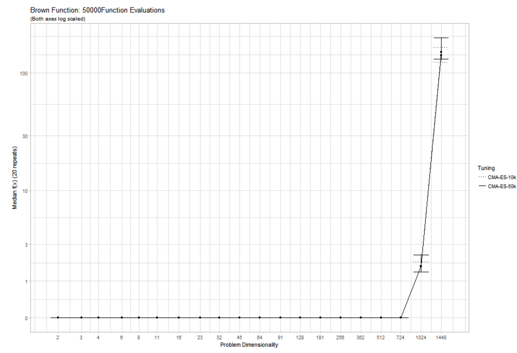
Bent Cigar Function at 10,000 Evaluations



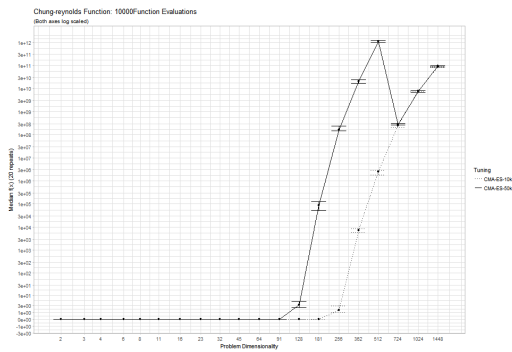
Bent Cigar Function at 50,000 Evaluations



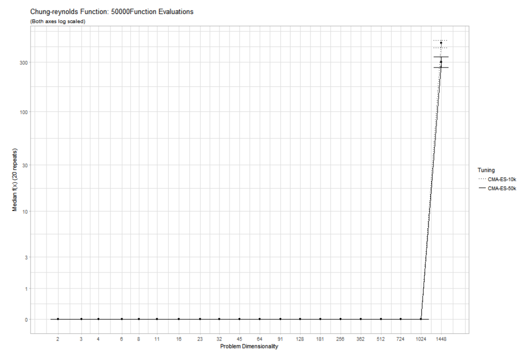
Brown Function at 10,000 Evaluations



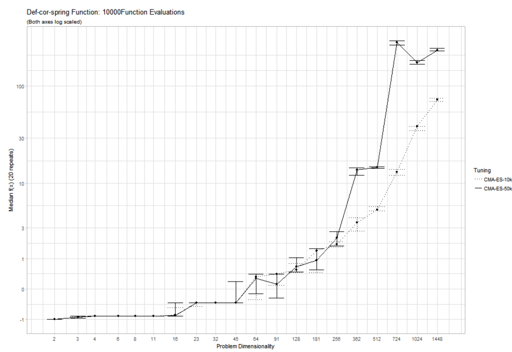
Brown Function at 50,000 Evaluations



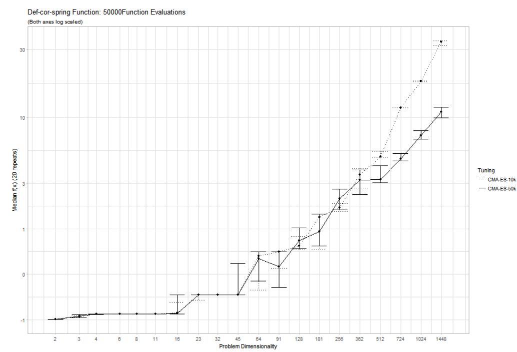
Chung-Reynolds Function at 10,000 Evaluations



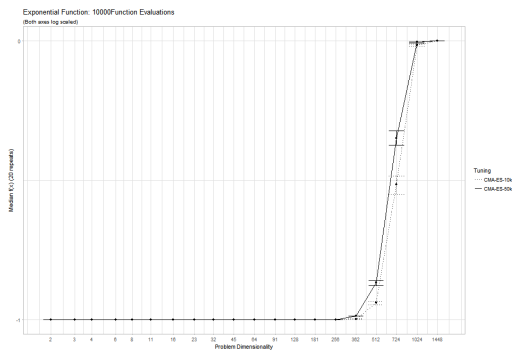
Chung-Reynolds Function at 50,000 Evaluations



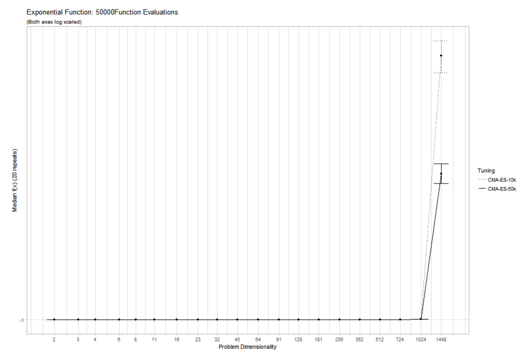
Deflected Corrugated Spring Function at 10,000 Evaluations



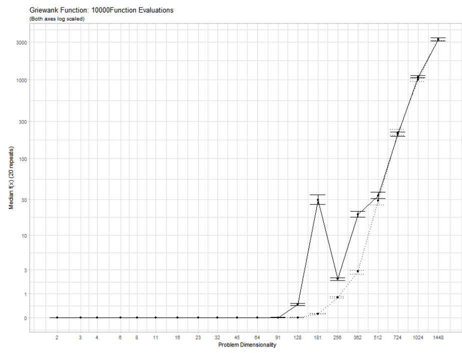
Deflected Corrugated Spring Function at 50,000 Evaluations



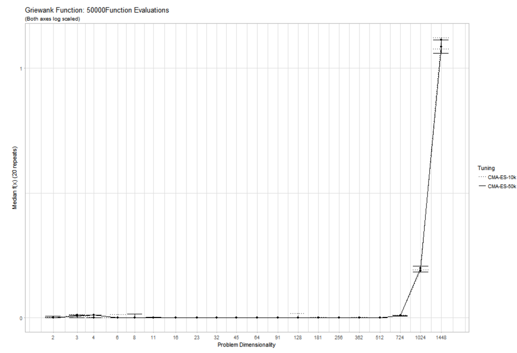
Exponential Function at 10,000 Evaluations



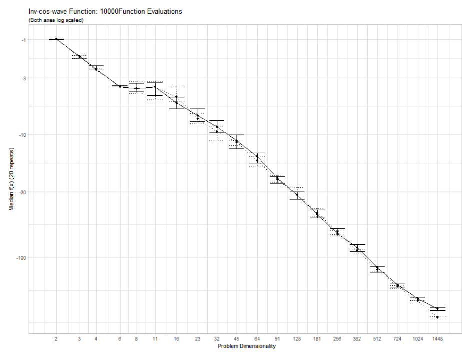
Exponential Function at 50,000 Evaluations



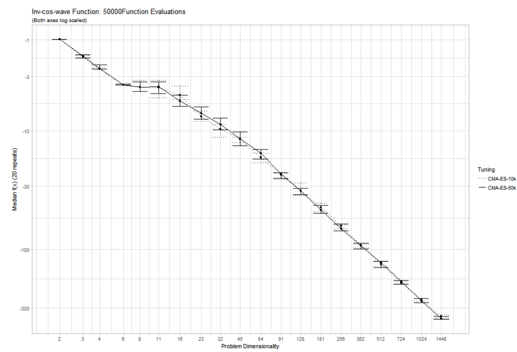
Griewank Function at 10,000 Evaluations



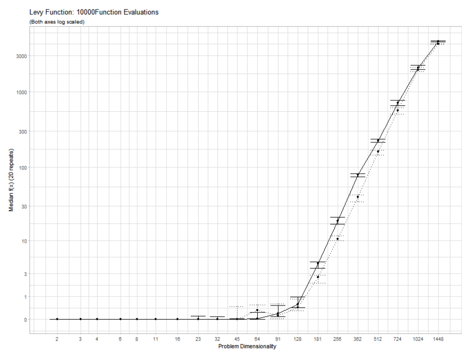
Griewank Function at 50,000 Evaluations



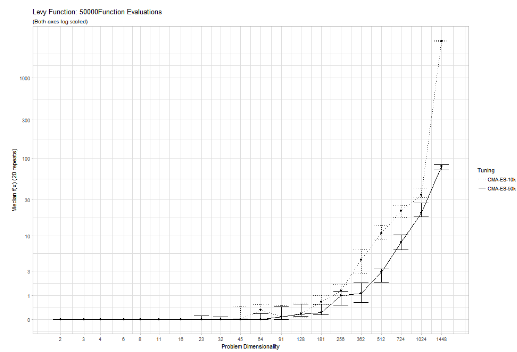
Inverted Cosine Wave Function at 10,000 Evaluations



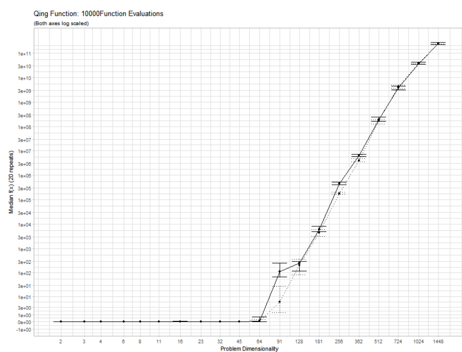
Inverted Cosine Wave Function at 50,000 Evaluations



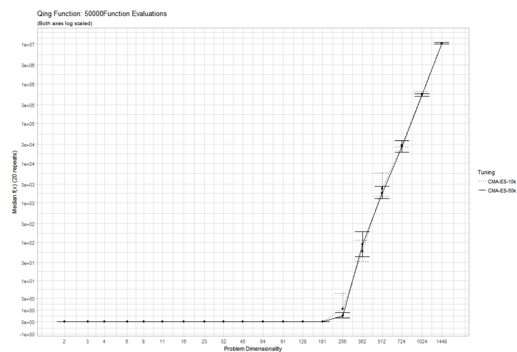
Levy Function at 10,000 Evaluations



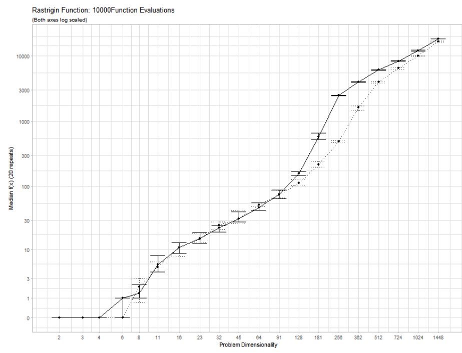
Levy Function at 50,000 Evaluations



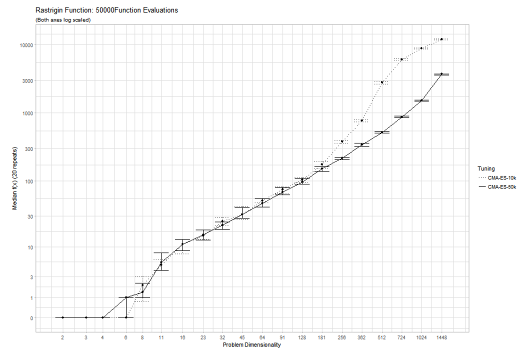
Qing Function at 10,000 Evaluations



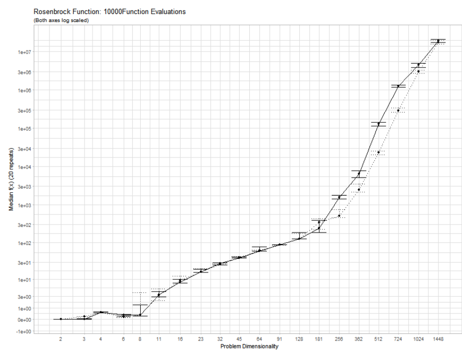
Qing Function at 50,000 Evaluations



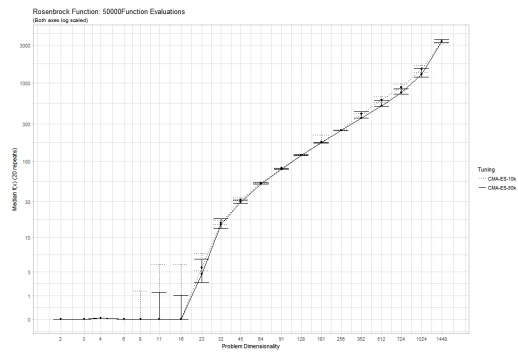
Rastrigin Function at 10,000 Evaluations



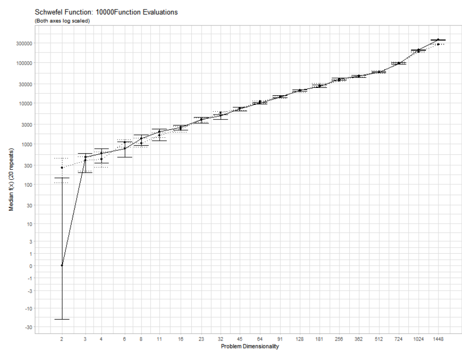
Rastrigin Function at 50,000 Evaluations



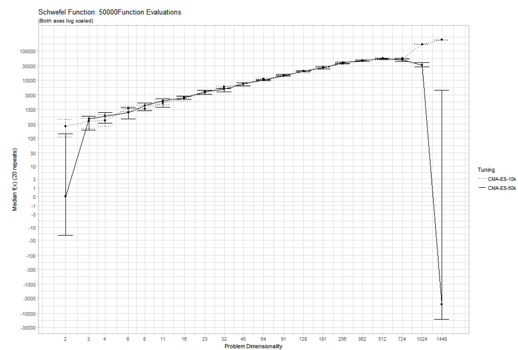
Rosenbrock Function at 10,000 Evaluations



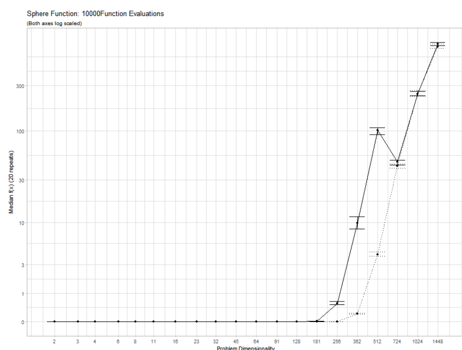
Rosenbrock Function at 50,000 Evaluations



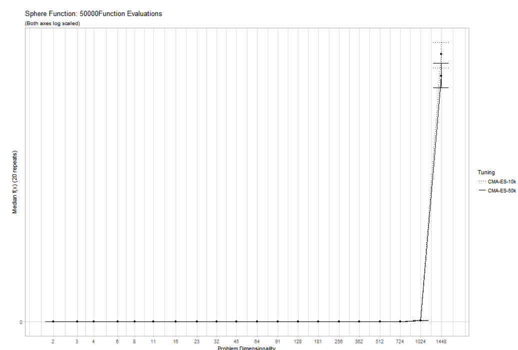
Schwefel Function at 10,000 Evaluations



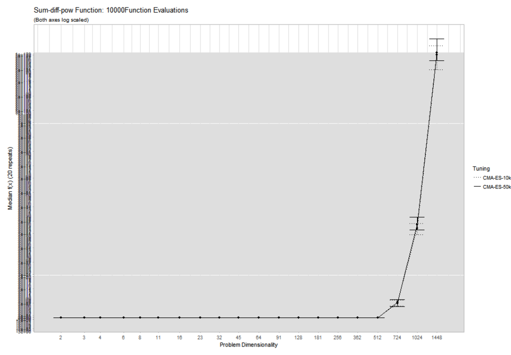
Schwefel Function at 50,000 Evaluations



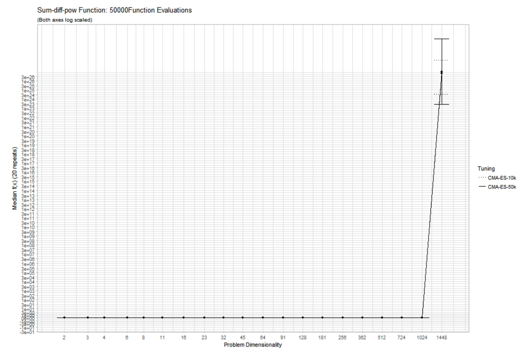
Sphere Function at 10,000 Evaluations



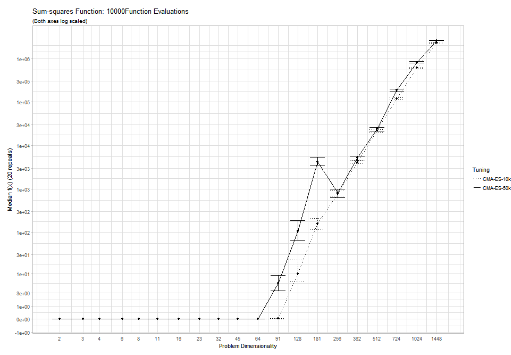
Sphere Function at 50,000 Evaluations



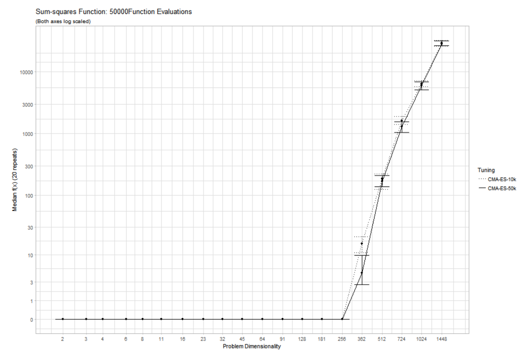
Sum of Different Powers Function at 10,000 Evaluations



Sum of Different Powers Function at 50,000 Evaluations



Sum Squares Function at 10,000 Evaluations



Sum Squares Function at 50,000 Evaluations

B

APPENDIX B: 10K & 50K EVALUATION TUNING BUDGET DESCRIPTIVE STATISTICS

Table B.1: sep-CMA-ES Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (10,000 Evaluation Tuning)

Dims.	Ackley						Alpine1						Bent Cigar					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	3E-15	2.61E-15	4.25E-15	3E-15	2.61E-15	4.11E-15	1.8E-143	1.3E-144	3.5E-141	0	0	0
3	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.92E-15	2.94E-15	1.1E-14	4E-15	2.54E-15	7.77E-15	7.7E-111	6.3E-112	7.4E-110	0	0	0
4	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	2.36E-14	4.45E-15	3.96E-14	1.42E-14	4.41E-15	2.8E-14	7.4E-87	1.05E-89	1.48E-84	0	0	0
6	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	1.05E-13	7.62E-14	1.54E-13	8.09E-14	1.84E-14	1.04E-13	2.86E-56	1.2E-58	7.2E-54	0	0	0
8	3.83E-10	1.95E-10	7.52E-10	4.44E-16	4.44E-16	4.44E-16	6.91E-19	1.66E-43	2.56E-17	3.17E-19	7.9E-243	1.92E-18	2.69E-34	3.15E-35	8.5E-33	6.9E-236	8.8E-239	8.3E-233
11	4E-15	4E-15	4E-15	4E-15	4E-15	4E-15	7.12E-21	1.01E-36	3.21E-15	3.6E-186	4.3E-241	9.5E-16	1.54E-13	1.69E-16	2.16E-11	3.1E-139	3.1E-143	8.4E-134
16	4E-15	4E-15	4E-15	4E-15	4E-15	4E-15	0.002358	1.26E-16	0.002505	2.15E-14	1.26E-16	0.002455	2.6E-38	8.63E-39	8.89E-38	2.5E-298	2.1E-301	1.6E-296
23	4E-15	4E-15	4E-15	4E-15	4E-15	4E-15	0.009005	0.006009	0.01249	0.004823	1.39E-05	0.006993	2.66E-26	8.27E-27	2.7E-25	2.8E-234	1.6E-235	2.6E-233
32	1.66E-10	1.08E-10	3.34E-10	4E-15	4E-15	4E-15	0.019163	0.012799	0.030297	0.007224	0.00479	0.009892	2.14E-12	1.64E-13	5.99E-12	1E-167	1.2E-168	1.4E-166
45	1.97E-08	1.42E-08	2.53E-08	4E-15	4E-15	4E-15	0.088877	0.043438	0.160211	0.01898	0.014865	0.022698	0.001034	0.000417	0.004096	2.4E-119	3E-120	1.1E-118
64	9.86E-06	7.53E-06	1.23E-05	4E-15	4E-15	7.55E-15	0.244061	0.10867	0.40066	0.053152	0.03668	0.086051	8.429856	4.945822	14.34163	2.6E-71	8.34E-72	6.57E-71
91	6.91E-05	5.78E-05	0.000102	1.11E-14	1.11E-14	1.47E-14	0.275676	0.170037	0.476805	0.171791	0.093125	0.212094	6.675954	1.159461	15.26123	4.86E-48	1.5E-48	1.22E-47
128	0.002647	0.001772	0.003221	1.47E-14	1.47E-14	2.18E-14	1.083879	0.801345	1.446541	0.321983	0.255336	0.447892	5.36327	2.034042	12.13223	1.5E-27	2.93E-28	2.56E-27
181	0.065409	0.056016	0.103422	2.53E-14	2.53E-14	2.89E-14	6.01139	4.603858	10.47752	0.82361	0.541352	1.78729	51.34948	35.07397	59.37319	2.26E-14	1.4E-14	4.54E-14
256	0.930774	0.853104	1.091811	2.97E-11	1.73E-11	4.74E-11	32.91603	22.25738	35.61941	1.333655	0.955537	2.210262	6263.074	5599.162	7340.425	0.004399	0.002488	0.00892
362	2.895003	2.795725	2.965213	1.62E-07	9.06E-08	2.25E-07	115.8144	108.1024	120.0842	82.73823	75.41908	85.40834	204725.6	181488.9	235386	14.79164	4.059236	22.83645
512	5.171349	5.066023	5.433745	0.000217	0.000135	0.000415	213.1185	209.8418	228.957	111.6917	103.465	120.1578	5032897	4647911	5606099	16.39641	1.774675	40.02427
724	9.087455	8.798223	9.456492	0.023295	0.008338	0.034043	465.8305	441.7909	475.4442	157.7768	149.0708	171.1556	58767522	55305447	62463622	11.35708	3.268771	38.69308
1024	12.84235	12.60942	13.06152	0.150523	0.128479	0.296142	900.2698	872.614	931.2216	266.7016	254.8341	280.4025	2.26E+08	2.16E+08	2.35E+08	228.3853	206.073	274.7108
1448	16.344	16.17579	16.54643	1.453542	1.422414	1.495074	1606.186	1575.878	1650.977	388.1734	376.5796	407.0378	7.44E+08	7.15E+08	7.59E+08	46686.33	44716.3	49206.33

Dims.	Brown						Chung-Reynolds						Deflected Corrugated Spring					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	2.1E-147	1.1E-149	4.6E-146	0	0	0	4E-287	1E-289	1.5E-282	0	0	0	-0.99529	-0.99742	-0.99109	-0.99529	-0.99742	-0.99109
3	3.5E-118	2.8E-120	5.2E-117	0	0	0	2.1E-224	7.8E-229	7.1E-220	0	0	0	-0.90525	-0.94646	-0.87805	-0.90525	-0.94646	-0.87805
4	4.12E-98	6.7E-100	2.2E-96	0	0	0	1.4E-180	6.2E-183	7.5E-176	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
6	9.32E-68	1.17E-70	2.55E-67	0	0	0	1E-122	1.8E-124	6.3E-116	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
8	3.11E-48	3.01E-50	8.03E-46	3.5E-250	6.5E-252	1.8E-247	1.61E-82	8.16E-87	1.96E-78	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
11	1.9E-31	1.54E-32	5.07E-30	1.1E-158	7.3E-163	8E-157	2.81E-56	1.41E-58	2.13E-53	4.68763768434652e-316	1.00394139234941e-320	2.49E-308	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
16	6.21E-20	1.9E-21	9.49E-19	5.32E-85	2.21E-89	1.88E-80	2.38E-32	9.8E-34	8.33E-31	6.9E-163	2.2E-170	1.2E-156	-0.81092	-0.84334	-0.53073	-0.81092	-0.84334	-0.53073
23	1.09E-13	2.59E-14	3E-12	1.03E-42	1.22E-44	2.11E-37	2.52E-25	7.92E-27	1.54E-22	5.1E-88	3.94E-98	2.66E-76	-0.37336	-0.48149	-0.37336	-0.37336	-0.48149	-0.37336
32	2.78E-12	6.82E-13	1.22E-10	4.5E-32	5.7E-34	2.89E-28	2.75E-20	1.13E-21	1.19E-19	2.3E-64	1.56E-71	7.77E-60	-0.37336	-0.37336	-0.37336	-0.37336	-0.37336	-0.37336
45	1.42E-09	1.83E-10	7.06E-09	4.05E-28	2.4E-30	6.58E-27	4.9E-17	9.25E-18	3.57E-16	3.12E-57	1.33E-60	2.48E-54	-0.37336	-0.37336	-0.3658	-0.37336	-0.37336	-0.3658
64	1.97E-07	8.12E-08	8.31E-07	8.87E-27	1.64E-27	1.06E-23	1.28E-11	7.87E-12	4.02E-11	8.28E-58	1.86E-60	5.86E-55	0.317491	-0.26942	0.409922	0.317491	-0.26942	0.409922
91	4.72E-08	2.95E-08	6.85E-08	2.77E-58	1.32E-58	6.91E-58	1.06E-18	5.98E-19	2.69E-18	4.2E-132	5.6E-133	1.7E-131	0.409922	0.090441	0.409922	0.409922	0.090441	0.409922
128	0.010175	0.007471	0.023735	1.74E-20	2.57E-21	3.85E-20	1.99E-11	9.11E-12	8.44E-11	4.4E-99	7.7E-100	1.67E-98	0.537561	0.449865	0.785403	0.537561	0.449865	0.785403
181	2.327604	1.667888	5.831744	9.59E-18	2.46E-18	4.18E-17	4.26E-05	2.34E-05	5.14E-05	7.7E-69	4.13E-69	1.49E-68	1.390559	0.448503	1.508681	1.390559	0.448503	1.508662
256	60.46724	38.32326	89.06033	6.84E-15	1.48E-15	9.17E-15	1.553352	1.073272	2.762488	2.31E-46	7.52E-47	6.09E-46	1.757313	1.611967	1.936745	1.757313	1.611967	1.936745
362	1116.147	795.2885	1239.804	1.35E-10	7.97E-11	2.36E-10	7218.986	5915.217	8545.234	7.66E-30	4.58E-30	2.19E-29	3.54952	2.731181	4.014638	3.54952	2.731181	4.014638
512	4569.117	3545.07	6324.245	1.66E-06	1.27E-06	1.97E-06	2445018	1827305	2869730	6.07E-18	4.4E-18	8.55E-18	5.015042	4.902034	5.511616	5.015042	4.902034	5.511616
724	70398.99	39730.01	187918.8	0.003898	0.00333	0.005497	2.56E+08	2.05E+08	3.04E+08	4.58E-09	3.29E-09	5.37E-09	13.30563	12.18632	14.20861	11.69826	11.69226	11.74532
1024	3234868	1265966	5214117	1.618264	1.381569	1.870411	7.73E+09	7.37E+09	8.23E+09	0.008386	0.006613	0.009742	39.03873	35.30874	39.21776	18.06107	17.95422	18.37786
1448	3.71E+08	1.86E+08	7.92E+08	139.5145	122.8335	162.2979	8.62E+10	8E+10	9.05E+10	463.2814	412.3538	490.4856	72.48447	69.77969	75.08992	33.72941	31.80633	34.382

Table B.2: sep-CMA-ES Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (50,000 Evaluation Tuning)

Dims.	Ackley						Alpine1						Bent Cigar					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	1.19E-15	6.11E-16	2.22E-15	1.19E-15	6.11E-16	2.22E-15	1.4E-142	2.3E-143	1.1E-140	0	0	0
3	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.55E-15	2.89E-15	1.19E-14	4.3E-15	2.89E-15	9.64E-15	1.1E-109	1.4E-111	2.2E-108	0	0	0
4	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	1.54E-14	2.79E-15	3.34E-14	1.23E-14	2.79E-15	2.84E-14	1.74E-87	2.2E-89	6.98E-86	0	0	0
6	1.27E-13	8.22E-14	2.39E-13	4.44E-16	4.44E-16	4.44E-16	1.27E-13	7.05E-14	1.69E-13	1.18E-13	6.09E-14	1.47E-13	2.14E-54	3.52E-56	2.22E-52	0	0	2.994329312529e-318
8	2.94E-10	1.39E-10	9.53E-10	4.44E-16	4.44E-16	4.44E-16	5E-18	4.17E-18	5.88E-18	5E-18	4.17E-18	5.88E-18	5.9E-34	1.1E-35	3.58E-32	5.5E-236	5.7E-242	3.5E-231
11	4E-15	4E-15	4E-15	4E-15	4E-15	4E-15	2.08E-17	1.8E-17	0.000259	1.98E-17	1.8E-17	2.6E-15	2.76E-13	5.46E-15	6.09E-12	3E-141	2.3E-146	2.4E-139
16	4E-15	4E-15	4E-15	4E-15	4E-15	4E-15	1.65E-05	1.23E-14	0.002487	6.32E-15	1.69E-17	3.07E-14	1.922084	0.318913	4.59197	3.73E-62	4.36E-65	8.07E-59
23	2.89E-14	2.09E-14	5.91E-14	4E-15	4E-15	4E-15	0.006598	0.002513	0.010312	0.002338	4.06E-13	0.00251	2.407068	0.500883	8.718204	1.15E-22	4.45E-27	2E-15
32	8.34E-12	5.65E-12	1.21E-11	4E-15	4E-15	4E-15	0.172243	0.102429	0.255854	0.006101	0.003679	0.008643	3.918815	0.992347	7.784476	2.44E-14	2.56E-16	2.06E-11
45	4.11E-10	3.06E-10	7.14E-10	4E-15	4E-15	7.55E-15	0.068735	0.038352	0.212237	0.014871	0.011984	0.020782	20.62594	8.518929	46.93707	1.9E-105	2.3E-106	1E-104
64	2.03E-06	1.41E-06	3.37E-06	7.55E-15	7.55E-15	7.55E-15	0.339691	0.202546	0.517202	0.041317	0.033129	0.054282	14.44415	3.890791	32.95751	1.28E-79	2.06E-80	4.71E-79
91	0.000458	0.000342	0.000522	7.55E-15	7.55E-15	7.55E-15	1.388429	0.806441	1.81424	0.139021	0.084565	0.17504	5084.795	1319.733	9805.743	1.3E-44	5.47E-45	3.89E-44
128	0.003115	0.002548	0.00373	1.47E-14	1.47E-14	1.47E-14	1.098613	0.550989	2.353469	0.377964	0.224182	0.618682	2131.913	474.8557	4099.894	2.79E-26	1.41E-26	3.93E-26
181	0.168384	0.148032	0.200669	2.89E-14	2.44E-14	3.24E-14	7.573897	4.408934	9.563051	0.554747	0.450973	1.927348	3859.18	3170.684	8396.514	4.25E-12	2.72E-12	1.17E-11
256	0.922975	0.819049	1.029368	3.42E-11	2.15E-11	4.72E-11	67.98074	60.75733	72.51004	0.932017	0.64304	1.510612	8776.596	6759.285	10344.96	0.002797	0.002012	0.006053
362	3.173795	3.096221	3.273139	2.37E-07	1.67E-07	3.06E-07	204.7434	189.3453	220.9826	1.700008	1.156676	2.056493	480066.6	416449.8	516179.2	12.96828	2.408358	25.59081
512	5.542518	5.478984	5.742155	7.7E-05	6.14E-05	9.1E-05	429.5345	419.0179	437.1629	3.087214	2.708299	6.517933	1.23E+08	1.19E+08	1.31E+08	3.230271	0.52342	9.1101
724	9.662787	9.492888	9.985928	0.006798	0.00532	0.008524	802.6308	781.5319	856.153	12.45139	10.91272	13.56473	79815188	75177988	83666610	10.82443	6.255314	24.40791
1024	13.59118	13.31548	13.78958	0.203708	0.172892	0.232754	1414.152	1386.456	1432.756	52.18092	46.31292	57.24082	2.27E+08	2.21E+08	2.31E+08	260.8096	239.176	273.6085
1448	16.38079	16.16259	16.80412	1.445439	1.396135	1.497215	2510.227	2472.491	2562.187	274.9363	247.4382	292.6627	8.86E+08	8.59E+08	8.97E+08	48926.78	44910.66	53193.23

Dims.	Brown						Chung-Reynolds						Deflected Corrugated Spring					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	2E-147	1.6E-149	8.4E-147	0	0	0	4.3E-282	1.5E-286	5.7E-279	0	0	0	-0.99119	-0.99586	-0.98386	-0.99119	-0.99586	-0.98386
3	1.3E-118	3.7E-120	2.6E-117	0	0	0	2.5E-222	5.3E-225	4.7E-218	0	0	0	-0.90328	-0.94986	-0.85608	-0.90328	-0.94986	-0.85608
4	8.5E-97	2.8E-99	1.36E-95	0	0	0	8.4E-179	2.9E-183	3.4E-175	0	0	0	-0.84334	-0.8462	-0.84334	-0.84334	-0.8462	-0.84334
6	1.53E-67	8.14E-69	1.1E-65	0	0	0	1.8E-118	1.8E-122	9.6E-116	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
8	2.49E-49	3.88E-50	2.23E-47	1.3E-250	3.7E-255	1.1E-247	1.87E-85	5.26E-88	2.87E-83	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
11	5.09E-31	4.22E-32	8.04E-30	1.1E-159	7E-163	2E-155	1.39E-54	8.59E-58	1.7E-51	1.71043669123705e-310	2.48368875240107e-317	8.56E-304	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
16	2.55E-19	1.3E-20	6.54E-18	4.96E-83	8.94E-90	1.91E-79	2.61E-33	2.97E-35	1.1E-31	1.9E-163	7.8E-173	1.1E-149	-0.82151	-0.84334	-0.37336	-0.82151	-0.84334	-0.37336
23	5.84E-15	5.97E-16	3.87E-14	7.05E-45	7.3E-49	7.08E-42	9.96E-24	9.94E-27	9.3E-23	7.38E-83	5.96E-90	3.85E-73	-0.37336	-0.37336	-0.37336	-0.37336	-0.37336	-0.37336
32	7.17E-12	1.23E-12	7.25E-11	6.19E-30	2.43E-33	1.11E-27	6.49E-21	1.41E-21	7.53E-20	2.25E-65	5.91E-68	1.22E-54	-0.37336	-0.37336	-0.37336	-0.37336	-0.37336	-0.37336
45	5.07E-10	1.2E-10	1.27E-08	2.17E-27	9.53E-30	1.52E-23	1.76E-17	3.56E-18	8.7E-17	1.32E-57	7.27E-62	6.41E-54	-0.37196	-0.37336	0.180816	-0.37196	-0.37336	0.180816
64	6.92E-07	3.53E-07	2.6E-06	1.79E-25	9.23E-27	3.61E-24	1.71E-11	9.06E-12	3.29E-11	2.18E-57	2.58E-58	2.71E-54	0.270901	-0.10909	0.409922	0.270901	-0.10909	0.409922
91	0.000121	3.83E-05	0.000487	7.14E-24	2.78E-24	1.24E-21	2.11E-05	1.66E-05	4.26E-05	1.46E-54	3.67E-56	9.47E-54	0.121815	-0.22454	0.409922	0.121815	-0.22454	0.409922
128	0.019071	0.008034	0.038892	1.39E-20	5.83E-22	5.15E-20	3.374503	2.32492	4.731296	6.17E-50	9.14E-51	1.47E-49	0.671351	0.47902	1.033728	0.671351	0.47902	1.033728
181	2.228441	1.533486	3.550197	1.4E-17	1.74E-18	6.53E-17	85334.6	50505.52	129660.6	3.23E-41	1.29E-41	6.34E-41	0.919469	0.541958	1.509144	0.919469	0.541958	1.506499
256	42.52684	35.84391	70.92572	4.58E-15	2.48E-15	7.32E-15	1.61E+08	1.5E+08	2.34E+08	7.63E-29	5.94E-29	9.4E-29	2.177973	1.668028	2.667111	2.177973	1.668028	2.660884
362	1009.632	742.0355	1313.535	1.34E-10	5.35E-11	2.39E-10	2.02E+10	1.65E+10	2.45E+10	3.08E-17	2.51E-17	4.04E-17	13.9017	12.19302	14.70252	3.209259	2.395163	3.918092
512	4932.304	4085.224	6152.43	1.93E-06	1.36E-06	2.3E-06	1.11E+12	9.63E+11	1.24E+12	3.59E-08	2.98E-08	5.11E-08	14.8226	14.59247	15.11854	3.250088	3.037189	4.231318
724	20999.45	14958.93	38226.62	0.001292	0.000685	0.001498	2.82E+08	2.61E+08	3.03E+08	4.45E-09	4E-09	5.31E-09	269.1237	253.9573	280.8773	4.813287	4.654118	5.30441
1024	4068914	1846438	9626774	1.657839	1.385098	2.276324	7.61E+09	6.69E+09	8.51E+09	0.00752	0.006331	0.00904	170.5096	164.6609	178.5525	7.335065	6.872482	7.934952
1448	7.6E+08	2.36E+08	1.57E+09	149.3479	130.132	195.9834	9.31E+10	8.7E+10	9.9E+10	301.3699	268.6477	337.1479	225.9915	221.5069	235.704	10.91701	9.85238	11.74971

Table B.3: sep-CMA-ES Descriptive Statistics: Exponential to Rastrigin Functions (10,000 Evaluation Tuning)

Dims.	Exponential						Griewank						Inverted Cosine Wave						Levy						Qing						Rastrigin					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	-1	-1	-1	-1	-1	-1	1.42E-06	0	0.002006	0	0	0.002006	-0.99271	-0.99797	-0.98393	-0.99271	-0.99797	-0.98393	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.97E-31	1.97E-31	1.97E-31	1.97E-31	1.97E-31	1.97E-31	0	0	0	0	0	0
3	-1	-1	-1	-1	-1	-1	0.009145	0.006308	0.009865	0.007914	0.006308	0.009859	-1.7541	-1.83965	-1.68469	-1.7541	-1.83965	-1.68469	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.5E-32	8.38E-31	5.92E-31	1.24E-30	5.92E-31	5.55E-31	8.38E-31	0	0	0	0	0	0
4	-1	-1	-1	-1	-1	-1	6.12E-05	1.11E-16	0.008611	0	0	0.007396	-2.47617	-2.47618	-2.41285	-2.47618	-2.47618	-2.41285	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.5E-32	8.87E-30	5.13E-30	9.71E-30	8.87E-30	5.13E-30	9.71E-30	0	0	0	0	0	0
6	-1	-1	-1	-1	-1	-1	0	0	0.009857	0	0	0.009857	-3.69046	-3.69046	-3.69038	-3.69046	-3.69046	-3.69046	1.5E-32	1.5E-32	1.81E-32	1.5E-32	1.5E-32	1.5E-32	1.39E-28	1.01E-28	2.43E-28	1.39E-28	1.01E-28	2.04E-28	0	0	0.994959	0	0	0.994959
8	-1	-1	-1	-1	-1	-1	8.82E-13	6.24E-14	1.21E-11	0	0	0	-3.84013	-4.28396	-3.27382	-3.84238	-4.28398	-3.27427	4.07E-20	4.76E-21	4.86E-19	1.5E-32	1.5E-32	1.52E-31	7.33E-28	5.23E-28	2.04E-27	6.46E-28	5.23E-28	1.24E-27	1.989918	0.746219	2.984877	1.989918	0.746219	2.984877
11	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	-3.68955	-4.92137	-3.27444	-3.92497	-4.92658	-3.2754	5.37E-31	4.13E-31	1.21E-30	5.37E-31	4.13E-31	1.21E-30	9.06E-29	7.17E-29	1E-28	9.06E-29	7.17E-29	1E-28	4.974795	3.979836	6.218494	4.974795	3.979836	6.218494
16	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	-4.65823	-5.08159	-3.7303	-4.66625	-5.08552	-3.7462	2.7E-30	1.92E-30	3.73E-30	2.7E-30	1.92E-30	3.73E-30	3.11E-28	2.45E-28	3.36E-28	3.11E-28	2.45E-28	3.36E-28	10.94454	7.710933	12.93446	10.94454	7.710933	12.93446
23	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	-7.37316	-8.15596	-6.61545	-7.38779	-8.18032	-6.6366	5.9E-09	1.59E-10	9.13E-08	1.12E-23	1.12E-25	7.89E-17	2.97E-22	6.5E-23	5.29E-21	6.59E-27	5.82E-27	7.44E-27	14.92438	12.93447	18.40673	14.92438	12.93447	18.40673
32	-1	-1	-1	-1	-1	-1	4.46E-09	4.81E-10	1.05E-08	0	0	0	-9.51021	-11.3673	-8.70256	-9.62277	-11.4205	-8.73277	1.58E-13	2.96E-15	0.089528	1.56E-28	7.88E-29	0.089528	1.84E-14	2.14E-16	6.48E-14	1.44E-26	1.26E-26	1.58E-26	25.00298	21.42335	28.21234	24.89465	21.22885	28.16684
45	-1	-1	-1	-1	-1	-1	1.23E-08	9.43E-09	3.49E-08	0	0	0	-11.7636	-12.539	-11.2175	-11.9098	-12.6835	-11.3299	5.66E-05	1.89E-06	0.476718	8.4E-09	1.73E-11	0.476706	1.01E-09	2.34E-10	8.61E-09	3.49E-26	2.99E-26	4.19E-26	31.38099	26.72839	41.85428	31.34754	26.6295	41.79009
64	-1	-1	-1	-1	-1	-1	6.26E-06	4.71E-06	8.19E-06	0	0	0	-16.7904	-18.758	-15.5669	-17.0681	-18.8962	-15.6736	0.318782	0.000658	0.544088	0.316713	1.13E-05	0.543853	0.0642	0.011277	0.244712	1.12E-25	1.03E-25	1.34E-25	52.17724	49.23213	56.00153	51.31841	47.24297	54.81988
91	-1	-1	-1	-1	-1	-1	0.000165	0.000119	0.000217	0	0	0	-23.9079	-25.0866	-22.1743	-24.1156	-25.45	-22.7983	0.128664	0.024969	0.613191	0.091711	0.001741	0.497395	5.548593	1.418956	27.24025	7.72E-25	6.74E-25	8.71E-25	74.44736	66.61404	86.30752	74.20402	65.62995	82.1644
128	-1	-1	-1	-1	-1	-1	0.003256	0.002536	0.014223	0	0	0.012316	-31.9034	-34.6777	-27.7441	-32.6122	-35.4691	-28.1519	0.433005	0.306364	0.844078	0.193431	0.059232	0.607174	206.734	84.23803	354.245	3.49E-20	5.24E-23	2.34E-18	112.5642	102.8185	128.8966	103.5368	95.28552	112.968
181	-1	-1	-1	-1	-1	-1	0.116807	0.110986	0.134019	9.99E-16	6.66E-16	0.001849	-44.4091	-47.8204	-41.017	-45.1173	-48.571	-41.5559	2.616587	2.023655	2.824	0.655237	0.550159	0.996959	4435.776	3100.27	5996.619	4.21E-07	6.68E-08	4.52E-06	216.3505	199.7493	243.4997	176.2013	155.2835	194.8224
256	-0.99992	-0.99993	-0.9999	-1	-1	-1	0.809318	0.770488	0.848399	8.02E-12	6.82E-12	9.98E-12	-63.046	-65.8405	-61.6079	-64.4062	-67.53	-62.9886	10.52662	9.962813	11.56945	1.286908	0.998065	1.726238	177342.4	162090.6	207286.5	1.114879	0.295732	4.224252	487.1902	472.9936	506.4383	376.5915	360.1747	393.0083
362	-0.9954	-0.99561	-0.99487	-1	-1	-1	2.834896	2.539239	2.983043	1.72E-07	1.39E-07	0.001849	-89.0829	-92.8636	-86.5091	-92.5144	-96.5663	-90.0984	39.7628	34.61896	42.66787	4.49899	2.714858	6.436853	3945239	3608511	4693160	58.71322	32.433	115.3873	1609.211	1459.859	1712.953	765.1216	742.2379	794.2239
512	-0.91698	-0.92676	-0.91153	-1	-1	-1	29.08458	25.40681	31.0702	7.81E-05	6.8E-05	0.000112	-125.492	-128.581	-117.793	-132.312	-135.577	-125.166	160.6797	146.2402	168.2746	10.74005	9.045613	13.8215	1.7E+08	1.29E+08	1.91E+08	2318.518	1463.165	5615.525	3946.56	3828.956	4061.828	2805.272	2627.923	2866.711
724	-0.43005	-0.46505	-0.39905	-1	-1	-1	210.9674	198.1512	235.7404	0.00615	0.005938	0.007102	-170.647	-176.485	-167.053	-188.114	-192.847	-181.979	566.9287	506.7774	619.4269	21.4174	17.69929	25.07221	4.32E+09	3.72E+09	4.8E+09	28076.53	25937.03	36460.67	6456.836	6268.392	6572.389	6044.835	5923.702	6197.809
1024	-0.01048	-0.01342	-0.00855	-0.99999	-0.99999	-0.99999	991.2177	944.9044	1046.321	0.13855	0.135231	0.142738	-221.186	-228.626	-218.565	-255.369	-263.01	-250.371	1937.772	1854.031	2082.563	34.29675	31.55198	42.41219	3.94E+10	3.67E+10	4.21E+10	558304.2	488391.9	652223.9	9925.508	9787.424	10123.14	8797.375	8681.713	8925.973
1448	-6.7E-09	-1.2E-08	-4.2E-09	-0.99804	-0.99816	-0.99793	3244.855	3103.801	3383.997	1.164091	1.110174	1.175932	-298.241	-307.032	-290.928	-353.066	-364.815	-341.882	4267.965	4207.888	4300.993	2862.475	2842.847	2924.734	2.5E+11	2.32E+11	2.77E+11	10523031	10140230	11201094	16455.91	16248.3	16743.93	11968.42	11781.1	12180.84

Table B.4: sep-CMA-ES Descriptive Statistics: Exponential to Rastrigin Functions (50,000 Evaluation Tuning)

Dims.	Exponential						Griewank						Inverted Cosine Wave						Levy						Qing						Rastrigin					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	-1	-1	-1	-1	-1	-1	0.000296	0	0.00521	0.000296	0	0.00521	-0.99565	-0.9973	-0.99153	-0.99565	-0.9973	-0.99153	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.97E-31	1.97E-31	1.97E-31	1.97E-31	1.97E-31	1.97E-31	0	0	0	0	0	0
3	-1	-1	-1	-1	-1	-1	0.005195	1.99E-11	0.008013	0.005195	0	0.007396	-1.73793	-1.81763	-1.64432	-1.73793	-1.81763	-1.64432	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.18E-30	1.04E-30	1.77E-30	5.92E-31	4.44E-31	1.04E-30	0	0	0	0	0	0
4	-1	-1	-1	-1	-1	-1	0.007396	4.17E-11	0.007396	0.007396	0	0.007396	-2.44304	-2.47618	-2.21427	-2.44512	-2.47618	-2.21427	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.5E-32	1.5E-32	7.03E-30	4.71E-30	9.61E-30	7.03E-30	4.71E-30	9.61E-30	0	0	0	0	0	0
6	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	-3.69046	-3.69046	-3.58005	-3.69046	-3.69046	-3.58012	2.1E-30	5.14E-31	5.73E-30	1.5E-32	1.5E-32	1.5E-32	1.29E-28	6.77E-29	1.84E-28	1.29E-28	6.77E-29	1.84E-28	0.994959	0	0.994959	0.994959	0	0.994959
8	-1	-1	-1	-1	-1	-1	5.33E-15	1.94E-16	0.010472	0	0	0.010472	-3.83023	-4.13266	-3.39893	-3.84264	-4.28395	-3.39893	4.15E-19	2.49E-20	3.32E-18	1.5E-32	1.5E-32	1.5E-32	6.63E-28	4.93E-28	2.47E-27	5.92E-28	3.98E-28	1.21E-27	1.365176	0.994959	2.238658	1.365176	0.994959	2.238658
11	-1	-1	-1	-1	-1	-1	0	0	0.001849	0	0	0.001849	-3.72925	-4.491	-3.38134	-3.80437	-4.49476	-3.39157	3.87E-31	2.14E-31	5.75E-31	3.87E-31	2.14E-31	5.75E-31	6.24E-17	6.25E-18	7.53E-13	5.06E-27	3.57E-27	6.8E-27	5.536191	3.98013	7.974181	5.472275	3.979836	7.959667
16	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	-5.27145	-5.97957	-4.62617	-5.27371	-5.98022	-4.62632	3.59E-29	9.98E-31	5.66E-28	1.5E-32	1.5E-32	1.5E-32	0.000454	1.4E-06	0.089985	1.05E-25	4.86E-26	2.79E-25	10.94455	8.705892	12.93446	10.94455	8.705892	12.93446
23	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	-6.91551	-7.7853	-5.99913	-6.9342	-7.7966	-6.01623	1.51E-29	1.24E-29	0.11191	1.51E-29	1.24E-29	0.11191	4.36E-27	2.78E-27	6.22E-27	4.36E-27	2.78E-27	6.22E-27	15.42479	12.68808	19.15822	15.42193	12.68573	18.44861
32	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	-8.67787	-9.74288	-7.63133	-8.79836	-9.77289	-7.65841	3.73E-11	1.53E-12	0.089528	4.56E-28	2.53E-28	0.089528	8.44E-14	4.6E-15	7.44E-13	8.22E-26	7.68E-26	9.11E-26	22.62591	19.25572	24.46617	21.89511	18.91846	24.39641
45	-1	-1	-1	-1	-1	-1	8.62E-05	3.02E-05	0.000152	2.78E-16	1.11E-16	4.44E-16	-11.4424	-13.2443	-10.132	-11.7058	-13.4402	-10.1867	7.02E-06	1.54E-06	0.022458	6.04E-11	4.3E-12	0.022382	8.49E-06	3.92E-06	4.14E-05	4.28E-26	3.62E-26	5E-26	31.52786	28.00425	40.33201	31.43025	27.90602	40.08944
64	-1	-1	-1	-1	-1	-1	0.000554	0.000439	0.000847	0	0	0	-15.4466	-17.4925	-14.4747	-15.6328	-17.6292	-14.5206	0.019708	0.002255	0.236034	0.003033	0.000153	0.1814	0.067346	0.008639	0.578037	1.29E-25	1.12E-25	1.5E-25	47.19048	42.88274	55.73848	46.32905	40.89572	54.39797
91	-1	-1	-1	-1	-1	-1	0.009721	0.007565	0.012443	0	0	0	-23.4296	-25.615	-22.5434	-23.6169	-25.7782	-23.0301	0.193965	0.090046	0.521856	0.089598	0.003972	0.458029	111.9713	68.12012	250.1199	1.46E-24	1.32E-24	1.62E-24	75.75394	65.49902	89.18047	68.69862	62.05705	79.81526
128	-1	-1	-1	-1	-1	-1	0.46674	0.426831	0.514872	0	0	0	-31.9729	-34.5085	-30.1968	-32.8563	-35.0505	-31.1817	0.575485	0.425612	0.985924	0.163935	0.097597	0.557684	255.0694	119.8415	291.5486	4.44E-21	2E-22	4.93E-20	157.4743	148.2883	170.4122	96.01961	89.00824	109.6467
181	-0.99998	-0.99998	-0.99997	-1	-1	-1	29.63809	26.03055	34.47005	2.39E-09	1.27E-09	5.14E-09	-46.1792	-48.8504	-42.0783	-47.2106	-50.0851	-43.0147	4.4765	3.746283	4.757992	0.232024	0.150462	0.565668	5978.876	4863.694	8096.988	1.12E-06	4.85E-07	1.28E-05	582.6356	528.5421	661.9889	150.5243	139.1031	163.7714
256	-0.99889	-0.99901	-0.9988	-1	-1	-1	2.08175	1.953161	2.215267	4.93E-08	4.13E-08	6.34E-08	-65.1106	-67.6682	-59.2358	-67.2267	-69.872	-61.568	19.00392	17.00827	21.0685	0.980034	0.518494	1.262772	443768.5	409260.9	516335.3	0.424834	0.274531	0.70554	2483.932	2452.049	2503.604	217.4229	206.7211	223.1133
362	-0.98107	-0.98167	-0.98008	-1	-1	-1	19.02392	17.61411	21.08898	4.18E-05	3.57E-05	4.55E-05	-84.1543	-89.7508	-79.5521	-92.8438	-99.0044	-88.6278	79.33492	74.2114	82.86984	1.134272	0.628986	1.894091	6623316	5872086	7103689	88.09504	43.06888	188.5342	3948.796	3912.042	4014.513	339.179	324.2253	357.0343
512	-0.82208	-0.83813	-0.81378	-1	-1	-1	33.47812	31.09334	37.1999	0.000125	0.000114	0.000133	-120.988	-131.187	-117.989	-128.371	-140.126	-124.993	225.9877	216.6875	236.6674	2.906322	1.924534	3.277621	1.86E+08	1.58E+08	2.37E+08	1734.921	1302.032	2611.629	6058.371	5984.792	6219.153	517.1515	506.743	534.0669
724	-0.27391	-0.29654	-0.25136	-1	-1	-1	205.4543	193.2783	217.939	0.005868	0.005508	0.006666	-165.119	-172.662	-162.294	-181.186	-190.905	-178.138	715.4218	650.805	765.1513	8.063731	6.322888	10.36255	3.81E+09	3.14E+09	4.35E+09	25867.36	19078.54	37305.6	8291.156	8126.626	8341.051	869.1374	846.0711	900.976
1024	-0.00311	-0.00549	-0.00285	-0.99999	-1	-0.99999	1079.901	1055.859	1119.113	0.146935	0.136438	0.154677	-213.03	-220.508	-206.436	-263.322	-270.392	-251.589	2066.436	1959.149	2221.348	20.20559	17.89795	27.12141	3.66E+10	3.41E+10	4.33E+10	531592	483267.7	572920	11999.21	11860.08	12229.62	1503.639	1477.081	1550.178
1448	-1.2E-08	-2.2E-08	-5.8E-09	-0.99892	-0.99899	-0.99884	3221.463	3081.168	3380.953	1.121865	1.08547	1.162995	-255.34	-261.729	-248.578	-364.73	-370.645	-352.59	4594.392	4515.608	4717.176	79.22757	71.35906	83.51556	2.44E+11	2.24E+11	2.68E+11	10515728	10153373	11237570	18069.21	17904.64	18167.26	3680.647	3597.843	3726.799

Table B.5: sep-CMA-ES Descriptive Statistics: Rosenbrock to Sum Squares Functions (10,000 Evaluation Tuning)

Dims.	Rosenbrock						Schwefel						Sphere						Sum of Different Powers						Sum Squares					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.004189	0.002816	0.004762	4.99E-08	2.93E-08	5.96E-08	256.6232	108.5696	434.2793	256.6232	108.5696	434.2793	1.4E-147	3.7E-149	1.5E-144	0	0	0	1.9E-175	3.6E-176	9.7E-174	0	0	0	4.8E-145	9.1E-147	2.7E-143	0	0	0
3	0.179581	0.167439	0.189781	0.005655	0.005452	0.005927	385.0309	212.229	483.6497	385.0309	212.229	483.6497	5.1E-118	1.7E-119	7.2E-116	0	0	0	1.2E-148	6.3E-151	1.2E-143	0	0	0	2.6E-115	8.8E-117	6.4E-114	0	0	0
4	0.534821	0.502123	0.578271	0.041871	0.039968	0.042909	414.5451	261.5694	636.6183	414.5451	261.5694	636.6183	1.25E-96	1.45E-97	1.23E-95	0	0	0	1.5E-111	1.4E-114	1.2E-109	0	0	0	3.5E-94	5.06E-97	2.76E-92	0	0	0
6	0.131431	0.096915	0.153526	1.55E-05	1.3E-05	2.15E-05	1085.718	834.1179	1243.668	1085.718	834.1179	1243.668	9.08E-67	6.04E-69	6.57E-66	0	0	0	3.64E-53	3.4E-56	1.23E-50	2.9E-240	7.5E-244	9.8E-236	1.24E-65	5.73E-66	3.85E-64	0	0	0
8	0.306172	0.212856	4.026269	1.1E-05	8.95E-06	1.304186	1016.613	814.2819	1373.944	1016.613	814.2819	1373.944	3.38E-50	4.22E-51	4.87E-48	7.4E-252	2.2E-255	1.9E-250	2.03E-25	6.56E-27	2.62E-24	2.4E-103	4.2E-106	7.8E-101	4.85E-47	3.31E-48	2.25E-45	1.9E-249	6E-251	3.3E-246
11	3.267219	2.080389	5.313997	0.000105	4.43E-05	3.986627	1618.738	1426.35	1840.919	1618.738	1426.35	1840.919	7.57E-33	1.42E-33	2.83E-31	6.7E-160	7.9E-164	3.9E-157	4.68E-15	6.81E-17	3.32E-14	6.3E-52	5.76E-56	4.3E-49	2.67E-31	4.72E-32	1.31E-29	2.5E-158	1.5E-163	2.4E-156
16	9.378098	8.009099	12.3728	0.010806	0.005498	3.99053	2323.358	1939.543	2655.376	2319.68	1939.533	2655.376	2.08E-20	1.18E-21	1.27E-18	1.11E-86	4.38E-88	6.29E-81	2.68E-10	1.89E-11	1.19E-09	1.43E-22	1.31E-24	2.14E-19	1.24E-16	4.58E-17	5.53E-14	1.86E-78	8.29E-84	1.86E-74
23	16.67645	15.99397	19.89285	3.54837	3.106767	5.899515	3819.952	3420.122	4308.508	3819.952	3420.122	4308.508	9.24E-16	2.53E-17	3.53E-14	3.22E-43	8.01E-47	1.68E-37	1.93E-08	5.55E-09	7.26E-08	4.84E-14	1.81E-15	3.08E-12	2.82E-10	2.34E-11	2.04E-09	1.07E-37	3.23E-44	8E-35
32	26.2276	25.17422	27.25149	15.80352	14.88251	17.20961	5636.161	5182.125	6273.201	5636.161	5182.125	6273.201	3.16E-14	6.85E-15	1.31E-13	7.46E-32	4.35E-35	3.06E-30	2.15E-07	5.2E-08	4.36E-07	2.85E-10	4.76E-11	6.01E-10	2.36E-29	9.2E-30	2.27E-28	7.9E-183	2.2E-183	7.4E-182
45	40.94554	39.54299	42.8069	31.86697	31.09282	33.76808	7374.449	6515.615	7893.861	7374.283	6515.599	7893.529	1.35E-12	6.9E-13	2.97E-12	2.45E-31	1.55E-34	7.7E-30	3.01E-07	1.61E-07	5.04E-07	8.04E-10	3.47E-11	2.65E-09	3.08E-17	1.22E-17	6.3E-17	2.8E-128	7.4E-129	1.3E-127
64	61.66474	59.85496	63.3725	52.95853	52.23285	53.94176	10839.34	9712.873	11474.52	10839.34	9712.873	11474.52	2.44E-10	1.92E-10	3.65E-10	3.47E-31	8.75E-33	1.43E-30	9.1E-07	6.66E-07	1.75E-06	3.48E-09	1.54E-09	1.37E-08	1.84E-07	1.07E-07	3.59E-07	8.03E-86	1.59E-86	1.28E-85
91	88.22467	87.02983	89.24518	81.82251	81.20364	82.3303	14184.57	13152.85	14663.85	14184.57	13152.85	14663.85	1.98E-07	1.59E-07	4.1E-07	6.91E-31	1.2E-31	7.24E-30	2.5E-06	1.59E-06	3.15E-06	2.14E-08	1.01E-08	4.16E-08	0.027598	0.01808	0.044246	1.75E-53	4.44E-54	3.33E-53
128	125.9458	125.0119	175.211	118.9741	118.5179	122.5281	19929.88	18098.35	20952.34	19929.88	18098.35	20952.34	9.58E-09	6.99E-09	1.33E-08	1.25E-52	5.7E-53	1.75E-52	2.31E-06	2.07E-06	3.33E-06	1.79E-08	1.13E-08	2.53E-08	10.03651	6.354836	21.9611	2.87E-30	1.83E-30	6.65E-30
181	339.8062	223.024	436.7772	177.0375	172.0052	216.3216	27214.47	25644.91	30177.08	27214.47	25644.9	30177.07	1.55E-05	1.22E-05	1.75E-05	2.88E-37	1.64E-37	3.43E-37	3E-06	2.34E-06	3.81E-06	5.52E-08	4.23E-08	8.69E-08	155.5997	113.433	207.2832	1.21E-13	3.26E-14	4.42E-13
256	502.9634	458.7916	748.9676	247.9929	247.1994	252.6853	35569.36	34502.17	37495.71	35567.2	34498.81	37490.94	0.003616	0.003149	0.003846	4.19E-26	3.42E-26	5.73E-26	6.08E-06	4.27E-06	8.23E-06	1.81E-07	1.37E-07	2.57E-07	767.1723	684.3171	944.9267	0.000184	0.000141	0.000258
362	2433.371	2119.742	3473.325	406.4753	355.5972	409.209	45316.15	43169.99	47908.36	45112.03	42917.44	47700.83	0.225597	0.205399	0.241328	7.34E-18	6.22E-18	8.22E-18	1.63E-05	1.42E-05	2.2E-05	5.35E-07	4.01E-07	6.63E-07	4055.93	3835.781	4590.048	15.70813	10.91494	20.48664
512	22584.1	20344.76	25004.93	608.1902	559.406	662.9307	57986.9	54146.2	62204.84	54243.16	48939.53	57661.01	4.090882	3.912519	4.427671	8.54E-12	6.75E-12	9.68E-12	0.000862	0.000171	0.008576	8.07E-07	7.66E-07	1.06E-06	21938.98	19790.01	23230.13	187.2143	124.3847	221.6883
724	287799.9	259458	333271.6	875.5359	823.5122	965.147	96765.59	93482.92	100591.4	54059.86	47832.34	58075.7	42.42986	40.03764	44.40158	1.75E-07	1.47E-07	1.97E-07	10128481	582221.8	3.78E+08	1.58E-06	1.4E-06	1.78E-06	120962	113272.1	126979.3	1602.316	1400.112	1909.937
1024	3027237	2741804	3265934	1504.047	1359.255	1663.323	178457.1	176760.4	182926.9	164471.5	163123.4	169127	244.076	232.7391	267.0657	0.000261	0.000223	0.000284	6.28E+43	4.25E+40	1.18E+46	3.57E-05	1.36E-05	0.000111	613927.9	594968.8	628500.2	6328.164	5743.767	7088.026
1448	17314572	15793076	20006372	3335.619	3154.341	3578.837	276642.6	273614.9	280951.1	237880.9	232569.7	240269.9	772.6778	746.2184	828.2262	0.050539	0.047823	0.052805	2E+128	1.6E+121	7.4E+132	6.37E+26	3.61E+24	2E+28	2305009	2246544	2370977	28985.09	25772.37	32314.52

Table B.6: sep-CMA-ES Descriptive Statistics: Rosenbrock to Sum Squares Functions (50,000 Evaluation Tuning)

Dims.	Rosenbrock						Schwefel						Sphere						Sum of Different Powers						Sum Squares					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	7.21E-05	4.1E-05	0.0001	2.25E-16	6.06E-17	4.95E-16	5.46E-06	-19.7381	143.1137	5.46E-06	-19.7381	143.1137	1E-146	1.5E-147	4.6E-145	0	0	0	1.2E-174	5E-177	5E-173	0	0	0	1E-147	3.1E-149	3.4E-146	0	0	0
3	0.006294	0.005263	0.036615	7.6E-09	4.34E-09	2.84E-08	463.9013	192.4643	570.4426	463.9013	192.4643	570.4426	2.7E-118	2.4E-119	1.8E-117	0	0	0	1.1E-148	8.3E-151	9.9E-146	0	0	0	8.1E-116	2.1E-117	9.8E-114	0	0	0
4	0.535684	0.491976	0.565328	0.042308	0.041048	0.043494	572.4711	335.578	764.9535	572.4711	335.578	764.9535	2.69E-97	4.7E-99	7.55E-96	0	0	0	1.1E-111	3E-113	7.3E-109	0	0	0	6.01E-95	1.29E-95	7.94E-94	0	0	0
6	0.283959	0.231289	0.331873	0.000436	0.000367	0.000619	750.1212	468.9254	1110.405	750.1212	468.9254	1110.405	7.03E-68	1.69E-69	5.89E-67	0	0	0	2.71E-53	1.65E-55	1.64E-51	5.9E-241	2.4E-246	1.8E-238	4.4E-66	3.85E-67	1.15E-64	0	0	0
8	0.324813	0.239225	1.412567	1.99E-05	1.36E-05	3.44E-05	1333.288	898.1728	1637.214	1333.288	898.1728	1637.214	5.16E-49	8.79E-50	8.54E-48	2.7E-249	2.3E-252	1.8E-248	1.68E-26	3.34E-28	3.27E-24	2.8E-104	5.5E-106	7.8E-103	1.04E-46	3E-48	1.43E-45	7.7E-251	1.2E-253	7E-247
11	3.397024	2.934374	4.399258	0.003274	0.001177	1.170008	1954.335	1184.419	2294.939	1954.335	1184.419	2294.939	2.33E-32	8.62E-34	1.7E-31	1.7E-159	1.9E-166	3.3E-155	1.35E-15	2.01E-17	8.82E-15	7.77E-54	2.44E-55	1.23E-52	4.35E-30	2.97E-33	3.49E-28	1.8E-156	3.4E-160	8.3E-153
16	8.5657	7.88759	10.34035	0.005638	0.004464	1.018048	2517.25	2166.931	2857.612	2517.25	2166.93	2857.612	5.59E-21	7.99E-22	2.19E-19	3.54E-86	1.7E-88	4.8E-81	6.96E-10	1.28E-11	3.41E-09	1.96E-21	3.04E-23	3.36E-19	3.75E-17	4.53E-19	1.22E-16	1.05E-79	6.65E-82	6.7E-72
23	16.27518	15.7337	19.33574	2.765163	1.955591	4.834969	3982.565	3232.662	4442.997	3982.557	3232.639	4442.997	1.55E-16	3.42E-17	5.33E-15	8.04E-48	7.78E-52	1.18E-41	8.9E-08	2.52E-08	2.32E-07	9.27E-13	5.83E-14	1.19E-11	5.81E-11	6.18E-12	1.86E-10	4.6E-38	4.9E-46	9.56E-35
32	27.80283	25.68559	29.62154	15.0394	13.26957	18.01945	4846.449	3938.484	5137.831	4846.449	3938.484	5137.831	2.86E-14	7.15E-15	1.06E-13	4.58E-34	3.81E-36	2.67E-29	1.32E-07	6.16E-08	6.27E-07	1.46E-10	1.03E-11	4.39E-10	1.66E-07	5.03E-08	3.7E-07	2.68E-25	2.1E-29	4.12E-23
45	39.67103	38.75464	41.61208	29.87025	28.69552	31.8184	7275.497	6332.491	7833.103	7275.497	6332.491	7833.103	1.62E-12	6.68E-13	3.61E-12	2.13E-32	3.44E-34	3.55E-31	3.57E-07	1.65E-07	7.82E-07	1.45E-09	1.34E-10	3.5E-09	4.39E-05	1.27E-05	0.000148	1.96E-19	1.45E-21	5.71E-18
64	60.86949	60.13865	78.23059	51.44102	50.68623	53.30751	9950.094	9467.034	10463.7	9949.82	9466.552	10463.33	2.53E-10	1.78E-10	3.05E-10	7.57E-32	1.12E-33	1.01E-30	1.05E-06	6.28E-07	1.78E-06	2.74E-09	1.55E-09	8.42E-09	0.008712	0.004592	0.020847	3.84E-16	9.46E-18	4.15E-15
91	88.67324	87.26193	89.72259	79.85046	78.77758	81.11847	13987.81	13552.8	15315.42	13987.81	13552.8	15315.42	2.79E-07	2.16E-07	4.6E-07	1.72E-30	1.61E-31	9.47E-30	2.12E-06	1.84E-06	2.66E-06	4.09E-09	2.28E-09	1.73E-08	5.677027	3.487057	9.269875	8.9E-13	1.36E-14	8.07E-12
128	126.0196	125.0185	187.0373	120.1964	119.728	121.0155	20433.1	19249.27	21158.89	20433.1	19249.27	21158.89	0.000104	8.88E-05	0.000128	9.28E-29	2.84E-29	1.46E-28	3.45E-06	2.56E-06	4.2E-06	3.92E-08	1.61E-08	6.6E-08	106.3496	64.12153	186.0524	2.48E-10	1.79E-10	1.47E-09
181	242.0148	183.9284	390.7117	172.8105	172.2804	174.4848	25713.88	24274.08	28078.26	25713.87	24274.06	28078.23	0.012082	0.011224	0.014272	4.59E-25	2.2E-25	6.42E-25	4.26E-06	3.15E-06	5.3E-06	9.02E-08	4.5E-08	1.17E-07	4107.795	3473.013	5408.201	1.01E-12	5.08E-13	3.2E-12
256	1551.887	1401.292	1759.024	249.4276	248.6361	250.2617	38787.66	36629.8	41099.22	38784.96	36627.16	41096.41	0.565334	0.510547	0.648812	4.76E-19	4.19E-19	5.87E-19	5.34E-06	4.6E-06	6.25E-06	1.51E-07	1.23E-07	2.05E-07	820.3814	632.2863	983.6771	0.000177	0.000102	0.000533
362	6456.715	5143.848	7681.782	357.9476	356.0672	427.2204	46200.31	42743.94	48449.6	45981.37	42500.39	48069.68	9.925186	8.482137	11.65313	2.13E-13	1.79E-13	2.47E-13	1.65E-05	1.08E-05	2.39E-05	6.33E-07	5.1E-07	7.72E-07	5195.886	4457.345	5655.407	4.628415	2.640262	9.90068
512	130491.4	114011.4	140626.9	506.8391	505.1571	600.0871	56755.82	55354.19	59911.8	52368.8	49956.85	54467.23	100.8863	91.20703	107.9088	6.82E-09	5.34E-09	7.75E-09	0.000568	0.000201	0.004435	8.3E-07	7.11E-07	9.67E-07	23950.58	21683.16	26019.92	168.1385	137.3243	208.2708
724	1255455	1173848	1345547	759.0818	718.3761	836.768	94059.63	88852.23	97727.43	48355.68	42340.41	53656.2	46.94054	43.26814	49.00449	2.9E-07	2.49E-07	3.15E-07	39915654	450481.5	1.03E+09	1.47E-06	1.19E-06	1.66E-06	187857.1	172685.6	196562.1	1312.555	1037.81	1561.483
1024	4519457	3923416	4918249	1273.873	1185.353	1514.347	199834.4	194901	203963.8	32107.25	28159.51	39572.03	245.6606	234.3097	259.7371	0.000263	0.000242	0.000276	3.05E+45	7.22E+42	1.25E+49	5.76E-05	1.56E-05	0.00019	816414.6	789552.2	852513	5952.67	5101.022	6899.09
1448	19265851	17697105	21253139	3368.445	3274.187	3579.38	354563.3	352924.2	361123.1	-4897.18	-15928.3	4393.335	826.1564	794.9083	858.7776	0.046329	0.04408	0.048756	2.3E+129	3.2E+125	1.7E+136	9.3E+26	2.78E+23	4.26E+30	2627493	2585397	2687968	28211.5	26622.78	31277.49

Table B.7: DE Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (10,000 Evaluation Tuning)

Dims.	Ackley						Alpine1						Bent Cigar						Brown						Chung-Reynolds						Deflected Corrugated Spring					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	2.74E-09	2.05E-09	2.74E-09	2.74E-09	2.05E-09	2.74E-09	0	0	0	0	0	0	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1
3	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	2.74E-09	2.74E-09	3.42E-09	2.74E-09	2.74E-09	3.42E-09	0	0	0	0	0	0	8.88E-16	8.88E-16	8.88E-16	8.88E-16	8.88E-16	8.88E-16	0	0	0	0	0	0	-0.97699	-0.99237	-0.92793	-0.9997	-0.9999	-0.99898
4	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	2.74E-09	2.74E-09	8.22E-09	2.74E-09	2.74E-09	8.22E-09	0	0	0	0	0	0	1.33E-15	1.33E-15	1.33E-15	1.33E-15	1.33E-15	1.33E-15	0	0	0	0	0	0	-0.84334	-0.84334	-0.84334	-0.99582	-0.99877	-0.84334
6	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	6.57E-07	1.68E-07	1.15E-06	6.85E-09	5.48E-09	8.22E-09	0	0	0	0	0	0	2.22E-15	2.22E-15	2.22E-15	2.22E-15	2.22E-15	2.22E-15	0	0	0	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
8	0.000184	0.000129	0.000368	4.44E-16	4.44E-16	4.44E-16	2.18E-07	5.41E-08	1.22E-06	1.1E-08	8.22E-09	1.37E-08	2.89E-06	1.76E-06	5.18E-06	0	0	0	6.02E-07	3.32E-07	8.47E-07	3.11E-15	3.11E-15	3.11E-15	0	0	0	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
11	4.68E-05	3.05E-05	7.59E-05	4.44E-16	4.44E-16	4.44E-16	0.000107	8.38E-05	0.000254	1.64E-08	1.3E-08	1.92E-08	3.45E-06	2.32E-06	8.84E-06	0	0	0	6.7E-11	4.99E-11	9.18E-11	1.11E-14	4.44E-15	1.11E-14	6.23E-16	3.21E-16	1.78E-15	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
16	0.004565	0.004116	0.005319	4.44E-16	4.44E-16	4.44E-16	0.00018	7.1E-05	0.00085	1.88E-07	6.37E-08	1.77E-06	0.173976	0.080468	0.274734	0	0	0	3.25E-07	2.51E-07	5.21E-07	1.83E-14	1.33E-14	2.22E-14	3.26E-08	7.38E-09	8.38E-08	0	0	0	-0.69833	-0.8388	-0.37336	-0.84334	-0.84334	-0.84334
23	0.008046	0.006242	0.009395	4.44E-16	4.44E-16	4.44E-16	0.005556	0.003927	0.007202	1.09E-06	2.08E-07	3.53E-05	1.519746	0.922012	2.131072	0	0	0	0.007735	0.006402	0.009639	9.77E-15	9.77E-15	9.77E-15	2.7E-05	8.42E-06	0.000104	0	0	0	-0.37336	-0.37336	-0.37336	-0.75861	-0.84274	-0.37336
32	0.153244	0.105833	0.198014	4.44E-16	4.44E-16	4.44E-16	0.038032	0.033191	0.051936	1.37E-06	3.41E-07	2.12E-06	1748.245	1076.4	2222.93	0	0	0	0.023635	0.018366	0.033595	1.38E-14	1.38E-14	1.38E-14	7.488214	4.761743	19.6937	0	0	0	-0.3733	-0.37336	-0.32681	-0.37336	-0.37336	-0.37336
45	0.56204	0.214253	0.941621	3.07E-06	8.73E-07	1.48E-05	0.278568	0.195022	0.34899	0.045367	0.02293	0.098446	37335.4	35090.77	52157.18	0	0	0	0.136031	0.108156	0.171311	1.95E-14	1.95E-14	2.62E-14	2742.784	1893.942	3645.41	0	0	0	0.409929	0.409923	0.410358	-0.37336	-0.37336	-0.37336
64	2.984553	2.508986	3.318856	0.029464	4.44E-16	0.533419	0.917265	0.650299	1.261141	0.113191	0.092808	0.190424	160651	132365.5	242622.7	1.6E-06	4.44E-07	10.68332	0.547831	0.459156	0.947068	1.64E-06	5.88E-13	0.003492	28586.32	20934.91	34852.89	0	0	2.02E-20	1.5065	1.377329	1.506516	0.409922	0.409922	0.409922
91	5.126877	4.620529	5.845301	0.869964	0.411947	1.243247	3.661796	3.130232	5.153223	1.949078	1.57066	2.304299	1993971	1840757	2231334	4194.58	1380.886	50324.8	4.659424	4.118655	5.822582	1.85E-08	1.27E-08	2.86E-08	2497393	1916365	3272132	4.96E-06	2.75E-10	0.044794	2.916388	2.88589	2.916669	1.506499	1.078373	1.506499
128	7.772489	7.162963	8.431741	2.625136	1.826509	3.562912	12.92934	11.87789	15.13849	7.784289	5.756914	8.82783	6777991	5687875	9440573	192018.8	125176.4	875113.7	12.89514	10.84828	15.33615	1.433324	0.311681	3.362869	6355207	4768779	10588976	13157.21	2379.134	38376.96	4.703618	4.643547	6.56131	2.916346	2.916346	2.916346
181	10.44219	9.679685	10.70449	5.586234	5.201168	6.287374	55.5385	52.56648	61.36325	22.46169	19.22373	24.182	28253908	22759717	31325460	6213774	4780751	8510901	457.5953	361.2786	529.7626	0.172477	0.140428	0.186203	3.29E+08	2.78E+08	3.93E+08	121524.7	27452.53	183741.5	9.075417	9.029304	11.34073	4.639431	4.639431	4.639431
256	14.24393	14.08089	14.51072	5.219676	4.861618	5.771967	100.9096	91.7364	109.2332	52.39165	51.12139	54.54498	91728579	84560000	1.01E+08	27914486	22547305	33809126	395.0462	323.3752	444.4054	32.85477	31.33201	37.78048	1.17E+09	9.59E+08	1.35E+09	1.57E+08	1.06E+08	1.92E+08	21.51957	18.35769	21.55434	11.68769	11.45303	11.68769
362	15.01553	14.7275	15.39322	11.36981	10.88356	11.59429	176.8133	171.1527	181.9655	147.206	143.587	153.0691	2.23E+08	2.1E+08	2.56E+08	82514466	73633133	1.03E+08	20597.78	8347.779	32184.91	32.5891	27.10648	36.9594	1.14E+10	9.66E+09	1.28E+10	9.31E+08	5.94E+08	1.38E+09	21.55307	18.14381	23.86264	21.55307	17.95174	22.53158
512	16.55489	16.38009	16.68843	13.38113	13.11584	13.5536	341.916	329.9889	354.9405	311.1647	299.9805	329.8755	3.43E+08	2.87E+08	3.66E+08	3.04E+08	2.57E+08	3.32E+08	23249.98	14815.9	49324.64	171.5628	164.317	186.3713	3.91E+10	3.53E+10	4.27E+10	1.09E+10	9.72E+09	1.3E+10	33.70716	29.69938	34.38979	29.79922	28.76175	34.23259
724	14.70599	14.38119	14.92378	14.30507	13.7816	14.43538	664.6573	644.6292	695.1714	523.1261	514.391	536.1577	5.13E+08	4.49E+08	5.93E+08	5.01E+08	4.36E+08	5.83E+08	56125.44	40156.35	168229.2	315.4335	246.6967	348.7636	4.15E+10	3.55E+10	5.5E+10	2.95E+10	2.26E+10	3.83E+10	50.62649	49.71615	61.46212	49.95256	49.04393	58.81753
1024	14.87986	14.56381	15.14961	14.74699	14.36205	14.99055	871.211	840.4662	926.4101	833.7133	788.3753	862.5107	7.22E+08	6.38E+08	8.89E+08	7.07E+08	6.14E+08	8.44E+08	419055	252222.1	928994.2	1842.955	1684.627	2541.535	8.27E+10	6.81E+10	1.14E+11	7.3E+10	6.44E+10	9.53E+10	89.80607	82.38384	97.34579	71.77447	66.76061	78.12777
1448	14.74588	14.517	15.15549	14.35466	13.84677	14.65751	1335.715	1299.707	1377.92	1199.727	1158.834	1242.007	1.06E+09	9.95E+08	1.17E+09	9.62E+08	8.83E+08	1.02E+09	7069836	2981147	16507418	6501541	2541014	13394544	1.71E+11	1.5E+11	2E+11	1.52E+11	1.29E+11	1.62E+11	113.1748	103.5652	128.5492	98.44771	88.16559	104.2567

Table B.8: DE Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (50,000 Evaluation Tuning)

Dims.	Ackley						Alpine1						Bent Cigar						Brown						Chung-Reynolds						Deflected Corrugated Spring					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	2.74E-09	0	2.74E-09	2.74E-09	0	2.74E-09	0	0	0	0	0	0	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	0	0	0	0	0	0	-0.99753	-0.99841	-0.99217	-1	-1	-1
3	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	2.74E-09	2.74E-09	5.48E-09	2.74E-09	2.74E-09	5.48E-09	0	0	0	0	0	0	8.88E-16	8.88E-16	8.88E-16	8.88E-16	8.88E-16	8.88E-16	0	0	0	0	0	0	-0.90449	-0.97347	-0.84334	-1	-1	-0.99999
4	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	5.48E-09	2.74E-09	8.22E-09	5.48E-09	2.74E-09	6.16E-09	0	0	0	0	0	0	1.33E-15	1.33E-15	1.33E-15	1.33E-15	1.33E-15	1.33E-15	0	0	0	0	0	0	-0.84334	-0.84512	-0.84334	-0.99279	-0.99684	-0.98511
6	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	3.81E-06	9.46E-08	1.32E-05	8.22E-09	5.48E-09	8.9E-09	0	0	0	0	0	0	2.22E-15	2.22E-15	2.22E-15	2.22E-15	2.22E-15	2.22E-15	0	0	0	0	0	0	-0.84334	-0.84334	-0.84334	-0.8798	-0.97628	-0.84334
8	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	0.00045	0.000135	0.00076	8.22E-09	7.53E-09	1.16E-08	0	0	0	0	0	0	2.31E-14	1.89E-14	3.36E-14	3.11E-15	3.11E-15	4.77E-15	0	0	0	0	0	0	-0.84333	-0.84334	-0.84332	-0.84334	-0.84334	-0.84334
11	9.12E-05	6.35E-05	0.000128	4.44E-16	4.44E-16	4.44E-16	0.003946	0.003152	0.00465	1.37E-08	1.1E-08	1.64E-08	5.98E-05	3.38E-05	9.72E-05	0	0	0	5.11E-11	2.54E-11	1.1E-10	6.11E-15	4.44E-15	1.11E-14	1.38E-15	3.55E-16	7.34E-15	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
16	0.008519	0.006167	0.012253	4.44E-16	4.44E-16	4.44E-16	0.06835	0.047875	0.092363	2.19E-08	1.57E-08	3.81E-08	0.176416	0.13123	0.301661	0	0	0	3.02E-07	2.08E-07	5.84E-07	1.33E-14	9.16E-15	2.05E-14	2.95E-08	1.52E-08	8.28E-08	0	0	0	-0.83688	-0.84144	-0.37336	-0.84334	-0.84334	-0.84334
23	0.128883	0.107912	0.170666	4.44E-16	4.44E-16	8.15E-07	0.044655	0.032698	0.056652	2.88E-08	2.46E-08	3.83E-08	40937.43	29041.44	58834.73	0	0	0	0.000134	8.47E-05	0.000228	3.25E-14	2.64E-14	4.03E-14	0.008807	0.004445	0.013762	0	0	0	-0.37336	-0.37336	-0.37335	-0.83944	-0.84332	-0.37336
32	1.319072	1.104681	1.554073	4.44E-16	4.44E-16	2.76E-06	0.70001	0.540775	1.179557	4.38E-08	4.11E-08	4.93E-08	8747.143	4139.811	11252.13	0	0	3.55E-07	0.046022	0.039853	0.071205	1.38E-14	1.38E-14	1.38E-14	27.36224	8.20748	44.46458	0	0	2.02E-20	-0.34639	-0.37059	-0.0983	-0.37336	-0.37336	-0.37336
45	3.462481	3.174789	3.554182	5.82E-07	4.44E-16	1.38E-06	3.656604	3.261996	4.41419	5.57E-06	3.73E-06	8.33E-06	176546.5	137142.7	206909.3	0	0	2.22E-14	0.459727	0.356327	0.526028	3.35E-12	2.24E-12	5E-12	15444.56	13322.64	21564.57	1.01E-20	0	9.59E-20	1.210238	0.485377	1.506581	-0.37336	-0.37336	-0.37336
64	5.921247	5.582315	6.528231	7.74E-05	6.51E-05	0.000133	36.97945	32.90137	39.57803	0.000813	0.000241	0.001663	3307427	2880989	3786355	0.008788	0.005905	0.010037	2.201503	2.107287	2.357505	1.02E-09	6.9E-10	1.33E-09	754804.8	668911.2	1047387	2.58E-13	2.19E-13	3.32E-13	2.91635	2.164063	2.918083	0.409922	-0.34745	0.409922
91	7.63218	7.112498	7.992729	0.000484	0.000358	0.000587	33.75965	30.59396	35.90677	0.00755	0.004345	0.012796	8966358	8435228	10469969	0.405725	0.314642	0.454786	41.85116	36.84607	47.12382	2.23E-05	1.79E-05	3.43E-05	7675482	5582492	10042589	2.24E-09	1.74E-09	3.12E-09	5.626081	4.678043	6.67573	0.411561	0.409933	0.790925
128	12.48756	12.20686	12.7411	0.024202	0.021227	0.037027	95.85325	90.30683	103.2877	0.088417	0.05806	0.139418	61733301	55153841	64448621	355.2085	301.9432	390.5285	81.16928	73.08468	88.01344	0.008867	0.007568	0.009441	3.55E+08	3.28E+08	4.06E+08	0.00524	0.004267	0.00676	14.85917	14.6644	16.16314	1.5065	1.506499	1.5065
181	14.90032	14.72997	15.12464	0.307892	0.208681	0.422061	163.0116	156.9675	168.6027	0.218319	0.176131	0.328731	1.34E+08	1.29E+08	1.43E+08	8448.615	7108.713	10517.93	12662.27	6064.986	20988.63	1.126688	0.784123	2.134679	1.95E+09	1.84E+09	2.16E+09	4.572203	3.338935	10.84745	18.92347	17.95916	20.6182	2.916346	2.916346	2.916489
256	16.91389	16.66151	17.08278	1.29532	1.154075	1.404087	277.2766	268.3385	288.6248	1.094409	0.820719	1.445104	3.39E+08	3.35E+08	3.5E+08	196485.9	163139.5	256235.2	538.7374	505.6563	612.7954	11.66869	9.895626	14.34335	1.15E+10	9.83E+09	1.27E+10	1704.686	840.3208	3747.003	44.36637	44.24936	47.11681	4.640506	4.639543	5.03806
362	18.55217	18.45957	18.65114	3.644307	3.523883	4.304199	468.2546	462.6516	480.5731	6.151896	5.33385	7.079403	6.06E+08	5.9E+08	6.27E+08	2307944	1969187	2961042	37275.91	14834.59	100680.2	33.17801	30.49561	36.63519	4.59E+10	4.21E+10	4.88E+10	576895.2	312341.7	1211836	78.11494	74.91574	81.14991	9.025156	9.025156	9.034916
512	19.40681	19.35113	19.50696	5.882911	5.69299	6.117149	718.3811	700.0523	733.5484	27.82893	24.38129	30.57503	1.22E+09	1.19E+09	1.28E+09	13124554	11835114	14957679	25505.94	12666.91	44301.6	50.55317	47.73446	54.92817	2.65E+11	2.4E+11	2.75E+11	35197274	31749498	47325640	166.2578	163.5285	169.334	14.66325	13.86853	14.66335
724	20.02704	19.95063	20.08075	8.526101	8.272105	8.640496	1125.665	1107.953	1143.393	85.5274	80.61242	89.7774	2.45E+09	2.41E+09	2.48E+09	71679420	66012955	74748770	856604.6	261049.2	2728598	187.6017	170.4999	208.7352	8.16E+11	7.97E+11	8.62E+11	1.87E+09	1.81E+09	2.54E+09	231.8663	226.7026	236.0796	21.55318	21.55308	21.57639
1024	20.33049	20.27603	20.36051	11.07556	10.91363	11.21186	1730.316	1691.731	1758.84	234.526	231.8923	245.0339	4.07E+09	3.98E+09	4.17E+09	2.15E+08	2.07E+08	2.2E+08	2.53E+11	1.02E+11	9.63E+11	1856.831	1517.619	2194.053	2.31E+12	2.24E+12	2.42E+12	6.8E+09	6.06E+09	7.88E+09	379.9798	375.4298	390.5486	34.23498	34.2328	34.34262
1448	20.29426	19.9417	20.56873	13.46341	13.23452	13.59843	2611.819	2579.548	2630.743	517.0039	502.2183	527.9696	6.34E+09	6.26E+09	6.42E+09	6.12E+08	6.02E+08	6.31E+08	1.31E+12	6.25E+11	3.19E+12	4985.487	4054.787	5756.805	6.24E+12	5.94E+12	6.37E+12	5.54E+10	5.16E+10	5.92E+10	607.091	597.0675	615.1003	55.50395	55.50393	58.68798

Table B.9: DE Descriptive Statistics: Exponential to Rastrigin Functions (10,000 Evaluation Tuning)

Dims.	Exponential						Griewank						Inverted Cosine Wave					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	-2	-2	-2	-2	-2	-2
4	-1	-1	-1	-1	-1	-1	1.19E-05	8.77E-07	9.22E-05	0	0	0	-3	-3	-3	-3	-3	-3
6	-1	-1	-1	-1	-1	-1	0.000243	3.25E-05	0.005881	0	0	0.001849	-4.87858	-4.95162	-4.46401	-5	-5	-5
8	-1	-1	-1	-1	-1	-1	0.008669	0.002478	0.01311	5.3E-06	0	0.000196	-6.2519	-6.36478	-5.94746	-7	-7	-6.63778
11	-1	-1	-1	-1	-1	-1	0.04459	0.025763	0.067129	0	0	0	-8.29008	-8.45544	-8.03763	-9.47543	-9.9982	-9.08303
16	-1	-1	-1	-1	-1	-1	0.020369	0.015951	0.028072	0.002484	0	0.009066	-11.7866	-12.1051	-11.4116	-13.2166	-13.7737	-12.8351
23	-1	-1	-1	-1	-1	-1	0.005424	0.00202	0.015402	0	0	0	-16.0801	-16.503	-15.4572	-16.7288	-17.4214	-16.3045
32	-1	-1	-1	-1	-1	-1	0.262569	0.148568	0.375733	1.64E-10	0	0.007396	-22.3864	-22.7797	-21.7244	-25.1711	-25.5636	-24.4973
45	-0.9999	-0.99991	-0.99989	-1	-1	-1	1.176822	1.151535	1.215404	0	0	2.81E-11	-30.0266	-31.1758	-29.3238	-32.0764	-32.7664	-30.894
64	-0.99692	-0.99751	-0.99612	-1	-1	-1	1.810099	1.516068	1.997116	0.030825	9.6E-08	0.154701	-37.576	-38.1049	-36.5795	-47.5404	-48.2059	-46.8438
91	-0.96332	-0.96793	-0.95814	-0.99828	-0.99973	-0.99389	7.88136	6.970066	9.304539	0.16469	0.009751	0.480556	-48.3661	-50.0821	-46.8516	-59.7483	-60.5851	-59.3191
128	-0.87214	-0.89109	-0.85036	-0.98041	-0.98752	-0.96704	28.17086	25.20702	30.93907	1.617482	1.315519	3.474353	-63.6357	-67.0959	-61.7789	-71.664	-73.7807	-69.0725
181	-0.60184	-0.63912	-0.56783	-0.83913	-0.87769	-0.80603	283.0023	275.1373	311.8635	3.862439	2.166153	5.048672	-84.3499	-86.1912	-80.6479	-89.6565	-95.3034	-86.9023
256	-0.15295	-0.17812	-0.13114	-0.50154	-0.52808	-0.48311	323.4248	288.5491	363.7746	133.7105	108.6738	155.6929	-93.846	-96.163	-90.1303	-116.291	-118.939	-113.17
362	-0.00606	-0.00737	-0.00426	-0.09442	-0.10987	-0.08597	971.8076	921.1578	1037.238	270.1036	248.0108	306.251	-124.834	-129.027	-122.242	-136.151	-137.518	-131.366
512	-6.8E-06	-1.1E-05	-4.2E-06	-0.00235	-0.00303	-0.0018	1918.265	1825.864	1993.398	923.2869	869.6737	1028.745	-156.632	-159.503	-149.492	-171.299	-177.612	-168.781
724	-2.9E-26	-5.5E-26	-1.6E-26	-8.3E-08	-1.4E-07	-4.3E-08	1778.029	1636.248	2137.279	1527.946	1460.566	1814.368	-195.387	-199.865	-187.712	-218.434	-224.177	-213.795
1024	-2.4E-45	-6.2E-41	-4.8E-50	-2.9E-16	-7.9E-16	-1.4E-17	2635.644	2349.659	3036.549	2540.61	2324.356	2988.767	-228.4	-238.095	-218.352	-262.982	-270.702	-257.611
1448	-3E-100	-4.1E-70	-1E-101	-2.8E-41	-4.9E-31	-2E-99	4050.118	3710.26	4368.197	3801.705	3432.84	4145.743	-287.046	-297.577	-274.556	-317.469	-325.144	-305.024

Dims.	Levy						Qing						Rastrigin					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evaluations			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	6.73E-14	6.73E-14	6.73E-14	6.73E-14	6.73E-14	6.73E-14	7.88E-10	7.88E-10	6.01E-09	7.88E-10	7.88E-10	6.01E-09	0	0	0	0	0	0
3	9.6E-14	9.6E-14	9.6E-14	9.6E-14	9.6E-14	9.6E-14	6.01E-09	7.92E-10	6.01E-09	6.01E-09	7.92E-10	6.01E-09	0	0	0	0	0	0
4	1.25E-13	1.25E-13	1.25E-13	1.25E-13	1.25E-13	1.25E-13	6.27E-09	4.97E-09	1.27E-08	6.27E-09	4.97E-09	1.27E-08	2.22E-10	1.29E-10	5.54E-10	0	0	0
6	1.82E-13	1.82E-13	1.82E-13	1.82E-13	1.82E-13	1.82E-13	1.78E-08	6.17E-09	1.87E-08	1.78E-08	6.17E-09	1.87E-08	1.91E-05	7.13E-06	3.59E-05	0	0	0
8	9.13E-08	5.09E-08	2.83E-07	2.4E-13	2.4E-13	2.4E-13	1.25E-05	2.39E-06	4.75E-05	1.91E-08	1.07E-08	2.93E-08	4.26E-08	3.12E-08	3.45E-07	0	0	1.85E-11
11	4.14E-09	3.34E-09	6.48E-09	3.26E-13	3.26E-13	3.26E-13	6.57E-05	1.19E-05	0.000133	4.77E-08	4.36E-08	5.4E-08	0.000983	0.000467	0.002026	0	0	0
16	0.00592	0.004414	0.008222	4.69E-13	4.69E-13	4.69E-13	4.598314	3.038681	7.416518	7.96E-08	6.06E-08	9.87E-08	0.995797	0.010213	1.004087	0.994959	0	0.994959
23	0.001113	0.000774	0.001414	6.7E-13	6.7E-13	6.7E-13	178.6239	123.9509	213.4083	1.46E-07	1.31E-07	1.68E-07	5.73162	4.040917	7.370884	5.363313	3.577168	6.87604
32	0.002269	0.001892	0.003676	9.29E-13	9.29E-13	1.36E-12	1441.93	1245.194	1622.075	3.46E-07	3.18E-07	3.71E-07	11.77285	10.29981	14.31112	1.492439	0.994959	2.238658
45	0.77419	0.635622	0.941803	1.3E-12	1.3E-12	1.3E-12	23096.14	16439.72	28582.05	7.77E-07	7.09E-07	9.32E-07	24.2471	18.44055	28.9561	10.45596	9.45214	14.92562
64	1.418146	1.02208	1.710503	0.361732	0.008412	0.544757	3860653	3136890	4688911	0.00027	0.000134	0.000613	89.00316	79.42115	93.41683	62.24189	56.10632	68.12304
91	13.23286	12.43318	16.24188	2.77E-07	1.56E-07	0.022394	8.33E+08	5.59E+08	1.01E+09	77.57168	21.00171	890.2737	150.7925	145.1615	166.1206	93.70796	90.22636	103.528
128	20.63335	15.68332	23.31718	4.590593	2.916736	6.369882	3.68E+09	3.06E+09	4.58E+09	366234.9	74373.34	8767433	301.0954	280.9238	309.8029	213.9816	192.7687	223.9138
181	42.35533	37.97211	45.14371	27.21776	24.29236	31.27041	4.48E+09	3.75E+09	5.8E+09	8.48E+08	3.85E+08	1.3E+09	717.0388	687.9418	740.4421	470.841	450.6722	487.9534
256	154.9248	141.1383	181.1107	60.23716	56.87189	62.96071	4.2E+10	3.72E+10	4.73E+10	1.59E+10	1.29E+10	1.88E+10	1018.337	997.5908	1077.392	871.5391	840.0018	903.7898
362	299.7989	271.8841	315.1003	211.7989	184.8302	226.7792	5.82E+10	4.28E+10	7.04E+10	3.84E+10	2.96E+10	4.38E+10	1887.931	1842.957	1941.415	1349.07	1329.907	1400.848
512	703.1143	680.4103	736.288	448.4534	422.2364	457.7364	8.43E+10	6.17E+10	1.06E+11	7.53E+10	4.9E+10	9.04E+10	3058.222	2997.631	3138.258	2629.748	2520.628	2711.992
724	1410.953	1207.821	1485.37	1364.22	1151.95	1469.986	1.49E+11	1.08E+11	1.83E+11	9.49E+10	8.35E+10	1.19E+11	5851.846	5756.121	5910.545	5261.09	5154.933	5319.736
1024	2486.887	2343.97	2596.555	2191.977	1999.703	2380.496	2.97E+11	2.12E+11	3.58E+11	2.01E+11	1.71E+11	2.67E+11	9582.314	9440.611	9636.206	6716.426	6593.204	6851.702
1448	3309.338	2948.239	3522.528	3284.785	2793.194	3484.318	3.85E+11	2.75E+11	4.27E+11	3.01E+11	2.46E+11	3.57E+11	14526.69	14386.5	14750.05	13637.27	13540.17	13883.86

Table B.10: DE Descriptive Statistics: Exponential to Rastrigin Functions (50,000 Evaluation Tuning)

Dims.	Exponential						Griewank						Inverted Cosine Wave					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	-1	-1	-1	-1	-1	-1	0.018157	0.00834	0.022488	0	0	4.8E-10	-0.99689	-0.99812	-0.99439	-1	-1	-1
3	-1	-1	-1	-1	-1	-1	0.047146	0.035982	0.056597	0	0	0	-1.82159	-1.85572	-1.69437	-2	-2	-2
4	-1	-1	-1	-1	-1	-1	0.0225	0.015661	0.029664	0	0	0	-2.47075	-2.54995	-2.44218	-3	-3	-3
6	-1	-1	-1	-1	-1	-1	0.061313	0.048165	0.073944	0	0	0	-4.04828	-4.2073	-3.9368	-5	-5	-5
8	-1	-1	-1	-1	-1	-1	0.29497	0.246141	0.368937	0	0	0	-5.81237	-6.08603	-5.56698	-7	-7	-6.99294
11	-1	-1	-1	-1	-1	-1	0.253006	0.207336	0.325433	3.39E-09	1.2E-10	3.67E-08	-8.10887	-8.25958	-7.68376	-9.91437	-9.98738	-9.81713
16	-1	-1	-1	-1	-1	-1	0.954948	0.914505	0.982959	0	0	0	-11.7268	-11.9932	-11.3982	-13.7491	-13.8839	-13.1093
23	-1	-1	-0.99999	-1	-1	-1	1.287455	1.229301	1.336958	0	0	2.53E-09	-15.8918	-16.1606	-15.7647	-18.4282	-18.9398	-17.9814
32	-0.99976	-0.99981	-0.99971	-1	-1	-1	1.276161	1.233292	1.306346	0	0	0	-21.5396	-21.6199	-21.3718	-25.189	-25.743	-24.7609
45	-0.99434	-0.99481	-0.99276	-1	-1	-1	3.428493	3.190735	4.059139	3.66E-08	2.24E-08	4.95E-08	-26.4796	-27.1199	-24.9777	-34.6539	-35.4418	-34.1639
64	-0.92074	-0.93362	-0.90655	-1	-1	-1	10.13586	8.765429	10.89504	1.15E-06	6.97E-07	1.62E-06	-24.5873	-25.3041	-23.2983	-48.7242	-49.3258	-47.9404
91	-0.8385	-0.84771	-0.82651	-1	-1	-1	46.7565	42.15352	58.38615	0.00064	0.000555	0.000809	-23.7336	-24.3899	-22.534	-66.7558	-68.0854	-66.3663
128	-0.37084	-0.40506	-0.36391	-0.99999	-0.99999	-0.99999	138.0393	120.1128	147.8723	0.018089	0.016979	0.024204	-35.4893	-36.755	-32.839	-92.0621	-93.0575	-90.4366
181	-0.08322	-0.10291	-0.07495	-0.99983	-0.99988	-0.99981	334.5262	310.318	355.4031	0.277776	0.209614	0.350359	-29.1292	-31.2766	-27.6581	-124.239	-125.884	-122.401
256	-0.0024	-0.00347	-0.00196	-0.99706	-0.99776	-0.99543	1062.829	998.8866	1132.952	1.688642	1.559251	1.800497	-37.2632	-38.2065	-35.2277	-167.809	-172.196	-162.747
362	-3.6E-06	-5.9E-06	-2.7E-06	-0.95643	-0.96775	-0.95163	1886.509	1809.642	1927.169	9.347444	6.604107	13.40585	-42.1074	-45.6672	-39.8035	-217.067	-220.475	-213.669
512	-4.3E-11	-8E-11	-2.8E-11	-0.75063	-0.79411	-0.72676	4885.994	4763.554	4960.786	63.79634	59.02257	69.73019	-47.0432	-47.9878	-44.0397	-270.084	-274.791	-265.063
724	-1.8E-18	-3.8E-18	-6.4E-19	-0.3339	-0.39114	-0.28853	7935.347	7800.999	8144.238	205.4639	194.6582	219.36	-119.394	-121.433	-116.884	-345.593	-354.827	-340.984
1024	-1.3E-32	-6.1E-32	-1.8E-33	-0.01691	-0.02337	-0.01127	14094.5	13929.34	14332.4	744.1137	679.6065	787.0687	-79.8653	-84.5258	-77.1361	-442.847	-448.035	-437.477
1448	-1E-99	-4.2E-66	-3E-101	-1.3E-35	-9.6E-31	-1.8E-56	22201.83	21845.91	22286.14	2091.85	2008.576	2189.339	-153.335	-158.595	-148.406	-549.859	-557.241	-542.711

Dims.	Levy						Qing						Rastrigin					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evaluations			10,000 Evaluations			50,000 Evaluations		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	6.73E-14	6.73E-14	6.73E-14	6.73E-14	6.73E-14	6.73E-14	7.88E-10	7.88E-10	6.01E-09	7.88E-10	7.88E-10	6.01E-09	0	0	0	0	0	0
3	9.6E-14	9.6E-14	9.6E-14	9.6E-14	9.6E-14	9.6E-14	7.92E-10	7.92E-10	6.01E-09	7.92E-10	7.92E-10	6.01E-09	0	0	0	0	0	0
4	1.25E-13	1.25E-13	1.25E-13	1.25E-13	1.25E-13	1.25E-13	8.84E-09	1.06E-09	1.27E-08	6.27E-09	1.06E-09	1.14E-08	1.193441	0.864411	1.664568	0	0	0
6	1.82E-13	1.82E-13	1.82E-13	1.82E-13	1.82E-13	1.82E-13	1.75E-07	2.38E-08	3.28E-06	1.56E-08	1.13E-08	2E-08	7.07E-07	1.43E-07	4.2E-06	0	0	0
8	7.23E-13	6.7E-13	1.1E-12	2.4E-13	2.4E-13	3.87E-13	0.002911	0.000392	0.017896	2.66E-08	1.98E-08	3.14E-08	8.563525	6.328055	9.08142	0	0	0
11	1.08E-09	6.92E-10	1.82E-09	3.26E-13	3.26E-13	7.56E-13	1.106295	0.349532	2.675092	5.45E-08	4.68E-08	6.08E-08	8.941958	8.101795	9.994735	0	0	0
16	1.26E-05	7.89E-06	2.45E-05	1.33E-12	9E-13	1.57E-12	75.18675	43.75691	115.8964	7.17E-08	6.31E-08	1.03E-07	17.22094	16.3651	19.93314	0	0	0
23	0.149896	0.133947	0.193405	6.7E-13	6.7E-13	6.7E-13	2099.954	1795.911	2679.267	1.42E-07	1.24E-07	1.48E-07	43.09229	39.37109	48.76566	8.81E-07	4.31E-07	1.34E-06
32	1.072689	1.00212	1.294477	9.29E-13	9.29E-13	2.32E-12	3219.633	2614.708	3599.413	5.72E-07	4.82E-07	8.59E-07	92.80695	85.31121	99.70355	2.72E-05	1.43E-05	4.12E-05
45	10.79421	9.781712	12.46324	1.27E-06	8.99E-07	1.44E-06	189387.3	102219	661711.8	9.2E-07	7.29E-07	1.2E-06	142.5422	136.389	164.7868	0.000203	0.000144	0.000378
64	15.949	14.26011	19.46165	5.8E-06	3.62E-06	6.46E-06	5.5E+08	3.95E+08	6.63E+08	512.1852	346.3825	619.2078	264.5373	251.3908	275.6569	0.297849	0.013825	1.010458
91	54.38697	49.88651	58.61587	0.000292	0.000235	0.022806	1.57E+09	1.28E+09	2.34E+09	1286.472	821.0767	1677.555	426.4507	415.1866	438.4688	5.74724	4.380086	7.497818
128	83.4717	72.66248	89.24837	0.000856	0.000577	0.18642	3.07E+09	2.77E+09	3.38E+09	1525.095	1380.237	1899.91	653.3724	635.5262	678.6919	35.87867	32.21682	40.99605
181	354.5275	324.1114	381.7005	0.569508	0.526956	0.629241	1.07E+11	9.83E+10	1.27E+11	1355716	1182834	1620374	1450.219	1418.647	1472.69	132.278	125.6887	141.5734
256	463.9195	439.345	487.753	2.834508	2.098476	4.44958	3.6E+11	3.11E+11	3.79E+11	82291178	65522678	1.08E+08	2350.202	2293.291	2372.729	259.6702	248.4986	268.0189
362	1365.904	1314.333	1448.114	19.37969	17.34824	20.57454	7.54E+11	7.14E+11	7.82E+11	1.89E+08	1.75E+08	2.38E+08	3603.705	3524.212	3659.031	491.5098	480.9751	520.3404
512	1974.743	1905.604	2079.158	45.01578	39.80767	48.4963	1.1E+12	1.03E+12	1.15E+12	5E+09	3.88E+09	6.5E+09	5425.836	5290.532	5445.583	935.1884	915.4254	972.6698
724	3596.682	3565.873	3737.535	145.7448	133.5139	153.3308	2.72E+12	2.6E+12	2.82E+12	1.35E+10	1.18E+10	1.5E+10	8122.839	7981.621	8233.108	1762.28	1692.221	1795.65
1024	5902.98	5815.776	6039.068	359.3153	345.3354	373.3283	4.47E+12	4.35E+12	4.57E+12	7.19E+10	6.78E+10	7.67E+10	12404.02	12267.66	12441.19	3286.272	3234.417	3343.999
1448	9639.332	9527.821	9726.023	882.7729	864.1913	963.3242	1.6E+13	8.98E+12	1.64E+13	2.01E+11	1.72E+11	2.54E+11	18759.55	18577.14	18946.33	6049.759	5957.976	6220.545

Table B.11: DE Descriptive Statistics: Rosenbrock to Sum Squares Functions (10,000 Evaluation Tuning)

Dims.	Rosenbrock						Schwefel						Sphere						Sum of Different Powers						Sum Squares					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	7.03E-11	7.03E-11	2.6E-10	7.03E-11	7.03E-11	7.03E-11	5.46E-06	5.46E-06	5.46E-06	5.46E-06	5.46E-06	5.46E-06	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1.41E-10	1.41E-10	2.18E-10	1.41E-10	1.41E-10	1.41E-10	8.18E-06	8.18E-06	8.18E-06	8.18E-06	8.18E-06	8.18E-06	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0.000101	2.78E-05	0.005105	2.11E-10	2.11E-10	2.11E-10	1.09E-05	1.09E-05	1.09E-05	1.09E-05	1.09E-05	1.09E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0.040868	0.00436	0.357048	3.52E-10	3.52E-10	3.52E-10	0.00023	0.000106	0.001174	1.64E-05	1.64E-05	1.64E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	2.456209	0.870801	3.005482	1.7E-09	6.17E-10	1.54E-08	0.016149	0.005459	0.05374	2.18E-05	2.18E-05	2.18E-05	0	0	0	0	0	0	6.71E-46	2.58E-49	1.66E-44	0	0	0	0	0	0	0	0	0
11	8.707168	7.712501	9.392098	6.56E-05	4.87E-06	0.000602	6.991426	4.064602	14.54995	3E-05	3E-05	3E-05	1.19E-10	5.14E-11	1.87E-10	0	0	0	9.97E-28	3.52E-30	3.13E-26	0	0	0	1.98E-09	9.79E-10	4.54E-09	0	0	0
16	13.985	13.79116	31.34028	12.27542	11.62305	12.82318	7.37141	5.82143	11.57975	0.329141	0.127622	1.199587	5.73E-07	3.37E-07	8.13E-07	0	0	0	1.99E-19	4.87E-20	9.81E-19	0	0	0	4.22E-06	3.11E-06	7.8E-06	0	0	0
23	68.69396	53.70721	101.7027	9.522948	5.693904	45.37057	147.6122	63.27091	175.7895	120.2237	6.52235	130.2924	2.23E-05	9.84E-06	3.95E-05	0	0	0	6.68E-14	5.59E-15	4.35E-13	1.07E-48	1.43E-59	3.4E-35	7.08E-05	4.68E-05	9.58E-05	0	0	0
32	268.244	196.1137	407.8521	28.36323	26.71508	84.42636	564.398	411.3592	697.2941	496.0373	336.339	616.2293	0.003165	0.001915	0.003943	0	0	0	5.21E-10	1.3E-10	1.8E-09	1.81E-20	2.69E-30	7.52E-18	0.031543	0.026287	0.056496	0	0	0
45	810.4601	705.368	1244.522	89.0038	53.88886	93.89932	975.861	774.4187	1178.504	769.015	519.5231	942.7627	0.290661	0.216951	0.359215	0	0	0	8.59E-14	8.28E-15	2.98E-13	1.6E-155	5.6E-168	9.7E-146	1.455255	1.081216	2.084456	0	0	1.63E-11
64	1637.341	1011.375	3116.311	194.4261	114.4129	426.3495	2868.453	2482.044	3159.961	1744.526	1499.943	1976.206	0.112263	0.061357	0.150962	1.89E-06	1.67E-10	0.007869	3.02E-10	8.66E-11	8.07E-10	7.49E-75	1.14E-78	2.68E-67	146.5146	124.972	178.1716	1.83E-08	9.36E-09	2.44E-08
91	23722.84	19608	29575.7	320.1808	251.9121	361.5818	5377.775	5062.894	6110.411	3494.543	3191.17	4452.297	1.616892	1.38562	2.014202	0.016629	0.001576	0.144937	2.41E-08	9.77E-09	3.75E-08	1.6E-52	6.99E-60	3.09E-45	314.0748	244.8301	358.6948	0.17644	1.92E-05	2.415407
128	141169.3	124841	170504	1128.113	663.4637	3303.212	9579.15	9161.52	10491.98	8367.608	7655.335	8755.797	10.68177	9.308819	12.00975	0.086541	0.002637	0.268577	2.02E-08	9.89E-09	7.13E-08	1.15E-27	2.88E-32	1.65E-25	1583.239	1342.449	2141.441	327.7819	84.09737	688.2525
181	618780.2	539665	803349.2	26047.18	11957.45	40651.87	19841.12	18957.67	20771.86	16257.66	15628.89	16942.02	29.67687	25.45165	30.81402	4.92433	2.977451	7.322008	1.37E-06	8.57E-07	2.8E-06	4.03E-28	1.02E-30	2.68E-23	11740.58	10757.32	13490.67	628.8542	377.3879	955.1992
256	2271812	2022131	2894830	1534298	1194979	1829678	34715.85	33610.74	36399.49	27985.81	26895.81	29874.75	95.10473	86.35557	102.0642	25.52988	18.86764	26.58695	2.25E-06	5.86E-07	3.77E-06	6.35E-17	2.49E-19	9.95E-12	36855.33	34622.71	41460.24	9240.593	7608.129	11930.46
362	8061716	6925571	8835061	1696983	1495704	2508936	55027.57	54148.16	56226.25	46541.93	45717.24	48222.99	229.7541	219.0624	240.2886	114.2006	103.5976	118.8597	1.24E-05	3.99E-06	3.46E-05	1.65E-17	6.96E-20	4.73E-12	148033.9	138052.2	158562.3	45869.48	36501.59	51620.37
512	6543852	5157709	7580294	6163065	5015800	7367129	93508.04	91674.82	94651.15	74102.21	73189.85	75190.72	343.9897	327.9946	403.4633	322.5803	281.7155	362.0952	0.000133	5.78E-05	0.000441	3.64E-08	1.25E-08	4.91E-06	435153.7	418324.2	468083.8	206378.9	181767.7	230246.7
724	16853945	12990073	17883707	12660046	11542627	14863147	150079.2	147902.5	151133.4	131151.2	129455	132266.9	566.4851	527.2354	602.2581	465.0025	432.6259	512.703	0.000462	0.000112	0.000946	1.58E-07	2.93E-09	1.47E-05	798246	740025.1	858980.5	611455.3	508904.7	652999
1024	21258684	17598358	25861611	19583448	14902624	23207412	245487.1	241302.5	246744.1	213979.1	211518	215238.6	797.5802	663.927	966.3771	770.1716	651.0026	942.1903	0.000942	0.000466	0.001777	0.000251	5.11E-05	0.000951	1489238	1304858	1642000	1457478	1286098	1618305
1448	27521677	23039563	35400496	26801964	22569575	32127314	378972.3	375343.9	381345.7	336841.9	334306.2	338826.5	1249.234	1123.483	1344.122	1096.817	1027.616	1237.067	0.000252	9.29E-05	0.0015	2.38E-05	1.48E-06	0.000773	3308511	2772980	3459176	3192047	2675716	3331167

Table B.12: DE Descriptive Statistics: Rosenbrock to Sum Squares Functions (50,000 Evaluation Tuning)

Dims.	Rosenbrock						Schwefel						Sphere						Sum of Different Powers						Sum Squares					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.015403	0.003736	0.024362	7.03E-11	7.03E-11	7.03E-11	1.813285	1.357363	6.187708	5.46E-06	5.46E-06	5.46E-06	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0.310243	0.195315	0.452924	1.41E-10	1.41E-10	1.41E-10	1.459245	0.822485	5.218958	8.18E-06	8.18E-06	8.18E-06	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	2.184297	1.664143	2.687314	2.11E-10	2.11E-10	2.11E-10	187.2959	124.4817	239.1911	1.09E-05	1.09E-05	1.09E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1.669465	1.287821	2.014755	3.52E-10	3.52E-10	3.52E-10	481.2476	386.9396	571.0262	1.64E-05	1.64E-05	1.64E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	5.076281	4.444576	5.596454	2.38E-09	8.24E-10	5.8E-09	144.3848	107.6022	242.9342	2.18E-05	2.18E-05	2.18E-05	0	0	0	0	0	0	8.14E-45	2.92E-49	4.68E-41	0	0	0	0	0	0	0	0	0
11	15.28556	12.64053	21.03255	0.000141	1.25E-05	0.002405	348.7877	258.5978	420.7068	3E-05	3E-05	3E-05	1.21E-10	6.5E-11	3.39E-10	0	0	0	3.79E-28	4.08E-30	8.61E-24	0	0	0	2.08E-09	1.05E-09	4.14E-09	0	0	0
16	140.9214	105.2181	244.001	6.875618	6.066341	8.228671	1963.055	1860.406	2133.701	6.26E-05	5.35E-05	6.89E-05	5.7E-07	3.27E-07	8.41E-07	0	0	0	5.74E-19	1.81E-19	2.15E-17	0	0	0	1.77E-05	1.29E-05	2.68E-05	0	0	0
23	749.2223	583.1085	982.31	13.36915	9.586061	17.79955	3204.456	2989.794	3381.957	0.006348	0.002051	0.009022	0.000171	0.000133	0.000381	0	0	0	1.36E-13	4.64E-14	1.23E-12	6.69E-48	1.87E-60	6.14E-37	0.007729	0.005029	0.010637	0	0	0
32	2186.344	1423.827	2828.633	28.0669	27.79454	29.29997	4064.511	3762.633	4205.232	0.152738	0.11453	0.226757	0.011538	0.008783	0.015113	0	0	3.73E-13	7.79E-10	2.06E-10	2.5E-09	2.33E-25	1.4E-31	1.51E-20	0.594259	0.420693	0.783176	0	0	8.17E-12
45	23347.66	16311.9	30225.81	41.91087	41.40306	82.68109	6532.168	6247.227	6726.524	9.898813	7.958302	12.83077	0.271261	0.230006	0.366035	3.73E-13	0	5.36E-13	1.44E-06	5.46E-07	2.82E-06	1.81E-19	1.25E-21	9.42E-16	22.08968	18.38354	27.91211	5.15E-12	0	1.63E-10
64	165229.2	147810.1	189146.4	68.91801	61.61967	102.7552	10760.74	10449.13	11033	150.3439	57.67432	195.5023	4.463842	3.669238	5.12898	4E-08	2.64E-08	7.47E-08	0.004089	0.003132	0.008134	9.95E-17	4.41E-18	1.04E-12	81.15916	66.36135	102.4442	1.88E-09	1.3E-09	2.43E-08
91	792375.1	699677.7	862957.1	283.2584	191.0206	365.7509	16985.05	16578.83	17540.24	298.2683	252.6472	430.5118	5.73513	4.891603	6.416966	2.3E-08	1.38E-08	4.2E-08	0.477894	0.265704	1.016302	8.86E-10	2.94E-11	7.36E-08	2851.94	2519.025	3148.69	0.004213	0.003439	0.004895
128	208552	150195.2	281114	582.8499	542.2227	672.6719	26925.83	26327.09	27328.88	1055.229	951.5723	1175.651	44.69758	40.83973	50.31435	0.000453	0.000347	0.001183	0.000651	0.000402	0.001058	3.7E-22	7.45E-23	9.69E-22	12008.83	11051.5	12502.97	0.108511	0.095436	0.12605
181	11825573	10591675	13154395	4116.73	3717.618	4431.026	39391.12	37891.17	40220.11	2852.007	2439.987	3136.02	123.2996	118.0473	133.4084	0.006169	0.005267	0.006744	1.55E-06	5.88E-07	7.14E-06	3.54E-10	1.22E-19	2.28E-07	37164.85	36012.06	40099.22	3.046074	2.535948	3.414729
256	32469950	29555249	33856227	24109.45	21040.12	27178.34	61636.56	60863.58	63092.19	6656.639	6291.229	7181.458	256.1627	243.6704	263.6844	0.164293	0.081556	0.341734	0.011744	0.007511	0.015425	1.89E-18	4.12E-21	2.54E-16	130550.4	127219.8	136489.9	118.9054	105.0038	135.7545
362	38902806	36570567	39946227	71291.7	53984.84	92349.5	92910.64	91979.26	94424.76	17475.92	16698.55	18246.47	602.5422	587.0288	622.562	1.859794	1.294324	2.68793	0.099997	0.05083	0.130526	5.05E-11	3.2E-11	9.55E-11	391239.5	367426.9	421725.2	1233.873	1009.502	1773.971
512	1.15E+08	1.07E+08	1.18E+08	262174.5	227810	305380.1	138268.5	136825.4	138760	31559.62	30207.81	33531.9	1515.856	1500.465	1533.361	26.07264	25.10232	28.39288	0.512876	0.42054	0.791741	1.29E-18	1.02E-20	1.81E-14	982505.3	963639.3	1020381	10827.97	9400.731	13876.4
724	2.32E+08	2.26E+08	2.41E+08	2345128	2269978	2494177	207642.5	205719.7	208934.7	59774.58	57274.47	61519.57	2216.013	2172.692	2264.692	55.06136	49.62393	63.83272	0.281815	0.219653	0.365045	1.43E-11	4.62E-12	2.88E-11	2932698	2892244	2988167	75824.09	73571.43	81142.46
1024	3.5E+08	3.41E+08	3.54E+08	5956274	5590188	6274815	301881.3	298963.6	304519.2	110411.1	109671.6	112289	3631.722	3568.287	3705.62	206.3734	202.4387	221.8445	0.506519	0.382467	0.638412	6.36E-12	3.17E-12	3.74E-11	6866688	6739373	7002178	345237.1	335301.9	364371.6
1448	2.31E+08	1.1E+08	4.51E+08	17471967	13489421	20031217	453669.6	451123.2	455919.7	193341.8	190787.7	195843.2	6481.652	6418.792	6562.737	594.401	564.8506	612.5538	0.756795	0.467216	0.994036	6.63E-11	4.41E-12	1.32E-09	17133204	16908697	17414421	1632998	1552798	1664034

Table B.13: GA Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (10,000 Evaluation Tuning)

Dims.	Ackley						Alpine1					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.030625	0.02134	0.048796	0.004854	0.00213	0.008411	0.000183	0.000111	0.000274	2.95E-05	1.66E-05	4.38E-05
3	0.050697	0.030754	0.094777	0.007443	0.004112	0.010913	0.000519	0.000285	0.000736	9.17E-05	7E-05	0.000117
4	0.098951	0.0701	0.112759	0.014938	0.00832	0.020206	0.000668	0.000492	0.001043	0.000145	9.46E-05	0.00019
6	0.136925	0.104118	0.172564	0.019098	0.014434	0.025414	0.00142	0.000964	0.001991	0.000385	0.000304	0.000457
8	0.217171	0.131452	0.265463	0.029378	0.022806	0.043376	0.003049	0.002228	0.00393	0.000657	0.000371	0.000742
11	0.411814	0.288636	0.517921	0.047024	0.038583	0.06679	0.006329	0.00546	0.007797	0.001209	0.001009	0.001377
16	0.587448	0.525757	0.701649	0.076985	0.063175	0.093917	0.014065	0.012167	0.014944	0.002861	0.002456	0.0031
23	1.055797	0.777852	1.175454	0.093774	0.075965	0.11325	0.031732	0.023109	0.038704	0.005995	0.005121	0.006566
32	1.608948	1.494764	1.916824	0.179105	0.149174	0.203543	0.057708	0.046249	0.07812	0.010748	0.009101	0.011685
45	2.07348	1.999747	2.197925	0.260413	0.209581	0.29866	0.165212	0.14384	0.201034	0.019651	0.018455	0.022213
64	2.878157	2.772012	2.980829	0.469311	0.431326	0.517	0.463495	0.423533	0.488713	0.038583	0.036593	0.045369
91	3.548475	3.377801	3.603201	0.708971	0.662829	0.758559	1.269367	1.175762	1.513999	0.087787	0.081225	0.092947
128	4.191178	4.134509	4.283962	1.11826	1.04634	1.24927	3.570681	3.24542	3.896914	0.200567	0.180205	0.216261
181	5.288783	5.142259	5.488774	1.786896	1.682013	1.828199	7.549899	7.092002	8.06249	0.444137	0.42113	0.475113
256	6.813491	6.695242	6.966278	2.41599	2.359994	2.484793	18.96913	18.04702	19.51977	1.091464	1.039234	1.162449
362	8.319806	8.223267	8.573507	3.047881	2.965164	3.104193	40.8874	39.9428	44.12951	3.028724	2.970316	3.159534
512	10.54503	10.29956	10.67292	3.778707	3.663879	3.847343	95.87863	93.87564	97.37565	8.310024	7.996121	9.087178
724	12.41354	12.32848	12.57445	4.655424	4.610316	4.72154	207.9493	201.7122	212.1673	22.09631	21.23791	22.70712
1024	14.68463	14.56541	14.93143	5.907743	5.850846	5.971002	425.0669	416.2276	436.0373	53.42238	51.62842	55.15626
1448	16.53034	16.40098	16.60274	7.247802	7.191746	7.402726	853.2582	847.1377	867.5667	128.6458	125.5796	129.9065

Dims.	Bent Cigar						Brown					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	1.455108	0.084051	6.865806	0.066783	0.011835	0.100647	7.8E-07	2.25E-07	1.52E-06	6.43E-08	9.62E-09	1.22E-07
3	4.405833	1.397609	11.14767	0.395786	0.081426	0.608637	3.63E-06	1.01E-06	6.91E-06	1.14E-07	4.36E-08	2E-07
4	13.01865	5.528133	30.34544	0.66671	0.321224	1.480726	1.02E-05	4.3E-06	2.61E-05	2.68E-07	8.16E-08	1.1E-06
6	102.4498	31.72465	150.212	1.306179	0.820599	3.678872	4.46E-05	2.28E-05	8.35E-05	1.07E-06	6.76E-07	2.19E-06
8	163.4887	127.0506	216.4767	9.268722	5.45208	12.29335	0.00012	7.98E-05	0.000172	2.56E-06	2.22E-06	4.58E-06
11	462.6106	267.5335	719.3295	33.24167	24.58597	39.77181	0.00029	0.000224	0.000425	1.36E-05	7.1E-06	2.77E-05
16	1762.026	1283.565	2921.787	93.91426	57.89705	130.4727	0.001178	0.000943	0.001779	3.16E-05	2.3E-05	5.07E-05
23	5265.92	4363.629	6537.438	253.8001	182.062	333.5913	0.003234	0.002715	0.00441	0.000114	9.9E-05	0.000183
32	21840.78	20167.21	32254.07	686.6847	525.1698	1182.322	0.010824	0.008412	0.012493	0.000342	0.000278	0.000417
45	61009.09	46579.32	70568.98	1731.907	1526.688	2249.767	0.028066	0.023642	0.03181	0.001063	0.000851	0.001566
64	185555.1	139971.3	222434.6	5491.62	4402.015	6942.244	0.093538	0.084941	0.110954	0.003075	0.002581	0.003687
91	466821	441607.7	510570.1	16261.41	14337.81	20652.96	0.320175	0.299174	0.370447	0.010496	0.008767	0.011979
128	1369388	1244723	1619134	47546.6	43163.06	57604.77	0.927896	0.811582	1.065958	0.03036	0.026558	0.035097
181	3868122	3633342	4133712	155117.6	139477.8	166801	2.18861	1.976537	2.450227	0.076799	0.070448	0.090939
256	11967466	11407993	12954944	440926.7	410886.6	479737.4	7.01717	6.203787	7.645007	0.244979	0.225746	0.269875
362	32564012	30935586	34958782	1392872	1283325	1467844	16.79576	15.45013	17.47086	0.651041	0.629684	0.694745
512	85739546	82062492	89995208	3664632	3548752	3785331	51.2075	48.17084	53.50009	2.017099	1.973717	2.155459
724	2.51E+08	2.45E+08	2.64E+08	11883495	11462737	12244167	133.9673	129.1173	142.4827	5.697502	5.427938	5.917319
1024	5.74E+08	5.41E+08	5.95E+08	30209281	29435360	31089693	431.8814	397.4127	458.5375	20.23366	19.46626	21.2205
1448	1.4E+09	1.37E+09	1.44E+09	83554180	80468617	86568106	2227.482	1784.312	8893.646	53.82518	53.19298	57.70895

Dims.	Chung-Reynolds						Deflected Corrugated Spring					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	5.86E-07	1.1E-07	3.77E-06	9.17E-10	4.8E-11	1.29E-08	-0.99988	-0.99995	-0.99973	-1	-1	-0.99999
3	2.66E-05	7.7E-07	0.000115	4.57E-09	2.04E-09	2.44E-08	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
4	0.000155	2.56E-05	0.000498	2.99E-07	4.95E-08	7.65E-07	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
6	0.001832	0.000262	0.005313	1.4E-06	4.97E-07	3.41E-06	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
8	0.009938	0.003992	0.041197	1.59E-05	7.36E-06	3.84E-05	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
11	0.056998	0.020502	0.129959	0.000128	4.05E-05	0.000345	-0.37336	-0.8418	-0.37336	-0.37336	-0.84334	-0.37336
16	0.921794	0.438701	1.739628	0.00224	0.000974	0.003289	-0.37336	-0.37336	-0.37336	-0.37336	-0.37336	-0.37336
23	8.791409	4.635247	15.77236	0.011485	0.007021	0.016511	0.409923	0.409922	0.409923	0.409922	0.409922	0.409922
32	78.38445	45.98408	142.6902	0.087278	0.049992	0.188565	1.506499	0.409923	1.5065	0.409922	0.409922	0.409923
45	605.882	546.1392	959.424	0.726539	0.548521	1.147391	2.916346	1.5065	2.916348	1.506499	1.506499	2.916346
64	5590.527	3777.397	7930.235	6.79333	4.224004	9.267034	4.639431	4.639431	4.639825	4.639431	2.916346	4.639431
91	52206.73	44726.39	69690.12	55.04989	43.30093	90.61441	11.68769	9.025156	11.68769	9.025156	6.675717	9.690789
128	371114.4	319127.9	526664.9	570.2488	441.1845	749.01	16.3075	14.66325	17.95174	14.66325	11.68769	14.66325
181	2468013	2237452	2743605	4151.103	3401.943	4528.446	25.4671	25.4671	29.69368	25.4671	21.55307	29.69368
256	27882819	21917722	31290314	39463.67	32067.69	45414.32	39.08356	34.23259	39.08356	39.08356	34.23259	39.08356
362	1.92E+08	1.67E+08	2.09E+08	285558.8	240036.6	333466.2	39.08356	39.08356	44.24621	39.08356	39.08356	44.24621
512	1.33E+09	1.23E+09	1.46E+09	2242288	2095247	2397754	44.24621	44.24621	49.71999	44.24621	44.24621	49.71999
724	8.44E+09	7.88E+09	9.02E+09	18421458	17762825	19719493	49.71999	48.37423	55.50393	49.71999	44.24621	49.71999
1024	5.29E+10	5.03E+10	5.69E+10	1.48E+08	1.4E+08	1.55E+08	58.42589	55.65446	60.47996	49.71999	49.71999	55.50393
1448	3.28E+11	3.15E+11	3.46E+11	1.23E+09	1.12E+09	1.35E+09	152.4904	149.1641	156.7647	55.50393	55.50393	55.50393

Table B.14: GA Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (50,000 Evaluation Tuning)

Dims.	Ackley						Alpine1					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.048594	0.020672	0.070232	0.006803	0.003889	0.010707	0.000192	0.000116	0.000318	2.79E-05	1.98E-05	4.44E-05
3	0.068618	0.03281	0.079069	0.011092	0.007928	0.018045	0.00046	0.000331	0.000704	7.2E-05	4.95E-05	0.000116
4	0.091834	0.065255	0.126914	0.01313	0.008728	0.01783	0.000758	0.000506	0.00088	0.000162	0.000126	0.00021
6	0.131386	0.070424	0.200767	0.022113	0.013403	0.02688	0.001843	0.001261	0.002438	0.000334	0.000229	0.000408
8	0.19305	0.122196	0.3304	0.025563	0.018551	0.036145	0.003593	0.00287	0.003938	0.00068	0.000482	0.000807
11	0.369623	0.275395	0.493355	0.046555	0.038238	0.060799	0.00655	0.00533	0.008154	0.001178	0.000927	0.001558
16	0.694868	0.593491	0.911209	0.075302	0.065206	0.092934	0.013074	0.01151	0.014525	0.002591	0.002179	0.002972
23	1.123802	0.849387	1.240294	0.115505	0.091452	0.126272	0.025872	0.020289	0.033963	0.005067	0.004075	0.005771
32	1.601161	1.452441	1.891587	0.193983	0.155138	0.216333	0.06141	0.051048	0.069221	0.010507	0.009368	0.011402
45	2.097591	2.011312	2.32716	0.278244	0.231075	0.307026	0.16402	0.137398	0.204293	0.020516	0.019037	0.022131
64	3.01677	2.826739	3.151605	0.497861	0.43007	0.568162	0.412665	0.358875	0.518436	0.043121	0.039616	0.04597
91	3.636122	3.527989	3.761903	0.74476	0.708391	0.827832	1.200284	1.127502	1.46903	0.085392	0.077198	0.09171
128	4.299324	4.211097	4.349279	1.216096	1.137977	1.295268	3.217696	2.89453	3.452667	0.180441	0.167609	0.192123
181	5.43123	5.280133	5.563518	1.815305	1.745512	1.887887	8.386993	8.268725	8.599224	0.44359	0.41388	0.508739
256	6.77958	6.645225	6.894622	2.468065	2.407839	2.51624	18.74412	17.48454	19.75385	1.118089	1.020731	1.221509
362	8.628605	8.452975	8.720051	3.079893	3.00577	3.158928	43.95585	42.6256	45.49786	3.200073	3.026997	3.4121
512	10.38258	10.18797	10.49973	3.709972	3.638657	3.815753	99.52432	97.40516	105.2507	8.540273	8.14699	8.914914
724	12.73147	12.60467	12.84596	4.660022	4.600911	4.700161	208.7351	200.3426	215.0928	21.3468	20.46003	22.09571
1024	14.63572	14.50642	14.79271	5.867116	5.799183	5.944921	430.4784	423.7408	446.5432	52.71581	50.89673	55.01907
1448	16.53314	16.38136	16.65693	7.267178	7.148245	7.344689	859.1299	837.7605	874.8426	122.2366	118.044	126.1242

Dims.	Bent Cigar						Brown					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.376171	0.121889	1.251712	0.020464	0.003179	0.054692	3.16E-07	1.9E-07	1.19E-06	5.09E-08	2.86E-08	9.15E-08
3	7.095605	3.297725	17.72958	0.241154	0.131439	0.661091	4.58E-06	1.29E-06	1.36E-05	1.67E-07	8.84E-08	2.78E-07
4	15.09093	6.469717	34.01559	0.729801	0.322777	1.433759	1.63E-05	3.62E-06	2.62E-05	3.46E-07	2.01E-07	8.01E-07
6	101.9988	22.47843	163.2487	2.448666	1.318412	4.782637	7.3E-05	5.18E-05	0.000139	1.72E-06	1.07E-06	2.52E-06
8	251.9915	114.174	374.9966	7.688641	6.046531	11.19951	0.000119	6.37E-05	0.000166	3.91E-06	2.39E-06	5.37E-06
11	638.7957	418.9166	913.4401	20.58757	12.8334	25.32331	0.000365	0.000232	0.000512	1.12E-05	5.98E-06	1.78E-05
16	1984.793	1627.508	2479.59	98.48625	70.57827	134.4367	0.001497	0.001109	0.001834	3.2E-05	2.5E-05	5.12E-05
23	6393.485	4473.235	8738.335	266.5476	226.1445	323.1982	0.003559	0.002931	0.004029	0.000152	0.000118	0.000188
32	21801.75	16157.08	26304.79	697.9771	547.4586	790.8481	0.00864	0.005825	0.012475	0.000386	0.000296	0.000456
45	51575.64	39907.94	63528.56	1795.075	1521.103	2361.016	0.032211	0.028702	0.035481	0.001007	0.000876	0.001201
64	169901.2	142517.7	204706.6	5886.803	5006.736	8022.404	0.102066	0.088548	0.117468	0.00294	0.002627	0.003364
91	566999.4	456461.8	604903.5	20257.2	16300.49	22097.49	0.296853	0.271074	0.350958	0.009907	0.008673	0.01102
128	1648340	1579369	1738973	53453.37	47578.58	61180.65	0.874107	0.746023	0.973565	0.026985	0.023654	0.029394
181	4719114	4361061	5287947	178256.1	153388.6	189709.6	2.754319	2.50309	2.940483	0.086351	0.080074	0.098393
256	11608006	11224005	12511514	426322.3	390756.1	433780.9	6.518334	6.333701	7.205059	0.21605	0.208982	0.245198
362	34939434	32110038	36412641	1316612	1201549	1446867	22.02767	20.82965	23.05584	0.812886	0.743506	0.840337
512	92210384	87144740	96313944	3636047	3261334	3924597	59.67129	55.15218	62.8338	2.336363	2.23942	2.530214
724	2.35E+08	2.25E+08	2.55E+08	10852651	10277394	11631010	142.3567	135.4494	155.8691	6.163218	5.60123	6.393365
1024	6.07E+08	5.85E+08	6.21E+08	31417020	30114437	32652305	361.6722	329.9347	381.3903	16.09541	15.17267	16.46518
1448	1.38E+09	1.35E+09	1.42E+09	85378123	82190507	90526034	3442.374	1918.12	6843.044	48.83662	47.18971	52.20566

Dims.	Chung-Reynolds						Deflected Corrugated Spring					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	7.82E-07	1.09E-07	5.51E-06	1.11E-09	1.6E-10	1.01E-08	-0.99801	-0.99971	-0.99346	-0.99999	-1	-0.99998
3	5.87E-06	1.39E-06	4.11E-05	4.5E-08	1.37E-09	1.36E-07	-0.84334	-0.86973	-0.84334	-0.84334	-0.8825	-0.84334
4	6.25E-05	2.79E-05	0.000464	2.76E-07	6.14E-08	6.36E-07	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
6	0.001684	0.000282	0.006576	2.69E-06	1.57E-06	5.44E-06	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
8	0.011466	0.005507	0.043458	1.88E-05	1.26E-05	3.65E-05	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
11	0.124142	0.066704	0.219085	8.67E-05	2.91E-05	0.000178	-0.84334	-0.84334	-0.37336	-0.84334	-0.84334	-0.84334
16	1.326805	0.359519	1.847579	0.001569	0.000735	0.002686	-0.37336	-0.37336	-0.37336	-0.37336	-0.37336	-0.37336
23	18.2597	8.710848	34.78801	0.014893	0.007958	0.027488	0.409922	0.409922	0.409923	0.409922	-0.37336	0.409922
32	79.33099	57.96083	181.8555	0.090238	0.056724	0.163675	1.5065	1.506499	1.506501	1.506499	1.506499	1.506499
45	755.0647	415.736	1167.984	1.077335	0.614119	1.511968	2.916346	1.507015	2.916346	1.506499	1.506499	2.916346
64	4762.034	3456.709	8063.542	5.883311	4.793573	8.210679	6.470461	4.639431	6.675717	4.639431	2.916346	4.639431
91	53701.63	45784.1	72719.07	61.00283	47.98758	72.99388	11.68769	9.025156	11.68769	9.025156	6.675717	11.68769
128	379154.7	292725.1	408384.3	388.7159	354.5116	549.7907	14.68421	14.66325	17.95174	11.68769	11.68769	14.66325
181	3190583	2511200	3551193	3865.239	3064.751	4647.48	25.4671	21.55384	25.4671	19.75241	17.95174	21.55307
256	26493358	22815939	29436028	38023	28441.26	48016.77	36.88531	34.23259	39.08356	34.23259	34.23259	39.08356
362	1.58E+08	1.4E+08	1.8E+08	280429	241172.8	317852	44.24621	39.67939	45.61466	39.08356	39.08356	44.24621
512	1.43E+09	1.29E+09	1.67E+09	2738533	2370651	3085277	53.17062	49.72356	55.50394	44.24621	42.95555	44.24621
724	9.01E+09	7.67E+09	9.57E+09	18864683	17014537	20838739	73.96606	70.06096	77.23238	49.71999	49.71999	49.71999
1024	5.84E+10	5.31E+10	6.22E+10	1.52E+08	1.38E+08	1.69E+08	142.8732	139.5201	148.834	49.71999	49.71999	49.71999
1448	3.2E+11	3.03E+11	3.36E+11	1.13E+09	1.04E+09	1.17E+09	238.4211	231.0148	244.75	55.50393	49.71999	55.50393

Table B.15: GA Descriptive Statistics: Exponential to Rastrigin Functions (10,000 Evaluation Tuning)

Dims.	Exponential						Griewank						Inverted Cosine Wave						Levy						Qing						Rastrigin					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	-1	-1	-1	-1	-1	-1	0.011475	0.00967	0.021395	0.007404	0.00062	0.007627	-0.9999	-0.99999	-0.73809	-1	-1	-0.73809	6.82E-06	1.75E-06	1.16E-05	2.36E-07	4.97E-08	7.87E-07	0.040229	0.023113	0.091307	0.001545	0.000387	0.002903	0.000596	0.000194	0.001332	1.42E-05	6.68E-06	2.64E-05
3	-1	-1	-1	-1	-1	-1	0.029659	0.022254	0.039517	0.010625	0.002456	0.012848	-1.99939	-1.99963	-1.99869	-2	-2	-1.99999	3.05E-05	1.07E-05	4.16E-05	7.77E-07	2.84E-07	2.57E-06	0.101166	0.056568	0.247685	0.006071	0.002833	0.01462	0.001361	0.00079	0.002646	6.99E-05	1.97E-05	0.000119
4	-1	-1	-1	-1	-1	-1	0.044561	0.035445	0.057365	0.015013	0.011561	0.02003	-2.99388	-2.99817	-2.98088	-2.99999	-2.99999	-2.99995	3.94E-05	3.1E-05	6.65E-05	1.05E-06	4.66E-07	2.99E-06	0.68512	0.399978	1.8653	0.01155	0.005021	0.018697	0.010264	0.004898	0.015047	0.000384	0.000253	0.00091
6	-1	-1	-1	-1	-1	-1	0.094774	0.0745	0.126018	0.035551	0.027056	0.041482	-4.95414	-4.98694	-3.69021	-4.99979	-4.99994	-3.69045	0.000139	0.000113	0.000244	6.06E-06	3.83E-06	1.07E-05	2.522931	1.361742	3.726315	0.112184	0.051529	0.220769	0.01955	0.015114	0.031648	0.00067	0.000353	0.00118
8	-1	-1	-0.99999	-1	-1	-1	0.23717	0.202222	0.286015	0.06696	0.04387	0.077124	-6.4161	-6.96316	-5.53885	-6.99911	-6.99969	-5.87527	0.000632	0.000243	0.001012	1.8E-05	1.23E-05	3.34E-05	10.42129	6.460715	20.25441	0.347381	0.14681	0.648962	0.048009	0.035721	0.072979	0.001303	0.000916	0.002676
11	-0.99999	-0.99999	-0.99999	-1	-1	-1	0.402001	0.353203	0.502016	0.098307	0.066504	0.108273	-9.10963	-9.8958	-7.83694	-9.99545	-9.99832	-7.904	0.001877	0.001311	0.0023	6.45E-05	4.87E-05	7.6E-05	33.35146	23.62112	65.08855	1.828857	0.931777	2.841159	0.111049	0.078577	0.170694	0.00607	0.003719	0.008897
16	-0.99996	-0.99997	-0.99994	-1	-1	-1	0.840738	0.634394	0.973232	0.111259	0.079707	0.129552	-12.1502	-12.8153	-11.5137	-13.2577	-14.9754	-12.5828	0.004967	0.003895	0.006589	0.000195	0.00013	0.000277	260.182	175.2575	386.2421	7.283404	3.983439	9.785093	0.573956	0.360452	0.683468	0.02203	0.015244	0.023492
23	-0.99983	-0.99987	-0.99979	-0.99999	-0.99999	-0.99999	1.026381	1.015646	1.033751	0.195982	0.174627	0.254148	-18.0202	-18.4429	-16.9907	-18.5492	-19.4816	-17.8071	0.013024	0.009893	0.019889	0.00063	0.000463	0.000747	861.1166	587.6696	1242.584	35.15261	19.80236	41.46491	1.568452	1.025933	2.109962	0.062663	0.044442	0.072049
32	-0.99955	-0.99961	-0.99945	-0.99998	-0.99999	-0.99998	1.083892	1.071036	1.098701	0.381168	0.349382	0.482864	-24.4715	-25.7838	-23.3719	-25.5681	-26.8917	-24.6748	0.034263	0.02714	0.039092	0.001355	0.001047	0.001528	7650.641	4152.469	12808.72	122.1873	100.7319	186.7709	3.976676	3.273634	4.762849	0.151216	0.133459	0.162725
45	-0.99881	-0.9989	-0.99851	-0.99996	-0.99997	-0.99994	1.20666	1.179979	1.255765	0.672002	0.583231	0.729892	-34.1346	-35.3224	-32.9741	-35.8838	-37.8134	-34.2397	0.118721	0.095209	0.142483	0.004462	0.00358	0.005803	37108.69	20943.41	49856.93	532.2529	393.2438	727.0029	8.702069	7.743074	10.15521	0.390385	0.331763	0.550144
64	-0.99558	-0.99619	-0.99519	-0.99986	-0.99988	-0.99984	1.575611	1.515892	1.749806	0.960745	0.85811	0.998768	-47.2194	-47.7719	-46.7497	-49.7299	-50.8427	-48.6602	0.352024	0.324203	0.397159	0.011906	0.010061	0.015401	185536.3	91631.48	354283.3	2040.147	1791.752	2195.514	19.73061	17.57268	22.28389	1.204041	0.995714	1.588956
91	-0.98917	-0.99058	-0.98733	-0.99963	-0.99972	-0.9996	3.496719	3.113482	3.658754	1.077811	1.067206	1.08782	-66.9239	-67.7671	-66.0095	-70.848	-72.3589	-69.4331	0.932154	0.784819	1.021351	0.038121	0.030543	0.043019	1325320	787454.6	1920524	9292.832	8065.552	10320.66	42.91758	40.11025	45.44489	3.851754	3.442174	4.469791
128	-0.96848	-0.97257	-0.96221	-0.99878	-0.9989	-0.99866	7.051048	6.136362	7.749183	1.214164	1.196579	1.236483	-92.5276	-93.0579	-91.3549	-100.922	-103.89	-99.0656	2.220237	2.017629	2.429364	0.113765	0.096946	0.146041	8126805	6123685	13110975	31423.03	27377.19	35444.65	94.20802	90.2983	100.2284	10.94459	9.940382	12.20123
181	-0.90629	-0.91215	-0.90299	-0.99661	-0.99719	-0.99625	16.9188	14.38947	18.62535	1.565495	1.502355	1.644902	-126.317	-127.467	-125.279	-140.752	-142.721	-139.901	5.843205	5.378306	6.129322	0.31721	0.269629	0.332154	57626460	47964491	79452065	168466.6	148276.7	190626.4	175.2644	163.202	184.0851	25.73404	23.9355	27.96898
256	-0.78846	-0.80311	-0.77491	-0.9913	-0.99229	-0.98937	46.73459	44.0045	48.68656	2.718124	2.610545	2.926083	-172.346	-173.526	-168.07	-194.398	-195.742	-194.002	14.27002	13.34843	15.22481	0.901206	0.829946	0.975668	2.21E+08	2.03E+08	2.95E+08	636611.4	597304.1	670579.2	342.5179	326.3227	355.0425	61.35886	58.03938	64.42964
362	-0.51517	-0.5289	-0.5059	-0.97418	-0.97643	-0.97277	118.5838	107.5256	124.3898	5.516442	5.203851	5.759939	-227.499	-229.544	-225.486	-272.273	-274.588	-270.915	29.98568	27.5855	31.86914	2.322476	2.147843	2.375557	1.52E+09	1.11E+09	2.21E+09	2664474	2310277	3655254	676.7276	658.2849	694.2383	139.1033	135.5474	141.9943
512	-0.17766	-0.19532	-0.15512	-0.92852	-0.93093	-0.92307	326.2344	304.2565	352.0743	14.313	13.40878	15.69499	-295.779	-298.332	-291.805	-379.588	-380.672	-377.211	81.11562	77.65935	89.18366	6.122128	5.909123	6.371253	6.86E+09	6.17E+09	8.34E+09	11886687	11206556	13041523	1311.628	1272.592	1330.247	287.9662	281.1857	295.5399
724	-0.0091	-0.01124	-0.00744	-0.80597	-0.81336	-0.79897	863.7753	815.4545	891.7502	40.3209	38.24767	41.86243	-372.147	-375.344	-367.218	-523.099	-525.08	-519.632	229.9251	222.4709	243.2356	15.45177	15.22416	15.89778	3.26E+10	2.69E+10	4.1E+10	61122899	56652943	71441157	2481.938	2412.21	2547.557	573.4087	560.2459	585.9933
1024	-8.5E-06	-1.1E-05	-5E-06	-0.53381	-0.55393	-0.51168	2097.951	2053.647	2291.336	112.9925	108.8742	114.5204	-451.783	-460.692	-443.138	-714.893	-722.396	-712.36	583.8123	565.7382	605.873	35.58224	34.4019	36.26156	1.56E+11	1.4E+11	1.77E+11	3.44E+08	3.06E+08	4.13E+08	4760.363	4674.785	4861.261	1123.451	1101.8	1142.127
1448	-7.6E-83	-2.2E-81	-5.1E-84	-3.2E-47	-7E-46	-5.1E-48	4908.524	4802.121	5169.287	304.0212	289.2313	311.074	-537.879	-544.247	-530.999	-960.864	-963.008	-956.8	1422.905	1401.747	1468.785	83.8043	82.44154	86.8297	5.48E+11	5.34E+11	6.02E+11	1.81E+09	1.69E+09	2E+09	8676.589	8595.89	8746.378	2195.566	2155.203	2210.87

Table B.16: GA Descriptive Statistics: Exponential to Rastrigin Functions (50,000 Evaluation Tuning)

Dims.	Exponential						Griewank						Inverted Cosine Wave						Levy						Qing						Rastrigin					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	-1	-1	-1	-1	-1	-1	0.010833	0.009087	0.014262	0.00477	0.001109	0.00747	-0.99992	-0.99998	-0.99979	-1	-1	-1	1.65E-06	6.37E-07	1.15E-05	1.57E-07	5.7E-08	7.31E-07	0.018346	0.005371	0.031286	0.000678	0.000214	0.001427	0.000243	0.000122	0.000677	2.22E-05	4.59E-06	7.9E-05
3	-1	-1	-1	-1	-1	-1	0.031607	0.019987	0.04214	0.010004	0.007753	0.011429	-1.47618	-1.99949	-1.47618	-1.47618	-1.99998	-1.47618	1.52E-05	1.05E-05	3.06E-05	5.37E-07	3.23E-07	1.69E-06	0.173819	0.122342	0.384387	0.007911	0.003995	0.016125	0.001582	0.000604	0.002588	9.4E-05	1.72E-05	0.000173
4	-1	-1	-1	-1	-1	-1	0.050529	0.032478	0.074	0.014274	0.011717	0.018699	-2.99678	-2.99899	-2.21427	-2.99992	-2.99998	-2.21427	6.53E-05	3.61E-05	0.000142	2.59E-06	1.7E-06	5.02E-06	0.621572	0.367998	0.795311	0.02866	0.013364	0.040029	0.004605	0.002794	0.007957	0.000183	8.5E-05	0.000251
6	-1	-1	-0.99999	-1	-1	-1	0.105768	0.089203	0.134353	0.03318	0.022289	0.044015	-4.94135	-4.96928	-4.77512	-4.9997	-4.99984	-4.9995	0.00028	0.000175	0.000361	9.87E-06	4.24E-06	1.62E-05	2.891094	2.176584	4.773798	0.129554	0.084192	0.286227	0.025299	0.010273	0.044961	0.000822	0.000334	0.001287
8	-0.99999	-1	-0.99999	-1	-1	-1	0.196584	0.16412	0.232859	0.046216	0.040533	0.058354	-6.85394	-6.95883	-5.60118	-6.99834	-6.99883	-6.6616	0.000464	0.000292	0.000834	1.79E-05	1.11E-05	2.72E-05	6.266598	3.488177	10.34408	0.626588	0.394703	0.810496	0.057571	0.028715	0.095294	0.0013	0.00105	0.002303
11	-0.99998	-0.99999	-0.99998	-1	-1	-1	0.58211	0.431959	0.646321	0.061664	0.049877	0.088721	-9.21734	-9.87665	-8.15422	-9.99436	-9.99685	-8.53391	0.001669	0.001189	0.002777	6.2E-05	2.87E-05	0.000111	35.54933	23.18747	63.85576	1.274438	0.911165	1.863773	0.159826	0.093325	0.263158	0.005897	0.004245	0.007443
16	-0.99996	-0.99997	-0.99993	-1	-1	-1	0.921027	0.845896	1.002263	0.114398	0.083011	0.126607	-11.8557	-12.3108	-11.5977	-13.6416	-14.8435	-13.5633	0.004636	0.002476	0.005777	0.000177	0.000105	0.00026	199.4211	152.1743	307.289	7.509783	3.920625	10.30552	0.386384	0.324364	0.526733	0.015544	0.011387	0.020581
23	-0.99984	-0.99986	-0.9998	-0.99999	-1	-0.99999	1.033905	1.024451	1.045639	0.184621	0.154384	0.26896	-16.5968	-17.2367	-16.0613	-18.1137	-18.8623	-16.86	0.0217	0.014278	0.024583	0.000571	0.000304	0.000745	926.4125	758.0706	1244.073	27.25641	19.14833	34.09795	1.587569	0.963392	2.284865	0.054699	0.039095	0.069927
32	-0.99954	-0.99961	-0.99945	-0.99998	-0.99999	-0.99998	1.10473	1.092719	1.139362	0.410634	0.282121	0.460039	-23.1835	-23.599	-22.7622	-25.9919	-27.2911	-24.7179	0.043587	0.027291	0.057136	0.001428	0.001155	0.002054	4875.414	3159.705	5775.167	124.244	110.9176	174.0468	4.264943	3.400588	4.819416	0.156595	0.123507	0.224965
45	-0.99874	-0.99888	-0.99849	-0.99995	-0.99997	-0.99994	1.214942	1.191548	1.303263	0.729804	0.665765	0.791257	-33.2597	-33.7063	-32.6642	-36.9237	-38.7013	-35.6295	0.105897	0.094762	0.137743	0.004005	0.00324	0.005758	23203.63	12150.18	40187.48	486.0656	396.7797	639.0511	8.851017	7.835262	9.993888	0.432836	0.345861	0.528399
64	-0.99562	-0.99645	-0.99489	-0.99987	-0.99988	-0.99984	1.828144	1.769013	1.892504	0.989401	0.948591	1.017816	-47.6953	-48.4754	-46.7006	-51.3268	-52.3235	-50.2464	0.34518	0.31101	0.392045	0.012675	0.011555	0.014101	155701.1	83979.65	296146.3	2069.964	1698.066	2398.855	21.87277	19.58107	22.7844	1.169427	1.000603	1.29399
91	-0.98898	-0.98973	-0.98736	-0.99957	-0.99964	-0.99951	2.791494	2.723604	2.971753	1.06283	1.0518	1.078449	-65.8208	-66.4943	-65.6257	-71.5311	-73.6976	-70.3052	0.895282	0.798894	0.963581	0.039583	0.033745	0.04318	1252798	804137.3	1599814	8335.796	7812.064	9907.923	45.00013	42.49424	47.38615	3.665926	3.275522	4.071112
128	-0.96644	-0.97024	-0.96379	-0.99887	-0.99906	-0.99881	7.546534	6.799014	8.147928	1.218503	1.192435	1.244053	-91.4587	-92.4997	-90.3208	-99.815	-102.041	-97.8706	2.331258	2.145629	2.516434	0.110841	0.085428	0.123084	8988348	7671696	13324902	37565.28	34320.72	41648.95	87.47554	82.03474	91.22045	9.972022	8.38109	10.84237
181	-0.9208	-0.9283	-0.91538	-0.99696	-0.99727	-0.99653	16.74871	15.98797	17.3811	1.540105	1.508003	1.588228	-126.96	-127.742	-125.757	-141.345	-142.753	-137.825	5.605723	5.235581	5.799761	0.319832	0.293431	0.332559	42240139	36099625	56288744	141617.1	132750.1	161774	182.008	171.3303	190.1753	28.09069	24.66629	30.33894
256	-0.7871	-0.7986	-0.76387	-0.99061	-0.99166	-0.98969	41.19851	40.39947	42.10385	2.730575	2.602256	2.827579	-167.996	-169.159	-166.372	-194.874	-196.426	-192.958	13.9753	12.43913	14.80335	0.831842	0.791673	0.8972	2.8E+08	2.18E+08	3.16E+08	606780.8	517794.1	648612.4	330.6857	323.1611	352.2466	62.29521	55.63502	66.0801
362	-0.47817	-0.51078	-0.43776	-0.97399	-0.97527	-0.97197	123.995	116.1081	132.8629	5.707018	5.470413	6.209865	-226.133	-229.259	-224.024	-272.603	-275.527	-270.153	32.39236	30.79406	33.35452	2.280849	2.152581	2.434376	1.36E+09	1.24E+09	1.73E+09	2382481	2196708	2877589	666.2597	654.836	696.6392	135.3299	130.8913	138.4875
512	-0.16301	-0.17889	-0.14396	-0.92763	-0.93014	-0.92576	304.4044	287.2542	326.7658	14.11583	13.32284	14.72669	-292.367	-298.322	-284.031	-378.509	-381.038	-375.961	90.58176	87.13408	95.06345	6.313621	6.130541	6.65792	9.41E+09	8.31E+09	1.04E+10	14555031	11761816	17181391	1331.259	1304.519	1375.587	287.6367	279.2055	295.1542
724	-0.00922	-0.01104	-0.00775	-0.80297	-0.81144	-0.79713	853.8979	816.6413	879.2046	39.68201	37.8846	42.12205	-370.432	-372.461	-364.028	-524.112	-527.512	-522.195	207.856	201.9527	215.0679	15.31483	14.80536	15.65445	3.04E+10	2.75E+10	3.48E+10	65255491	57571545	67535449	2505.489	2451.121	2536.509	564.1841	544.086	579.4025
1024	-1E-05	-1.4E-05	-4.8E-06	-0.5354	-0.54896	-0.52381	2173.991	2102.334	2233.204	108.5605	106.5849	112.6006	-442.837	-448.531	-437.059	-712.873	-715.614	-709.986	560.8379	549.1171	590.744	34.73058	34.00131	36.20229	1.54E+11	1.39E+11	1.65E+11	3.72E+08	3.28E+08	4.57E+08	4689.803	4650.584	4766.467	1126.107	1106.862	1145.24
1448	-5.7E-13	-9.8E-13	-2.5E-13	-0.18152	-0.1901	-0.17246	5074.753	4912.992	5151.08	316.3571	310.3367	325.0712	-536.629	-544.063	-530.891	-962.467	-967.366	-958.328	1448.012	1413.769	1473.033	85.21369	83.58979	87.05804	5.77E+11	5.51E+11	6.36E+11	1.85E+09	1.74E+09	2.12E+09	8913.129	8842.178	8943.311	2226.553	2190.467	2261.017

Table B.17: GA Descriptive Statistics: Rosenbrock to Sum Squares Functions (10,000 Evaluation Tuning)

Dims.	Rosenbrock						Schwefel						Sphere						Sum of Different Powers						Sum Squares					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.028758	0.011924	0.128805	0.007014	0.0014	0.026199	0.002192	0.000837	0.006059	0.000174	6.28E-05	0.000553	3.69E-06	8.21E-07	6.55E-06	1.05E-07	1.75E-08	2.35E-07	4.31E-08	1.1E-08	1.59E-07	1.3E-09	3.06E-10	6.19E-09	1.34E-05	1.03E-05	2.09E-05	4.16E-07	9.3E-08	1.87E-06
3	0.750167	0.597904	1.19049	0.196077	0.026189	0.527469	0.005728	0.003667	0.021305	0.000489	0.000205	0.001083	6.85E-06	2.15E-06	1.03E-05	2.86E-07	1.52E-07	4.94E-07	1.03E-07	1.81E-08	1.93E-07	1.25E-09	4.54E-10	4.25E-09	9.21E-05	3.89E-05	0.000134	2.82E-06	7.61E-07	6E-06
4	1.636931	0.554345	2.383085	1.36192	0.521223	1.933931	0.03318	0.01565	0.038558	0.001752	0.000781	0.003334	2.66E-05	6.04E-06	5.07E-05	6.34E-07	3.14E-07	1.43E-06	1.62E-07	7.01E-08	4.5E-07	3.89E-09	5.94E-10	1.24E-08	0.000206	9.87E-05	0.000352	9.49E-06	5.86E-06	1.33E-05
6	3.357356	1.515759	4.740803	2.414045	1.028991	3.802719	0.196646	0.074473	0.237208	0.004302	0.003272	0.006835	0.000105	4.54E-05	0.000185	6.03E-06	3.12E-06	9.16E-06	4.19E-07	8.08E-08	1.4E-06	2.08E-08	2.11E-09	5.77E-08	0.001335	0.000902	0.002318	5.25E-05	3.69E-05	8.47E-05
8	8.48655	6.89638	15.21878	3.366428	1.296571	5.55699	0.409364	0.19936	0.479048	0.011892	0.006547	0.018748	0.000283	0.000194	0.000341	1.43E-05	9.42E-06	1.93E-05	3.38E-07	1.69E-07	3.93E-06	2.75E-08	3.36E-09	4.87E-08	0.005225	0.003429	0.01021	0.000171	8.49E-05	0.000221
11	62.35256	31.26567	74.071	6.733401	2.915878	9.803885	0.789151	0.562089	1.276431	0.040744	0.02269	0.051131	0.000779	0.00049	0.001092	2E-05	1.34E-05	4.12E-05	6.94E-07	1.1E-07	1.81E-06	3.52E-08	7.24E-09	7.72E-08	0.018974	0.011746	0.025083	0.000773	0.000404	0.001023
16	72.31506	37.81019	97.70042	9.888156	4.45368	23.58801	1.930726	1.444732	3.397077	0.08011	0.062932	0.121944	0.003681	0.00226	0.004028	0.000109	6.42E-05	0.000164	1.38E-06	2.86E-07	5.63E-06	5.1E-08	8.58E-09	1.08E-07	0.06585	0.054635	0.083453	0.003255	0.001581	0.004426
23	116.9447	105.4564	127.6358	68.22008	17.34984	80.84342	8.952761	6.855796	12.10056	0.338246	0.274032	0.393124	0.007431	0.005046	0.009777	0.00027	0.000206	0.00039	4.11E-06	4.14E-07	1.31E-05	1.7E-07	4.14E-08	3.87E-07	0.366538	0.264492	0.591137	0.015144	0.010657	0.021983
32	177.4997	155.1924	238.9289	86.27265	80.46126	102.8296	25.62997	19.91567	31.30683	0.86473	0.683128	1.347403	0.020839	0.01579	0.025841	0.000683	0.000553	0.000816	9.51E-06	2.16E-06	2.49E-05	2.01E-07	6.99E-08	7.93E-07	1.002885	0.81487	1.456626	0.044069	0.033509	0.068482
45	371.0316	313.7452	427.2211	147.368	124.9446	167.9816	64.32316	51.70494	76.26802	2.947021	2.473912	3.666669	0.05494	0.051942	0.069793	0.001803	0.001418	0.002314	6.21E-06	2.64E-06	3.14E-05	1.89E-06	1.11E-07	3.42E-06	4.892334	4.489955	6.360566	0.144687	0.122747	0.191202
64	862.7755	752.9335	1085.001	241.2157	227.7622	284.6264	258.5445	204.8605	273.596	7.04672	6.663045	8.379784	0.183102	0.156277	0.211834	0.006399	0.005122	0.007662	3.96E-05	7.27E-07	0.000126	4.43E-07	5.63E-08	4.86E-06	21.31683	18.08767	23.28036	0.816572	0.673731	0.95502
91	1978.81	1653.893	2298.881	403.803	375.9192	437.7214	640.1509	580.9822	754.7219	24.64011	20.21163	25.64467	0.602847	0.48719	0.700443	0.021791	0.017875	0.024827	7.1E-05	1.53E-05	0.000196	2.12E-06	8.13E-07	6.1E-06	91.97703	81.45095	99.91956	3.474196	2.69653	3.818279
128	4475.064	3817.977	5147.521	672.8888	624.6505	749.6182	1516.413	1453.45	1653.375	69.80472	57.79074	79.77304	1.460093	1.380322	1.626187	0.052339	0.048775	0.059025	4.93E-05	1.62E-05	0.000104	3.29E-06	1.29E-06	1.14E-05	371.4734	332.3212	426.0699	14.70847	12.03765	16.05931
181	15001.41	11136.04	17339.51	1155.549	1108.545	1253.743	3956.186	3594.005	4113.655	180.9955	171.3667	197.7486	4.269995	3.700881	4.968113	0.161784	0.146759	0.173022	0.000319	0.000132	0.00103	9.8E-06	2.64E-06	2.53E-05	1461.23	1377.436	1612.682	56.23767	48.78344	60.39295
256	50348.7	40138.77	63673.26	2236.161	2119.873	2554.026	9321.77	8700.947	9806.269	557.907	521.4149	620.9783	11.83482	11.15385	14.2199	0.4887	0.416082	0.542436	0.000471	4.8E-05	0.001123	3.06E-05	7.41E-06	6.88E-05	5637.655	5459.795	5782.11	227.3279	214.0086	253.9281
362	189628.2	167284.8	199977	4966.79	4620.223	5229.098	19333.75	18166.47	20049.61	1665.186	1522.863	1736.49	38.42797	35.83239	39.63131	1.402447	1.344204	1.462636	0.000566	0.000247	0.002875	3.83E-05	1.46E-05	9.21E-05	20453.86	20020.51	21879.16	863.9098	791.6265	974.1403
512	752783.9	684541.4	859438.1	11688.91	10820.74	12019.63	39020.66	37569.42	40184.16	4026.538	3899.128	4270.114	95.84842	93.39029	100.3439	3.897618	3.631895	4.200977	0.002159	0.000492	0.004744	9.97E-05	3.77E-05	0.00057	91001.39	81854.93	97992.34	3870.529	3746.787	4176.715
724	3088564	2979920	3150919	29781.57	27544.33	31896.79	74701.67	73282.55	76713.56	10840.56	10500.9	11184.01	242.0772	236.7526	255.5323	11.27116	10.99558	11.72217	0.004755	0.000526	0.018104	0.000113	6.51E-05	0.000359	337880.4	318043	357925	15996.37	15192.95	16670.88
1024	13266623	12451590	14479343	104147.1	98512.88	111108.6	137126.9	134031.8	138877.4	26380.61	25939.63	27685.47	627.7402	607.2576	644.1198	32.19729	30.30458	33.44184	0.012036	0.004139	0.02011	0.000264	0.000139	0.000593	1065913	1040059	1085018	57317.7	54912.6	59601.75
1448	48124258	44160409	49221079	372328.8	349148.1	397028.2	240076.3	237846.2	241745.4	60176.79	59262.93	61201.98	1464.831	1432.091	1500.498	91.57275	87.04018	94.0888	0.017862	0.005176	0.04002	0.00021	8.53E-05	0.001647	3782338	3656823	3911876	231276.4	221273.2	242543.1

Table B.18: GA Descriptive Statistics: Rosenbrock to Sum Squares Functions (50,000 Evaluation Tuning)

Dims.	Rosenbrock						Schwefel						Sphere						Sum of Different Powers						Sum Squares					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.017267	0.006635	0.059116	0.002993	0.000907	0.012168	0.003415	0.000481	0.005297	8.3E-05	4.81E-05	0.000192	1.1E-06	2.67E-07	3.91E-06	1.43E-07	5.67E-08	3.4E-07	4.7E-08	6.08E-09	1.25E-07	6.96E-10	2.69E-10	3.97E-09	1.56E-05	2.73E-06	3.39E-05	3.38E-07	1.81E-07	8.96E-07
3	0.537161	0.411496	0.911406	0.406275	0.272952	0.73179	0.010206	0.006055	0.016734	0.000379	0.000152	0.00071	9.47E-06	3.51E-06	1.31E-05	3.14E-07	2.07E-07	6.97E-07	1.69E-07	2.01E-08	2.82E-07	4.25E-09	7.39E-10	1.56E-08	4.32E-05	2.78E-05	0.000121	3.54E-06	2.13E-06	4.19E-06
4	0.83147	0.254605	1.33505	0.405703	0.14501	0.990386	0.02747	0.020996	0.071069	0.001776	0.000506	0.003433	3.54E-05	2.37E-05	7.23E-05	1.41E-06	6.69E-07	2.05E-06	2.48E-07	5.11E-08	8.84E-07	7.43E-09	1.23E-09	2.77E-08	0.000154	0.000113	0.00029	6.75E-06	2.77E-06	1.12E-05
6	4.262903	3.3532	5.054703	2.325184	1.60635	3.005987	0.116783	0.081042	0.204687	0.005631	0.0021	0.007393	0.000132	7.01E-05	0.000188	3.72E-06	1.83E-06	6.11E-06	1.88E-07	8.79E-08	4.14E-07	5.25E-09	9.93E-10	9.05E-09	0.001525	0.000539	0.002569	5.89E-05	3.98E-05	9.55E-05
8	6.842301	5.49856	11.72564	3.185253	1.782366	5.82901	0.365658	0.171969	0.701353	0.01157	0.007836	0.020356	0.000253	0.000169	0.000348	1.23E-05	8.25E-06	1.6E-05	1.32E-06	1.82E-07	3.64E-06	2.52E-08	5.59E-09	8.54E-08	0.0047	0.002937	0.008058	0.000185	0.000101	0.000484
11	25.97931	11.00088	68.5823	6.584531	2.175316	8.354278	0.801423	0.599811	1.045966	0.027601	0.013429	0.040776	0.00065	0.000351	0.00081	2.66E-05	1.59E-05	3.59E-05	8.02E-07	2.34E-07	1.51E-06	2.51E-08	7.8E-09	7.65E-08	0.021173	0.013319	0.031132	0.000656	0.000499	0.000821
16	77.85579	21.20334	93.44231	13.58172	7.599835	74.01726	3.958528	2.416305	4.709394	0.144261	0.108877	0.181772	0.001968	0.001374	0.003307	9.53E-05	7.09E-05	0.000117	3.25E-06	1.32E-06	1.77E-05	4.55E-08	9.02E-09	1.55E-07	0.085576	0.054712	0.130589	0.002632	0.001941	0.00407
23	132.2465	111.8962	179.9186	72.91638	19.27156	82.08839	8.162728	6.298956	10.4075	0.385148	0.303103	0.494161	0.007171	0.00494	0.009898	0.000279	0.000239	0.000357	1.46E-06	6.99E-07	1E-05	1.89E-07	1.47E-08	5.53E-07	0.456676	0.241355	0.601545	0.013776	0.010072	0.019792
32	199.6373	188.5438	235.0797	88.07367	78.44614	111.5358	29.4569	20.99754	36.85522	0.9739	0.884124	1.173481	0.025664	0.021225	0.029096	0.000825	0.000736	0.001009	6.19E-06	1.93E-06	1.63E-05	5.26E-08	2.14E-08	2.03E-07	1.333654	1.10431	1.561413	0.043118	0.030904	0.066127
45	390.9617	304.0701	458.195	148.0962	103.6022	161.0378	79.79764	60.67605	97.29203	2.808217	2.344115	3.218398	0.084981	0.064355	0.09055	0.002251	0.002004	0.002879	1.29E-05	1.07E-06	2.44E-05	2.51E-07	9.81E-08	7.33E-07	5.974012	4.578485	6.625231	0.194317	0.16051	0.236774
64	786.9051	635.628	905.387	253.1646	219.4693	297.4136	222.432	191.5415	246.732	8.070458	6.971821	8.749971	0.216493	0.179778	0.275136	0.006949	0.005208	0.008738	6.08E-05	2.68E-05	0.000201	1.51E-06	1.31E-07	1.21E-05	27.27649	24.31206	33.15495	0.924479	0.749274	1.184972
91	1965.187	1807.42	2476.026	407.8613	360.8413	438.542	594.1356	543.7275	713.1578	24.65332	21.02454	26.63172	0.625815	0.524266	0.685952	0.021698	0.016652	0.022408	4.63E-05	1.57E-05	0.000117	5.51E-07	2.35E-07	3.81E-06	117.8535	94.95545	149.7998	3.584859	2.832131	4.284857
128	4418.179	3591.165	4858.167	669.6785	599.9686	723.0398	1604.602	1472.836	1892.597	68.85588	65.71956	71.79723	1.791182	1.570821	2.015876	0.055478	0.049013	0.063703	0.000161	2E-05	0.000344	5.62E-06	1.48E-06	2.09E-05	359.5646	325.3897	380.192	13.14244	11.95437	15.73643
181	17664.49	16023.87	20467.55	1236.776	1136.614	1299.882	3959.034	3809.849	4298.69	184.057	174.0617	211.2209	4.791521	4.314396	5.728288	0.170045	0.148228	0.195542	0.000504	0.000125	0.001441	2.74E-05	5.48E-06	6.16E-05	1621.791	1200.029	1776.463	46.91734	41.62066	55.47459
256	52693.17	45753.96	60246.1	2550.603	2240.354	2689.841	9215.685	8723.911	9508.417	535.8848	487.0566	559.5889	13.68751	12.88483	14.67232	0.481696	0.446306	0.523508	0.00031	0.000145	0.001169	1.34E-05	4.62E-06	4.55E-05	5187.622	4765.183	6196.532	208.0275	193.9892	221.124
362	214290.2	192823.9	262105.5	5030.877	4405.335	5614.742	20259.37	19298.27	20490.57	1543.72	1444.71	1696.092	34.26061	30.81527	37.52191	1.373815	1.264565	1.47142	0.001023	0.000461	0.003896	1.83E-05	1.68E-06	9.04E-05	22420.52	20463.95	24166.64	913.4438	801.9946	970.3503
512	908705.3	819963.7	946436.7	12380.77	11494.76	13923.36	38629.08	37579.16	39331.42	4139.217	3873.964	4280.247	93.65136	89.45141	97.21721	3.81527	3.585418	3.867818	0.00232	0.001178	0.004238	9.73E-05	2.17E-05	0.000668	88229.52	83520.79	94063.92	3640.688	3430.329	3819.2
724	3399833	3232319	3887507	31367.91	28870.68	33404.15	77385.28	76012.35	78143.63	11178.38	10967.96	11316.29	253.703	240.1387	265.9315	11.72971	10.9793	12.23578	0.002846	0.001055	0.006217	0.000166	7.73E-05	0.000596	321833.2	313496.3	332943.4	14419.44	14027.64	16930.39
1024	12358262	10952604	13977932	104875.2	95135.31	107251.2	135431.3	133342.5	137505.8	26627.14	25219.65	27260.51	619.6331	586.1414	636.0495	31.63822	30.49781	32.50785	0.007599	0.002432	0.023675	0.000218	5.96E-05	0.000413	1128523	1086130	1198540	57110.02	56062.29	59881.31
1448	50040024	45998129	53064057	404261.1	375047.9	421157.2	239606.1	235123	242384.6	57871.75	56920.52	58551.92	1503.175	1463.846	1518.895	88.95021	85.41085	94.21564	0.013932	0.006006	0.038887	0.001813	0.000209	0.00282	4085344	3845629	4152584	223276.5	219162.7	234176.3

Table B.19: PSO Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (10,000 Evaluation Tuning)

Dims.	Ackley						Alpine1						Bent Cigar					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	2.74E-09	0	2.74E-09	2.74E-09	0	2.74E-09	0	0	0	0	0	0
3	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.25E-08	2.74E-09	1.73E-06	4.25E-08	2.74E-09	1.73E-06	4.44E-14	0	1.42E-12	0	0	0
4	0.001365	0.00089	0.001712	4.44E-16	4.44E-16	4.44E-16	5.26E-07	2.74E-09	2.96E-06	2.53E-07	2.74E-09	2.08E-06	0.000103	4.92E-06	0.001185	0	0	0
6	0.011341	0.00928	0.015936	4.44E-16	4.44E-16	4.44E-16	0.001772	0.000349	0.005562	1.74E-06	3.56E-07	3.46E-06	0.005003	1.94E-06	0.094317	0	0	0
8	0.011577	0.008163	1.297131	4.44E-16	4.44E-16	1.29673	0.010531	0.004584	0.07295	0.000104	2.92E-06	0.002775	1.331858	0.448136	5.839209	0	0	0
11	1.921629	1.099581	2.275627	1.920957	1.098887	2.275124	0.175046	0.062565	0.348683	0.082244	0.016676	0.229528	2.190095	0.485533	3.565571	0.013877	0	0.34381
16	2.299684	2.079102	3.337472	2.245385	1.940581	2.935833	0.919242	0.7973	1.381481	0.245601	0.113552	0.510397	76.15877	49.50161	153.167	0.68661	0.231327	1.808727
23	4.346659	4.183852	5.02485	3.310807	2.83362	4.01948	3.494569	2.67996	4.306831	1.260592	0.78129	1.925275	18847.12	11724.96	24226.38	477.9093	159.5466	1278.664
32	4.576591	3.947626	5.608753	0.161291	0.137925	0.22545	6.583737	5.063601	9.381624	4.691517	2.832557	6.512119	250322.3	213377.3	317847.9	24347.16	15676.71	34143.5
45	6.625906	5.649672	7.554068	0.466626	0.359077	0.578771	12.2391	10.81233	15.39439	8.470229	6.762302	11.52902	1034968	804509.3	1239689	269856.3	188074.9	315427.4
64	7.592656	6.951095	9.634243	1.028607	0.656585	1.350756	22.76138	19.43421	26.80738	19.08436	16.43378	21.79541	3062796	2452206	5085810	22920.59	14563.48	25955.31
91	8.46884	7.956892	8.985593	2.607926	2.262783	3.232507	38.21754	33.52569	40.4329	30.52966	24.94902	32.39659	9290117	5306492	13925936	25888.99	19146.55	42297.11
128	9.019062	8.001254	10.87884	1.224123	0.826542	1.634681	56.84421	41.65515	67.78247	6.72003	3.285392	12.17166	8630183	4986882	14628043	19411.26	9362.787	25392.22
181	10.3049	9.776501	11.70272	1.572308	1.233008	2.077223	79.29581	65.3184	96.13856	7.413134	5.273258	11.00645	34009101	31363296	36503964	12222523	10467733	13692108
256	10.71134	9.162905	11.51114	2.184858	1.39986	3.273801	101.5033	81.23487	118.3394	8.038778	3.872505	12.86638	51140800	47576845	55715130	35490694	33239178	39116804
362	10.88654	9.84907	11.9744	1.762352	1.387564	4.192258	146.7996	113.9231	184.4203	17.88497	10.79643	22.55396	58588423	43005585	74768220	264994.5	120149.1	419520.6
512	10.97699	10.04296	12.08146	2.499381	2.01784	3.490506	196.8367	163.6525	247.8848	21.22502	14.04933	31.02621	94294320	81379018	1.74E+08	1824965	1298153	2744005
724	10.66929	9.862572	11.36385	2.208861	2.043835	2.895881	329.5039	277.371	353.7852	49.40156	26.65963	64.47378	1.06E+08	66990791	1.4E+08	2512309	723393	25364887
1024	11.80844	9.723831	13.47942	3.629398	2.764615	5.24263	397.7072	312.6741	466.6804	59.28129	31.78884	82.73163	2.3E+08	1.55E+08	2.67E+08	1349033	648237.3	3458760
1448	10.77205	10.05312	11.85737	2.674438	2.144799	3.565488	545.5537	381.9841	772.4473	73.46117	45.57462	98.6729	4.23E+08	3.31E+08	4.79E+08	8147424	6211968	26336385

Dims.	Brown						Chung-Reynolds						Deflected Corrugated Spring					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1
3	8.88E-16	8.88E-16	8.88E-16	8.88E-16	8.88E-16	8.88E-16	0	0	0	0	0	0	-0.99744	-0.99995	-0.84334	-1	-1	-1
4	3E-15	1.33E-15	8.27E-15	1.33E-15	1.33E-15	1.33E-15	0	0	0	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
6	4.79E-12	1.59E-12	1.93E-11	2.22E-15	2.22E-15	2.22E-15	1.08E-15	5.47E-17	4.55E-15	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
8	1.64E-14	5.61E-15	5.66E-13	3.11E-15	3.11E-15	3.11E-15	3.88E-10	2.58E-11	2.12E-09	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
11	3.46E-07	1.76E-07	7.69E-07	4.44E-15	4.44E-15	4.44E-15	3.87E-07	1.5E-07	8.13E-07	0	0	0	-0.84334	-0.84334	-0.72581	-0.84334	-0.84334	-0.84334
16	0.003874	0.001623	0.009187	4.45E-13	6.66E-15	6.46E-10	0.022824	0.007633	0.054494	0	0	3.76E-12	-0.37336	-0.43383	-0.37336	-0.84334	-0.84334	-0.84334
23	0.123618	0.104196	0.160529	7.99E-05	3.19E-05	0.000111	211.4825	103.3652	476.5122	0.003498	0.001443	0.053739	-0.37336	-0.37336	-0.37336	-0.84334	-0.84334	-0.84334
32	0.848477	0.522813	0.96709	0.008715	0.005471	0.013678	13266.45	11722.75	21714.36	62.5334	20.09197	226.1527	-0.3733	-0.37336	-0.03852	-0.84334	-0.84334	-0.84334
45	7.495122	5.698712	13.50584	0.224863	0.164607	0.268597	284393.1	248365.1	370732.7	55383.61	38285.32	85840.23	-0.37318	-0.38301	-0.33223	-0.84334	-0.84334	-0.84334
64	36.30297	17.30825	46.7639	1.075955	0.695928	1.210609	3766151	3258954	5127322	520317.5	279743.1	569438.2	1.508107	0.582038	1.999965	-0.68847	-0.83473	-0.36987
91	170.9161	132.701	202.0673	6.642404	6.017318	7.932288	17117656	12384375	23091329	1150382	711371.6	1807538	0.627355	0.410457	1.510879	-0.84331	-0.84334	-0.37336
128	936.3301	665.2149	1085.668	106.4389	84.3134	139.9649	53771877	42694525	65903411	17305860	14305029	25079509	2.916788	1.516256	3.828612	-0.54182	-0.8431	-0.32826
181	2756.772	2361.168	3459.988	1930.262	1384.225	2168.508	1.46E+08	43792685	2.25E+08	6618.028	3515.856	10205.63	4.639509	3.560637	6.675717	-0.37336	-0.37336	-0.37336
256	7360.025	6553.798	8111.013	3445.144	2911.302	3672.558	7.15E+08	3.97E+08	8.73E+08	9806325	3952516	16134378	3.817787	2.890734	8.352888	-0.36985	-0.3732	-0.31435
362	14474.08	12924.85	15733.56	8678.899	8313.307	9379.284	4.63E+08	2.71E+08	1.33E+09	42967.14	2237.631	303891.2	6.705964	6.248984	9.038087	-0.37334	-0.37336	0.409922
512	24498.49	22268.33	26487.7	18401.67	17375.81	20057.4	2.53E+09	2.14E+09	3E+09	9E+08	7.08E+08	1.12E+09	12.10684	9.027701	14.70584	0.410005	0.409922	1.506499
724	37805.93	33959.56	43931.72	30080.63	27725.51	35126.7	2.9E+09	1.9E+09	5.02E+09	345236.4	134009.6	705371.4	11.64982	8.832581	15.15997	0.409946	-0.37157	2.0581
1024	58986.19	55436.64	65328.55	48125.05	44037.15	50345.5	1.21E+10	8.47E+09	1.53E+10	75091397	42157118	1.37E+08	21.81348	18.43874	25.33518	0.446073	0.346725	1.543311
1448	94144.65	86241.65	102626.7	75901.03	68860.73	82380.48	2.2E+10	1.52E+10	3.14E+10	1.69E+08	1.28E+08	5.73E+08	34.23259	25.46725	37.99203	2.916346	2.916346	4.639431

Table B.20: PSO Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (50,000 Evaluation Tuning)

Dims.	Ackley						Alpine1						Bent Cigar					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	2.74E-09	2.74E-09	5.48E-09	2.74E-09	2.74E-09	5.48E-09	0	0	0	0	0	0
3	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	5.8E-08	2.74E-09	3.64E-07	4.11E-09	2.74E-09	3.48E-07	3.55E-13	0	3.2E-12	0	0	0
4	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	1.42E-05	4.35E-07	0.000121	5.26E-07	7.53E-09	2.08E-06	2.76E-07	4.49E-09	1.37E-06	0	0	0
6	0.299168	0.236502	1.030678	4.44E-16	4.44E-16	3.19E-06	0.003763	0.000379	0.007109	1.73E-06	1.1E-08	4.21E-06	16.18794	10.84289	25.83411	0	0	0
8	1.434427	1.00042	2.07922	4.91E-05	3.83E-05	0.000126	0.033545	0.008323	0.08367	0.000687	0.000113	0.002811	1.095609	0.498277	2.088517	0	0	0
11	2.220885	1.766681	3.273452	0.071183	0.051171	0.108693	0.526074	0.378794	0.738113	0.011115	0.005352	0.070524	95.38295	60.86864	196.35	1.29E-12	0	3E-06
16	3.228499	2.578186	3.500186	0.080786	0.066265	0.095481	1.869284	1.194802	3.099439	0.042323	0.020067	0.106932	10620.97	6087.331	25470.7	0.78159	0.201051	1.442341
23	3.876497	3.627642	5.121096	0.162836	0.136616	0.256462	5.077082	3.088031	6.151145	0.114184	0.066847	0.29845	134487.2	102616.5	168243.1	2.000865	0.745213	3.332835
32	5.036161	4.13138	5.897272	0.165304	0.106788	0.289629	10.34213	6.913191	11.36176	0.544723	0.403828	0.840245	742630.1	610630.6	1012909	250.1509	190.6067	426.403
45	5.701876	5.111698	7.368463	0.358346	0.250488	0.708917	12.75143	9.600907	14.93657	0.653636	0.369596	1.76487	2738219	2138822	4414920	7637.162	4961.956	12041.71
64	7.495188	5.6383	10.69307	0.416156	0.301762	0.699594	30.28059	24.12713	33.40064	3.475529	1.707864	4.665606	4743357	3012023	5882629	8057.364	4916.09	11182.41
91	8.864287	7.117587	10.31774	0.77189	0.555488	1.035036	44.58211	35.97465	52.64642	2.386628	1.219378	5.902843	9331831	6654515	14070512	24451.83	13415.61	33165.29
128	9.789019	8.631198	11.01374	0.950736	0.579278	3.372207	47.43589	38.67716	59.85028	2.478982	1.485272	5.763992	16902382	14519318	19515990	104054.6	73347.27	168427.2
181	10.87832	9.650047	11.8923	0.920029	0.757889	1.339574	86.15949	70.57995	91.35789	9.920937	5.486044	14.59588	37719195	26248072	44424153	192131.8	113883.4	393496.1
256	9.989655	8.65758	11.01048	1.234189	0.811789	2.164375	109.5012	97.60155	121.2978	14.25429	10.9809	18.43292	58545302	42463465	80141912	405251.2	219724	499345.8
362	9.769204	8.119372	10.12987	1.482708	1.304943	2.008557	165.1897	143.5717	197.7763	25.32933	12.99531	37.90326	62232315	42421233	75981956	442911.8	297853.7	606627
512	11.1587	10.19798	13.05704	2.346277	1.516275	3.212432	211.9952	168.6138	262.1177	35.3223	25.49719	45.12052	1.28E+08	1.02E+08	1.59E+08	2120318	1526511	2674374
724	11.55646	10.50247	12.93461	2.845782	2.020309	4.265917	357.9144	316.7906	391.814	57.41544	36.01807	65.75255	1.65E+08	1.11E+08	2.15E+08	1236674	644221.4	1794250
1024	10.34284	9.634785	11.56259	2.48899	1.948666	3.003312	445.4855	380.9291	487.1626	61.10358	42.7291	104.3655	2.92E+08	2.2E+08	3.61E+08	5625930	3141020	13007876
1448	12.08587	11.49215	12.47759	3.034674	2.339189	3.44734	648.2832	593.6103	750.7902	97.51595	82.16361	172.3268	3.41E+08	2.72E+08	4.73E+08	3262060	2584749	9977287

Dims.	Brown						Chung-Reynolds						Deflected Corrugated Spring					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	4.44E-16	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1
3	8.88E-16	8.88E-16	8.88E-16	8.88E-16	8.88E-16	8.88E-16	0	0	0	0	0	0	-0.84334	-0.9859	-0.84334	-1	-1	-1
4	7.99E-15	1.33E-15	1.16E-14	1.33E-15	1.33E-15	1.33E-15	0	0	1.26E-21	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
6	1E-11	3.69E-12	1.68E-11	2.22E-15	2.22E-15	2.22E-15	8.12E-16	1.2E-17	1E-14	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
8	2.18E-08	6.97E-09	5.34E-08	3.11E-15	3.11E-15	3.11E-15	2.18E-10	4.43E-11	2.01E-09	0	0	0	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334	-0.84334
11	0.000191	9.75E-05	0.000498	4.44E-15	4.44E-15	4.44E-15	1.8E-10	3.46E-11	4.29E-08	0	0	0	-0.84334	-0.84334	-0.37336	-0.84334	-0.84334	-0.84334
16	0.004794	0.002543	0.007505	1.64E-11	3.97E-12	6.87E-11	148.8304	99.91507	192.799	2.73E-12	1.15E-12	8.49E-12	-0.6078	-0.84334	-0.37336	-0.84334	-0.84334	-0.84334
23	0.231587	0.125779	0.34025	1.02E-05	4.76E-06	2.82E-05	2067.602	1545.519	3417.45	5.23E-09	6.93E-10	7.63E-08	-0.37293	-0.57205	-0.36706	-0.84328	-0.84333	-0.84321
32	2.365778	1.601448	2.801667	0.005928	0.003588	0.019759	128899.1	77084.6	194535.4	0.010905	0.005047	0.097007	-0.33955	-0.37255	0.011451	-0.84329	-0.84333	-0.84268
45	4.836483	3.497127	5.591305	0.138735	0.100149	0.21337	1060039	710678.4	2621419	8.553046	5.425813	15.19184	-0.3642	-0.37326	0.409939	-0.84334	-0.84334	-0.84332
64	51.24288	38.27108	71.02543	0.982365	0.798228	1.148263	5041718	3761709	8418667	48.33666	18.07354	78.86616	0.410702	-0.0856	1.84926	-0.84271	-0.84332	-0.65303
91	205.1248	165.7165	254.554	4.282202	3.841028	5.035857	16297916	10319399	21724312	88.66872	20.78383	184.1328	1.506543	0.410126	1.641335	-0.60776	-0.84333	-0.37336
128	1477.448	1198.801	1786.3	18.19488	15.86936	25.94103	1.17E+08	66797942	2.28E+08	2112.52	1279.249	5475.766	4.639484	3.985661	4.658075	-0.37336	-0.37336	0.409922
181	7196.821	5343.032	9412.196	209.0628	146.9134	315.1454	2.7E+08	1.49E+08	5.71E+08	12882.48	5057.931	19175.96	4.639498	2.920895	6.67653	-0.37336	-0.5019	-0.37336
256	9731.347	8460.519	10686.62	1372.658	1033.48	2058.59	3.89E+08	2.14E+08	6.5E+08	7407.955	3229.608	28431.74	3.269249	1.538295	5.548629	-0.40964	-0.84211	-0.37211
362	23034.48	19166.66	27771.53	5018.834	4203.766	5423.506	1.19E+09	5.56E+08	1.81E+09	138000.3	35738.23	416477	4.798906	2.764536	7.10192	-0.37317	-0.80175	0.162105
512	51746.56	36814.92	59130.32	11085.78	10309.1	11543.66	3.54E+09	1.63E+09	4.57E+09	831522.6	580135.8	1942495	7.944024	4.666596	14.90037	0.056593	-0.37058	0.423884
724	95761.13	77395.24	277140.3	17796.1	17419.66	19008.88	8.5E+09	4.3E+09	1.05E+10	2632651	410721.2	3424364	19.59229	15.35512	27.27985	2.03566	0.416447	3.104271
1024	149695.4	72780.33	359710.6	26316.31	26065.75	27252.77	1.75E+10	1.2E+10	2.47E+10	2172701	464959.9	4134537	21.81295	19.86412	27.102	1.506499	0.409943	2.916346
1448	337634.4	181555.1	688702.5	39393.45	38855.88	41225.3	1.6E+10	6.46E+09	4.15E+10	5726810	898836.6	12503192	45.14507	37.87081	50.21905	2.916346	1.506501	5.148502

Table B.21: PSO Descriptive Statistics: Exponential to Rastrigin Functions (10,000 Evaluation Tuning)

	Exponential						Griewank						Inverted Cosine Wave						Levy						Qing						Rastrigin					
Dims.	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	-1	-1	-1	-1	-1	-1	0	0	0.007396	0	0	0.001849	-1	-1	-1	-1	-1	-1	6.73E-14	6.73E-14	6.73E-14	6.73E-14	6.73E-14	6.73E-14	6.01E-09	7.88E-10	6.01E-09	6.01E-09	7.88E-10	6.01E-09	0	0	0	0	0	0
3	-1	-1	-1	-1	-1	-1	0.009873	0.007401	0.02774	0.008627	0.007396	0.014171	-1.99954	-1.99993	-1.99712	-2	-2	-2	9.6E-14	9.6E-14	9.6E-14	9.6E-14	9.6E-14	9.6E-14	6.01E-09	7.92E-10	6.01E-09	6.01E-09	7.92E-10	6.01E-09	0.000858	9.21E-05	0.013201	0	0	4.62E-11
4	-1	-1	-1	-1	-1	-1	0.064076	0.047024	0.086852	0.064076	0.043128	0.084413	-2.73805	-2.98429	-2.47606	-3	-3	-2.86905	1.25E-13	1.25E-13	1.25E-13	1.25E-13	1.25E-13	1.25E-13	1.14E-08	6.27E-09	1.71E-08	8.84E-09	6.27E-09	1.14E-08	0.995092	0.029269	1.222087	0	0	3.84E-08
6	-1	-1	-1	-1	-1	-1	0.166292	0.116668	0.210575	0.166287	0.116424	0.210574	-3.69046	-4.01723	-3.69045	-3.95237	-4.54166	-3.69046	0.000128	8.29E-05	0.000203	1.82E-13	1.82E-13	1.82E-13	1.26E-08	7.47E-09	1.46E-08	1.26E-08	7.47E-09	1.46E-08	3.1647	2.418679	5.499264	0.727525	8.03E-06	1.989918
8	-1	-1	-1	-1	-1	-1	0.173959	0.10334	0.270237	0.172371	0.102768	0.268833	-5.16642	-5.42132	-5.15673	-5.29758	-5.42855	-5.16664	0.003568	0.001127	0.023938	2.4E-13	2.4E-13	2.4E-13	1.84E-05	5.69E-06	9.53E-05	2.32E-08	2.03E-08	2.74E-08	5.749938	4.452263	8.261465	1.503626	0.007926	4.477318
11	-1	-1	-1	-1	-1	-1	0.126952	0.081465	0.273245	0.115644	0.079411	0.270568	-7.35262	-7.38477	-7.09817	-7.38092	-7.44466	-7.3807	0.065098	0.006984	0.115463	1.03E-12	3.26E-13	0.089528	0.018605	0.001193	0.037067	5.2E-08	4.57E-08	6.24E-08	13.18299	8.793167	15.48717	9.951142	7.959671	15.17312
16	-1	-1	-0.99999	-1	-1	-1	0.23282	0.183889	0.312324	0.050668	0.043443	0.068098	-9.63475	-10.0095	-9.21373	-10.0764	-10.5398	-9.43773	0.418053	0.310782	0.529898	0.179057	0.089529	0.268585	136.6594	87.38005	206.9802	0.001577	1.02E-07	0.397236	22.89311	20.54992	25.20317	21.04278	13.39332	24.03313
23	-0.99921	-0.99946	-0.99884	-0.99995	-0.99997	-0.99986	1.0814	1.05804	1.128415	0.413594	0.247327	0.565211	-11.5149	-12.1051	-10.5371	-12.6176	-13.2225	-11.3944	1.159321	0.767481	1.381891	0.316381	0.116491	0.479806	43156.67	14050.87	67823.61	165.5926	94.30403	889.6188	40.98029	32.37649	47.71848	38.39651	26.35375	45.5286
32	-0.99547	-0.9963	-0.99223	-0.99958	-0.99982	-0.9993	2.134033	1.976167	2.671688	1.130372	1.077296	1.171199	-14.972	-16.2812	-13.5581	-15.9653	-17.0099	-13.9903	3.31619	2.300399	4.504082	1.123785	0.69195	1.639901	1345621	673898.1	3238168	54092.72	15460.11	145073.1	63.05042	53.77137	78.06497	55.81373	46.68724	63.10553
45	-0.97137	-0.97424	-0.96642	-0.99621	-0.99674	-0.99563	5.847318	5.174685	7.953013	2.673171	2.253384	3.657444	-20.8872	-22.4317	-19.683	-22.2665	-24.6349	-20.998	4.742456	3.974166	6.90398	2.235251	1.478898	3.337563	2503424	140972.3	5852414	9773.205	8181.566	10789.53	115.7537	105.6342	131.7358	99.63084	90.99336	110.5357
64	-0.92828	-0.94601	-0.91821	-0.99988	-0.99991	-0.99977	17.7628	16.0667	20.98417	11.10522	8.63923	12.5354	-23.9246	-27.4195	-22.2634	-28.8599	-32.4902	-25.6395	12.69215	9.827166	16.33332	5.997833	4.377923	8.900632	1.06E+08	74859149	1.34E+08	7048592	5717644	12944932	248.2872	238.1962	265.5036	230.6384	213.0599	245.3427
91	-0.87288	-0.9088	-0.80139	-0.99922	-0.99933	-0.99841	37.38016	34.62201	39.85112	22.86225	19.65599	26.04439	-32.4267	-35.7023	-29.1207	-35.9137	-38.0434	-31.9002	24.30398	20.59626	28.29457	11.22209	8.694893	14.56622	3.22E+08	2.72E+08	4.6E+08	97299913	68428451	1.65E+08	443.1201	414.449	467.2047	346.7643	323.1432	375.0479
128	-0.73302	-0.78044	-0.586	-0.99885	-0.9992	-0.99847	50.56928	28.48079	75.13092	1.038361	0.755874	1.137196	-37.1445	-39.8255	-34.5296	-40.0168	-43.3841	-36.5411	43.4316	36.16271	50.76171	23.07573	19.6225	29.47112	9.17E+08	7.67E+08	1.11E+09	26546771	19812929	48063505	711.1505	677.9178	756.3926	678.0071	630.2055	708.5406
181	-0.5233	-0.5844	-0.48159	-0.91115	-0.93301	-0.88631	80.45327	53.8958	100.6197	1.181237	1.116068	1.319882	-44.7009	-51.4281	-41.9214	-47.9903	-54.3418	-44.4997	76.30732	67.43056	92.25596	54.73335	42.84035	61.55935	1.53E+09	1.19E+09	2.07E+09	7.79E+08	5.49E+08	1.02E+09	1167.865	1136.235	1208.787	1060.52	1038.929	1097.832
256	-0.33297	-0.37382	-0.28822	-0.90759	-0.92656	-0.86765	171.851	108.0065	197.2372	1.997592	1.75961	2.126831	-49.5232	-54.3564	-45.1877	-56.2078	-61.719	-53.7199	75.99778	52.81433	130.5356	23.69476	23.54151	24.19891	2.65E+09	2.01E+09	3.51E+09	1.34E+09	9.39E+08	1.55E+09	1752.717	1571.89	1911.889	597.3964	452.8209	828.1616
362	-0.2278	-0.29547	-0.17502	-0.95528	-0.96504	-0.93313	247.8617	225.4157	314.5837	2.642252	1.900447	3.996416	-70.0233	-74.0898	-60.8422	-77.7959	-79.103	-67.8859	186.5754	104.8641	255.169	34.33295	31.86656	50.89709	5.34E+09	4.93E+09	5.92E+09	2.68E+09	2.44E+09	2.88E+09	2396.007	2083.867	2559.581	709.1947	580.3129	924.7604
512	-0.09603	-0.1758	-0.05907	-0.95612	-0.97175	-0.93778	437.6936	385.2004	566.9588	4.476348	2.907189	7.395712	-73.9755	-82.9704	-65.3564	-75.7963	-85.1609	-68.0972	241.032	203.3505	274.1597	53.39326	50.76393	58.41922	9.39E+09	7.56E+09	1.11E+10	2.18E+09	1.38E+09	3.27E+09	3095.908	2486.92	3584.339	1123.768	850.894	1542.504
724	-0.06183	-0.08144	-0.01925	-0.90568	-0.91738	-0.8516	610.2521	524.7991	721.9029	10.57992	6.611398	15.40285	-83.8997	-108.685	-79.7353	-105.656	-127.961	-97.6655	338.3609	277.3841	444.4827	72.34372	66.85411	88.88392	1.19E+10	1.06E+10	1.38E+10	7.77E+09	6.88E+09	8.3E+09	4407.009	4016.9	5164.619	1351.934	883.1081	1670.61
1024	-0.00015	-0.00301	-1.2E-05	-0.57776	-0.74095	-0.25749	537.6913	347.3365	781.8214	10.05267	4.88808	47.73208	-122.371	-131.073	-100.606	-158.958	-177.558	-130.152	527.2897	473.6379	630.5301	94.7695	92.93828	102.9777	2.08E+10	1.71E+10	2.32E+10	1.31E+10	1.05E+10	1.52E+10	5977.903	4830.225	7240.728	2539.696	1679.84	3148.238
1448	-0.00042	-0.00159	-0.00026	-0.40258	-0.49519	-0.34756	1230.498	976.2399	1497.438	26.90886	19.04275	34.13093	-130.845	-160.697	-94.9765	-407.362	-529.367	-317.992	421.9267	320.3638	583.7439	132.8819	132.2152	135.4892	2.62E+10	2.29E+10	3.13E+10	1.85E+10	1.62E+10	2.31E+10	8626.909	7890.938	9350.79	3438.963	1985.536	3850.188

Table B.22: PSO Descriptive Statistics: Exponential to Rastrigin Functions (50,000 Evaluation Tuning)

	Exponential						Griewank						Inverted Cosine Wave						Levy						Qing						Rastrigin					
Dims.	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	-1	-1	-1	-1	-1	-1	0.007494	0.000494	0.008181	0	0	0	-1	-1	-1	-1	-1	-1	6.73E-14	6.73E-14	6.73E-14	6.73E-14	6.73E-14	6.73E-14	6.01E-09	7.88E-10	6.01E-09	6.01E-09	7.88E-10	6.01E-09	0	0	0	0	0	0
3	-1	-1	-1	-1	-1	-1	0.032472	0.023436	0.03813	0.009861	0.007396	0.014171	-2	-2	-1.99999	-2	-2	-2	9.6E-14	9.6E-14	9.6E-14	9.6E-14	9.6E-14	9.6E-14	6.01E-09	7.92E-10	6.01E-09	6.01E-09	7.92E-10	6.01E-09	0.058254	0.002053	0.99624	0	0	7.85E-11
4	-1	-1	-1	-1	-1	-1	0.180482	0.129774	0.21649	0.035732	0.012982	0.063446	-2.75125	-2.96779	-2.68605	-3	-3	-3	1.25E-13	1.25E-13	4.44E-13	1.25E-13	1.25E-13	1.25E-13	1.66E-08	1.14E-08	2.47E-08	8.84E-09	6.27E-09	1.27E-08	1.06285	0.181529	2.01851	1.32E-05	2.59E-06	7.93E-05
6	-1	-1	-1	-1	-1	-1	0.570562	0.350183	0.706194	0.154304	0.088248	0.179498	-3.67666	-3.76598	-3.64092	-4.86308	-4.9997	-4.39536	0.000145	8.35E-05	0.000371	1.82E-13	1.82E-13	1.82E-13	1.06E-05	3.36E-06	5.33E-05	1.61E-08	7.47E-09	1.87E-08	3.125613	1.996125	4.177852	0.000218	2.26E-05	0.002345
8	-1	-1	-1	-1	-1	-1	0.515977	0.415514	0.629544	0.18461	0.133084	0.263412	-5.03515	-5.1851	-4.43324	-5.7402	-6.47083	-5.16664	0.053778	0.008208	0.090596	5.49E-11	1.05E-11	3.01E-10	0.01296	0.008107	0.101221	2.71E-08	2.1E-08	2.83E-08	7.604324	6.570615	10.30658	0.021057	0.013167	0.042856
11	-1	-1	-1	-1	-1	-1	0.917254	0.850059	0.95731	0.118166	0.08187	0.156846	-6.58787	-7.18056	-6.15974	-7.63713	-7.96938	-7.38079	0.21681	0.112702	0.313555	0.00016	4.77E-05	0.000336	78.33715	56.94047	113.2734	4.95E-08	4.25E-08	5.63E-08	18.65833	13.58601	23.82735	2.028792	0.841562	5.112894
16	-0.99943	-0.99962	-0.9992	-1	-1	-1	1.15384	1.115576	1.227335	0.032114	0.012343	0.081211	-8.82477	-9.15575	-8.1521	-11.07	-11.5827	-10.83	0.762663	0.609297	1.246252	0.005159	0.003701	0.014062	704.7179	455.3156	1274.898	2.45E-06	1.04E-07	0.000874	43.52122	33.13587	51.44237	0.748966	0.289904	1.707911
23	-0.99779	-0.99839	-0.99699	-1	-1	-1	1.801092	1.660648	2.081748	0.02926	0.015666	0.044884	-10.7879	-11.4987	-9.4694	-15.2826	-15.7797	-14.6079	2.186482	1.410882	4.019438	0.033219	0.019972	0.198498	614370.3	321443.6	867736.3	2.022795	0.907968	4.833063	94.68604	82.09222	117.1875	18.35616	3.908238	26.99548
32	-0.98694	-0.99044	-0.98374	-0.99999	-0.99999	-0.99998	4.349765	3.742509	5.03481	0.321082	0.217799	0.416732	-11.4981	-11.9345	-9.98956	-19.3474	-21.1467	-18.2969	2.929574	2.541531	3.644838	0.769307	0.470283	1.036343	13520144	11439462	24555450	258.342	194.3611	379.6026	128.5929	114.6123	151.7321	39.00746	31.22215	45.3022
45	-0.9641	-0.96926	-0.94544	-0.99992	-0.99996	-0.99987	8.18912	4.82681	12.60085	0.916036	0.778163	0.990621	-14.8224	-16.351	-13.8232	-26.152	-26.937	-24.8491	8.454145	7.382913	10.93652	1.79838	1.40944	2.049581	29634690	22914615	55436498	7400.72	6714.428	8164.696	247.9172	221.9375	276.8838	64.51159	59.67037	75.50572
64	-0.87126	-0.90146	-0.84266	-0.9998	-0.99988	-0.99976	19.72679	16.97985	28.16394	0.899045	0.712508	1.0306	-19.9952	-21.3258	-18.8758	-32.5283	-33.5373	-30.8578	20.29463	13.1134	27.73812	3.519182	3.087554	3.925355	1.5E+08	41820055	6.75E+08	27363.4	23921.9	29257.58	375.5084	369.4724	402.652	124.6147	109.1054	138.3632
91	-0.82169	-0.83852	-0.67548	-0.99974	-0.99981	-0.99963	34.36764	23.73098	47.30382	1.057167	1.028884	1.102149	-22.7235	-23.6878	-20.8758	-39.6179	-41.4226	-36.066	25.87149	19.26271	37.70328	8.263531	8.045691	8.438603	4.05E+08	2.16E+08	7.15E+08	86971.12	75174.94	96771.1	598.0688	547.1064	624.1831	169.0573	149.461	214.5512
128	-0.69547	-0.74454	-0.56271	-0.99914	-0.99933	-0.99839	91.27557	69.56064	105.1357	1.189946	1.144515	1.33909	-25.0237	-28.4442	-23.7592	-45.7515	-50.2782	-42.229	40.17722	29.81586	87.23882	11.12214	10.95802	11.55567	7.95E+08	4.09E+08	2.09E+09	243634.8	225209.9	272322.7	771.1512	693.1832	849.113	205.2604	143.1022	322.4453
181	-0.52641	-0.64471	-0.35886	-0.99788	-0.99881	-0.99717	137.4777	91.28356	186.4279	1.628779	1.422632	1.812056	-34.5791	-37.9051	-27.034	-70.9327	-73.5315	-56.5586	56.15818	35.19716	93.36714	16.6738	16.44665	16.88235	1.15E+09	4E+08	2.74E+09	718746.4	630634.6	943986.5	1235.046	1113.808	1336.239	408.736	271.6825	487.2682
256	-0.30445	-0.46715	-0.13154	-0.997	-0.99845	-0.99579	170.5574	134.4427	268.8486	1.670833	1.457625	2.23562	-38.143	-42.5002	-36.0082	-77.056	-83.2342	-69.9344	139.0416	105.6409	164.7103	25.07546	23.45222	28.41976	7.42E+09	4.66E+09	9.14E+09	2153743	1908930	2636567	1726.564	1608.9	1971.099	491.1124	406.2289	528.8326
362	-0.30109	-0.41683	-0.18226	-0.98455	-0.99055	-0.97804	269.7298	227.0112	335.1361	2.766827	2.077808	3.373104	-64.9282	-80.2224	-38.6674	-124.216	-133.761	-96.5153	150.2848	130.344	193.0359	34.27323	33.53107	36.32198	1.28E+10	9.65E+09	2.04E+10	6996132	6680426	8069162	2256.286	2057.298	2563.595	629.6738	507.4129	766.0198
512	-0.11145	-0.1502	-0.02691	-0.97503	-0.98643	-0.96724	509.9375	372.1963	674.4635	4.867965	3.392151	6.770069	-58.1047	-76.664	-44.3442	-152.471	-226.193	-123.351	227.5788	114.0003	255.5899	47.09557	46.97317	47.58283	1.66E+10	7.1E+09	2.4E+10	18847593	17903365	20884972	3431.777	3087.381	3591.145	936.2493	736.0287	1122.996
724	-0.0218	-0.03303	-0.00431	-0.97004	-0.97716	-0.93413	585.7511	527.816	935.8041	7.05984	4.676653	13.66934	-68.0018	-96.1275	-57.6093	-267.974	-337.08	-227.222	270.5178	174.5901	396.9861	66.45878	66.20342	69.77352	3.23E+10	1.38E+10	4.2E+10	60336424	55800245	69528347	4609.646	3861.839	5343.693	1367.408	1114.246	2102.404
1024	-0.00319	-0.02442	-0.00091	-0.96189	-0.97238	-0.92672	1035.695	653.8732	1464.353	9.105408	5.007342	14.28438	-97.2175	-149.415	-62.131	-229.2	-324.674	-174.941	599.9543	500.9404	784.7384	115.4451	105.645	122.518	5.88E+10	4.29E+10	8.46E+10	2.09E+08	1.73E+08	2.34E+08	7118.307	6548.504	7937.327	2328.865	1768.007	2789.343
1448	-0.00056	-0.01172	-0.00018	-0.95453	-0.97789	-0.57707	1604.024	1314.933	1860.473	31.62722	22.77384	68.76743	-147.459	-163.918	-114.596	-507.144	-659.49	-352.433	979.1104	733.6835	1152.892	145.714	141.2614	158.9547	6.02E+10	3.13E+10	8.36E+10	4.86E+08	3.61E+08	5.59E+08	10027.55	8566.627	11341.55	2647.076	2185.352	3115.716

Table B.23: PSO Descriptive Statistics: Rosenbrock to Sum Squares Functions (10,000 Evaluation Tuning)

Dims.	Rosenbrock						Schwefel						Sphere						Sum of Different Powers						Sum Squares					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	7.03E-11	7.03E-11	7.03E-11	7.03E-11	7.03E-11	7.03E-11	0.000794	0.00061	0.002199	5.46E-06	5.46E-06	5.46E-06	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1.41E-10	1.41E-10	1.41E-10	1.41E-10	1.41E-10	1.41E-10	16.94301	0.007949	118.4601	8.18E-06	8.18E-06	8.29E-06	0	0	0	0	0	0	2.02E-28	0	9.98E-23	0	0	0	0	0	0	0	0	0
4	2.11E-10	2.11E-10	2.11E-10	2.11E-10	2.11E-10	2.11E-10	122.2769	99.18668	238.4751	9.013231	0.025357	118.4835	0	0	0	0	0	0	1.62E-19	4.48E-21	6.51E-18	0	0	0	0	0	0	0	0	0
6	1.269813	0.043295	4.00095	3.87E-10	3.52E-10	3.973941	237.2701	118.4476	355.4155	236.8767	118.4384	355.315	3.97E-11	1.14E-11	8.18E-11	0	0	0	9.45E-13	9.5E-14	4.99E-12	0	0	0	1.19E-09	4.05E-10	4.37E-09	0	0	0
8	4.61961	3.267177	6.975846	4.92E-10	4.92E-10	6.25E-07	525.9404	353.0395	796.0925	473.7534	236.8767	596.9463	2.27E-08	1.14E-08	5.58E-08	0	0	0	6.73E-11	3.63E-12	3.05E-10	0	0	0	2.55E-08	8.4E-09	4.77E-08	0	0	0
11	9.939332	7.259112	63.71413	3.64267	2.209837	8.755116	953.1447	712.651	1195.133	939.9302	667.3639	1077.714	8.5E-09	2.96E-09	3.92E-08	0	0	0	1.93E-09	3.53E-10	4.64E-09	3.25E-16	2.23E-17	2.16E-14	2.94E-05	1.3E-05	9.75E-05	0	0	0
16	19.75025	16.30029	39.04602	13.60381	12.33157	15.04748	1707.991	1586.834	1936.904	1397.724	1224.139	1685.463	0.000494	0.000358	0.000979	1.57E-06	1.18E-07	1.54E-05	9.45E-09	5.69E-09	4.63E-08	1.42E-11	1.94E-12	2.25E-10	0.284171	0.107431	0.463435	1.7E-06	4.05E-08	6.08E-05
23	120.5999	83.14343	161.8369	57.32126	26.49695	83.41796	3106.616	2900.2	3443.71	2303.324	1999.323	2725.175	0.025353	0.012279	0.031815	0.003243	0.000957	0.005123	4.33E-08	2.37E-08	1.27E-07	5.36E-10	6.94E-11	2.15E-09	3.060071	2.30641	5.226287	1.132885	0.439784	1.753034
32	444.9872	370.7818	709.6938	140.9425	90.17301	191.8829	4802.982	4481.779	5029.992	3338.996	3102.343	3591.824	0.341979	0.239734	0.418829	0.133111	0.092944	0.212098	6.21E-08	1.86E-08	3.06E-07	5.66E-10	1.29E-10	1.88E-09	29.12336	20.97683	38.10832	5.770461	2.763861	8.552789
45	2755.207	1986.935	3299.214	824.4558	683.7056	1041.539	8079.015	7025.06	8720.32	6990.796	5935.617	8131.573	1.35851	1.099524	1.711948	0.30181	0.202078	0.426707	6.06E-08	1.9E-08	1.63E-07	2.29E-09	4.28E-10	4.15E-09	149.1227	124.7297	168.8149	70.89633	53.05354	89.97264
64	11282.87	9640.38	14255.01	2714.322	2156.641	3175.508	12876.82	11668.45	14115.1	10802.45	9483.21	12325.4	4.480179	4.031144	5.183949	1.451746	1.184769	1.756355	2.58E-07	1.07E-07	4.77E-07	5.86E-09	1.18E-09	5.18E-08	452.941	412.1626	538.4499	243.1941	221.8342	280.1777
91	37129.19	28908.71	44320.4	8549.579	5274.583	11643.8	21111.43	19932.92	22488.03	17353.82	15574.86	18115.47	11.04507	7.778414	14.61978	0.035316	0.026828	0.060683	9.96E-08	3.31E-08	2.32E-07	4.4E-09	1.47E-09	1.26E-08	1611.256	1439.642	1854.662	118.0969	66.49456	158.3953
128	65688.96	52214.24	80289.42	26953.42	19360.3	34654.07	32536.37	31351.81	33674.18	28607.46	27996.75	29782.84	16.06282	12.27921	21.07572	0.222471	0.137718	0.335007	2.55E-07	6.47E-08	4.5E-07	6.79E-09	1.01E-09	2.52E-08	3904.846	3679.74	4364.549	1812.383	1636.187	2019.174
181	156416.7	135335.8	195499.1	83329.96	74960.31	106999.5	51603.84	49997.1	53532.87	48879.63	47389.67	50369.47	24.9964	15.92571	33.27571	0.106985	0.072074	0.20238	2.46E-07	1.28E-07	5.35E-07	2.98E-08	7.81E-09	5.88E-08	11172.09	9341.986	12409.27	692.4986	468.2956	909.085
256	290302.3	246925.6	361101.1	170289.6	145878.8	223646.1	77714.49	76358.97	79684.24	75336.04	73279.94	76489.59	49.93659	37.75297	69.35901	0.204679	0.121385	0.288158	3.7E-07	1.32E-07	1.31E-06	9.99E-09	5.96E-09	3.14E-08	24744.8	16862.86	26512.97	160.3594	115.3772	293.5507
362	444372.9	395181.5	534310	152618.6	132020.9	209736.8	113654.7	112118.6	116593.9	107933.5	105635.8	110529	71.29329	33.41633	82.76354	0.836367	0.345948	1.78097	1.62E-07	3.73E-08	3.25E-07	6.06E-08	1.53E-08	9.92E-08	50295.44	35530.05	67610.88	351.5794	157.5992	555.3305
512	861878.8	732943.1	1000880	443595	429210.7	538910.1	168057.3	166033.4	171406.7	163213.3	161961.7	166616.7	110.1748	67.37138	147.1403	0.489484	0.224395	1.270085	3.83E-07	1.07E-07	9.78E-07	2.95E-08	1.2E-08	5.52E-08	121493.4	107777.1	140037.2	5492.58	3755.518	9641.441
724	1390589	1216691	1527043	926643.3	850712.1	1080805	253203.9	249553.8	255080.7	238701.8	237464	243155.2	179.9624	103.3115	207.7952	0.561663	0.255048	26.83608	3.09E-07	8.81E-08	8.77E-07	7.72E-09	2.68E-09	2.86E-08	224055.5	179497.8	284450.5	3176.727	1873.769	12924.72
1024	2147281	1824566	2344240	1062687	912795.8	1205568	365450.1	364160.4	369432.5	354762.6	351294.9	357404.7	251.179	190.5918	283.0043	5.656843	3.166025	13.20809	3.41E-07	2.45E-07	5.95E-07	7.14E-09	1.09E-09	2.64E-08	582334.4	340493.6	697915.7	14178.39	6134.848	57557.68
1448	3261649	2820641	3425333	2024217	1723848	2230353	531465.9	527027	534251.9	521323.8	518570.4	525509.5	360.6326	329.5853	409.2758	9.671846	5.347577	13.5586	2.57E-07	1.3E-07	3.53E-07	2.81E-08	1.72E-08	5.11E-08	889789.5	726527.3	1228517	40224.78	28884.21	45770.63

Table B.24: PSO Descriptive Statistics: Rosenbrock to Sum Squares Functions (50,000 Evaluation Tuning)

Dims.	Rosenbrock						Schwefel						Sphere						Sum of Different Powers						Sum Squares					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	9.19E-11	9.19E-11	9.78E-11	9.19E-11	9.19E-11	9.19E-11	0.000587	0.000269	0.002317	5.46E-06	5.46E-06	5.46E-06	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	5.56E-08	6.61E-10	0.02394	1.84E-10	1.84E-10	1.84E-10	0.070135	0.030219	118.5901	8.45E-06	8.3E-06	4.1E-05	0	0	0	0	0	0	9.79E-27	1.55E-28	1.6E-22	0	0	0	0	0	0	0	0	0
4	4.96E-10	2.76E-10	0.027368	2.76E-10	2.76E-10	4.96E-10	119.8644	118.8754	154.6079	0.021242	0.000365	118.4384	0	0	9.31E-14	0	0	0	5.38E-20	1.33E-20	1.98E-17	0	0	0	0	0	2.66E-12	0	0	0
6	5.456695	4.12065	16.78132	4.59E-10	4.59E-10	1.37E-06	300.6749	210.4877	476.6223	236.8772	118.4394	268.0146	1.02E-10	3.23E-11	2.35E-10	0	0	0	2.06E-12	4.4E-13	2.35E-11	0	0	0	1.04E-09	3.14E-10	5.68E-09	0	0	0
8	6.243646	5.188989	10.6403	3.28E-09	6.43E-10	3.985888	386.1956	313.5137	681.674	355.3445	207.2849	356.8686	3E-08	2.12E-08	2.29E-07	0	0	0	1.75E-10	2.71E-11	9.57E-10	0	0	0	1.03E-06	4.63E-07	1.89E-06	0	0	0
11	29.0376	13.93591	75.46278	2.961962	0.239819	5.175136	1114.002	780.3159	1231.381	772.1288	475.2714	1125.827	9.25E-05	5.44E-05	0.000203	0	0	0	7.37E-09	9.23E-10	2.31E-08	3.56E-15	4.79E-18	1.95E-13	0.012584	0.006727	0.033906	0	0	0
16	312.0023	235.8473	472.7701	15.35069	12.33228	75.40054	1961.927	1675.047	2202.537	1190.035	1019.901	1315.768	0.006267	0.004496	0.009196	2.74E-11	0	2.42E-09	1.21E-08	2.21E-09	1.91E-08	7.86E-12	1.58E-12	4.85E-11	0.741618	0.469794	1.074528	1.13E-09	1.82E-10	5.98E-09
23	4650.353	2570.046	15524.03	24.92689	23.51669	27.17511	3110.914	2813.363	3432.893	2091.864	1697.948	2650.86	0.144906	0.114794	0.240844	5.98E-07	2.93E-07	9.17E-07	3.45E-08	1.08E-08	7.48E-08	2.08E-10	9E-12	1.75E-09	8.452844	6.793461	14.31621	0.001465	0.000505	0.004936
32	69263.55	26680.18	102392.3	38.60697	33.44564	43.27332	5324.149	4485.566	6141.763	3384.798	2975.672	4977.889	0.823367	0.700031	0.998145	0.001043	0.000658	0.001487	9.86E-08	1.81E-08	2.23E-07	6.94E-10	1.01E-10	1.35E-08	41.74902	25.96369	51.7301	0.202262	0.104733	0.392154
45	79879.4	40291.07	129260.8	55.94213	50.38272	62.47843	7968.939	7405.751	8754.215	5272.263	4622.876	6517.765	1.149851	0.809495	1.853535	0.002594	0.001592	0.003795	8.01E-08	1.83E-08	2.41E-07	4.28E-10	7.53E-11	2.97E-09	311.1083	241.5028	411.7198	0.852659	0.5263	1.0931
64	76709.48	26199.81	240746.3	75.90445	70.47684	81.90267	13358.31	12212.29	14608.08	9526.299	8597.707	10550.58	4.599455	3.175232	6.154265	0.010679	0.006086	0.018082	1.85E-07	2.73E-08	4.16E-07	1.56E-09	2.15E-10	1.72E-08	467.6991	336.1777	623.7749	2.234233	1.038307	2.860915
91	157440.8	100580	276053.7	107.1891	99.71879	118.1036	21790.26	20815.43	22941.46	15354.09	14310.65	16686.18	11.53952	6.168574	15.86658	0.015752	0.013261	0.029035	2.17E-07	1.03E-07	4.88E-07	1.26E-08	3.21E-09	4.25E-08	1783.315	1203.665	2719.412	6.07067	4.28464	8.858823
128	2865148	1580474	6053104	343.5276	253.3698	1289.389	34274.66	32488.18	35379.53	25474.65	24692.71	26499.27	20.85711	13.35555	33.84154	0.065394	0.042496	0.108385	2.15E-07	7.12E-08	4.66E-07	1.4E-08	1.37E-09	6.36E-08	4416.363	2396.813	5828.869	31.35077	15.65736	39.56301
181	4038201	1334208	7493143	515.6038	467.1916	612.7898	48901.42	47056.67	52361.34	37653.47	36061.2	46010.43	31.69453	27.20946	39.32099	0.130616	0.100966	0.277407	3.28E-07	1.5E-07	7.26E-07	1.42E-08	4.92E-09	5.79E-08	14470.79	9210.634	16381.17	39.78921	26.84616	57.00919
256	5581136	4097460	7306314	1860.181	804.1776	3131.006	78561.51	75757.47	80484.35	63469.43	59569.68	67294.01	65.40026	54.65423	78.15862	0.29956	0.101385	0.465507	3.4E-07	8.51E-08	7.63E-07	1.79E-08	4.31E-09	3.63E-08	29740.21	20763.53	34383.6	218.9007	163.654	319.3963
362	14162557	9680579	26732280	2275.062	918.7585	4140.509	115596.1	113792.2	118813.6	99439.6	96573.46	101877.1	94.02223	51.35865	123.9407	0.779376	0.612363	1.174975	4.33E-07	2.16E-07	7.19E-07	1.98E-08	6.07E-09	2.84E-08	47060.83	36725.04	64816.92	240.55	181.6739	400.99
512	17490673	13162530	40919125	5007.441	2868.149	20086.03	171703.3	164220	173703.8	149155.8	137203.6	156735	104.5445	73.54045	121.377	2.085531	1.283869	2.800503	2.38E-07	1.07E-07	1.53E-06	6.72E-09	3.36E-09	3.43E-08	91396.04	77389	138517.2	1657.393	707.2627	2235.271
724	29973684	18988677	48793694	16925.53	9175.303	34275.2	250029.9	248100.1	255037.9	227938.8	223942.9	230721.5	132.2229	120.9978	190.854	1.676943	1.167034	2.833149	6.6E-07	4.09E-07	1.15E-06	9.66E-08	3.43E-09	1.81E-07	270730.3	203487.5	320591.8	2499.092	1511.714	4239.43
1024	1.01E+08	53976157	1.46E+08	54932.55	18472.65	196503.7	371012.5	367936.9	373995.5	334775	328938.9	341110.9	239.1819	158.655	274.2961	2.187715	1.083912	11.07955	2.2E-07	7.13E-08	1.44E-06	6.44E-09	9.14E-10	2.7E-08	542364.1	435020.5	638538.8	3852.225	2741.633	11061.8
1448	1.46E+08	1.1E+08	2.31E+08	48102.76	31398.47	93919.74	530982.7	528650.8	534518.7	492533.7	486846.6	499669.3	409.9074	303.6601	484.0557	12.47843	4.513406	29.91898	5.41E-07	2.56E-07	1.82E-06	2.95E-08	5.56E-09	8.85E-08	848054.4	758259.2	968565	30505.88	22698.35	34681.09

Table B.25: RMHC Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (10,000 Evaluation Tuning)

Dims.	Ackley						Alpine1					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.000774	0.000347	0.001157	0.000143	8.54E-05	0.000245	2.5E-05	1.18E-05	5.16E-05	5.59E-06	3.39E-06	1.41E-05
3	0.001447	0.001005	0.002012	0.000228	0.00017	0.000354	8.68E-05	6.82E-05	0.000131	1.19E-05	5.05E-06	2.75E-05
4	0.002031	0.001621	0.002519	0.000347	0.000195	0.000465	0.000135	5.05E-05	0.000196	3.16E-05	2.36E-05	4.57E-05
6	0.005357	0.002921	0.006447	0.000661	0.000548	0.00083	0.000342	0.000252	0.00042	5.79E-05	4.35E-05	9.25E-05
8	0.004896	0.004277	0.005817	0.00086	0.000776	0.001243	0.000501	0.000421	0.000672	8.99E-05	6.47E-05	0.000113
11	0.009554	0.007746	0.0119	0.001516	0.001294	0.001923	0.001339	0.001016	0.001831	0.000201	0.000126	0.000288
16	0.019149	0.01492	0.022083	0.00179	0.001429	0.002583	0.003204	0.002836	0.004163	0.000431	0.000404	0.00052
23	0.033825	0.029908	0.040621	0.00336	0.002939	0.00387	0.00926	0.007502	0.011068	0.001097	0.000907	0.001242
32	0.067529	0.063701	0.081062	0.007909	0.007214	0.008508	0.022173	0.019752	0.026299	0.003331	0.002816	0.003838
45	0.132008	0.116165	0.167962	0.012358	0.010844	0.013429	0.069101	0.058165	0.081292	0.009825	0.009047	0.011008
64	0.370529	0.336452	0.441396	0.025521	0.024475	0.028143	0.196697	0.163489	0.220268	0.025685	0.023389	0.029372
91	0.914621	0.839287	1.013263	0.05139	0.049047	0.055306	0.53209	0.442794	0.631581	0.050837	0.048033	0.057773
128	1.718266	1.67386	1.907068	0.125875	0.114068	0.131102	1.362811	1.239834	1.563779	0.109729	0.10093	0.128099
181	2.656489	2.528155	2.720039	0.23316	0.213647	0.254706	3.657099	3.354073	4.226199	0.236921	0.22821	0.257628
256	3.708472	3.617474	3.793795	0.626615	0.590735	0.642582	9.393719	8.812817	9.631456	0.547636	0.508061	0.634715
362	5.100556	4.96402	5.27433	1.147382	1.113553	1.211919	22.92817	20.98876	25.06134	1.487486	1.375984	1.604117
512	7.116714	6.825382	7.394587	2.158961	2.098491	2.212212	55.55936	52.61452	59.79283	3.817803	3.59418	4.141977
724	9.563608	9.4002	9.700872	3.185792	3.144321	3.257137	133.6992	127.4487	136.3805	10.58229	10.17406	11.12459
1024	12.04075	11.9168	12.25009	4.268004	4.241191	4.393308	300.9105	291.9734	307.6618	25.84179	24.7796	26.95036
1448	14.41455	14.24708	14.62726	5.841251	5.784623	5.88867	615.4409	607.5178	624.7862	77.50374	75.77252	80.15943

Dims.	Bent Cigar						Brown					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	1.51E-06	8.89E-08	3.82E-06	2.22E-07	3.55E-13	3.55E-07	4.42E-12	1.12E-12	1.24E-11	9.09E-14	2.49E-14	3.33E-13
3	2.18E-05	8.28E-06	0.000108	7.11E-07	3.55E-07	2.31E-06	4.27E-11	1.57E-11	4.93E-10	3.97E-13	2.17E-13	8.16E-13
4	0.000117	7.1E-05	0.000783	2.53E-06	1.27E-06	3.57E-06	5.48E-09	1.63E-09	1.07E-08	1.49E-10	1.05E-10	2.69E-10
6	0.00179	0.000922	0.002848	3.26E-05	1.94E-05	6.44E-05	2.28E-09	1.4E-09	6.59E-09	3.92E-11	1.44E-11	6.4E-11
8	0.012333	0.006032	0.023899	0.000125	7.29E-05	0.000311	3.49E-08	2.25E-08	4.57E-08	7.81E-10	3.19E-10	1.43E-09
11	0.059405	0.041534	0.087248	0.0008	0.000477	0.001319	4.47E-07	3.09E-07	5.46E-07	1.04E-08	6.92E-09	1.65E-08
16	0.382243	0.294099	0.564501	0.007868	0.005836	0.012932	6.1E-07	5.06E-07	8.12E-07	9.17E-09	6.95E-09	1.29E-08
23	2.699506	2.239906	3.323001	0.033387	0.025963	0.051753	3.56E-06	2.34E-06	4.09E-06	4.55E-08	3.2E-08	6.12E-08
32	14.87309	11.48805	21.69675	0.270439	0.209815	0.32113	2.2E-05	1.69E-05	4.85E-05	2.27E-07	1.96E-07	3.24E-07
45	83.74	70.62547	95.79642	1.53915	1.295244	1.801265	9.97E-05	7.2E-05	0.000116	1.39E-06	1.1E-06	2.26E-06
64	469.9463	445.7146	585.1608	9.187947	8.150691	10.09967	0.000683	0.000502	0.000903	1.08E-05	8.49E-06	1.36E-05
91	3499.937	2802.328	4330.429	64.03297	54.56715	70.67659	0.004198	0.003548	0.004776	9.19E-05	7.75E-05	9.83E-05
128	22323.14	18676.03	29591.37	472.3166	429.2158	550.2567	0.024431	0.021178	0.028582	0.000538	0.000471	0.000586
181	110313.7	101632.3	127301.6	2501.706	2104.636	2872.705	0.147676	0.133249	0.165092	0.003357	0.002956	0.003772
256	613232.9	560863.4	780549.8	15426.73	14478.44	16778.54	0.895624	0.817416	1.018174	0.023564	0.021294	0.025321
362	3584694	3089016	4749926	92118.2	87123.63	96670.27	4.346477	3.774509	4.695971	0.113249	0.107805	0.120636
512	20279273	17010939	23736815	485893.5	475203.4	524616.3	19.25505	18.38549	24.95344	0.526514	0.4908	0.573142
724	84421267	77045709	91871349	2382494	2294958	2492801	83.54762	73.64458	97.02719	1.926552	1.850959	1.968468
1024	2.69E+08	2.62E+08	2.88E+08	9703430	9426360	9947975	325.2359	288.7001	400.1532	5.41189	5.303015	5.868452
1448	7.51E+08	7.16E+08	7.68E+08	41985377	40625933	45139532	28518.69	4010.382	94713615	15.54582	14.93267	15.96589

Dims.	Chung-Reynolds						Deflected Corrugated Spring					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	1.15E-17	9.93E-19	6.42E-17	2.59E-20	1.26E-21	8.61E-20	-0.84334	-0.99997	-0.84334	-0.84334	-1	-0.84334
3	6.96E-16	2.55E-16	2.2E-15	8.08E-20	2.87E-20	4.13E-19	-0.84334	-0.84334	-0.37336	-0.84334	-0.84334	-0.37336
4	2.74E-14	1.97E-15	1.35E-13	3.55E-18	1.62E-19	1.34E-17	-0.60835	-0.84334	-0.37336	-0.60835	-0.84334	-0.37336
6	1.33E-12	2.85E-13	5.08E-12	4.81E-16	3.47E-16	1.73E-15	-0.37336	-0.37336	0.409922	-0.37336	-0.37336	0.409922
8	4.35E-11	9.56E-12	1.57E-10	1.91E-14	1.29E-14	3.2E-14	0.409922	-0.37336	0.409922	0.409922	-0.37336	0.409922
11	5.48E-09	1.05E-09	9.08E-09	2.9E-12	1.51E-12	5.07E-12	1.506499	0.409922	1.5065	1.506499	0.409922	1.506499
16	3.68E-08	1.9E-08	1.2E-07	1.89E-11	8.81E-12	3.95E-11	1.506499	0.409923	2.916346	1.506499	0.409922	2.916346
23	1.44E-06	1.13E-06	3.51E-06	1.47E-10	8.44E-11	2.62E-10	6.675717	4.208659	7.263076	6.675717	4.208659	7.263076
32	3.59E-05	2.46E-05	6.53E-05	1.33E-08	7.93E-09	1.86E-08	6.675717	6.675717	9.025156	6.675717	6.675717	9.025156
45	0.001339	0.000874	0.002874	2.65E-07	2.2E-07	3.25E-07	11.68769	9.025156	14.66325	11.68769	9.025156	14.66325
64	0.066991	0.03375	0.085641	1.77E-05	1.02E-05	2.33E-05	17.95174	13.91936	17.95174	17.95174	13.91936	17.95174
91	2.109556	1.627134	2.85934	0.000837	0.000615	0.001305	19.75241	17.95174	25.4671	19.75241	17.95174	25.4671
128	80.59232	71.26561	128.3714	0.067368	0.04848	0.082366	25.4671	24.4886	29.69368	25.4671	24.4886	29.69368
181	2474.834	1992.575	3401.532	1.604921	1.358137	2.114952	25.4671	25.4671	30.8284	25.4671	25.4671	30.8284
256	105966.3	92714.95	120975.2	63.35262	57.66098	77.91253	29.69368	28.63703	34.23259	29.69368	28.63703	34.23259
362	2748621	2057301	4923251	2064.555	1899.399	2293.973	31.96313	29.69368	39.08356	31.96313	29.69368	39.08356
512	71194489	54821481	81709654	45310.05	41560.13	49876.42	39.08356	37.87081	39.08356	39.08356	37.87081	39.08356
724	9.7E+08	8.77E+08	1.3E+09	924862.6	875446.5	1031035	39.08356	39.08356	44.24621	39.08356	39.08356	44.24621
1024	1.15E+10	1.01E+10	1.24E+10	15790414	15053979	18857553	49.71999	44.24621	49.71999	44.24621	42.95555	49.71999
1448	8.63E+10	8.2E+10	9.13E+10	2.87E+08	2.7E+08	3.04E+08	75.64883	74.32612	80.23528	49.71999	48.35155	49.71999

Table B.26: RMHC Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (50,000 Evaluation Tuning)

Dims.	Ackley						Alpine1					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.00115	0.000431	0.001788	0.000161	0.000104	0.000216	1.85E-05	1.03E-05	4.3E-05	4.57E-06	2.77E-06	8.34E-06
3	0.001616	0.000817	0.0023	0.000302	0.00025	0.000407	8.25E-05	4.93E-05	0.000137	1.7E-05	8.41E-06	2.3E-05
4	0.002421	0.001845	0.003102	0.000323	0.000187	0.000535	0.000175	0.000106	0.000253	2.43E-05	1.06E-05	3.3E-05
6	0.00408	0.002902	0.005266	0.000551	0.000508	0.000805	0.000336	0.000182	0.000437	5.92E-05	5.06E-05	8.67E-05
8	0.005978	0.004953	0.014779	0.001014	0.000782	0.001266	0.000617	0.000457	0.000957	0.000158	6.4E-05	0.000195
11	0.010119	0.008482	0.014627	0.001389	0.001117	0.001612	0.001529	0.00113	0.001862	0.000248	0.000143	0.000338
16	0.02057	0.017643	0.025721	0.001967	0.001591	0.002531	0.003356	0.002927	0.003988	0.0005	0.000415	0.000592
23	0.048969	0.039887	0.076603	0.002759	0.002467	0.003492	0.00818	0.006525	0.011795	0.001023	0.000902	0.001229
32	15.12261	11.45996	16.3328	0.00395	0.003451	0.004595	1.53656	0.834231	2.185398	0.002321	0.00197	0.002615
45	15.22628	14.6582	16.6361	0.006868	0.00614	0.007859	14.69213	10.96036	19.10318	0.004718	0.003648	0.005342
64	18.2899	17.96855	18.80292	0.012237	0.011727	0.012753	55.84512	49.87217	67.53628	0.009599	0.008348	0.011159
91	18.71633	18.45989	19.01402	0.021767	0.019138	0.02372	122.5649	111.8975	128.4634	0.021457	0.01945	0.023169
128	18.20334	17.88094	18.54804	0.048182	0.046174	0.050837	182.8698	172.6071	195.1765	0.057546	0.050724	0.061121
181	18.94985	18.76849	19.04436	0.087756	0.080452	0.095265	284.5445	273.1749	303.481	0.168428	0.158291	0.181673
256	18.724	18.58385	18.90557	0.211257	0.197328	0.219429	382.2144	369.9873	404.1651	0.480849	0.461976	0.511943
362	18.79336	18.64705	18.87162	0.481137	0.461746	0.498435	525.9654	492.4162	540.4009	1.366329	1.248197	1.392504
512	18.88536	18.80195	18.97927	1.189483	1.106404	1.245371	55.10915	52.45431	60.02277	3.75976	3.446688	4.003443
724	18.67032	18.58381	18.84987	2.196211	2.141925	2.25199	133.893	129.9569	136.2912	9.689619	9.199165	10.10717
1024	18.68149	18.60062	18.72727	3.01415	2.975743	3.092075	306.1532	292.2091	315.7846	25.24041	24.40726	26.70586
1448	18.79416	18.68337	18.8481	4.192086	4.081397	4.234972	650.3139	639.8985	673.4781	64.6883	63.74843	67.43894

Dims.	Bent Cigar						Brown					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	2.71E-06	1.42E-06	8.88E-06	3.55E-07	8.88E-08	3.55E-07	3.93E-12	1.81E-12	7.69E-12	5.21E-14	2.49E-14	1.54E-13
3	2.72E-05	1.73E-05	0.000153	5.77E-07	2.89E-07	2.49E-06	3.98E-11	1.48E-11	1.96E-10	5.95E-13	3.1E-13	1.46E-12
4	0.000174	5.62E-05	0.000459	2.13E-06	9.33E-07	4.97E-06	0.02451	3.29E-10	0.401194	2.07E-12	5.68E-13	3.35E-12
6	1670.472	0.003023	131652.3	1.01E-05	7.26E-06	2.59E-05	4.08183	1.522009	5.989405	1.13E-11	6.36E-12	1.75E-11
8	3327109	2462638	5193415	3.99E-05	2.32E-05	4.85E-05	8.267518	6.247683	26.70058	2.34E-11	1.11E-11	3.26E-11
11	12852463	10608173	17004438	0.000159	9.21E-05	0.000249	590.9689	61.57829	13578.17	1.07E-10	6.77E-11	1.63E-10
16	42428186	35918176	50822825	0.000623	0.000432	0.000969	4.8E-07	3.23E-07	6.87E-07	1.18E-08	6.86E-09	1.46E-08
23	62721361	51463379	74346836	0.004431	0.003368	0.006524	7.18E-06	4.98E-06	4.57E-05	3.23E-08	2.73E-08	4.03E-08
32	86285920	69528197	1.04E+08	0.022102	0.017545	0.028573	5.45E-05	2.66E-05	0.149234	1.66E-07	1.05E-07	2E-07
45	1.25E+08	1.07E+08	1.43E+08	0.145207	0.107804	0.20538	18.7417	15.40825	31.15616	3.84E-07	3.36E-07	5.01E-07
64	1.84E+08	1.63E+08	2.03E+08	0.843053	0.638833	1.17519	3443.475	614.6455	23174.78	1.12E-06	8.2E-07	1.25E-06
91	2.9E+08	2.66E+08	3.14E+08	5.908823	4.610924	6.778786	10999.4	1678.25	55453.91	6.62E-06	5.57E-06	7.87E-06
128	3.47E+08	3.26E+08	3.69E+08	35.59192	27.6382	42.42295	2200712	1336623	15085200	3.61E-05	3E-05	4.17E-05
181	5.24E+08	4.94E+08	5.45E+08	158.2113	139.3582	177.8394	69023213	11501150	2.64E+08	0.000209	0.00018	0.000251
256	6.46E+08	6.06E+08	6.92E+08	979.7936	858.8372	1206.719	6957469	589091.7	34213386	0.001276	0.001088	0.00142
362	8.69E+08	8.35E+08	9.24E+08	5712.974	5381.476	6059.884	72694903	3421763	2.12E+09	0.007224	0.006791	0.007516
512	1.14E+09	1.11E+09	1.2E+09	33662.23	32141.53	36173.74	52845721	22447641	1.01E+08	0.042721	0.040203	0.045386
724	1.64E+09	1.6E+09	1.71E+09	203394.7	199587.3	218253.7	3.47E+08	1.28E+08	1.61E+09	0.246677	0.237165	0.293612
1024	1.91E+09	1.83E+09	1.97E+09	1284804	1180355	1392516	5.21E+08	48639959	8.86E+09	1.51898	1.482737	1.711262
1448	3.06E+09	2.99E+09	3.14E+09	7063892	6501628	8277508	2.94E+10	2.49E+09	2.04E+11	8.106002	7.421377	9.09091

Dims.	Chung-Reynolds						Deflected Corrugated Spring					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	2.57E-17	8.38E-18	1.42E-16	1.47E-19	2.02E-20	3.65E-19	-0.84334	-0.99998	-0.84334	-0.84334	-1	-0.84334
3	6.88E-16	1.37E-16	2.57E-15	3.23E-19	8.08E-20	2.37E-18	-0.84334	-0.84334	-0.72584	-0.84334	-0.84334	-0.72585
4	1.59E-14	2.75E-15	6.21E-14	3.73E-18	7.89E-19	9.78E-18	-0.84334	-0.84334	-0.37336	-0.84334	-0.84334	-0.37336
6	317.6045	1.43E-12	11300.37	7.23E-17	8.57E-18	1.93E-16	0.01828	-0.37336	0.409923	0.01828	-0.37336	0.409922
8	2512628	636027.3	3621233	2.45E-16	8.85E-17	7.63E-16	-0.37336	-0.37336	0.409922	-0.37336	-0.37336	0.409922
11	50138363	35557835	65719634	3.85E-15	2.14E-15	1.12E-14	1.506499	0.409922	1.5065	1.506499	0.409922	1.506499
16	2.72E+08	1.86E+08	3.92E+08	7.37E-14	4.72E-14	9.79E-14	2.916346	1.506499	3.347117	2.916346	1.506499	3.347117
23	1.29E+09	9.89E+08	1.69E+09	7.83E-12	3.96E-12	1.83E-11	3.777888	2.563884	4.639431	3.777888	2.563884	4.639431
32	55091409	32011388	72822552	4.33E-10	2.94E-10	9.8E-10	7.850436	4.639431	9.025156	7.850436	4.639431	9.025156
45	4.81E+09	4.12E+09	6.95E+09	6.42E-09	4.14E-09	2.48E-08	11.68769	9.025156	14.66325	11.68769	9.025156	14.66325
64	1.46E+09	1.32E+09	1.76E+09	2.23E-07	1.31E-07	4.07E-07	17.95174	14.66325	18.85208	17.95174	14.66325	18.85208
91	8.58E+09	7.12E+09	1.11E+10	5.15E-06	2.77E-06	8.39E-06	21.55307	20.65274	25.4671	21.55307	20.65274	25.4671
128	2.6E+10	2.12E+10	2.96E+10	0.00016	0.000112	0.000199	23.51009	20.65274	25.4671	23.51009	20.65274	25.4671
181	4.45E+10	4.12E+10	5.33E+10	0.003913	0.003313	0.005742	29.69368	25.4671	29.69368	29.69368	25.4671	29.69368
256	6.87E+10	5.97E+10	7.94E+10	0.188396	0.144137	0.201288	29.69368	25.4671	34.23259	29.69368	25.4671	34.23259
362	1.29E+11	1.25E+11	1.36E+11	5.901101	5.014863	6.970081	34.23259	29.69368	34.23259	34.23259	29.69368	34.23259
512	1.94E+11	1.78E+11	2.13E+11	204.9512	173.6866	240.3648	39.08356	34.23259	44.24621	39.08356	34.23259	40.37422
724	3.12E+11	2.94E+11	3.47E+11	6753.136	5577.282	7788.778	44.24621	39.08356	44.24621	39.08356	39.08356	44.24621
1024	5.71E+11	5.47E+11	6.27E+11	269210.6	228261.6	298686.6	44.24621	44.24621	49.71999	44.24621	39.08356	44.24621
1448	1.23E+12	1.16E+12	1.26E+12	8473058	7319649	9577228	83.14844	81.59715	89.37631	44.24621	44.24621	49.71999

Table B.27: RMHC Descriptive Statistics: Exponential to Rastrigin Functions (10,000 Evaluation Tuning)

Dims.	Exponential						Griewank						Inverted Cosine Wave					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	-1	-1	-1	-1	-1	-1	0.007396	2.87E-05	0.007398	0.007396	1.66E-06	0.007396	-0.73809	-0.80357	-0.73809	-0.73809	-0.80357	-0.73809
3	-1	-1	-1	-1	-1	-1	0.007402	6.67E-06	0.012322	0.007396	6.07E-07	0.012321	-1.47618	-1.99973	-1.47618	-1.47618	-1.99999	-1.47618
4	-1	-1	-1	-1	-1	-1	0.01235	0.007426	0.012442	0.012321	0.007397	0.012325	-2.21427	-2.6063	-2.21426	-2.21427	-2.99998	-2.21427
6	-1	-1	-1	-1	-1	-1	0.017387	0.012863	0.027249	0.01724	0.01233	0.027078	-3.69041	-3.94397	-3.68875	-3.69045	-4.21413	-3.69043
8	-1	-1	-1	-1	-1	-1	0.043578	0.025594	0.053103	0.038206	0.024646	0.050499	-5.6723	-6.98696	-5.16501	-6.08251	-6.99973	-5.35488
11	-1	-1	-1	-1	-1	-1	0.033217	0.026673	0.056965	0.028515	0.016824	0.054203	-7.62931	-8.37767	-7.36738	-7.89898	-8.4823	-7.37829
16	-1	-1	-1	-1	-1	-1	0.051269	0.03166	0.070683	0.029274	0.009808	0.053246	-11.8245	-12.7901	-11.1806	-12.3678	-13.161	-11.4717
23	-1	-1	-1	-1	-1	-1	0.068815	0.041066	0.102763	0.052688	0.013724	0.084262	-17.5186	-17.9103	-16.8263	-17.8795	-18.6044	-17.467
32	-1	-1	-1	-1	-1	-1	0.099911	0.070271	0.140486	0.037947	0.015093	0.095726	-24.693	-26.783	-23.2143	-26.6075	-28.1164	-23.8743
45	-1	-1	-1	-1	-1	-1	0.161225	0.107462	0.214001	0.021537	0.006616	0.053063	-34.1231	-35.5074	-33.1844	-36.4281	-37.6205	-35.0681
64	-0.99999	-0.99999	-0.99998	-1	-1	-1	0.376708	0.300011	0.418048	0.033791	0.02389	0.049973	-48.6273	-49.5442	-47.8088	-51.1327	-52.5608	-49.5117
91	-0.99993	-0.99994	-0.99992	-1	-1	-1	0.71293	0.568043	0.814413	0.043882	0.028945	0.069649	-67.6798	-68.8463	-66.4121	-71.1506	-73.0274	-69.252
128	-0.9996	-0.99963	-0.99949	-0.99999	-0.99999	-0.99999	1.072407	1.06057	1.090426	0.104963	0.09165	0.121279	-93.9232	-95.0316	-93.2604	-100.256	-102.411	-98.582
181	-0.99748	-0.99764	-0.9971	-0.99995	-0.99995	-0.99994	1.430152	1.361033	1.499906	0.421706	0.392259	0.456836	-131.273	-132.748	-130.558	-141.018	-141.996	-139.326
256	-0.98642	-0.98772	-0.98563	-0.99968	-0.99969	-0.99964	3.468098	3.324323	3.864918	0.96761	0.91427	1.007649	-180.844	-182.574	-179.66	-198.045	-200.1	-195.091
362	-0.92399	-0.92867	-0.91138	-0.99806	-0.99815	-0.99781	14.95207	12.62591	18.07325	1.370266	1.350047	1.396305	-245.368	-247.273	-242.673	-274.421	-277.176	-272.812
512	-0.65423	-0.71928	-0.60185	-0.98941	-0.99002	-0.9888	71.26256	63.25114	74.21316	3.033673	2.910976	3.204082	-325.765	-327.897	-322.643	-384.215	-387.798	-381.774
724	-0.2009	-0.24436	-0.16395	-0.9555	-0.95768	-0.95286	301.8944	280.0932	322.8166	10.39607	9.895569	10.62587	-419.093	-424.361	-414.702	-534.86	-537.907	-532.826
1024	-0.00465	-0.00593	-0.00324	-0.84519	-0.85109	-0.83856	1002.397	926.6719	1065.328	35.51984	33.19113	36.7947	-514.894	-523.199	-509.373	-740.15	-742.371	-737.92
1448	-2.2E-07	-3.5E-07	-1.1E-07	-0.45251	-0.46477	-0.43582	2727.683	2628.704	2905.512	148.274	141.9109	152.5924	-625.559	-630.094	-617.09	-1011.54	-1014.98	-1006

Dims.	Levy						Qing						Rastrigin					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	6.55E-07	2.33E-07	1.45E-06	1.73E-08	5.7E-09	4.33E-08	7.58E-07	2.68E-07	2.47E-06	1.12E-08	6.37E-09	1.69E-08	1.88E-05	4.86E-06	3.51E-05	4.61E-07	2.35E-07	9.29E-07
3	1.89E-06	5.16E-07	4.25E-06	1.15E-07	4.74E-08	2.04E-07	5.18E-06	1.42E-06	1.83E-05	1.11E-07	6.63E-08	3.07E-07	4.94E-05	2.31E-05	0.000196	3.11E-06	1.27E-06	6.34E-06
4	3.86E-06	2.84E-06	9.31E-06	3.48E-07	8.14E-08	5.35E-07	4.12E-05	4.91E-06	0.000125	2.75E-07	1.28E-07	7.06E-07	0.000291	0.000152	0.000504	9.94E-06	1.2E-06	1.53E-05
6	3.02E-05	9.67E-06	4.73E-05	8.23E-07	3.82E-07	1.1E-06	0.000541	0.000324	0.00085	4.35E-06	3.5E-06	7.06E-06	0.000789	0.000483	0.001144	3.12E-05	1.42E-05	4.41E-05
8	5.61E-05	2.99E-05	0.000102	1.51E-06	7.09E-07	3.34E-06	0.002442	0.001457	0.004236	3.24E-05	2.33E-05	6.57E-05	0.001311	0.000635	0.003432	5.5E-05	3.53E-05	8.55E-05
11	0.000143	0.000118	0.000199	6.46E-06	4.65E-06	9.04E-06	0.01116	0.008435	0.017567	0.000209	0.00011	0.000508	0.004961	0.002266	0.006338	0.000173	0.00012	0.000281
16	0.000601	0.00045	0.001018	1.94E-05	1.67E-05	2.49E-05	0.07301	0.058839	0.094792	0.001817	0.001271	0.002366	0.015791	0.011558	0.023055	0.000762	0.000521	0.000876
23	0.001633	0.001012	0.001986	6.26E-05	4.37E-05	7.2E-05	0.506423	0.379895	0.628253	0.01046	0.008635	0.013889	0.050245	0.035307	0.064231	0.001839	0.001551	0.002292
32	0.004648	0.003478	0.005727	0.000151	0.000125	0.000191	2.519221	1.77685	3.287036	0.06184	0.045412	0.077677	0.170835	0.130776	0.222486	0.005533	0.004609	0.006889
45	0.015467	0.011608	0.019017	0.000538	0.000412	0.000716	26.05523	16.38075	31.34345	0.320372	0.279918	0.610944	0.508009	0.390747	0.594833	0.014794	0.013076	0.01898
64	0.039815	0.031393	0.046906	0.001353	0.00121	0.00169	189.271	162.2186	257.132	3.996043	3.51426	4.76114	1.490516	1.150724	1.897235	0.056385	0.045099	0.062489
91	0.132466	0.116468	0.147914	0.004709	0.00441	0.005471	1613.765	1336.831	1963.89	33.39253	29.6954	44.01492	5.542732	4.971302	6.430345	0.163419	0.13734	0.177879
128	0.406323	0.38729	0.44516	0.014536	0.012847	0.0162	15116.12	12813.53	17031.94	308.0089	276.863	353.4908	20.33858	15.05558	25.18293	0.436908	0.396661	0.496664
181	1.333163	1.217377	1.503096	0.047074	0.045107	0.050119	125617.7	112980.1	134146.8	3570.363	3332.655	4147.814	60.34258	53.93789	66.56975	1.480489	1.23514	1.731114
256	3.552273	3.447806	3.978649	0.143077	0.132732	0.167922	1831326	1540608	2690202	41378.13	38604.01	48838.38	152.3633	144.8513	161.4324	13.16424	12.72684	14.31088
362	13.51542	11.35908	15.99039	0.542734	0.506585	0.575039	43876559	35834405	54100610	336131.8	315908.7	365143.3	347.4622	337.8723	356.1794	36.6801	34.10313	37.17097
512	35.28386	32.43731	35.8737	2.626351	2.500211	2.693559	5.33E+08	4.07E+08	1.57E+09	2148418	1926909	2321099	786.0264	764.6393	811.3563	134.72	132.9478	139.6045
724	100.4816	92.62859	104.4971	7.781864	7.596675	8.15009	6.52E+09	5.4E+09	1.24E+10	14715535	13502207	16262183	1623.064	1591.16	1674.46	298.7849	289.4433	307.9993
1024	286.2457	265.6099	298.1652	20.20271	19.64396	20.91227	4.19E+10	3.52E+10	5.42E+10	1.3E+08	1.15E+08	1.53E+08	3357.768	3276.536	3407.477	616.032	604.4408	621.3864
1448	809.4084	776.1789	827.9361	48.92501	47.05596	49.99738	2.31E+11	2.15E+11	2.57E+11	6.15E+08	5.34E+08	6.88E+08	6504.641	6399.367	6600.014	1471.811	1453.582	1500.156

Table B.28: RMHC Descriptive Statistics: Exponential to Rastrigin Functions (50,000 Evaluation Tuning)

Dims.	Exponential						Griewank						Inverted Cosine Wave					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	-1	-1	-1	-1	-1	-1	0.008235	0.007441	0.009778	0.007408	0.005632	0.007422	-0.73809	-0.73809	-0.73809	-0.73809	-0.80357	-0.73809
3	-1	-1	-1	-1	-1	-1	0.007399	2.38E-05	0.012328	0.007397	9.7E-07	0.012321	-1.47618	-1.99953	-1.47618	-1.47618	-1.99998	-1.47618
4	-1	-1	-1	-1	-1	-1	0.012588	0.009971	0.015685	0.012327	0.009867	0.015402	-2.21427	-2.99987	-2.21426	-2.21427	-3	-2.21427
6	-0.99994	-1	-0.99439	-1	-1	-1	0.02665	0.022595	0.036898	0.024949	0.02111	0.034551	-3.81939	-4.99303	-3.68956	-4.60667	-4.99994	-3.69044
8	-0.95562	-0.97925	-0.90787	-1	-1	-1	0.032238	0.026604	0.055112	0.028749	0.02168	0.052471	-5.28238	-5.73464	-5.15639	-5.68167	-6.40799	-5.16587
11	-0.72764	-0.78722	-0.67227	-1	-1	-1	0.048958	0.02067	0.07129	0.033812	0.001845	0.064238	-7.63689	-8.1166	-7.35707	-7.89951	-8.22936	-7.37849
16	-0.43496	-0.51623	-0.36146	-1	-1	-1	0.07041	0.04049	0.094044	0.032217	0.012663	0.066303	-11.6045	-12.1407	-11.0443	-11.8221	-13.1441	-11.2268
23	-0.14804	-0.18623	-0.10586	-1	-1	-1	0.082237	0.062827	0.1072	0.017257	0.007451	0.031997	-17.1121	-18.1952	-16.488	-18.0766	-18.9007	-16.695
32	-1	-1	-1	-1	-1	-1	0.192526	0.160329	0.231737	0.018964	0.011105	0.034927	-23.9709	-24.9647	-23.118	-24.2825	-25.6554	-23.715
45	-1	-1	-1	-1	-1	-1	0.269728	0.24735	0.318909	0.022803	0.01791	0.035618	-34.1546	-35.3422	-33.1406	-35.3836	-36.6216	-34.349
64	-0.0324	-0.06467	-0.0273	-1	-1	-1	0.402961	0.344454	0.448097	0.029631	0.019366	0.063122	-48.1753	-48.8478	-47.1677	-49.8745	-51.607	-48.965
91	-0.07581	-0.10035	-0.04024	-1	-1	-1	0.694629	0.650883	0.780987	0.032805	0.023114	0.042599	-68.9811	-70.2331	-66.3249	-72.9599	-74.549	-69.6426
128	-0.01413	-0.01768	-0.01019	-1	-1	-1	2.861503	1.500452	5.023721	0.07327	0.051959	0.08681	-93.9667	-94.3883	-92.2579	-98.9852	-101.632	-97.1484
181	-0.0001	-0.0002	-6.7E-05	-1	-1	-1	98.84553	81.11075	116.3316	0.115581	0.090708	0.154728	-129.919	-131.716	-129.036	-141.247	-143.275	-138.004
256	-2.7E-06	-5.6E-06	-1.1E-06	-0.99998	-0.99998	-0.99998	1073.822	994.2048	1167.517	0.251465	0.192146	0.300128	-180.826	-181.891	-179.139	-197.354	-200.903	-195.962
362	-2.2E-08	-3.9E-08	-6.6E-09	-0.99989	-0.9999	-0.99989	2442.694	2363.078	2631.784	0.589115	0.525887	0.640274	-243.168	-245.878	-240.19	-275.378	-280.122	-274.215
512	-2E-10	-5.7E-10	-1.1E-10	-0.99931	-0.99936	-0.99927	4466.067	4299.551	4617.567	1.083092	1.029368	1.116929	-322.107	-324.144	-317.416	-386.053	-388.429	-382.846
724	-1.7E-14	-5.9E-14	-5.6E-15	-0.9959	-0.99609	-0.99569	5458.916	5375.854	5578.762	1.70373	1.679602	1.75867	-410.059	-413.815	-406.568	-536.008	-537.166	-533.282
1024	-2.7E-18	-4.5E-18	-9.3E-19	-0.97588	-0.97709	-0.97399	8138.017	7898.782	8487.687	5.542906	5.139419	5.789382	-520.403	-524.925	-512.574	-737.905	-740.749	-734.799
1448	-6E-27	-3.1E-26	-6.8E-28	-0.8711	-0.88853	-0.84342	10736.05	10544.21	10986.83	27.89755	25.8293	30.03728	-622.496	-627.969	-608.453	-1010.23	-1014.76	-1008.21

Dims.	Levy						Qing						Rastrigin					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	1.02E-06	3.98E-07	2.41E-06	2.15E-08	6.21E-09	6.06E-08	9.92E-07	5.88E-07	1.6E-06	1.55E-08	6.49E-09	3.28E-08	9.81E-06	4.81E-06	2.05E-05	4.22E-07	1.95E-07	1.11E-06
3	1.79E-06	1.12E-06	1.02E-05	1.21E-07	4.55E-08	2.82E-07	5.15E-06	2.7E-06	1.43E-05	1.15E-07	6.98E-08	2.39E-07	6.55E-05	3.31E-05	0.00013	3.38E-06	1.31E-06	5.49E-06
4	4.68E-06	2.03E-06	1.19E-05	1.12E-07	5.56E-08	2.91E-07	1.86E-05	7.51E-06	4.85E-05	3.23E-07	1.89E-07	5.68E-07	0.000196	5.12E-05	0.000279	5.66E-06	2.28E-06	7.93E-06
6	1.48E-05	8.8E-06	2.21E-05	6.27E-07	3.66E-07	1.32E-06	91576.45	0.000854	9524237	2.38E-06	1.13E-06	4.13E-06	0.000811	0.000483	0.001141	3E-05	1.72E-05	5.83E-05
8	4.81E-05	2.99E-05	7.87E-05	2.27E-06	1.4E-06	3.43E-06	3.14E+08	32176361	7.01E+08	8.01E-06	4.25E-06	1.45E-05	0.00629	0.002459	0.287541	4.25E-05	3.19E-05	5.76E-05
11	0.000152	0.000106	0.000221	6.38E-06	3.49E-06	8.93E-06	4.28E+09	1.12E+09	7.04E+09	3.42E-05	1.7E-05	6.17E-05	0.008038	0.003377	0.013659	0.000139	9.48E-05	0.000179
16	0.000513	0.000376	0.000693	1.66E-05	1.25E-05	2.61E-05	2.81E+10	1.96E+10	3.24E+10	0.000202	0.00015	0.000309	1.068123	1.041728	2.532279	0.000508	0.000405	0.000686
23	0.001764	0.001038	0.002169	4.89E-05	3.7E-05	8.8E-05	5.85E+10	4.93E+10	7.77E+10	0.002208	0.001526	0.004303	0.064851	0.034141	0.094223	0.001808	0.001287	0.002327
32	0.00473	0.003306	0.005506	0.000201	0.000147	0.000268	5.85E+10	4.52E+10	7.04E+10	0.014949	0.010084	0.029636	0.385584	0.233726	0.701431	0.004455	0.003394	0.006537
45	0.017987	0.008476	0.962463	0.00051	0.000376	0.000641	8.31E+10	5.65E+10	1.04E+11	0.101883	0.072556	0.144372	0.517367	0.438873	0.569123	0.014395	0.012299	0.017026
64	0.109634	0.048093	5.441435	0.001416	0.00118	0.001579	1.36E+11	1.28E+11	1.53E+11	1.043466	0.871396	1.155102	1.660771	1.312376	2.00937	0.042554	0.036833	0.052293
91	2.06051	0.249616	5.523549	0.004218	0.003444	0.004593	1.92E+11	1.75E+11	2.09E+11	8.472919	7.548736	9.925001	10.04144	8.954815	13.16278	0.14353	0.13028	0.158547
128	6.983978	4.156654	8.832843	0.012443	0.010716	0.01419	2.75E+11	2.56E+11	3.11E+11	68.85213	56.63118	78.82698	34.54311	29.14108	44.16906	0.409489	0.354877	0.445306
181	12.32186	10.97256	20.24649	0.033409	0.031239	0.038861	1.3E+10	9.95E+09	1.64E+10	490.3263	416.1533	531.9687	99.74462	87.66619	109.9515	1.230245	1.119035	1.312822
256	35.75349	29.5814	42.62966	0.114055	0.101779	0.126933	6.31E+10	5.44E+10	7.28E+10	2676.746	2323.575	3064.028	234.6908	220.8943	251.7931	4.118585	3.719256	4.567282
362	83.69911	77.33754	96.42966	0.333139	0.282918	0.373037	1.68E+11	1.53E+11	1.9E+11	16936.63	15926.11	18851.78	720.2034	682.2745	754.7331	12.5496	11.3205	12.88577
512	142.5877	128.6218	157.3605	1.050279	0.953479	1.086204	5.91E+11	5.3E+11	6.29E+11	105786.9	100365	123112.7	1491.397	1418.447	1528.61	43.2689	38.27502	45.0386
724	307.366	281.3698	327.3078	3.234993	3.165332	3.367429	7.87E+11	7.65E+11	8.53E+11	825658.8	771360.5	868165.4	2902.252	2829.6	2938.502	133.9725	129.0671	137.1808
1024	622.6259	579.4301	633.3824	9.850373	9.504947	10.36467	1.17E+12	1.02E+12	1.2E+12	7195435	6603084	7541549	5607.184	5478.812	5702.874	442.1768	425.6076	453.0432
1448	1368.819	1344.609	1454.828	31.87338	29.1185	34.71062	1.11E+12	1.03E+12	1.16E+12	82385027	77193459	86934294	7728.339	7614.866	7828.956	1015.061	1003.901	1041.438

Table B.29: RMHC Descriptive Statistics: Rosenbrock to Sum Squares Function (10,000 Evaluation Tuning)

Dims.	Rosenbrock						Schwefel						Sphere						Sum of Different Powers						Sum Squares					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.556661	0.09901	3.541868	0.552665	0.097056	3.540758	118.4388	0.001393	118.4396	118.4384	8.52E-05	118.4384	9.17E-12	5.1E-12	2.24E-11	3.73E-13	9.31E-14	1.28E-12	9.24E-14	1.42E-14	1.8E-12	3.55E-15	2.12E-22	1.42E-14	4.51E-11	1.37E-11	1.41E-10	1.78E-12	0	6.75E-12
3	1.740927	0.723758	5.483881	1.737559	0.71913	5.463996	118.4461	118.4394	236.8772	118.4385	118.4384	236.8768	6.71E-11	4.2E-11	1.93E-10	1.58E-12	5.59E-13	3.19E-12	2.01E-13	1.42E-14	7.5E-13	1.42E-14	2.66E-15	1.42E-14	5.18E-10	1.96E-10	1.6E-09	1.65E-11	3.55E-12	2.79E-11
4	2.0502	0.696691	7.058937	1.976918	0.612473	7.035565	236.8884	207.2684	355.3184	236.8772	207.2673	355.3152	4.34E-10	2.15E-10	7.9E-10	5.08E-12	3.86E-12	1.01E-11	1.09E-12	1.18E-13	3.42E-12	1.42E-14	3.55E-15	3.82E-14	2.75E-09	1.4E-09	7.41E-09	2.68E-11	1.31E-11	4.78E-11
6	2.439388	1.11312	4.005956	2.164845	0.603692	3.655207	355.338	236.9161	473.7698	355.3156	236.8786	473.7541	3.66E-08	1.75E-08	7.45E-08	9.31E-10	6.1E-10	1.52E-09	1.15E-10	1.91E-11	5.37E-10	4.16E-12	9.1E-13	2.08E-11	2.29E-07	1.11E-07	4.08E-07	9.16E-09	5.04E-09	1.19E-08
8	2.608709	1.123299	4.098626	1.978238	0.988458	3.551682	355.4298	355.3465	473.8992	355.3172	355.316	473.7643	3.34E-07	2.22E-07	4.92E-07	1.31E-08	1.12E-08	1.71E-08	8.06E-09	2.89E-09	2.85E-08	1.38E-09	2.25E-10	2.28E-09	3.5E-07	2.56E-07	6.84E-07	8.35E-09	3.2E-09	1.19E-08
11	5.044504	2.392209	12.56523	1.62477	0.414299	5.461068	592.3214	473.8981	710.7832	592.1992	473.7616	710.6372	7.15E-08	4.07E-08	1.16E-07	7.66E-10	5.86E-10	1.19E-09	1.72E-10	5.62E-11	4.83E-09	3.78E-11	5.98E-12	8.28E-11	1.82E-06	8.65E-07	3.15E-06	1.55E-08	9.28E-09	2.61E-08
16	66.31333	9.483054	74.80606	7.145411	3.606993	71.21591	770.6991	681.5077	948.1696	769.8767	681.0452	947.529	5.24E-07	3.32E-07	6.95E-07	1.24E-08	8.25E-09	1.69E-08	4.7E-09	1.13E-09	1.52E-08	1.55E-10	1.65E-11	4.88E-10	1.45E-05	1.17E-05	2.92E-05	2.18E-07	1.63E-07	2.84E-07
23	65.53176	15.71765	78.55998	10.4258	2.740468	71.13566	1069.196	950.8313	1335.894	1066.127	947.6779	1332.52	3.27E-06	2.22E-06	4.16E-06	4.54E-08	3.09E-08	5.93E-08	5.83E-09	2.23E-09	1.88E-08	1.55E-10	5.94E-11	1.12E-09	0.000125	8.42E-05	0.000175	1.4E-06	1.2E-06	1.93E-06
32	126.1771	96.66851	162.1083	74.89929	65.41043	79.49649	1608.707	1428.409	1698.17	1599.138	1421.55	1688.136	1.92E-05	1.53E-05	2.38E-05	3.13E-07	2.36E-07	4.29E-07	4.48E-08	1.23E-08	1.52E-07	6.26E-10	8.65E-11	3.39E-09	0.001038	0.000712	0.001384	1.46E-05	1.11E-05	1.7E-05
45	192.2481	182.0495	228.4107	95.77171	82.76558	154.8104	2398.059	2149.237	2742.585	2369.646	2132.929	2724.737	0.000129	9.43E-05	0.000156	1.55E-06	1.18E-06	1.89E-06	1.91E-08	7.44E-09	5.84E-08	5.6E-10	1.12E-10	2.33E-09	0.007226	0.005242	0.010837	0.000117	9.03E-05	0.000143
64	237.4834	151.8707	413.2688	87.65866	11.60317	192.9608	3377.8	3123.065	3520.347	3318.146	3081.612	3437.059	0.000591	0.000416	0.000649	9.54E-06	7.74E-06	1.31E-05	8.01E-08	3.16E-08	3.01E-07	1.25E-08	2.13E-09	1.56E-08	0.074522	0.059052	0.080552	0.001393	0.001241	0.002052
91	359.5125	313.3215	514.7428	162.3674	143.1546	220.2995	4837.954	4478.508	5010.313	4625.947	4358.367	4862.483	0.004207	0.003086	0.004765	5.78E-05	4.84E-05	6.69E-05	3.61E-07	1.05E-07	1.35E-06	6.48E-09	3.35E-10	1.65E-08	0.642517	0.485351	0.757123	0.011955	0.010672	0.014085
128	751.8534	629.7377	825.6632	257.0809	205.2032	329.7439	7005.972	6560.489	7273.118	6595.536	6146.583	6914.874	0.01859	0.016148	0.024143	0.000385	0.000361	0.000467	2.14E-06	8E-07	4.04E-06	2.66E-08	2.07E-09	1.86E-07	4.848837	4.266006	5.198557	0.111078	0.090621	0.134487
181	1730.556	1454.901	1992.791	527.263	440.5395	630.5286	10382.24	9646.051	10879.15	9292.96	8434.148	9582.968	0.1382	0.119581	0.163925	0.002773	0.002576	0.00313	9.03E-06	2.8E-06	3.87E-05	9.97E-08	1.89E-08	5.16E-07	41.76964	37.89229	45.5892	0.957259	0.855665	1.096932
256	5401.656	4671.393	6235.558	965.6899	931.9749	1076.054	16392.36	15944.71	16922.97	13859.78	13284.17	14313.36	0.783764	0.68009	0.844878	0.020239	0.018603	0.024457	2.61E-05	8.79E-06	5.17E-05	1.93E-06	5.78E-07	7.35E-06	320.7339	278.8456	368.9703	8.638093	8.009866	9.394006
362	26155.01	21313.97	31551.88	1995.826	1940.574	2094.635	27472.55	26390.98	28089.53	19940.66	19513.38	20758.81	3.864688	3.491899	4.751326	0.098862	0.095679	0.107339	0.000174	4.17E-05	0.000327	2.93E-06	6.09E-07	1.12E-05	3092.994	2788.396	4013.898	69.21025	66.08685	78.08714
512	136933.4	125685	160534.9	4907.671	4750.161	5204.11	46645.06	46029.11	47333.39	27495.38	27027.68	28140.43	24.96389	19.37563	30.50647	0.44331	0.40664	0.468658	0.000571	0.000175	0.001287	1.25E-05	1.76E-06	4.69E-05	19785.14	16542.16	23374.63	532.7913	490.7973	579.127
724	866825.4	747339.9	1052204	14998.47	13844.2	15443.61	82532.1	80537.77	84239.98	40238.04	39513.15	41367.6	85.22166	76.68516	89.24791	2.444172	2.33914	2.520801	0.001835	0.000747	0.00512	4.57E-05	1.8E-05	0.000159	114622.1	102074.7	125939.7	3250.643	3056.103	3476.85
1024	4144260	3839437	6384030	50662.34	47894.34	52605.54	144991.4	143853.3	146069.2	59794.97	58896.17	60879.39	277.5258	270.2892	302.2465	9.089932	8.873598	9.965803	0.004641	0.001613	0.014822	7.36E-05	2.12E-05	0.000205	561206.3	536338.6	583032.8	18632.29	18177.6	19291.12
1448	21462452	15122604	24068843	173811.3	157022	176721.3	249388.9	246904.6	253193.9	96321.42	95034.84	97623.55	786.7639	772.8416	803.2531	45.98176	42.52486	46.623	0.008858	0.003806	0.023601	0.000479	7.6E-05	0.00102	2137509	2034789	2233341	118777.1	116082.1	122656.4

Table B.30: RMHC Descriptive Statistics: Rosenbrock to Sum Squares Function (50,000 Evaluation Tuning)

Dims.	Rosenbrock						Schwefel						Sphere						Sum of Different Powers						Sum Squares					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	2.19702	0.312791	4.344044	2.194636	0.312095	4.334645	59.22167	0.001111	118.4398	59.21946	5.39E-05	118.4384	7.17E-12	3.31E-12	2.1E-11	3.73E-13	1.63E-13	7.68E-13	1.28E-13	2.75E-14	5.58E-13	3.55E-15	1.69E-21	1.87E-14	9.15E-11	4.74E-11	2.37E-10	2.49E-12	7.11E-13	4.8E-12
3	0.796352	0.292354	3.506001	0.626672	0.289991	3.49569	118.4509	118.4392	236.8813	118.4388	118.4384	236.8767	7.23E-11	2.75E-11	1.08E-10	1.72E-12	3.73E-13	2.77E-12	9.24E-14	1.42E-14	1.5E-12	8.88E-15	1.36E-20	2.49E-14	2.81E-10	1.48E-10	9.02E-10	8.88E-12	2.4E-12	2.1E-11
4	0.588375	0.487528	4.196617	0.133514	0.125137	4.119014	236.8802	118.4514	355.3183	236.8769	118.4384	355.3154	3.01E-10	1.15E-10	5.68E-10	4.98E-12	1.93E-12	6.52E-12	1.42E-12	1.63E-13	4.22E-12	1.42E-14	3.55E-15	7.46E-14	1.86E-09	1.18E-09	9.92E-09	3.62E-11	2.17E-11	5.3E-11
6	1.69148	0.372032	4.203767	1.164342	0.067495	3.50325	355.325	236.8901	384.9642	355.3155	236.8775	384.9306	0.020517	4.28E-09	0.43443	1.65E-11	8.92E-12	2.47E-11	3.87E-08	4.23E-11	3.85E-05	5.68E-14	3.55E-15	1.39E-13	1.059651	1.79E-08	4.525923	1.42E-10	9.86E-11	3.45E-10
8	2.974347	0.777236	5.736926	2.933449	0.699405	5.143751	355.4291	355.3606	592.2785	355.3214	355.3176	592.1974	3.20468	1.634011	4.564991	4.61E-11	3.41E-11	6.63E-11	0.005512	0.000601	0.043342	2.01E-13	4.62E-14	5.12E-13	76.89623	63.6782	104.3368	8.72E-10	5.36E-10	1.41E-09
11	8.360463	5.275486	31.64123	1.660115	0.4642	2.95179	533.3287	474.0581	710.9678	532.9848	473.7587	710.6357	13.09847	10.91442	19.01606	1.78E-10	1.07E-10	2.54E-10	0.049464	0.005756	0.127366	3.55E-13	1.45E-13	6.11E-13	325.9655	257.8681	560.0472	3.06E-09	2.1E-09	3.79E-09
16	52.50219	15.32249	77.08941	5.025831	2.757002	72.23222	829.8053	711.8559	948.8524	829.1164	710.6887	947.5522	45.74615	33.74242	61.75941	7.19E-10	6.25E-10	1.1E-09	0.287245	0.087905	0.394967	5.56E-13	5.68E-14	1.39E-12	1339.362	1170.108	1661.349	2.91E-08	1.66E-08	4.35E-08
23	81.14109	67.76123	93.13945	72.20025	49.12051	79.10328	1246.836	1068.842	1456.098	1243.706	1066.089	1450.963	70.54893	50.31843	84.62012	4.34E-09	3.15E-09	4.84E-09	0.377694	0.249862	0.557451	1.24E-12	3.23E-13	5.14E-12	3791.168	3263.455	4876.586	3.55E-07	1.4E-07	9.4E-07
32	96.86456	72.21593	135.218	76.83982	14.58496	82.22005	1670.826	1426.743	1899.932	1658.508	1421.575	1895.457	65.08039	56.73956	78.04251	2.54E-08	2.16E-08	2.91E-08	6.96E-07	2.29E-07	4.02E-06	2.16E-08	6.99E-09	8.44E-08	6570.057	5402.876	7218.392	1.56E-06	1.02E-06	1.98E-06
45	213.0832	155.7065	243.3542	144.1181	94.17381	173.2407	2270.576	2115.382	2417.44	2251.155	2102.884	2399.526	100.9232	93.45373	117.8759	1.5E-07	1.13E-07	2.03E-07	5.25E-07	8.76E-09	1.53E-06	7.24E-09	1.22E-09	4.01E-08	11595.75	9820.232	12942.34	1.55E-05	1.07E-05	1.79E-05
64	287.5545	255.8477	316.211	145.9605	112.6016	162.3267	3205.791	2841.854	3431.066	3141.04	2785.577	3348.777	160.0406	145.9396	172.5303	9.17E-07	7.91E-07	1.15E-06	0.000331	1.02E-06	0.002063	8.84E-10	1.71E-10	2.48E-09	29877.29	25666.17	33347.25	0.000118	9.11E-05	0.000152
91	30657.61	19484.13	69904.77	298.7315	251.1892	391.2486	4596.762	4505.892	4772.322	4390.097	4357.561	4625.059	310.1681	294.3935	329.5344	5.71E-06	4.83E-06	6.14E-06	2.78E-05	1.84E-07	0.000295	4.07E-09	7.75E-10	7.52E-09	50240.26	49063.73	53969.68	0.000949	0.00066	0.001246
128	1125.85	906.5736	1460.118	431.0106	347.9048	663.3331	6838.38	6319.691	7385.56	6417.34	5907.688	6915.398	391.0634	369.9652	424.3882	3E-05	2.63E-05	3.75E-05	2.55E-06	6.06E-07	5.6E-05	1.2E-08	2.36E-09	6.03E-08	83728.6	78466.69	92118.52	0.0081	0.00625	0.009248
181	2142259	1861671	2876022	562.1081	513.9519	827.3734	10551.97	10054.8	11013.57	9250.6	8761.145	9890.341	545.4628	523.1997	562.1461	0.000168	0.000144	0.000212	0.024061	0.000272	0.074076	1.74E-08	4.91E-09	6.98E-08	164547.2	158817.6	170346.1	0.066988	0.055267	0.07366
256	13067265	10620846	14179956	945.8034	879.1779	1135.946	16671.08	15891.75	17034.31	13005.42	12184.13	13425.31	613.9904	565.7073	643.1282	0.001139	0.00101	0.0012	0.020413	0.010696	0.053098	4.79E-08	1.11E-08	2.24E-07	310120.2	290759.9	341575.6	0.492801	0.409969	0.555181
362	37866738	32950462	39892275	1454.856	1380.47	1500.904	28141.42	27112.54	29113.4	18626.98	17305.88	19184.16	937.9524	871.0384	958.6956	0.005879	0.005461	0.00641	0.223968	0.13783	0.414093	2.23E-07	2.67E-08	6.41E-07	558251.6	531955.6	585771.3	4.213233	3.88133	4.838971
512	42144176	40125479	44698942	2104.445	1973.015	2213.489	49234.95	47862.53	51239.58	27288.83	26600.54	27639.77	1158.158	1086.443	1205.66	0.03515	0.031078	0.039792	0.312741	0.096534	0.577252	4.43E-07	1.04E-07	3.15E-06	1300286	1263837	1344509	35.15913	32.04665	39.04214
724	54296100	50989051	59348979	4605.381	4409.273	4766.895	85617.73	84384.39	87505.02	39693.4	38044.18	40581.65	1661.106	1598.925	1776.31	0.207907	0.200832	0.228008	0.191314	0.129916	0.229945	2.66E-06	8.8E-07	5.74E-06	2200991	2125248	2269795	292.8351	276.1659	309.5556
1024	81159744	76568928	85288578	12300.64	11787.36	12906.05	145219.4	143991	147606.4	60968.05	58151.32	61718.4	2145.214	2132.205	2173.475	1.342954	1.248805	1.546963	0.178488	0.110365	0.309674	8.4E-06	1.2E-06	6.67E-05	4624339	4460971	4760801	2454.991	2285.461	3482.912
1448	1.11E+08	1.04E+08	1.17E+08	45006.87	43606.91	48717.65	249683.8	246305.4	252009.8	95607.21	93502.34	97417.53	3030.024	2929.501	3088.789	7.910145	7.483692	8.649408	0.096926	0.031906	0.236446	8.18E-05	1.61E-05	0.000485	8835251	8332490	9208520	19864.71	17724.01	20246.21

Table B.31: SA Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (10,000 Evaluation Tuning)

Dims.	Ackley						Alpine1					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.038029	0.027768	0.048049	0.005542	0.003596	0.008976	0.003486	0.002818	0.004717	0.001818	0.001277	0.002327
3	0.092962	0.065365	0.119114	0.019794	0.010588	0.022453	0.05078	0.032854	0.085825	0.003082	0.002422	0.004278
4	0.682477	0.461336	0.991866	0.042078	0.029619	0.059779	0.036768	0.024013	0.068497	0.003802	0.003104	0.005249
6	4.681222	3.860838	12.94673	0.0706	0.060567	0.086548	0.230407	0.204076	0.257022	0.006663	0.005898	0.009987
8	2.425482	2.126078	2.557572	0.094288	0.08919	0.108142	0.852376	0.743546	1.153853	0.033627	0.023417	0.043928
11	3.74447	3.466683	3.95647	0.259101	0.220752	0.290852	1.879418	1.560688	2.017042	0.03983	0.033046	0.060077
16	18.92187	18.60179	19.40913	0.17399	0.145344	0.220471	1.619896	1.348573	1.755725	0.054865	0.05043	0.070354
23	19.44683	19.1764	19.86031	0.358059	0.308412	0.408282	3.854049	3.479285	4.124825	0.125878	0.112791	0.155796
32	19.90216	19.64216	20.00299	1.047932	0.939763	1.25399	5.898791	5.213943	6.339901	0.276651	0.239548	0.362444
45	19.99146	19.74259	20.18364	2.046994	1.831349	2.16313	10.98343	10.42313	11.93294	0.426191	0.382735	0.496047
64	20.28878	20.21897	20.38356	3.13812	3.053058	3.335815	18.17546	16.76798	18.9076	0.903187	0.79212	1.047544
91	20.4091	20.35242	20.47485	5.617164	5.385127	5.818401	24.35943	23.49147	26.13821	1.230608	1.103848	1.401254
128	20.62791	20.59994	20.67931	9.037491	8.536895	9.168115	35.40743	34.20058	37.35517	2.669836	2.599479	2.920315
181	20.75757	20.71282	20.79614	13.53702	13.12303	14.09199	62.02528	60.67329	65.71398	4.374836	4.133415	5.004555
256	20.81211	20.79409	20.83553	16.83674	16.53766	17.48697	80.97813	79.05578	83.07216	7.546349	7.200392	8.109054
362	20.88239	20.8576	20.90849	18.29803	18.12147	18.52123	113.5971	106.9035	119.7426	13.87217	13.24176	14.61969
512	20.91895	20.87653	20.93861	19.21621	19.06356	19.34533	201.4692	196.4489	204.9879	26.18284	24.52974	26.51628
724	20.94974	20.93267	20.97444	20.19699	20.13712	20.24694	285.6091	279.3331	291.2145	44.49149	41.91595	45.80538
1024	20.98926	20.9741	21.00625	20.69952	20.66945	20.72617	452.1505	445.4853	466.117	84.10081	81.09935	86.29897
1448	21.02796	21.00476	21.04555	20.76718	20.7342	20.7909	785.1712	778.4928	794.4247	159.8038	156.6783	163.97

Dims.	Bent Cigar						Brown					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.002205	0.00051	0.009397	8.56E-05	1.4E-05	0.000172	0.000267	0.000102	0.00055	6.03E-06	2.1E-06	1.19E-05
3	0.147295	0.099374	0.38951	0.010469	0.007428	0.02177	0.002392	0.000703	0.00381	4.2E-05	1.76E-05	8.79E-05
4	0.178504	0.112155	0.265515	0.03599	0.025001	0.05268	1.052206	0.164866	1.826029	4.63E-05	2.86E-05	6.89E-05
6	1.494075	0.735565	2.750225	0.050456	0.028976	0.068116	0.656693	0.521379	0.780567	0.000184	7.43E-05	0.000267
8	0.937789	0.775849	1.377699	0.107201	0.083679	0.174085	0.493109	0.416075	0.737553	0.000327	0.000274	0.00049
11	2.504338	1.779958	4.248682	0.136218	0.126397	0.186629	2.176638	1.901112	2.758136	0.004192	0.00304	0.005957
16	6.444819	4.443559	9.899033	0.394002	0.29917	0.456054	1.460473	1.317266	1.715612	0.01232	0.010497	0.016146
23	8.314344	6.012291	10.82929	1.100561	0.890408	1.344255	3.892365	3.503844	4.428921	0.02063	0.016596	0.025985
32	21.93391	15.59257	29.92653	2.252222	1.808988	2.759391	4.531342	4.208605	5.253815	0.081974	0.075432	0.10746
45	103.3131	75.6237	111.6424	4.490924	3.829367	5.478469	9.004781	8.461985	9.523869	0.108111	0.097493	0.128905
64	577.16	516.5518	658.6984	55.26821	48.5776	65.79299	8.401063	8.05377	8.969864	0.270329	0.243693	0.345805
91	3431.704	3080.391	3870.844	84.11082	81.30688	92.66206	23.94794	22.44465	25.00629	0.397548	0.376418	0.440651
128	22260.86	18006.23	26014.43	491.6071	444.1116	569.962	34.72092	31.88455	35.27049	0.905803	0.835601	0.986213
181	96765.1	81378.24	115156.9	2270.023	2088.772	2513.375	51.66366	49.1078	53.94189	1.813497	1.635165	1.920733
256	618359.1	539179.4	789774.1	15005.03	13805.95	16276.11	82.4892	78.74811	85.86099	3.214162	3.126342	3.439278
362	3420323	3004547	4074179	86883.76	82328.6	90562.13	133.1914	122.3809	140.1918	6.614985	6.38189	6.745227
512	18400962	15265874	20455066	418525.9	387103.3	456083	245.3328	235.2255	252.3171	32.61769	31.62258	33.82296
724	72834978	62869898	82859083	2092649	1941554	2171516	194.3917	187.9692	202.0464	42.15657	40.24024	43.2258
1024	2.53E+08	2.4E+08	2.64E+08	8289414	8040110	8751388	597.8805	570.9837	682.3518	181.0026	170.9614	185.0925
1448	6.89E+08	6.45E+08	7.24E+08	38550798	37627441	40301220	98726.12	12964.66	1427847	1824.701	1754.061	1859.932

Dims.	Chung-Reynolds						Deflected Corrugated Spring					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	4.82E-09	1.4E-09	4.55E-08	6.21E-11	8.62E-12	1.93E-10	-0.99486	-0.9974	-0.98065	-0.99919	-0.99964	-0.99848
3	9.15E-06	3.68E-06	3.16E-05	5.44E-08	1.7E-08	2.48E-07	-0.84334	-0.90999	-0.84334	-0.96552	-0.98401	-0.92759
4	0.000598	0.000259	0.000875	1.11E-06	5.92E-07	2.02E-06	-0.84333	-0.84334	-0.84333	-0.84334	-0.84558	-0.84334
6	0.362174	0.221219	0.481548	3.23E-05	2.22E-05	0.000121	-0.84331	-0.84334	-0.8432	-0.84334	-0.84334	-0.84334
8	0.067153	0.045955	0.086674	0.000128	4.11E-05	0.000219	-0.40781	-0.81389	-0.37336	-0.84334	-0.84334	-0.84334
11	0.775456	0.509534	1.070251	0.002914	0.001541	0.006167	-0.37335	-0.37336	-0.37333	-0.84314	-0.84332	-0.80322
16	1.256676	1.13041	1.862921	0.005962	0.003851	0.009981	0.410037	0.409925	0.410629	-0.37336	-0.37336	-0.37336
23	7.943176	6.285641	9.010558	0.077594	0.053471	0.097852	1.510992	1.507965	2.09083	0.409923	0.409922	0.409939
32	1.293167	0.707611	1.554015	0.077574	0.055243	0.092504	2.925785	2.917337	3.307665	1.506499	1.468988	1.5065
45	1.225407	0.57273	1.874449	0.124908	0.077009	0.143982	4.639478	4.639436	4.640207	2.916346	2.529141	2.916346
64	4.974664	3.852288	5.278199	0.35538	0.277969	0.394738	9.025157	6.67909	9.025175	4.639431	4.639431	6.675717
91	14.4967	13.45965	16.5111	1.184911	1.055731	1.340028	11.6877	11.68769	11.68861	6.675719	6.675717	8.918549
128	70.23421	49.37197	101.2555	2.662011	2.091212	3.005861	21.55307	21.55307	22.44601	14.66325	13.91936	14.66325
181	3126.102	2407.24	3768.055	11.01555	9.009481	11.96581	30.87031	29.57543	33.40116	21.28354	17.95174	21.55307
256	97610.13	79769.78	130769	104.5533	92.26686	123.7146	50.85397	49.39402	55.12917	29.69368	29.69368	34.14265
362	1981123	1763807	2766552	1589.576	1326.479	1798.184	74.16733	67.76619	80.25913	44.24621	39.08356	44.24621
512	48871822	37265662	73227601	36241.29	32591.14	38262.07	112.4908	109.6069	114.5221	49.71999	49.71999	49.71999
724	9.01E+08	6.79E+08	1.21E+09	629401.9	585695.1	692944.9	152.8411	148.3173	158.3384	55.50393	55.50393	55.50393
1024	1.01E+10	9.65E+09	1.09E+10	11198068	10525142	12165486	270.6001	260.2937	273.6968	60.77195	58.01702	61.59567
1448	7.69E+10	6.92E+10	8.4E+10	2.33E+08	2.15E+08	2.61E+08	329.1169	315.8455	334.0049	76.20642	74.72783	77.28252

Table B.32: SA Descriptive Statistics: Ackley to Deflected Corrugated Spring Functions (50,000 Evaluation Tuning)

Dims.	Ackley						Alpine1					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.013266	0.007563	0.015418	0.003346	0.002358	0.004272	0.041475	0.013865	0.623303	0.000494	0.000283	0.001118
3	1.004392	0.632466	1.807815	0.017742	0.012454	0.023409	1.560158	0.142601	3.189828	0.001654	0.00128	0.00282
4	0.224103	0.16616	0.306993	0.028804	0.022519	0.036689	3.986563	2.087947	5.171151	0.003833	0.003004	0.004962
6	2.14329	2.022887	2.352775	0.063799	0.038109	0.079424	8.932991	6.363673	14.83774	0.014569	0.012247	0.019328
8	10.77302	3.206338	19.24981	0.05895	0.050027	0.074348	1.02118	0.807644	1.129903	0.025445	0.022419	0.042966
11	19.17829	17.89897	19.45104	0.121458	0.101601	0.131974	0.926979	0.666102	0.97313	0.035256	0.021567	0.048691
16	19.69198	19.26833	20.02056	0.178762	0.168268	0.204799	2.614207	2.404847	2.773444	0.057934	0.05187	0.083881
23	20.17749	19.8848	20.29623	0.208589	0.183829	0.234752	10.4294	9.893894	12.12516	0.257298	0.222464	0.302209
32	20.32248	20.20177	20.54587	0.456155	0.401005	0.557335	39.19114	37.62289	42.79627	0.241757	0.210597	0.272237
45	20.55755	20.44177	20.65474	1.116642	0.899223	1.238564	12.55204	11.49015	12.94509	0.438866	0.374364	0.502975
64	20.68006	20.58748	20.71517	3.046677	2.747386	3.231025	51.6989	46.31357	64.33163	0.641357	0.582243	0.729746
91	20.62716	20.53238	20.6356	5.231202	5.000754	5.33495	91.18889	81.82117	99.38806	1.463413	1.231254	1.637625
128	20.66475	20.63687	20.70809	7.767416	7.237778	8.272994	194.103	181.6008	202.1985	2.304377	2.15667	2.757427
181	20.70101	20.6374	20.76547	12.45594	12.13648	13.18554	54.71225	53.1248	56.02729	4.92049	4.028302	5.244241
256	20.81956	20.78815	20.84675	15.25152	14.9908	15.40907	85.17337	83.04989	87.05644	10.0954	9.570969	10.70677
362	20.86708	20.84462	20.88723	18.05505	17.50988	18.29044	117.7784	111.8839	122.8925	14.06959	13.66391	14.57777
512	20.92302	20.90334	20.93659	19.3013	19.20242	19.41412	206.6316	204.8177	215.5892	26.30804	25.0805	26.91013
724	20.95279	20.93741	20.96417	20.19745	20.15759	20.25223	315.8288	312.1713	323.6775	46.98729	46.18485	49.47834
1024	20.98495	20.96701	21.00063	20.47635	20.41811	20.53006	454.8511	441.686	462.6872	79.48711	78.13456	81.50846
1448	21.01079	20.99563	21.02692	20.78129	20.76593	20.81113	817.7923	803.1484	837.6463	146.6737	143.8527	149.1938

Dims.	Bent Cigar						Brown					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	2.139937	0.583855	9.036058	0.00014	3.2E-05	0.000379	8.41E-05	4.13E-05	0.000155	1.59E-06	7.86E-07	3.28E-06
3	7.184205	2.223402	12.34444	0.028413	0.010821	0.074221	0.01411	0.009387	0.027949	3.87E-05	1.55E-05	9.08E-05
4	0.160625	0.057341	0.303988	0.023755	0.011706	0.044384	0.341672	0.181941	0.704655	5.7E-05	2.15E-05	8.05E-05
6	1.327431	0.635604	2.371986	0.230015	0.149085	0.301873	1.882911	0.717815	2.957316	2.51E-05	1.65E-05	3.04E-05
8	1.353231	0.789685	1.812155	0.160354	0.099225	0.225642	3.539286	2.822994	4.498428	0.000221	0.000151	0.000343
11	2.255439	1.55063	2.83776	0.542055	0.426133	0.847965	1.108025	0.915544	1.440209	0.000689	0.00053	0.001085
16	13.87037	10.0246	18.39023	0.727392	0.645187	0.790704	5.967557	4.610751	6.400381	0.005652	0.004315	0.006861
23	10.43232	6.322473	12.76599	0.960131	0.679774	1.155073	7.896796	6.525286	9.750118	0.011235	0.00969	0.01249
32	22.2021	19.90409	31.82342	2.412427	1.932542	2.75283	9.976559	8.509617	11.15421	0.010957	0.009096	0.011755
45	98.42548	89.97927	103.3829	3.100769	2.649761	4.170084	6.490767	5.283485	7.462668	0.038925	0.036992	0.050707
64	40189690	30268077	47828750	5.028312	4.46571	5.499545	14.98756	13.8254	15.86409	0.244723	0.224983	0.276625
91	2.15E+08	1.97E+08	2.32E+08	10.97222	10.13667	11.74342	95.33398	85.62365	107.2309	0.302857	0.260465	0.351343
128	3.07E+08	2.89E+08	3.22E+08	35.80077	33.58482	40.5549	26.48765	24.76171	27.08677	0.513478	0.475106	0.571609
181	5.02E+08	4.7E+08	5.09E+08	181.9545	159.4634	207.5509	63.09658	59.63789	67.40423	1.268525	1.14668	1.406644
256	6.56E+08	6.26E+08	6.85E+08	929.1338	851.76	1025.839	36332.49	16704.68	154974.6	2.162347	1.941007	2.295167
362	8.51E+08	8.04E+08	8.7E+08	5270.61	4776.279	5966.103	279.7448	253.4202	313.3449	4.880331	4.64632	5.217234
512	1.13E+09	1.07E+09	1.17E+09	32711.83	30441.91	36106.69	182.4313	172.0841	199.283	9.511814	9.210529	10.17101
724	1.48E+09	1.44E+09	1.54E+09	202207.6	192522.7	212364	5.61E+09	1.19E+09	4.12E+10	18.37674	17.79907	19.35978
1024	1.92E+09	1.86E+09	2.01E+09	1183652	1126015	1301445	153891.6	37712.42	2617585	47.372	45.6595	49.17159
1448	2.64E+09	2.55E+09	2.68E+09	7289247	6986870	7855337	7.87E+10	1.74E+09	2E+12	76.40704	73.84588	78.79339

Dims.	Chung-Reynolds						Deflected Corrugated Spring					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	3.97E-09	4.29E-10	2.85E-08	2.41E-11	1.09E-12	9.4E-11	-0.99172	-0.99813	-0.98434	-0.99987	-0.99997	-0.99969
3	4.81E-06	1.17E-06	2.79E-05	6.07E-08	1.96E-08	2.36E-07	-0.86242	-0.9342	-0.84334	-0.97706	-0.98899	-0.93575
4	0.000693	0.000169	0.001202	5.52E-06	2.64E-06	9.13E-06	-0.84334	-0.84334	-0.84333	-0.84334	-0.84334	-0.84334
6	71.09507	13.54044	3966.275	1.17E-06	3.15E-07	2.09E-06	-0.8433	-0.84333	-0.84297	-0.84334	-0.84334	-0.84334
8	773049.2	193637	3026884	0.000343	0.000181	0.000565	-0.83419	-0.8399	-0.76799	-0.84334	-0.84334	-0.84334
11	31245610	16124170	50213655	0.000161	5.37E-05	0.000284	-0.37335	-0.37336	-0.37332	-0.77952	-0.84317	-0.39661
16	59270.16	7978.917	221219.2	0.00263	0.002182	0.004178	0.409932	0.409923	0.409958	-0.37336	-0.37336	-0.37324
23	1.09E+09	8.51E+08	1.3E+09	0.002817	0.002377	0.003495	1.5065	0.419393	1.5065	0.409922	0.409922	0.409923
32	1.68E+09	1.41E+09	2.29E+09	0.011738	0.007046	0.013416	2.916346	2.393781	2.916357	1.506499	1.506499	1.506499
45	5.99E+08	4.12E+08	7.88E+08	0.047461	0.039899	0.064022	4.639439	4.571545	4.639491	2.916346	2.916346	2.916346
64	1760322	608257.3	2834182	0.20331	0.182116	0.240453	9.02516	8.431409	9.02517	4.639431	4.639431	6.675717
91	4.78E+09	3.8E+09	6.08E+09	0.358175	0.327432	0.434275	13.3834	11.6877	14.53691	9.025156	6.675717	9.025156
128	4.28E+09	3.54E+09	5.32E+09	0.740254	0.631032	0.866692	19.22053	17.95175	21.55307	11.68769	11.68769	14.66325
181	1.98E+10	1.85E+10	2.31E+10	1.613417	1.344868	1.983382	44.23957	43.41783	44.24734	21.55307	21.55307	22.35144
256	3.9E+10	3.37E+10	4.79E+10	4.368707	3.857625	4.718846	55.39441	54.87638	59.1874	29.69368	29.69368	34.23259
362	8.76E+10	7.54E+10	9.38E+10	13.10841	11.21441	13.76953	80.14021	74.69076	81.66205	44.19056	39.08356	44.24621
512	1.57E+11	1.47E+11	1.67E+11	195.5937	157.232	237.6641	103.7197	102.2657	103.9979	49.71999	49.71999	49.71999
724	3.21E+11	2.9E+11	3.39E+11	6837.149	5802.066	8185.737	178.8694	169.2372	188.3706	55.50393	55.50393	55.50393
1024	6.56E+11	6.13E+11	6.85E+11	228725.2	203049.7	261116.3	294.5308	287.7535	301.1288	57.84385	55.6918	61.2128
1448	1.12E+12	1.08E+12	1.26E+12	6834381	6137285	7957843	356.8364	342.5884	359.8381	65.60801	62.34912	67.43942

Table B.33: SA Descriptive Statistics: Exponential to Rastrigin Functions (10,000 Evaluation Tuning)

Dims.	Exponential						Griewank						Inverted Cosine Wave					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	-0.99997	-0.99999	-0.99993	-0.99999	-1	-0.99999	0.020263	0.015759	0.031548	0.004133	0.001664	0.007814	-0.98612	-0.99367	-0.97434	-0.99978	-0.99991	-0.9994
3	-0.99787	-0.9986	-0.99604	-0.99999	-1	-0.99998	0.076438	0.041175	0.086187	0.011466	0.010229	0.013635	-1.79267	-1.9044	-1.64282	-1.99806	-1.9985	-1.99656
4	-0.9913	-0.99544	-0.98426	-0.99997	-0.99998	-0.99996	0.235916	0.202188	0.272254	0.016954	0.013565	0.019677	-2.44639	-2.48125	-2.41652	-2.99762	-2.99799	-2.99601
6	-0.8609	-0.89186	-0.83934	-0.99994	-0.99997	-0.99991	0.641178	0.549953	0.744792	0.048173	0.041618	0.056684	-3.6608	-3.73103	-3.56684	-4.99122	-4.99325	-4.98938
8	-0.73477	-0.76397	-0.7065	-0.99983	-0.9999	-0.99976	1.08987	1.025253	1.122806	0.0767	0.058126	0.101679	-4.08759	-4.37537	-3.81061	-6.98342	-6.98664	-6.97809
11	-0.61849	-0.69602	-0.53261	-0.99977	-0.99987	-0.99969	1.530546	1.474953	1.592127	0.156725	0.132311	0.178515	-6.10829	-6.39007	-5.92701	-9.95955	-9.97009	-9.9467
16	-0.45995	-0.51047	-0.42631	-0.99776	-0.99829	-0.99731	33.05654	25.37219	41.44089	0.621346	0.52094	0.748996	-7.92351	-8.65574	-7.70356	-14.894	-14.9178	-14.8669
23	-0.2157	-0.24767	-0.19992	-0.99312	-0.99378	-0.9853	23.40548	19.04037	28.2038	1.009697	0.884268	1.019941	-10.389	-11.0027	-9.51091	-21.8655	-21.9038	-21.7953
32	-0.11647	-0.1274	-0.09907	-0.97987	-0.98499	-0.97708	10.38051	8.438876	12.12915	0.969253	0.863624	1.015432	-13.186	-14.4242	-12.6306	-30.6596	-30.7699	-29.4478
45	-0.03897	-0.05589	-0.02908	-0.77881	-0.81799	-0.76308	12.89958	11.63973	14.02134	1.047863	1.041586	1.055844	-17.2464	-18.0052	-16.4305	-42.1595	-43.3355	-41.3826
64	-0.00426	-0.00691	-0.00377	-0.43231	-0.48522	-0.29639	25.83385	23.40936	27.56219	1.219435	1.168316	1.234836	-24.886	-26.0521	-23.4078	-59.3011	-60.3639	-57.5235
91	-0.00018	-0.00025	-0.00012	-0.00033	-0.0004	-0.00025	23.80837	22.22996	26.27836	1.370773	1.325356	1.42957	-31.0374	-32.1941	-28.5144	-81.9933	-83.2037	-79.191
128	-9.2E-10	-1.8E-09	-1.2E-10	-9.8E-10	-1.9E-09	-1.3E-10	38.88943	35.78175	41.04219	1.602984	1.549817	1.668347	-41.8919	-44.5846	-40.162	-110.26	-111.426	-106.972
181	-1.7E-13	-5.1E-13	-4.9E-14	-2.1E-13	-5.2E-13	-6.3E-14	43.1168	41.88789	45.15333	2.445673	2.361254	2.564448	-66.3561	-69.4117	-65.052	-143.101	-144.59	-139.522
256	-7.9E-19	-2.1E-18	-1.2E-19	-8.3E-19	-2.2E-18	-1.2E-19	60.29299	57.30617	64.96844	4.624872	4.472689	4.859428	-83.1008	-87.3321	-80.0117	-199.208	-201.121	-197.652
362	-6.4E-27	-1E-26	-3E-27	-6.7E-27	-1.1E-26	-3.4E-27	86.57121	82.00368	89.3887	10.98442	10.50761	11.58731	-118.029	-119.97	-114.066	-267.092	-268.852	-265.208
512	-1.4E-37	-1.7E-36	-1E-39	-1.5E-37	-1.9E-36	-1.4E-39	166.6371	158.5699	176.0268	21.08306	20.6971	22.60165	-158.863	-166.498	-154.746	-373.111	-376.807	-371.122
724	-4.7E-53	-1.2E-52	-6.1E-54	-5E-53	-1.3E-52	-6.4E-54	386.5417	376.7781	406.1708	50.11885	47.29997	52.82436	-209.112	-214.408	-200.146	-511.603	-515.984	-510.555
1024	-2.5E-74	-1E-73	-1.8E-76	-2.6E-74	-1.1E-73	-2.4E-76	1091.704	1021.344	1170.998	110.6773	107.6087	113.4367	-273.755	-279.341	-268.259	-699.073	-702.795	-694.13
1448	-2E-106	-3E-104	-4E-109	-2E-106	-3E-104	-5E-109	2612.161	2454.049	2729.136	268.9262	265.006	279.0114	-316.84	-334.153	-309.975	-929.832	-936.788	-926.241

Dims.	Levy						Qing						Rastrigin					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.000339	0.000145	0.000556	3.8E-05	2.52E-05	5.29E-05	0.000358	0.00017	0.000537	1.75E-05	1.02E-05	2.94E-05	0.002659	0.001619	0.004103	0.000465	0.000154	0.000546
3	0.00602	0.004141	0.010119	0.000191	0.000133	0.000463	0.006609	0.004983	0.013187	0.000717	0.000406	0.001019	0.03316	0.012353	0.059314	0.002467	0.001604	0.003446
4	0.288234	0.197559	0.405469	0.003174	0.002433	0.008357	0.035042	0.028389	0.059343	0.001798	0.000937	0.00277	0.054066	0.034328	0.094984	0.00418	0.001954	0.006778
6	0.165306	0.145152	0.209727	0.005863	0.003857	0.00874	2.974551	1.673048	3.742383	0.060404	0.04001	0.095492	0.23642	0.168963	0.334194	0.020719	0.014992	0.035265
8	0.270026	0.207951	0.296821	0.013644	0.011515	0.017822	0.732366	0.529681	0.974949	0.157994	0.119171	0.185179	0.533274	0.408239	0.798208	0.036737	0.026104	0.04686
11	0.758504	0.65235	0.820593	0.014061	0.010096	0.020568	1.242375	0.946364	1.482589	0.143092	0.121934	0.173118	1.130633	0.849398	1.394042	0.068369	0.057789	0.109905
16	1.511974	1.330433	1.692741	0.050525	0.032178	0.070387	2.724657	2.061857	3.146328	0.282632	0.205965	0.353433	3.27857	2.918237	3.703606	0.105972	0.080198	0.137358
23	2.650459	2.357881	2.945716	0.126748	0.098635	0.151377	6.418717	5.091761	7.113258	0.588659	0.457186	0.657897	10.94375	10.26568	11.75308	0.360262	0.308367	0.401561
32	4.229399	3.622234	4.366278	0.143211	0.117888	0.168196	15.78409	12.73218	18.50764	1.223063	1.020563	1.545902	9.239077	8.237981	10.39428	0.45713	0.385906	0.528129
45	6.401655	6.040388	6.756573	0.296486	0.259007	0.349176	36.11475	31.01581	42.13286	2.97954	2.757982	3.789209	26.91586	26.20114	28.61454	1.075019	0.942332	1.172734
64	11.66239	11.01641	12.5714	0.491786	0.438425	0.579481	184.532	161.2284	249.9615	6.939219	6.112518	7.597305	26.43733	24.95405	28.06258	1.369154	1.184455	1.525352
91	14.4938	13.47688	14.61437	0.776706	0.735402	0.82025	1882.313	1544.526	2046.369	59.64612	51.56783	70.13715	35.57729	31.59843	39.34976	2.376218	2.158146	2.783262
128	21.56842	20.52372	22.46724	1.677581	1.596815	1.766891	12479.69	9462.327	15213.35	317.8572	284.7989	341.0964	53.41754	50.533	55.8544	4.57648	4.388664	5.031078
181	36.43262	34.63525	38.40776	4.088976	3.822152	4.2606	103255.6	88172.7	124693.2	3379.869	3141.567	3984.998	132.6759	121.8866	142.2038	10.61119	9.750498	11.22275
256	66.11904	63.5322	70.40155	6.713394	6.42274	6.981288	1337024	1055930	1686398	40973.47	34662.5	43768.16	207.0908	197.3934	211.9956	30.87591	29.00789	32.9437
362	79.48039	76.53054	82.61352	12.32468	12.03256	12.83966	30036475	25052439	37908022	344554.7	304897.7	368061.2	397.5402	384.6267	409.0387	57.50339	55.03731	59.11632
512	123.5943	118.8034	128.9065	21.05343	20.19882	21.75854	4.69E+08	3.91E+08	5.68E+08	2241439	2127008	2322365	907.825	878.0615	933.2682	167.0461	161.4899	171.8753
724	195.4188	186.5215	200.9435	38.6463	36.8987	39.57217	6.76E+09	5.04E+09	8.27E+09	11005667	10092498	11646853	1718.576	1695.999	1764.657	409.0169	399.2502	415.8551
1024	373.1104	359.9339	391.0157	61.86948	61.09548	62.48606	3.85E+10	2.89E+10	5.49E+10	84650852	75931674	89485732	3318.457	3266.107	3340.344	708.2693	695.1478	721.5724
1448	891.1519	825.3269	922.2313	115.5785	114.8404	118.6578	1.89E+11	1.81E+11	2.19E+11	5.84E+08	5.38E+08	6.52E+08	6559.059	6510.88	6728.218	1430.482	1413.775	1446.756

Table B.34: SA Descriptive Statistics: Exponential to Rastrigin Functions (50,000 Evaluation Tuning)

Dims.	Exponential						Griewank						Inverted Cosine Wave					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	-0.99996	-0.99998	-0.99995	-1	-1	-0.99999	0.181885	0.106507	0.648113	0.008167	0.006791	0.013418	-0.99176	-0.99537	-0.98293	-0.99987	-0.99993	-0.99974
3	-0.99893	-0.99917	-0.99877	-0.99998	-0.99999	-0.99996	0.117235	0.088128	0.147619	0.010879	0.00893	0.012703	-1.71913	-1.80501	-1.63801	-1.99757	-1.99846	-1.99661
4	-0.91728	-0.94839	-0.83574	-1	-1	-0.99999	0.288893	0.240163	0.349772	0.028551	0.017882	0.034463	-2.42979	-2.48247	-2.31705	-2.99765	-2.99861	-2.99662
6	-0.85062	-0.90655	-0.80078	-0.99998	-0.99999	-0.99997	0.853728	0.747889	0.946555	0.042226	0.034297	0.050218	-3.57366	-3.65694	-3.48383	-4.98502	-4.99236	-4.97956
8	-0.80226	-0.83919	-0.73822	-0.99989	-0.99991	-0.99983	1.270388	1.205306	1.378626	0.072065	0.065451	0.081417	-4.80489	-5.01736	-4.66417	-6.97813	-6.98478	-6.97134
11	-0.51455	-0.60206	-0.38797	-0.99994	-0.99995	-0.99993	1.337416	1.272933	1.409211	0.134257	0.11396	0.163528	-6.1794	-6.62495	-6.01229	-9.94669	-9.96326	-9.93069
16	-0.3885	-0.41963	-0.35047	-0.99862	-0.99888	-0.99817	131.1032	114.1095	148.4595	0.115731	0.068507	0.18053	-9.32183	-9.71492	-8.98518	-14.8822	-14.8935	-14.8496
23	-0.16243	-0.1885	-0.14928	-0.98695	-0.9914	-0.98578	176.9441	145.6224	202.8533	0.348563	0.299052	0.512803	-7.89797	-8.49618	-7.72177	-21.7508	-21.7763	-20.6207
32	-0.10427	-0.11524	-0.09243	-0.97145	-0.97533	-0.964	7.749634	6.968125	8.135732	1.031078	1.014499	1.040352	-13.2809	-13.9718	-12.3788	-30.0165	-30.721	-29.0535
45	-0.0387	-0.04637	-0.03188	-0.82631	-0.83692	-0.78985	347.2701	309.1986	377.8863	1.05624	1.047154	1.09068	-16.8987	-18.038	-16.4148	-42.2999	-43.4097	-41.8353
64	-0.00455	-0.00602	-0.00336	-0.52858	-0.56443	-0.47002	465.607	441.4425	513.7991	1.039716	1.027163	1.05519	-24.05	-25.2664	-22.9939	-59.5618	-60.8629	-58.1613
91	-0.00019	-0.00027	-0.00016	-0.00028	-0.0004	-0.00025	178.7344	156.2277	203.9982	1.17807	1.165261	1.196718	-26.5729	-28.7803	-24.5455	-79.4499	-80.9003	-77.3073
128	-5.8E-10	-2.9E-09	-1.9E-10	-6.6E-10	-3.1E-09	-1.9E-10	966.4873	912.9184	1011.29	1.348605	1.299802	1.391135	-37.4755	-38.5249	-33.4122	-109.181	-111.613	-107.045
181	-1.2E-13	-5E-13	-1.9E-14	-1.2E-13	-5.1E-13	-1.9E-14	846.9821	797.6623	928.302	1.788149	1.746424	1.846075	-52.5307	-54.3692	-49.1053	-149.965	-152.608	-148.484
256	-4.2E-19	-3.3E-18	-1.3E-19	-4.8E-19	-3.6E-18	-1.4E-19	1324.354	1301.526	1430.394	2.234859	2.184249	2.341277	-64.1314	-67.6073	-60.5473	-199.954	-202.168	-196.67
362	-6E-26	-2.7E-25	-1.4E-27	-6.7E-26	-2.8E-25	-1.6E-27	4330.202	4135.507	4420.414	4.449184	4.317655	4.750436	-95.4145	-97.9288	-89.5835	-274.457	-277.245	-272.231
512	-1.3E-37	-1.1E-36	-1.8E-38	-1.4E-37	-1.1E-36	-1.8E-38	5475.514	5249.603	5607.845	9.006524	8.535539	9.345729	-128.754	-130.138	-123.279	-374.63	-378.35	-372.157
724	-9.9E-54	-2.2E-52	-6.9E-55	-9.9E-54	-2.2E-52	-7E-55	2878.146	2799.762	2998.99	18.45309	17.95941	19.08507	-197.909	-213.737	-194.67	-513.564	-517.513	-511.765
1024	-2.5E-74	-4.6E-73	-5E-76	-2.6E-74	-4.8E-73	-5.4E-76	10952.08	10616.27	11288.74	40.29085	36.14999	41.66841	-283.38	-288.163	-273.418	-698.609	-701.318	-695.186
1448	-5E-105	-6E-103	-4E-107	-6E-105	-7E-103	-5E-107	12626.63	12261.23	12802.57	96.61454	92.87832	103.5286	-363.86	-370.379	-352.847	-942.066	-949.507	-936.991

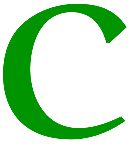
Dims.	Levy						Qing						Rastrigin					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	7.94E-05	4.85E-05	0.000187	1.15E-05	5.67E-06	3.22E-05	0.000262	0.000111	0.00047	1.66E-05	3.28E-06	4.94E-05	0.002164	0.000826	0.005015	0.000266	0.000102	0.000331
3	0.005225	0.002274	0.006483	0.000448	0.000272	0.000578	0.005894	0.003649	0.012033	0.000425	0.00033	0.000613	0.012064	0.006249	0.021603	0.001098	0.000887	0.002787
4	0.03069	0.023006	0.056586	0.002276	0.001515	0.003405	0.036401	0.020724	0.053578	0.00211	0.001223	0.003239	0.052313	0.040273	0.07589	0.003659	0.002323	0.004729
6	0.11796	0.090832	0.140586	0.005025	0.003319	0.007371	78349.35	0.436742	4365240	0.009538	0.005282	0.011762	0.878826	0.34261	1.197535	0.024182	0.014459	0.027588
8	0.293756	0.229787	0.355116	0.008364	0.005987	0.010142	6.18E+08	2.29E+08	1.25E+09	0.01992	0.014693	0.024493	0.657576	0.607461	0.911134	0.050608	0.03232	0.059068
11	0.49884	0.435399	0.53469	0.015669	0.008741	0.020256	7.72E+09	4.73E+09	1.23E+10	0.046845	0.034817	0.063741	2.061404	1.237755	2.354029	0.068052	0.057168	0.105105
16	1.083023	1.017591	1.216596	0.047505	0.038308	0.074682	3.48E+10	2.19E+10	4.58E+10	0.142432	0.072465	0.168071	4.067626	3.468027	4.945135	0.136183	0.095088	0.200423
23	1.969216	1.800668	2.068881	0.081735	0.064538	0.09957	8.64E+10	6.68E+10	9.73E+10	0.198299	0.175862	0.236643	5.490464	4.2867	6.316149	0.241138	0.197256	0.302332
32	3.182293	2.943091	3.39677	0.127812	0.094249	0.158356	1.69E+10	1.13E+10	1.98E+10	0.943976	0.794364	1.255672	11.90798	10.17953	13.52093	0.390768	0.354137	0.439326
45	6.346731	5.996513	6.585916	0.262307	0.230541	0.29546	1.4E+10	1.2E+10	2.01E+10	2.414506	2.223822	2.812487	18.43525	16.83873	19.5578	0.959552	0.755846	1.048009
64	14.83846	12.09219	16.63167	0.56502	0.480995	0.631713	3.93E+10	3.37E+10	4.38E+10	4.502942	3.992602	5.200702	30.70466	28.81176	34.10303	1.177781	1.090116	1.299361
91	23.52701	21.44121	27.41412	1.205897	1.083218	1.349184	1.18E+11	9.06E+10	1.3E+11	15.35125	13.60159	17.4511	36.18856	34.4202	38.96319	2.41241	2.057721	2.578993
128	46.26889	40.09967	51.19878	1.502152	1.352151	1.620984	2.2E+11	2.07E+11	2.69E+11	73.71522	68.38275	90.77246	67.14048	63.6825	75.30186	4.909115	4.333318	5.432951
181	45.36107	42.09764	49.33962	2.839857	2.69967	2.983894	7.18E+09	4.83E+09	9.95E+09	441.6258	389.0578	531.4432	153.3297	143.6919	162.4497	8.640953	8.086018	9.247516
256	68.00507	64.94743	74.94676	6.098995	5.698537	6.311802	4.94E+10	3.91E+10	5.54E+10	2677.965	2415.902	3038.626	387.0977	362.8298	403.5409	13.01684	12.20911	13.83082
362	207.4376	199.9286	217.0254	8.746091	8.391711	9.955575	1.82E+11	1.73E+11	1.97E+11	16250.53	15432.23	19066.84	567.2369	528.1509	600.6509	30.38272	29.19079	31.08359
512	211.556	194.0456	237.6802	16.7062	15.94458	17.14526	5.73E+11	5.27E+11	6.02E+11	101102.5	96389.21	106681.2	1484.693	1434.04	1510.316	63.16382	60.18961	64.92216
724	461.7891	444.7364	479.0969	27.86395	26.79277	28.42531	7.04E+11	6.78E+11	7.47E+11	875325.7	801539	919859	2982.744	2860.528	3031.92	163.6042	152.1894	171.2687
1024	738.4498	710.3289	762.9169	50.62178	48.83384	51.70819	8.66E+11	8.06E+11	9.31E+11	7774867	7302214	8295749	5504.962	5360.925	5580.81	496.721	457.8528	505.6953
1448	1693.83	1647.108	1782.932	85.45449	84.11552	86.98864	1.23E+12	1.14E+12	1.35E+12	71403244	66412695	75052332	7456.907	7328.981	7533.725	1065.673	1052.346	1077.921

Table B.35: SA Descriptive Statistics: Rosenbrock to Sum Squares Functions (10,000 Evaluation Tuning)

Dims.	Rosenbrock						Schwefel						Sphere						Sum of Different Powers						Sum Squares					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.014832	0.007769	0.030501	0.002933	0.000619	0.006199	0.064103	0.023208	0.167752	0.005549	0.003426	0.012528	0.000117	4.28E-05	0.000241	1.22E-05	6.46E-06	2.99E-05	2.07E-05	8.33E-06	3.23E-05	1.53E-07	5.54E-08	3.87E-07	0.020438	0.004683	0.267858	9.69E-05	3.73E-05	0.000181
3	0.222428	0.131207	0.339153	0.036625	0.020635	0.077254	1.21292	0.519716	2.198259	0.094044	0.059195	0.193236	0.586438	0.200996	1.085099	5.71E-05	3.78E-05	0.000144	0.00054	0.000313	0.001111	3.51E-06	1.46E-06	7.51E-06	0.00451	0.002523	0.009901	4.93E-06	2.97E-06	1.21E-05
4	2.302914	1.689719	2.663921	0.385756	0.272855	0.549327	7.426816	3.730951	12.03422	0.334536	0.187054	0.504724	0.779729	0.322781	1.701002	0.000136	8.61E-05	0.0002	0.007301	0.001952	0.024256	1.58E-06	9.35E-07	4.2E-06	0.011123	0.007563	0.016971	0.000269	0.000104	0.000473
6	2.316435	0.819913	5.378943	1.733443	0.300189	4.321038	21.22405	16.08432	28.38379	1.418986	1.02443	2.090864	0.064943	0.051743	0.084761	0.000527	0.000301	0.000773	0.033755	0.022915	0.057265	6.19E-06	2.92E-06	1.16E-05	1.693905	1.213469	2.389566	0.000662	0.000386	0.000859
8	2.75577	1.796852	5.01113	1.765375	0.552987	4.414793	53.06271	33.56468	68.86802	3.108571	2.195121	5.218235	14.91966	10.87956	18.99649	0.00044	0.000294	0.000726	0.05304	0.036435	0.06782	3.07E-05	2.04E-05	5.17E-05	19.965	16.30065	25.35815	0.001765	0.001048	0.002068
11	7.911913	4.26272	10.93386	3.899175	1.039111	8.045576	180.9892	144.0753	214.2629	8.530047	6.445396	52.58982	0.504027	0.484648	0.559639	0.006359	0.003892	0.006802	0.051041	0.026591	0.080813	4.11E-05	1.21E-05	9.38E-05	34.60704	23.88977	40.14617	0.012326	0.008273	0.016328
16	66.27958	16.75152	87.07973	23.05665	6.600146	73.36977	296.1782	266.7515	493.8705	16.6874	10.92774	130.9911	3.162998	2.571344	3.879679	0.007083	0.006391	0.008199	0.111904	0.091146	0.189164	2.6E-05	1.12E-05	8.73E-05	31.03829	19.93127	35.66681	0.023612	0.018912	0.026195
23	81.67637	54.05142	90.03605	12.67173	3.543422	73.59036	1086.66	889.5276	1187.075	147.5022	137.7244	263.29	9.005609	7.120689	10.62962	0.033324	0.029757	0.038496	0.141456	0.107484	0.184336	9.54E-05	3.92E-05	0.000296	6.122212	5.261145	8.424276	0.034465	0.028149	0.042215
32	110.0979	98.15046	147.762	78.28994	15.27061	85.20484	1188.756	1040.363	1463.289	272.9665	234.2518	424.9824	15.68224	12.97895	18.67543	0.006732	0.005616	0.008525	0.075558	0.046141	0.088475	0.000923	0.000436	0.001358	5.945359	5.519983	6.400577	0.088381	0.080683	0.109068
45	192.6543	172.8006	208.1891	102.4922	88.61068	127.539	3316.996	3112.84	3722.903	563.106	414.9933	788.9489	11.74268	10.99748	12.96119	0.072075	0.056859	0.077942	0.134051	0.098408	0.17582	0.000813	0.000617	0.00558	10.40343	9.529268	11.72058	0.167728	0.140143	0.193511
64	586.8297	516.0693	659.6971	168.2489	157.9614	179.3028	5396.596	4958.153	5747.989	1241.132	1144.716	1365.843	11.8556	11.02671	13.11281	0.100335	0.085423	0.109836	0.158071	0.106034	0.205003	0.001279	0.000391	0.002904	12.44145	11.37601	12.94755	0.471729	0.406055	0.510939
91	820.4725	757.7326	894.1491	156.6637	120.9243	188.9644	6518.628	6303.619	6865.904	1780.428	1532.29	1934.08	24.68125	21.69368	27.38223	0.251963	0.221141	0.2647	0.173524	0.10419	0.194793	0.005924	0.002309	0.007943	27.04768	24.62145	28.86685	1.884172	1.697465	2.038583
128	719.6085	657.6946	792.1644	271.0161	244.2509	286.6494	12787.56	11704.22	13232.61	3122.595	2803.119	3394.516	39.97819	38.76979	43.21599	0.72142	0.631035	0.776198	0.231348	0.16102	0.345345	0.00582	0.002352	0.011914	43.5836	40.76652	46.97948	3.502259	3.329974	4.002944
181	1758.259	1580.093	1973.91	492.9893	435.4926	589.9218	10000.17	9495	10649.06	8769.455	8255.846	9458.973	58.32066	55.52927	62.27923	1.063389	1.015739	1.164437	0.23224	0.181714	0.364536	0.011423	0.00738	0.019482	70.84453	66.25211	73.81648	9.648159	8.832653	10.25273
256	6133.881	5734.134	6832.868	1200.156	1078.004	1276.851	16164.86	15935.29	16805.49	13636.65	13161.64	13944.86	93.98989	88.71136	95.66997	2.774528	2.701881	2.979555	0.278343	0.234733	0.380766	0.023501	0.016294	0.033007	387.1683	350.3038	458.1419	37.88191	35.00568	39.53999
362	24868.24	22235.54	29948.57	2714.302	2510.903	2820.846	26287.59	25521.88	27460.12	18685.77	18286.77	19589.22	78.72096	77.44291	82.31551	5.217734	4.997491	5.397827	0.438601	0.364093	0.481281	0.033934	0.02071	0.039244	2824.358	2407.842	3106.439	100.556	95.70039	103.5511
512	117532	101604.1	150171.1	5005.319	4789.806	5251.294	45385.89	44475.69	46572.06	27277.73	26553.66	28295.41	116.0618	109.6036	118.6745	12.31013	11.85958	12.64962	0.465569	0.400873	0.54281	0.035112	0.02636	0.046197	20371.49	18143.56	24826.48	563.992	537.4434	605.073
724	714776.1	615691.8	1148173	15135.85	13850.42	15593.15	83088.35	80436.9	83656.84	40931.41	40058.87	42923.48	297.6061	286.6219	305.9553	30.91098	28.13232	32.05312	0.464986	0.394687	0.659832	0.043377	0.025189	0.052929	103923.8	94370.34	113456.8	3661.468	3376.605	3818.852
1024	3498044	3042095	4374403	46096.72	42352.07	49309.85	145135.5	143608.9	148343.8	60092.23	58845.61	61002.77	437.389	416.3114	443.1638	51.79503	50.39548	53.16996	0.766716	0.569668	0.924258	0.061155	0.049436	0.06793	477745.9	449281.3	526729.1	18729.19	17477.04	19723.44
1448	19283745	17677248	20976375	173203.7	167482.4	180050.4	245406.6	242748.6	248242.1	97517.08	95747.91	99263.42	871.6069	834.7866	930.1425	101.8344	98.80442	105.9286	0.870396	0.688844	1.492729	0.087349	0.059794	0.137443	1944039	1842093	2085626	108854.9	107418.4	110935.2

Table B.36: SA Descriptive Statistics: Rosenbrock to Sum Squares Functions (50,000 Evaluation Tuning)

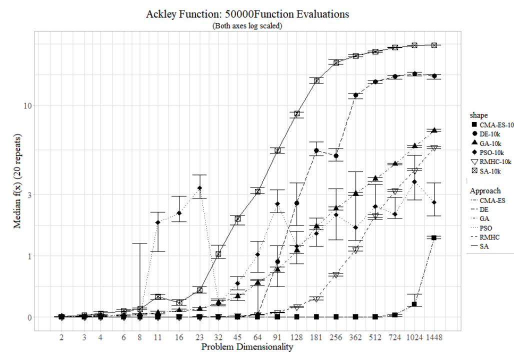
Dims.	Rosenbrock						Schwefel						Sphere						Sum of Different Powers						Sum Squares					
	10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions			10,000 Evaluations			50,000 Evalutions		
	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3	Median	Q1	Q3
2	0.015051	0.005735	0.040167	0.002843	0.000792	0.007966	0.137704	0.082313	0.389008	0.010177	0.006938	0.019966	11.00064	6.434143	13.24475	2.23E-07	1.24E-07	7.57E-07	9.97E-05	2.5E-05	0.00624	2.5E-07	1.2E-07	4.03E-07	10.6857	3.726748	15.18417	6.59E-07	2.84E-07	1.96E-06
3	0.542825	0.143738	0.716889	0.05706	0.031365	0.099434	0.719062	0.233237	1.026334	0.080802	0.05128	0.118345	9.804215	7.27746	22.35693	9.55E-05	6.23E-05	0.000164	0.000141	7.8E-05	0.000244	2.63E-06	9.3E-07	3.84E-06	41.15995	28.33943	56.44722	3.36E-05	1.94E-05	5.97E-05
4	1.417597	0.845649	2.251415	0.280118	0.209114	0.481058	2.569395	1.455399	3.247134	0.414405	0.249831	0.503785	0.297251	0.20967	0.800916	8.07E-05	5.14E-05	0.000117	0.05803	0.023538	0.104593	9.2E-07	4.32E-07	1.94E-06	25.24308	14.61679	45.00716	3.93E-05	1.85E-05	5.08E-05
6	9.489017	7.19601	12.06144	2.057051	0.979714	2.373189	18.33426	11.62653	21.60743	1.415081	0.926539	2.198041	19.50651	15.91113	24.09748	7.32E-05	3.51E-05	0.000179	0.008192	0.00402	0.01435	1.18E-05	6.09E-06	3.89E-05	147.159	126.6936	175.6639	1.99E-05	1.66E-05	2.71E-05
8	2.593246	1.153414	5.566575	2.254591	0.52516	4.928545	38.15142	30.50794	49.39824	2.559407	1.949046	3.517193	40.14727	25.91494	45.47283	0.000359	0.000266	0.000477	0.016256	0.008397	0.022891	2.4E-05	1.58E-05	5.11E-05	262.7122	172.6065	326.2246	3.34E-05	2.81E-05	4.12E-05
11	7.943329	3.016336	10.0055	3.489764	1.424441	7.283625	80.78586	58.96909	181.5218	3.839975	3.197001	40.17808	15.26078	11.7969	18.00595	7.21E-05	6.38E-05	0.000122	0.009679	0.007202	0.012726	0.000123	4.34E-05	0.00025	2.758238	2.209509	3.385244	0.000661	0.000524	0.000825
16	10.48523	8.216456	16.466	6.502074	3.808708	9.525983	416.5445	332.3695	543.2732	129.803	64.73452	146.4431	1.529858	1.237303	1.785235	0.007054	0.005317	0.009271	0.094783	0.076592	0.162368	0.000321	0.000127	0.000452	2.078693	1.821263	2.584372	0.005543	0.003581	0.007985
23	74.67562	40.63954	83.71851	16.59116	6.265036	71.08516	628.1623	514.0961	716.6062	142.1273	133.8561	218.7863	5.984387	4.409498	6.651231	0.003753	0.002783	0.00462	0.138865	0.076928	0.189941	0.000129	5.75E-05	0.000164	11.42351	7.356274	14.50467	0.030063	0.025939	0.038298
32	378.6383	321.6547	396.7853	35.59041	34.35554	51.43372	1265.345	1100.852	1460.024	273.9493	155.2029	432.7619	54.30855	48.33981	60.56905	0.011516	0.009985	0.014035	0.125009	0.055818	0.204685	0.000153	6.85E-05	0.001159	204.7775	151.6531	249.4869	0.04534	0.036056	0.054963
45	1225.472	904.7579	2288.29	46.63886	45.21312	106.0356	2350.206	2054.576	2543.388	486.7651	346.7033	680.5384	9.414668	8.505267	10.25209	0.077299	0.072408	0.082171	0.290824	0.215454	0.407688	0.005114	0.000515	0.022859	9.858828	9.024567	11.07949	0.218966	0.196343	0.269513
64	622809.3	569075.7	774105.4	67.36382	66.00793	135.3797	3494.143	3426.498	3899.808	983.2095	848.9836	1166.589	34.25092	30.39018	36.47849	0.111757	0.092611	0.12623	0.114262	0.083637	0.153103	0.004038	0.002215	0.008902	10133.36	7974.985	11624.09	0.262096	0.23553	0.305979
91	340193	288520	499398.5	111.8338	110.37	195.3205	6231.359	5851.322	6582.227	1772.232	1616.438	2002.81	65.55755	57.28001	71.4315	0.1766	0.167291	0.213885	0.115268	0.093574	0.154513	0.002718	0.001986	0.005065	53941.83	50107.27	57082.08	1.050893	0.888578	1.208947
128	699497.9	470999.8	1051439	158.9109	148.9724	216.8473	9769.332	9268.649	9965.199	2836.293	2588.692	3164.259	163.7145	153.3015	170.8731	0.403875	0.3417	0.470085	0.127104	0.109274	0.200175	0.008494	0.005103	0.011158	121390.1	112716.3	134862.1	1.495983	1.303928	1.756738
181	11109927	8539272	12349272	342.831	281.5833	409.1582	18680.38	18189.38	19816.01	4747.374	4393.275	4962.639	431.3651	408.1308	472.5028	0.592524	0.53872	0.643043	0.313203	0.223627	0.444614	0.016973	0.01284	0.021711	216454.9	208865.8	226927.7	1.867883	1.697808	2.039783
256	711441.2	539510.2	1005226	662.1536	579.3257	728.9633	31279.48	30705.51	32160.55	8190.228	7576.036	8756.945	871.4815	849.1392	899.8806	1.179204	1.102387	1.346364	0.26269	0.196888	0.338947	0.015427	0.009664	0.023742	438911.1	406412.5	457662.8	7.31583	6.889131	8.335883
362	12459080	11304878	13113301	1177.804	1037.416	1307.101	47311.72	46528.09	48503.3	13421.74	12951.21	13936.84	379.8645	356.7544	401.0035	3.016708	2.912851	3.22237	0.335348	0.269662	0.448804	0.021735	0.012569	0.037947	320199.7	308905.1	330201.3	17.20947	16.61332	18.44048
512	29450955	26165575	33223522	2091.488	1997.822	2305.507	80000.07	78232.59	82206.88	23577.51	22187.16	24266.54	1447.679	1397.506	1535.862	6.556301	6.204869	6.762484	0.412207	0.358442	0.590855	0.022882	0.014778	0.031406	1082220	1033621	1157776	62.31684	59.40851	65.51611
724	53683844	51721776	57919202	4612.953	4448.391	4806.399	118309.7	117354.7	120389.7	38695.55	37810.16	39862.48	1795.168	1739.445	1850.78	11.13769	10.62894	11.96346	0.600559	0.406297	0.796744	0.056699	0.035299	0.089559	2089033	2034349	2124437	325.2947	312.722	357.5271
1024	79568980	73366644	85448608	12132.43	11760.25	12716.31	144796.6	142597.6	148280.2	60692.19	59590.13	61647.22	2451.192	2403.929	2536.989	25.71451	25.16061	26.32008	0.742583	0.578792	0.9674	0.049952	0.026596	0.089754	4436951	4176558	4613041	2486.269	2357.992	2932.015
1448	1.18E+08	1.12E+08	1.24E+08	42971.25	41704.81	45842.17	248489.1	246919.6	249349.7	96879.23	94129.02	97813.31	4471.319	4438.536	4531.386	47.28566	43.81982	49.14607	0.995415	0.732842	1.27858	0.085777	0.05742	0.146576	8848410	8718769	9165161	19222.9	17815.21	22333.79



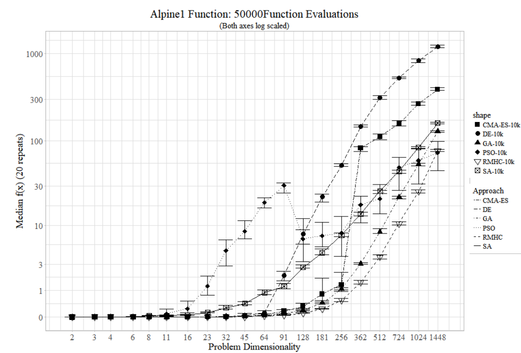
APPENDIX C: SMALL TO LARGE-SCALE METAHEURISTIC COMPARISON MATERIAL

C.1 METAHEURISTIC COMPARISON PLOTS

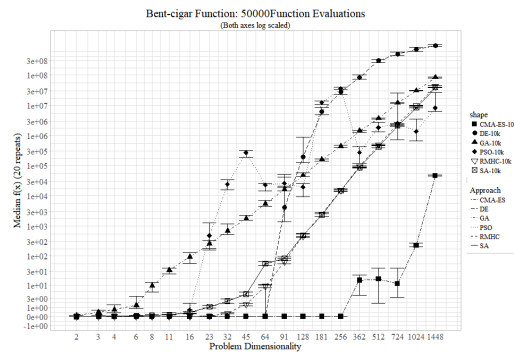
C.1.1 *Comparison of Approaches Tuned at the 10,000 Evaluation Budget (With Q1 & Q3 Error Bars)*



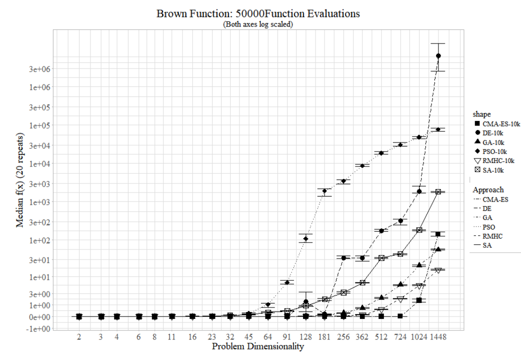
Ackley Function at 50,000 Evaluations



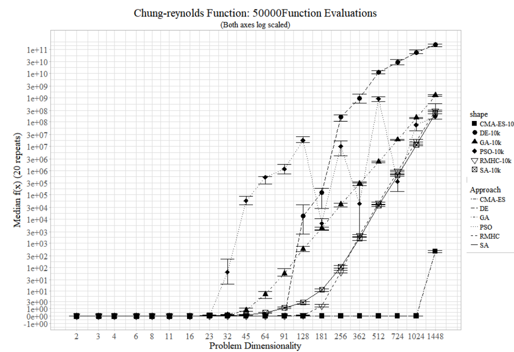
Alpine n.1 Function at 50,000 Evaluations



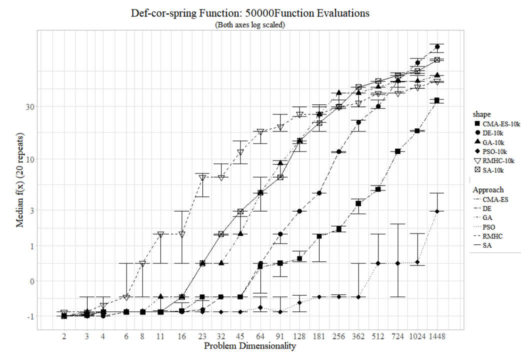
Bent Cigar Function at 50,000 Evaluations



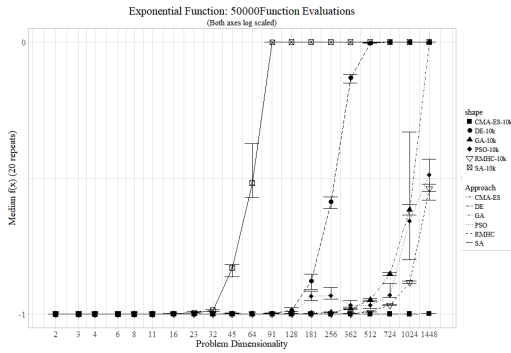
Brown Function at 50,000 Evaluations



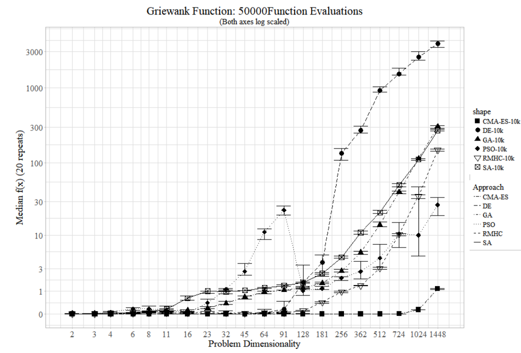
Chung-Reynolds Function at 50,000 Evaluations



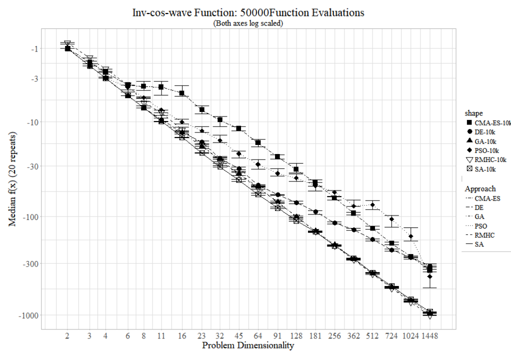
Deflected Corrugated Spring Function at 50,000 Evaluations



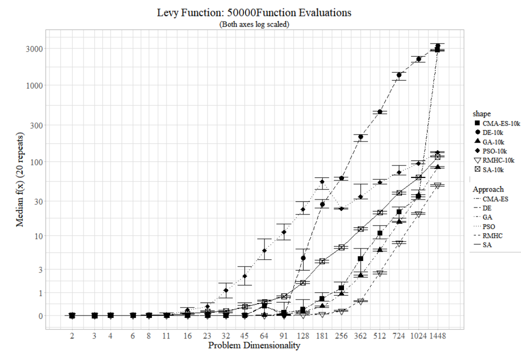
Exponential Function at 50,000 Evaluations



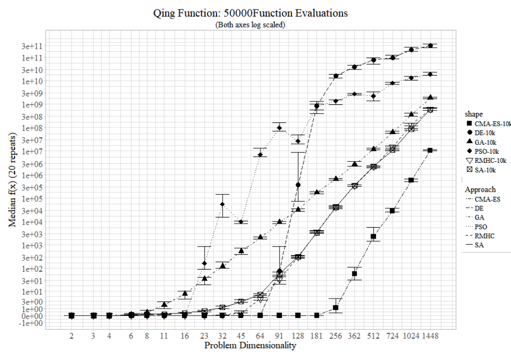
Griewank Function at 50,000 Evaluations



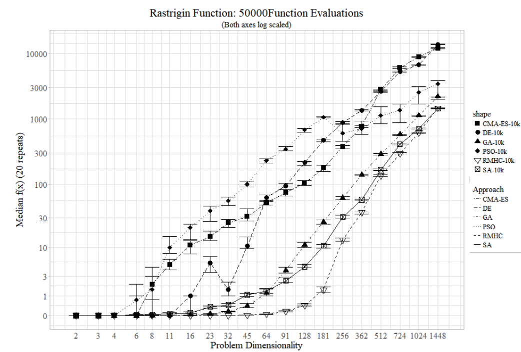
Inverted Cosine Wave Function at 50,000 Evaluations



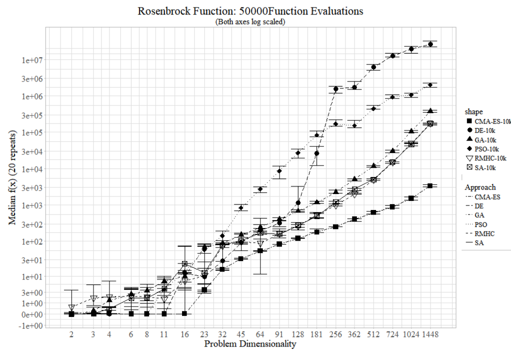
Levy Function at 50,000 Evaluations



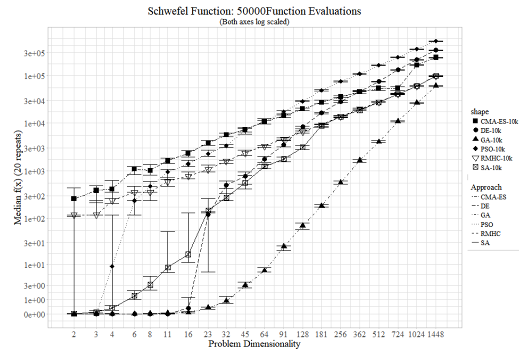
Qing Function at 50,000 Evaluations



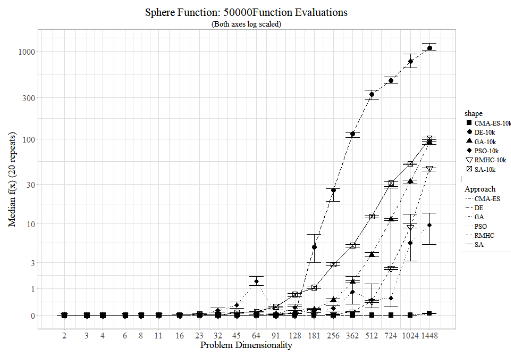
Rastrigin Function at 50,000 Evaluations



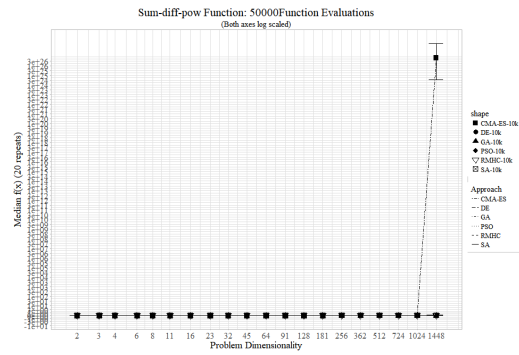
Rosenbrock Function at 50,000 Evaluations



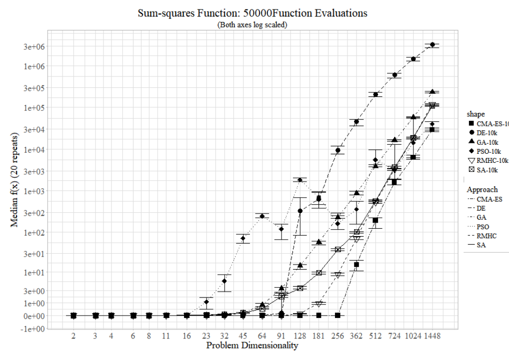
Schwefel Function at 50,000 Evaluations



Sphere Function at 50,000 Evaluations

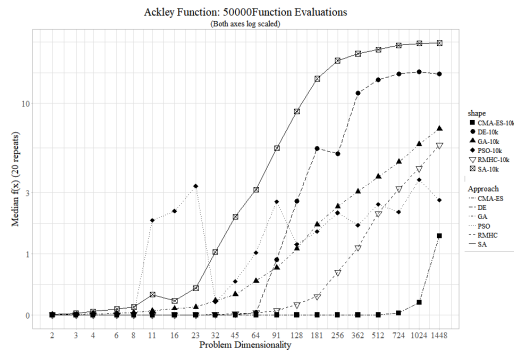


Sum of Different Powers Function at 50,000 Evaluations

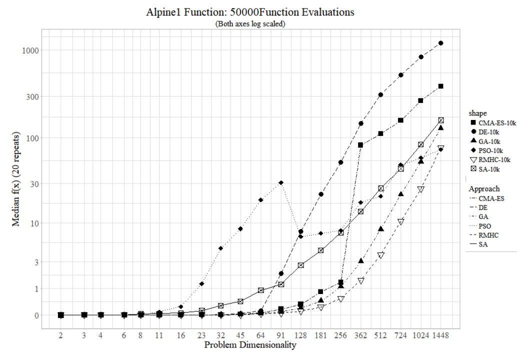


Sum Squares Function at 50,000 Evaluations

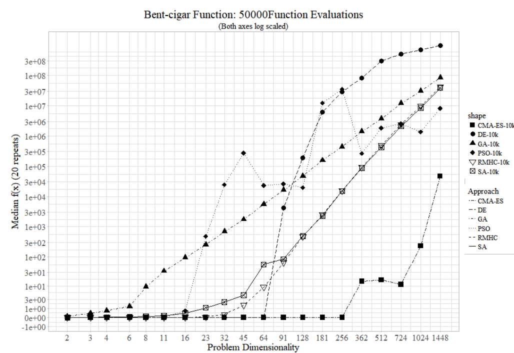
C.1.2 *Comparison of Approaches Tuned at the 10,000 Evaluation Budget (No Error Bars)*



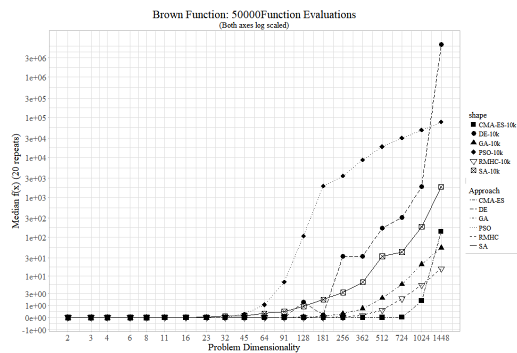
Ackley Function at 50,000 Evaluations



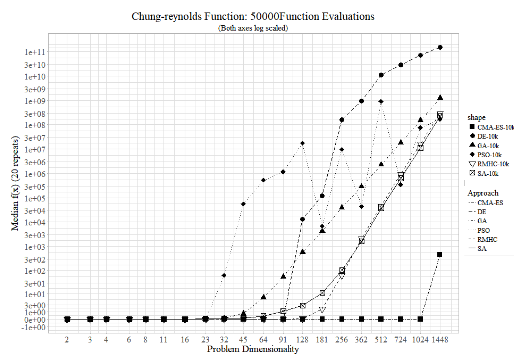
Alpine n.1 Function at 50,000 Evaluations



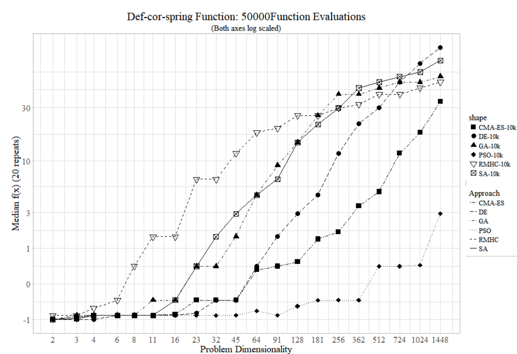
Bent Cigar Function at 50,000 Evaluations



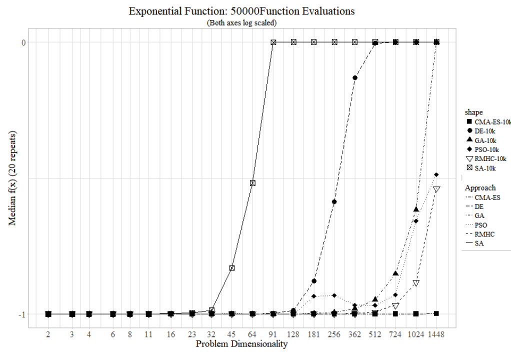
Brown Function at 50,000 Evaluations



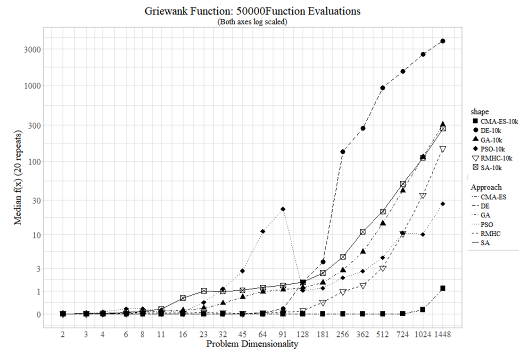
Chung-Reynolds Function at 50,000 Evaluations



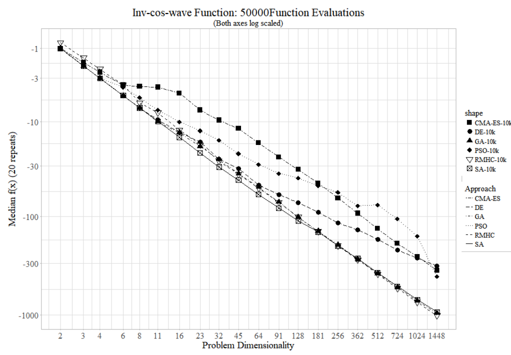
Deflected Corrugated Spring Function at 50,000 Evaluations



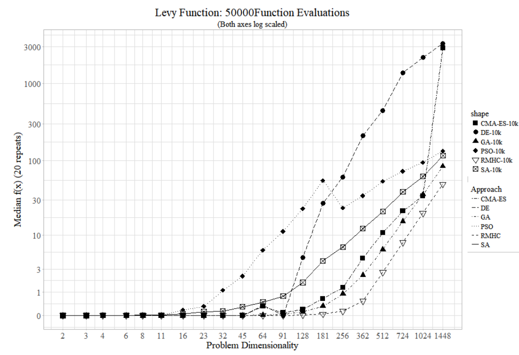
Exponential Function at 50,000 Evaluations



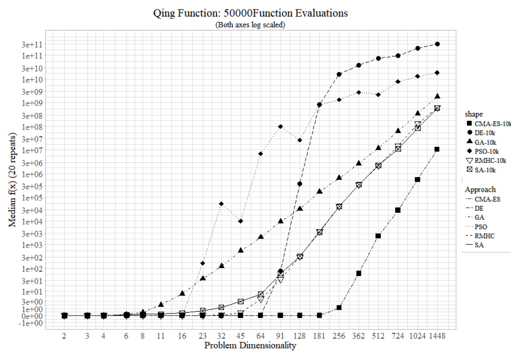
Griewank Function at 50,000 Evaluations



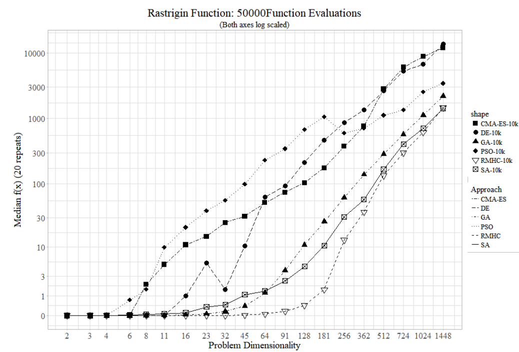
Inverted Cosine Wave Function at 50,000 Evaluations



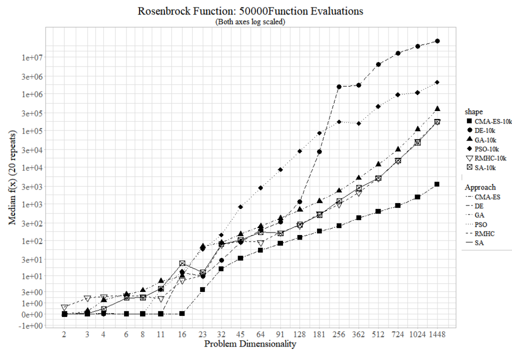
Levy Function at 50,000 Evaluations



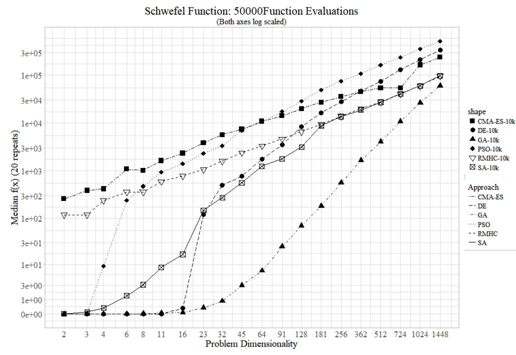
Qing Function at 50,000 Evaluations



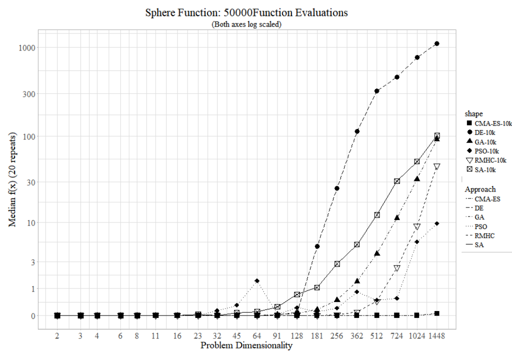
Rastrigin Function at 50,000 Evaluations



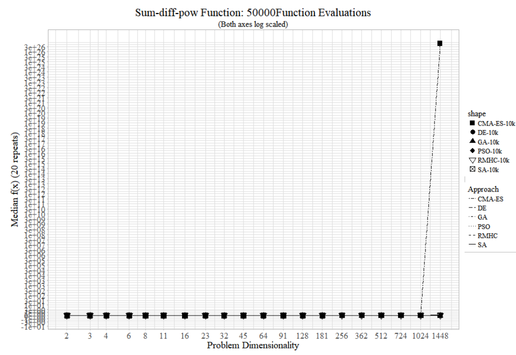
Rosenbrock Function at 50,000 Evaluations



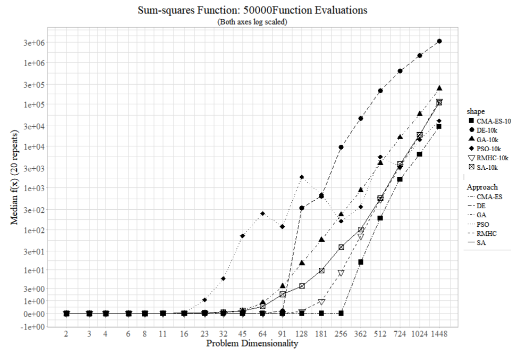
Schwefel Function at 50,000 Evaluations



Sphere Function at 50,000 Evaluations

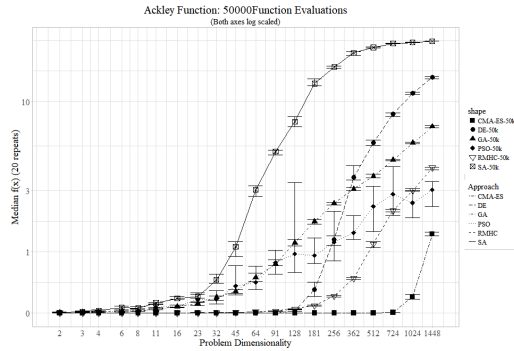


Sum of Different Powers Function at 50,000 Evaluations

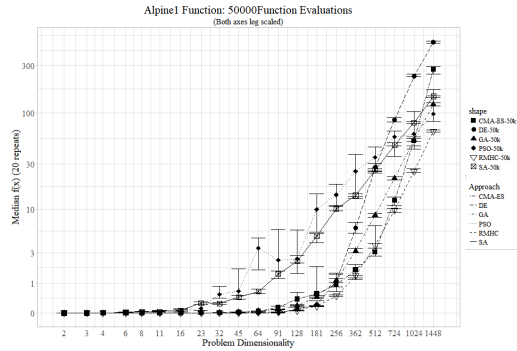


Sum Squares Function at 50,000 Evaluations

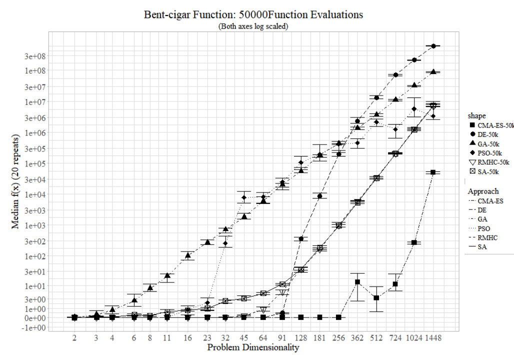
C.1.3 *Comparison of Approaches Tuned at the 50,000 Evaluation Budget (With Q1 & Q3 Error Bars)*



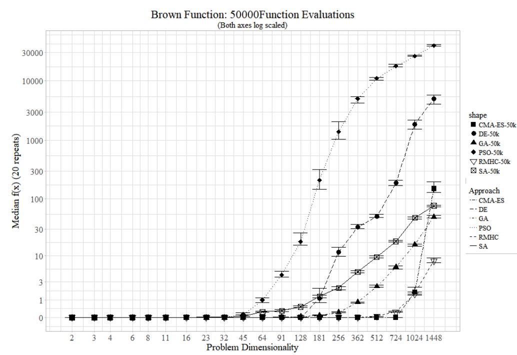
Ackley Function at 50,000 Evaluations



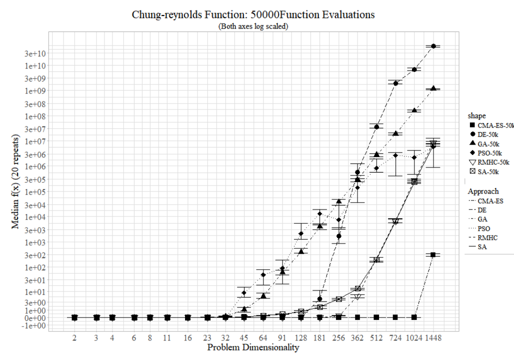
Alpine n.1 Function at 50,000 Evaluations



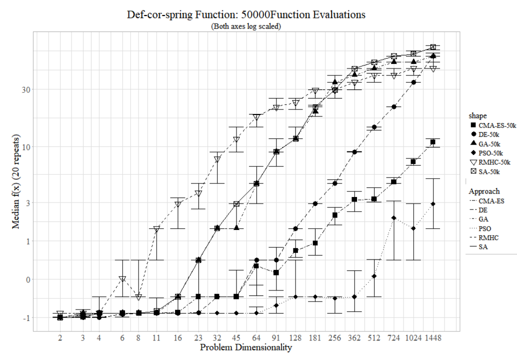
Bent Cigar Function at 50,000 Evaluations



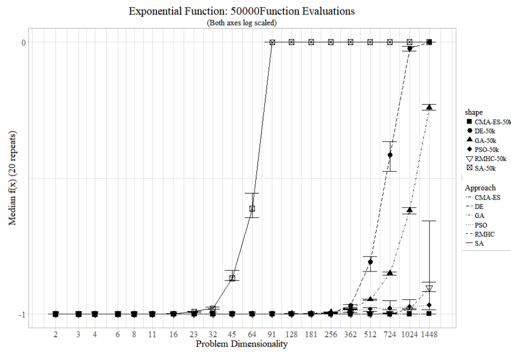
Brown Function at 50,000 Evaluations



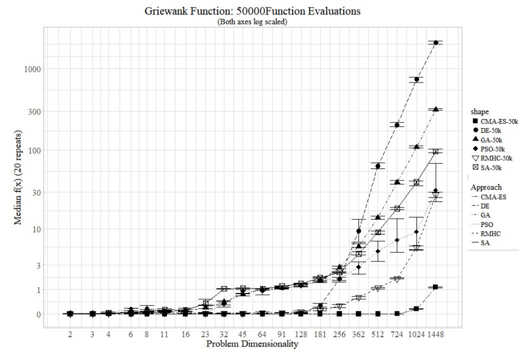
Chung-Reynolds Function at 50,000 Evaluations



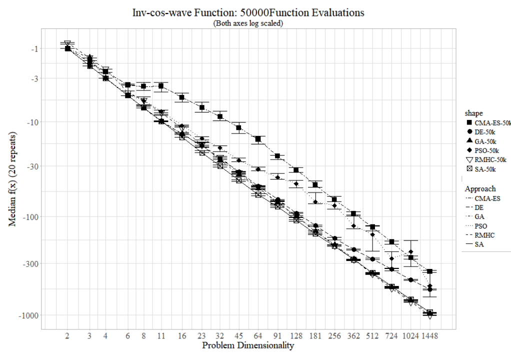
Deflected Corrugated Spring Function at 50,000 Evaluations



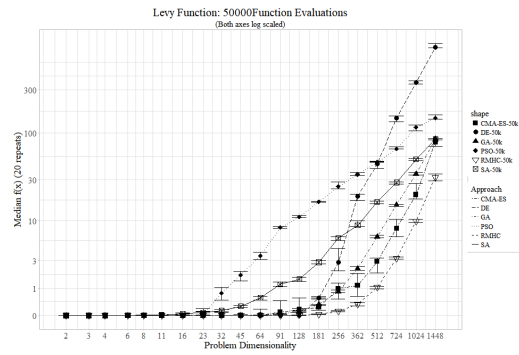
Exponential Function at 50,000 Evaluations



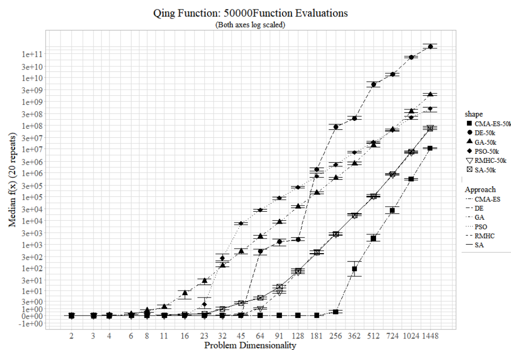
Griewank Function at 50,000 Evaluations



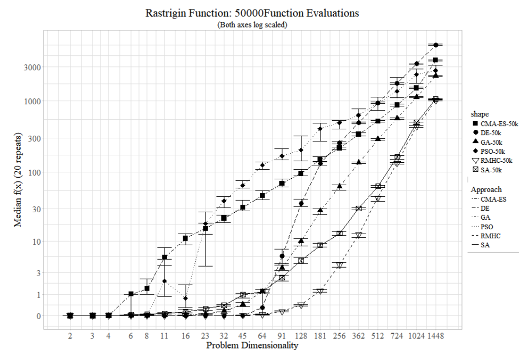
Inverted Cosine Wave Function at 50,000 Evaluations



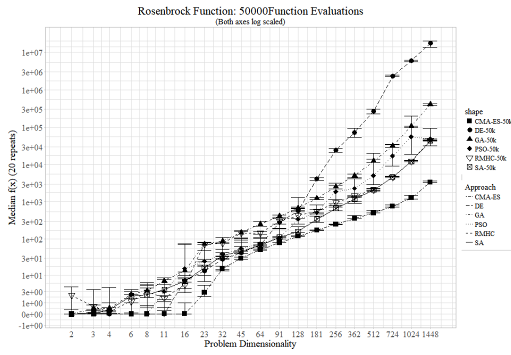
Levy Function at 50,000 Evaluations



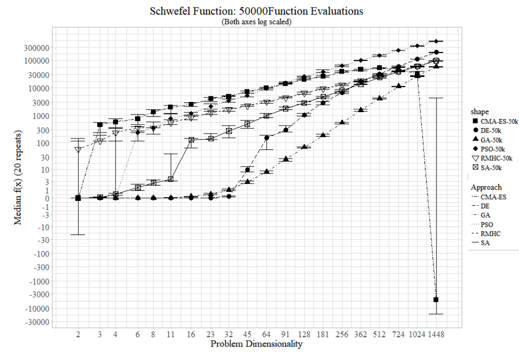
Qing Function at 50,000 Evaluations



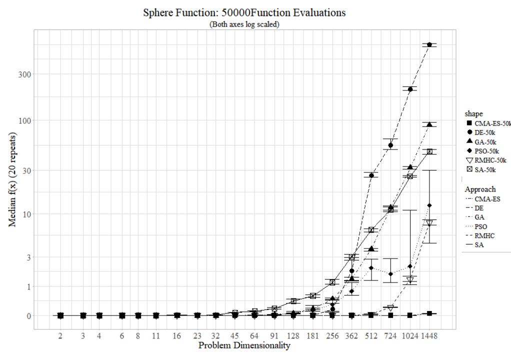
Rastrigin Function at 50,000 Evaluations



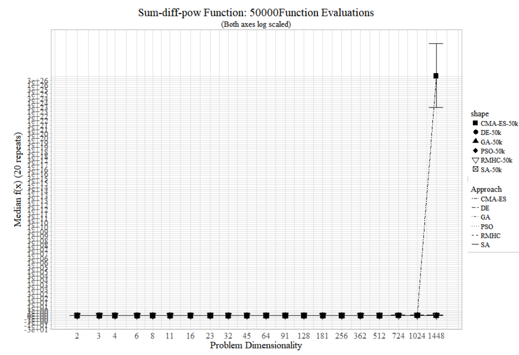
Rosenbrock Function at 50,000 Evaluations



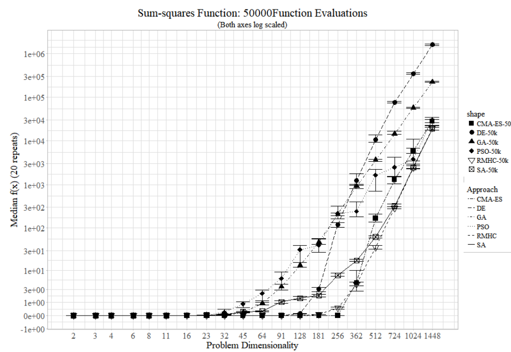
Schwefel Function at 50,000 Evaluations



Sphere Function at 50,000 Evaluations

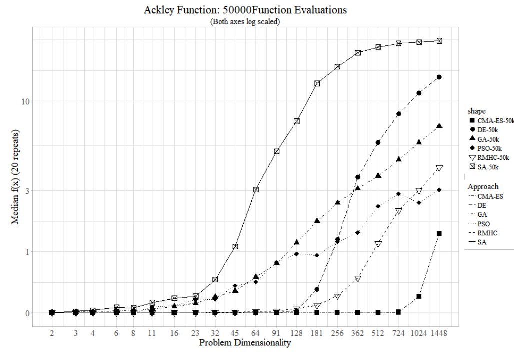


Sum of Different Powers Function at 50,000 Evaluations

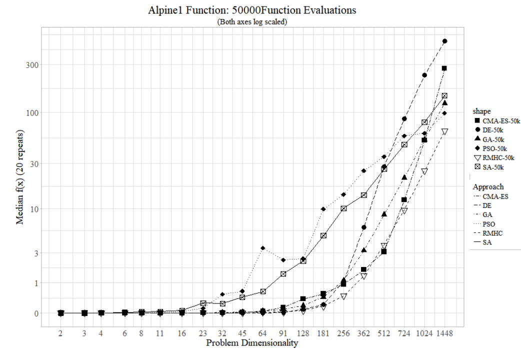


Sum Squares Function at 50,000 Evaluations

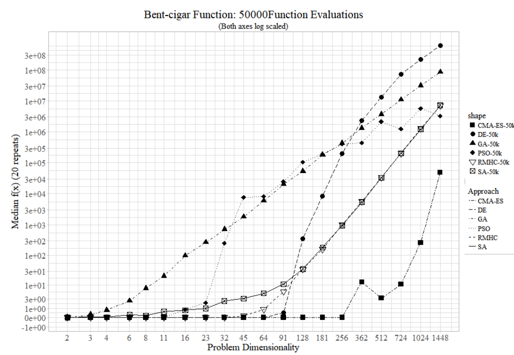
c.1.4 *Comparison of Approaches Tuned at the 50,000 Evaluation Budget (No Error Bars)*



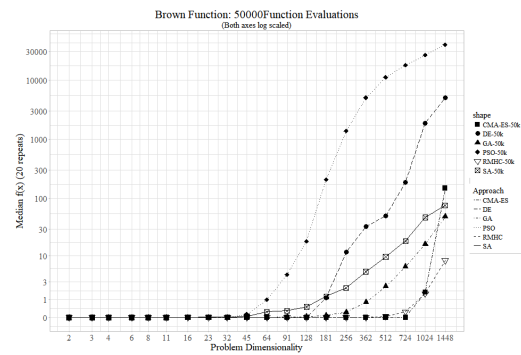
Ackley Function at 50,000 Evaluations



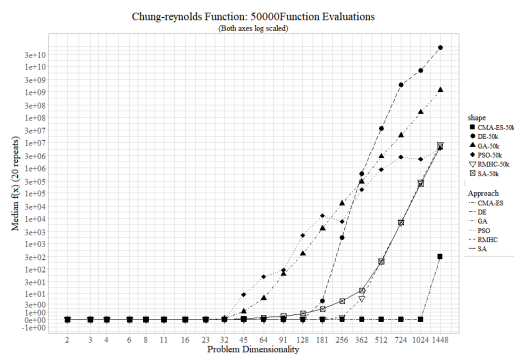
Alpine n.1 Function at 50,000 Evaluations



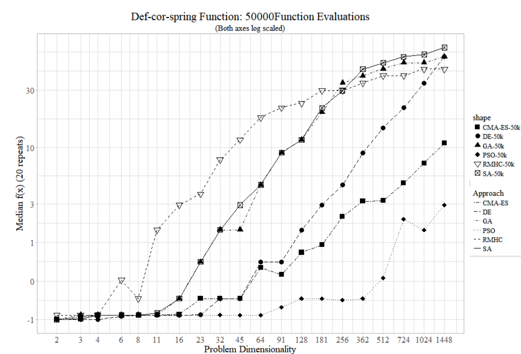
Bent Cigar Function at 50,000 Evaluations



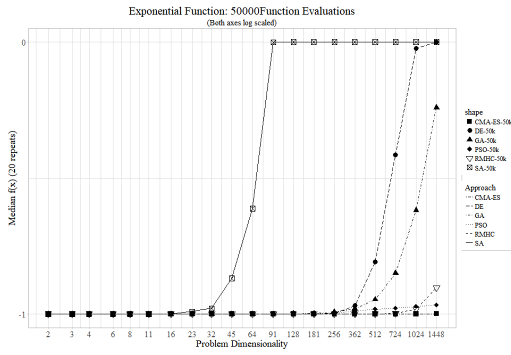
Brown Function at 50,000 Evaluations



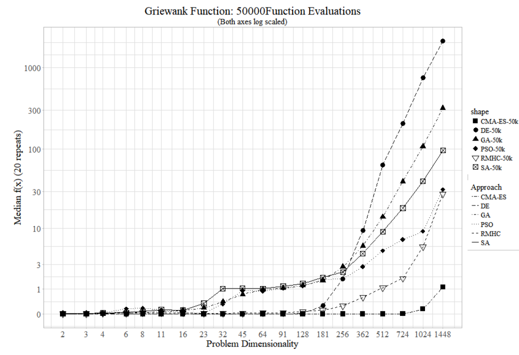
Chung-Reynolds Function at 50,000 Evaluations



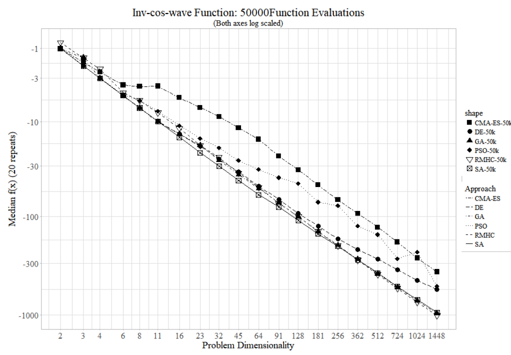
Deflected Corrugated Spring Function at 50,000 Evaluations



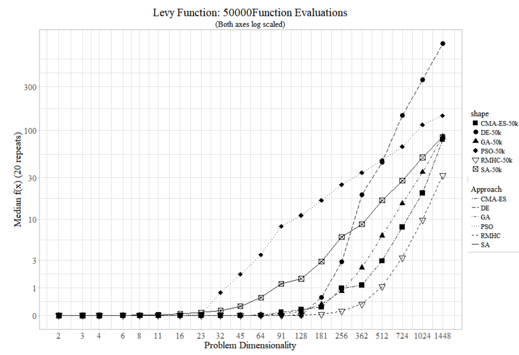
Exponential Function at 50,000 Evaluations



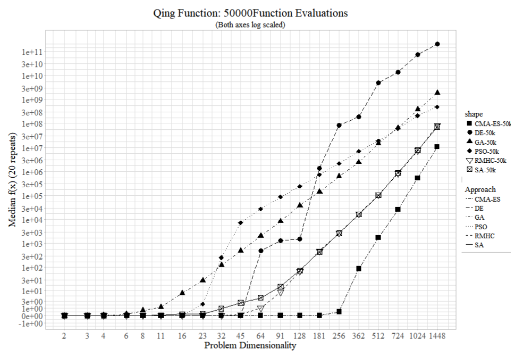
Griewank Function at 50,000 Evaluations



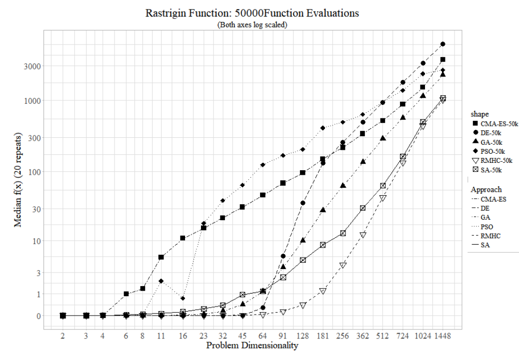
Inverted Cosine Wave Function at 50,000 Evaluations



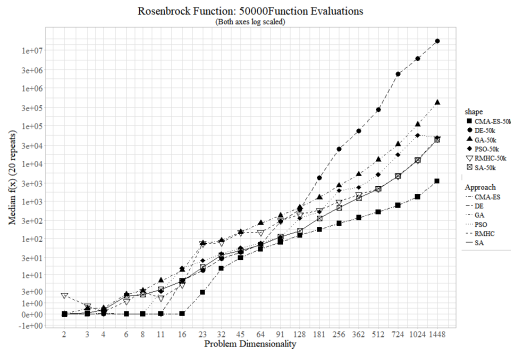
Levy Function at 50,000 Evaluations



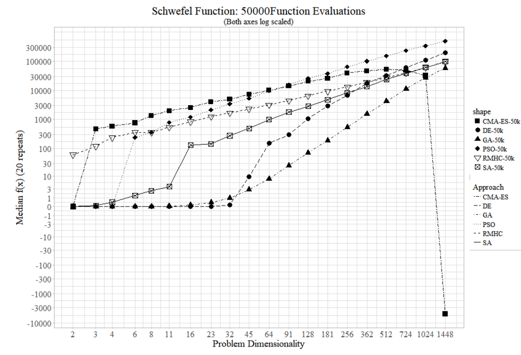
Qing Function at 50,000 Evaluations



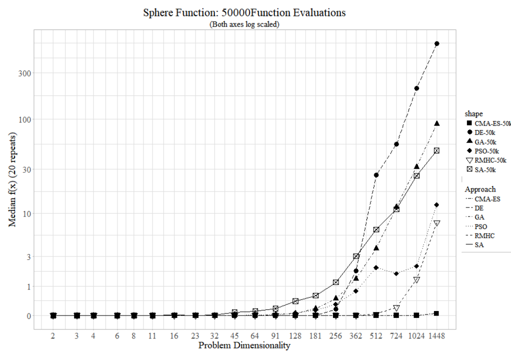
Rastrigin Function at 50,000 Evaluations



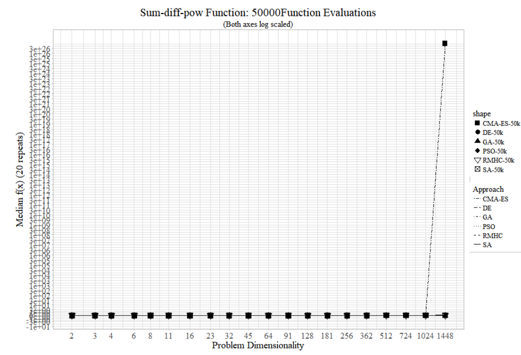
Rosenbrock Function at 50,000 Evaluations



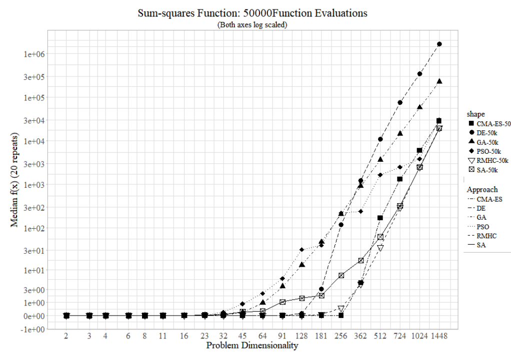
Schwefel Function at 50,000 Evaluations



Sphere Function at 50,000 Evaluations

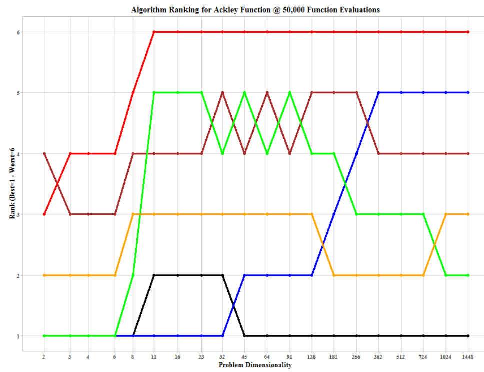


Sum of Different Powers Function at 50,000 Evaluations

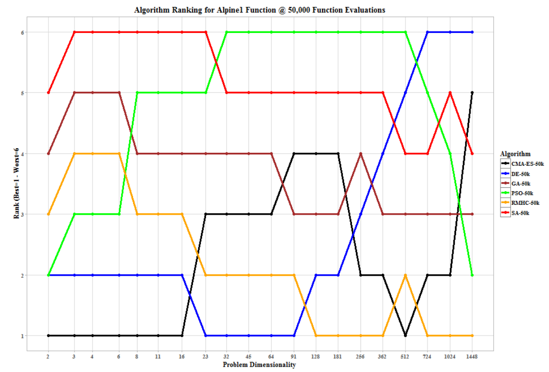


Sum Squares Function at 50,000 Evaluations

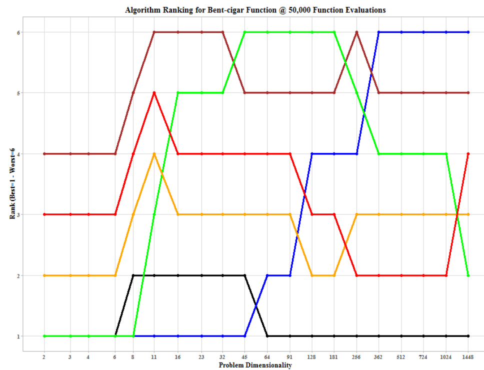
C.2.1 *Rank Ordering By Benchmark Function*



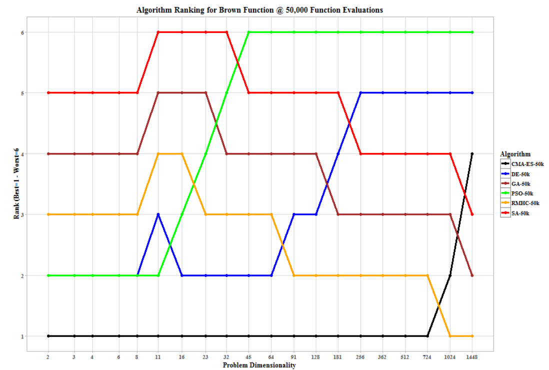
Rankings for Ackley Function



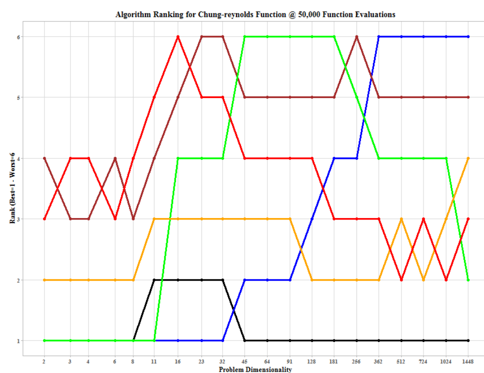
Rankings for Alpine no.1 Function



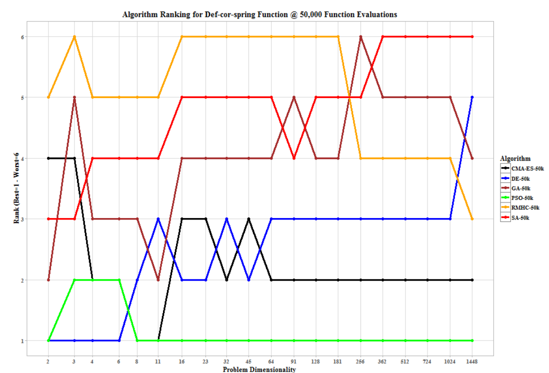
Rankings for Bent Cigar Function



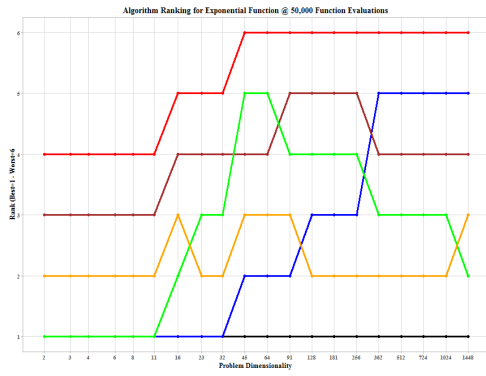
Rankings for Brown Function



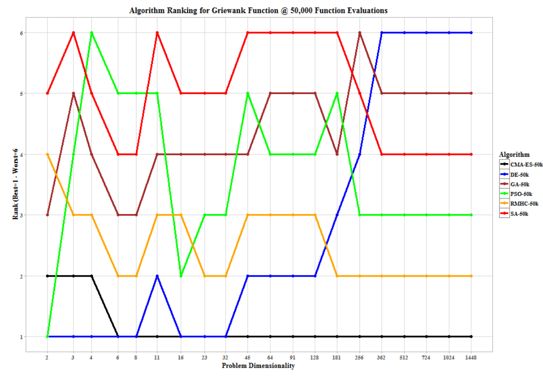
Rankings for Chung Reynolds Function



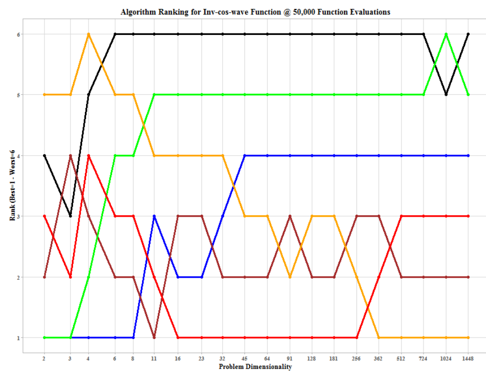
Rankings for Deflected Corrugated Spring Function



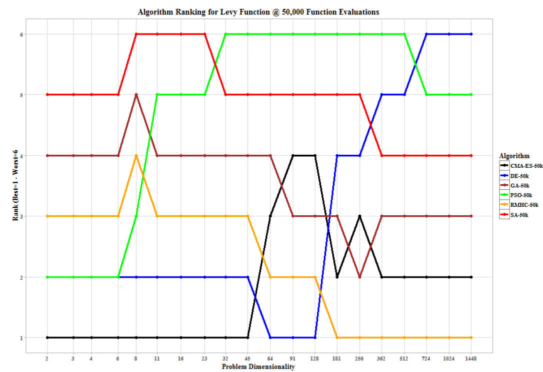
Rankings for Exponential Function



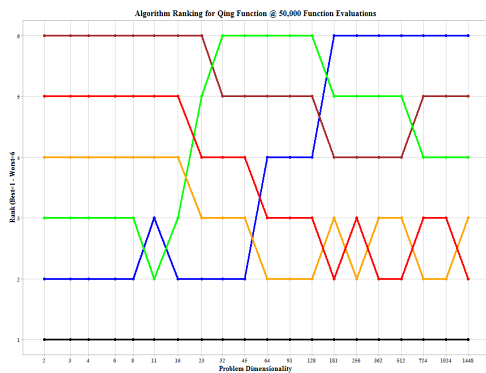
Rankings for Griewank Function



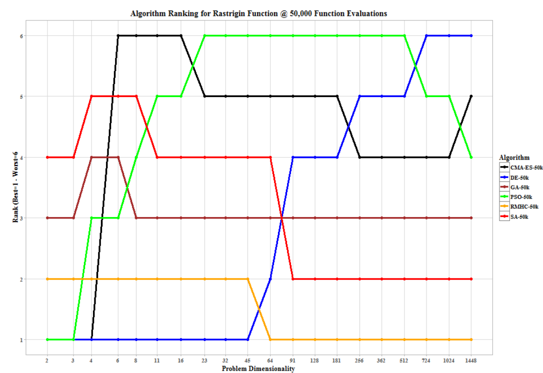
Rankings for Inverted Cosine Wave Function



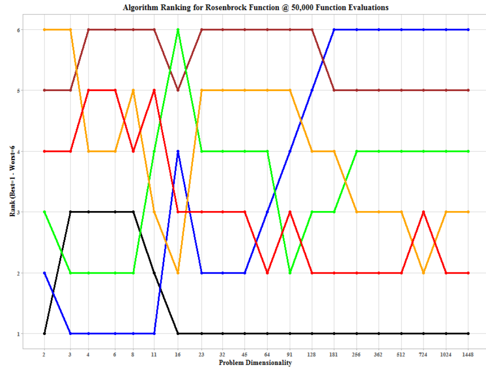
Rankings for Levy Function



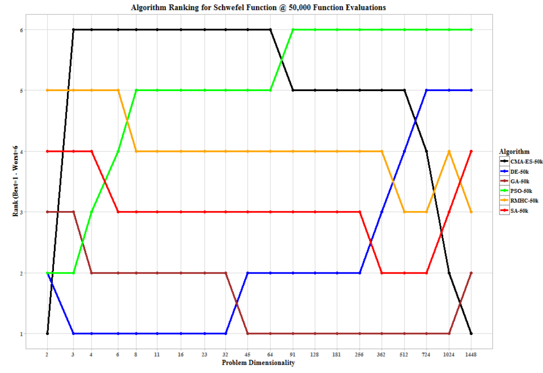
Rankings for Qing Function



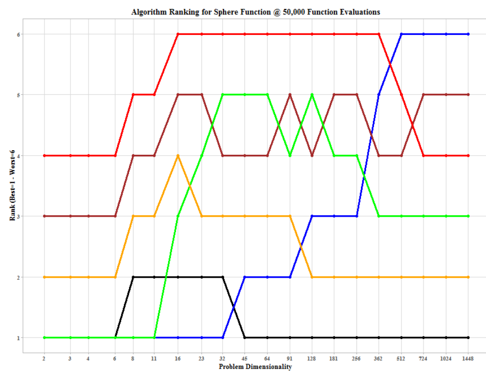
Rankings for Rastrigin Function



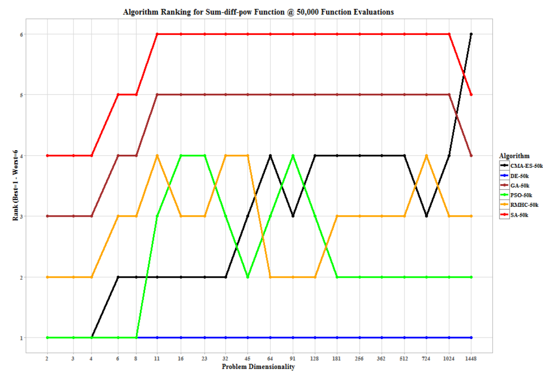
Rankings for Rosenbrock Function



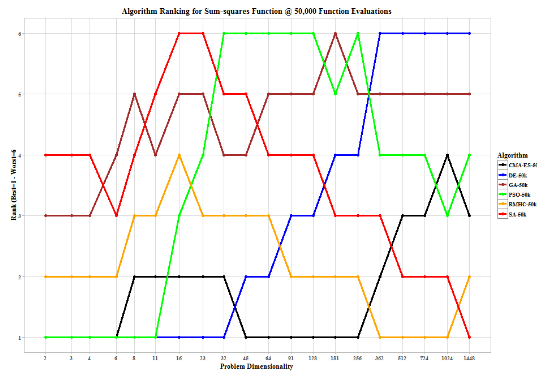
Rankings for Schwefel Function



Rankings for Sphere Function



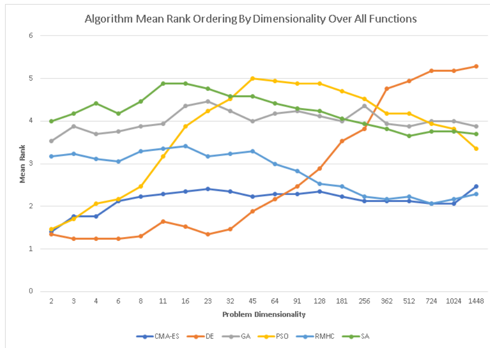
Rankings for Sum of Different Powers Function



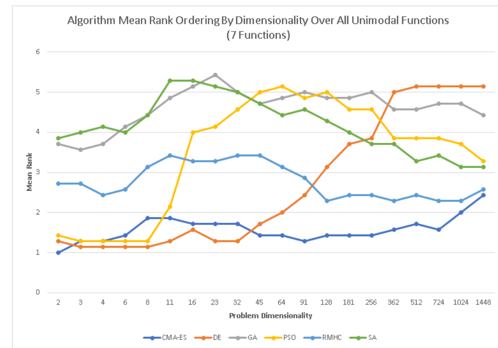
Rankings for Sum Squares Function

C.2.2 *Rank Ordering: Mean Aggregate and Pooled Mean Over Subsets of the Benchmark Suite*

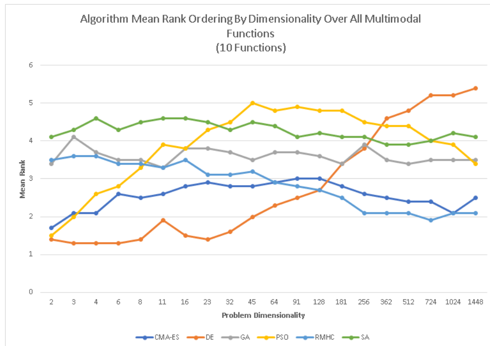
C.2.2.1 *Mean Aggregated Rank Ordering Over All Dimensionalities*



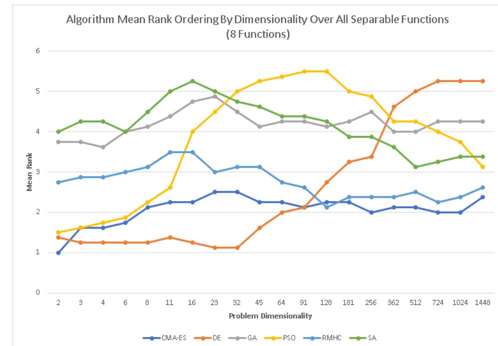
Mean Aggregate Rankings Over Entire Benchmark Suite



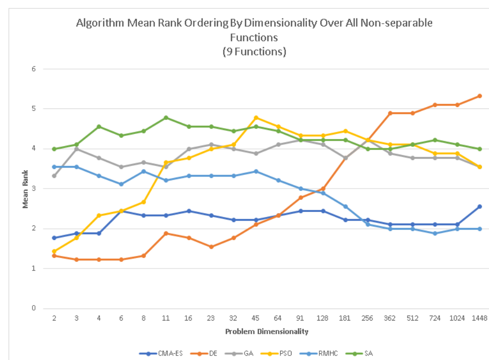
Mean Aggregate Rankings Over All Uni-modal Problems



Mean Aggregate Rankings Over All Multi-modal Problems

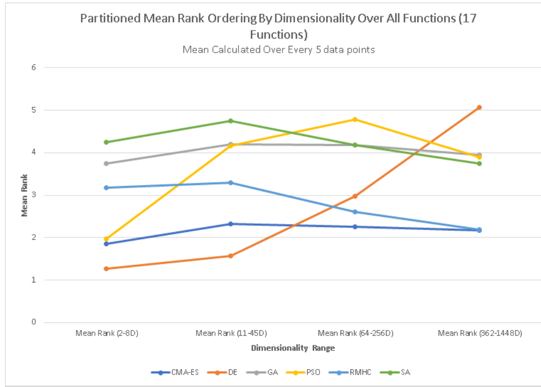


Mean Aggregate Rankings Over All Separable Problems

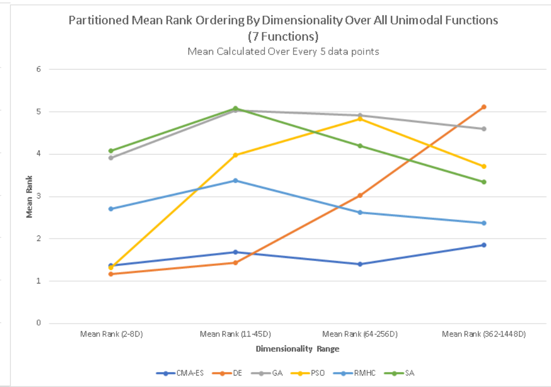


Mean Aggregate Rankings Over All Non-separable Problems

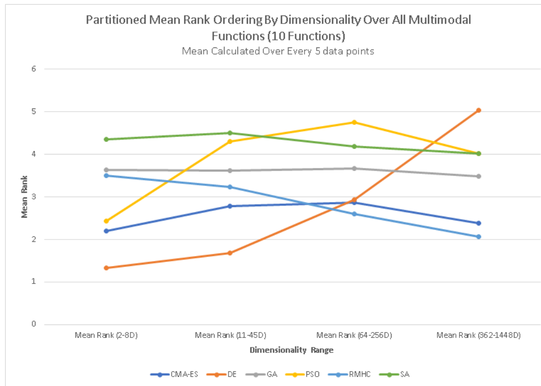
C.2.2.2 *Pooled Mean Rank Ordering Over Several Dimension Ranges*



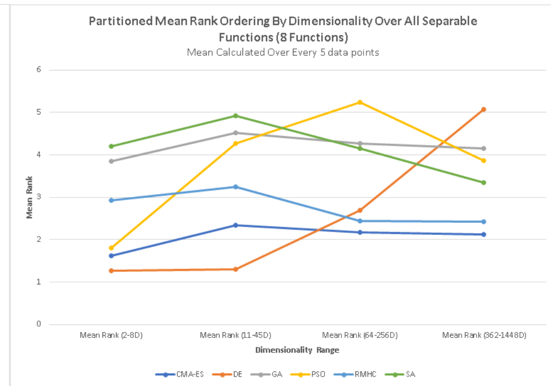
Pooled Mean Rankings Over Entire Benchmark Suite



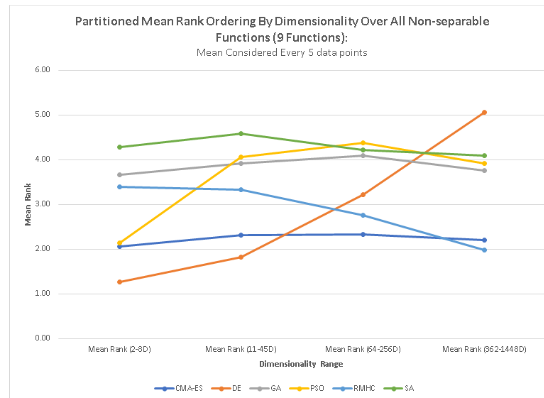
Pooled Mean Rankings Over All Uni-modal Problems



Pooled Mean Rankings Over All Multi-modal Problems



Pooled Mean Rankings Over All Separable Problems



Pooled Mean Rankings Over All Non-separable Problems

C.3 ALGORITHM RANKING TABLES

C.3.1 *Algorithm Rank Ordering By Benchmark Function*

Table C.1: Algorithm Rankings for Ackley Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean*
Sep-CMA-ES	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	*1.2*
DE	1	1	1	1	1	1	1	1	1	2	2	2	2	3	4	5	5	5	5	5	*2.45*
GA	4	3	3	3	4	4	4	4	5	4	5	4	5	5	5	4	4	4	4	4	*4.1*
PSO	1	1	1	1	2	5	5	5	4	5	4	5	4	4	3	3	3	3	2	2	*3.15*
RMHC	2	2	2	2	3	3	3	3	3	3	3	3	3	2	2	2	2	2	3	3	*2.55*
SA	3	4	4	4	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	*5.5*

Table C.2: Algorithm Rankings for Alpine n.1 Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean*
Sep-CMA-ES	1	1	1	1	1	1	1	3	3	3	3	4	4	4	2	2	1	2	2	5	*2.25*
DE	2	2	2	2	2	2	2	1	1	1	1	1	2	2	3	4	5	6	6	6	*2.65*
GA	4	5	5	5	4	4	4	4	4	4	4	3	3	3	4	3	3	3	3	3	*3.75*
PSO	2	3	3	3	5	5	5	5	6	6	6	6	6	6	6	6	6	5	4	2	*4.8*
RMHC	3	4	4	4	3	3	3	2	2	2	2	2	1	1	1	1	2	1	1	1	*2.15*
SA	5	6	6	6	6	6	6	6	5	5	5	5	5	5	5	5	4	4	5	4	*5.2*

Table C.3: Algorithm Rankings for Bent Cigar Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean*
Sep-CMA-ES	1	1	1	1	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	*1.3*
DE	1	1	1	1	1	1	1	1	1	1	2	2	4	4	4	6	6	6	6	6	*2.8*
GA	4	4	4	4	5	6	6	6	6	5	5	5	5	5	6	5	5	5	5	5	*5.05*
PSO	1	1	1	1	1	3	5	5	5	6	6	6	6	6	5	4	4	4	4	2	*3.8*
RMHC	2	2	2	2	3	4	3	3	3	3	3	3	2	2	3	3	3	3	3	3	*2.75*
SA	3	3	3	3	4	5	4	4	4	4	4	4	3	3	2	2	2	2	2	4	*3.25*

Table C.4: Algorithm Rankings for Brown Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean*
Sep-CMA-ES	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	4	*1.2*
DE	2	2	2	2	2	3	2	2	2	2	2	3	3	4	5	5	5	5	5	5	*3.15*
GA	4	4	4	4	4	5	5	5	4	4	4	4	4	3	3	3	3	3	3	2	*3.75*
PSO	2	2	2	2	2	2	3	4	5	6	6	6	6	6	6	6	6	6	6	6	*4.5*
RMHC	3	3	3	3	3	4	4	3	3	3	3	2	2	2	2	2	2	2	1	1	*2.55*
SA	5	5	5	5	5	6	6	6	6	5	5	5	5	5	4	4	4	4	4	3	*4.85*

Table C.5: Algorithm Rankings for Chung-Reynolds Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean*
Sep-CMA-ES	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	*1.2*
DE	1	1	1	1	1	1	1	1	1	2	2	2	3	4	4	6	6	6	6	6	*2.8*
GA	4	3	3	4	3	4	5	6	6	5	5	5	5	5	6	5	5	5	5	5	*4.7*
PSO	1	1	1	1	1	1	4	4	4	6	6	6	6	6	5	4	4	4	4	2	*3.55*
RMHC	2	2	2	2	2	3	3	3	3	3	3	3	2	2	2	2	3	2	3	4	*2.55*
SA	3	4	4	3	4	5	6	5	5	4	4	4	4	3	3	3	2	3	2	3	*3.7*

Table C.6: Algorithm Rankings for Deflected Corrugated Spring Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean*
Sep-CMA-ES	4	4	2	2	1	1	3	3	2	3	2	2	2	2	2	2	2	2	2	2	*2.25*
DE	1	1	1	1	2	3	2	2	3	2	3	3	3	3	3	3	3	3	3	5	*2.5*
GA	2	5	3	3	3	2	4	4	4	4	4	5	4	4	6	5	5	5	5	4	*4.05*
PSO	1	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	*1.15*
RMHC	5	6	5	5	5	5	6	6	6	6	6	6	6	6	4	4	4	4	4	3	*5.1*
SA	3	3	4	4	4	4	5	5	5	5	5	4	5	5	5	6	6	6	6	6	*4.8*

Table C.7: Algorithm Rankings for Exponential Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	***Mean***
Sep-CMA-ES	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	*1*
DE	1	1	1	1	1	1	1	1	1	2	2	2	3	3	3	5	5	5	5	5	*2.45*
GA	3	3	3	3	3	3	4	4	4	4	4	5	5	5	5	4	4	4	4	4	*3.9*
PSO	1	1	1	1	1	1	2	3	3	5	5	4	4	4	4	3	3	3	3	2	*2.7*
RMHC	2	2	2	2	2	2	3	2	2	3	3	3	2	2	2	2	2	2	2	3	*2.25*
SA	4	4	4	4	4	4	5	5	5	6	6	6	6	6	6	6	6	6	6	6	*5.25*

Table C.8: Algorithm Rankings for Griewank Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	***Mean***
Sep-CMA-ES	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	*1.15*
DE	1	1	1	1	1	2	1	1	1	2	2	2	2	3	4	6	6	6	6	6	*2.75*
GA	3	5	4	3	3	4	4	4	4	4	5	5	5	4	6	5	5	5	5	5	*4.4*
PSO	1	4	6	5	5	5	2	3	3	5	4	4	4	5	3	3	3	3	3	3	*3.7*
RMHC	4	3	3	2	2	3	3	2	2	3	3	3	3	2	2	2	2	2	2	2	*2.5*
SA	5	6	5	4	4	6	5	5	5	6	6	6	6	6	5	4	4	4	4	4	*5*

Table C.9: Algorithm Rankings for Inverted Cosine Wave Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean*
Sep-CMA-ES	4	3	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	5	6	*5.65*
DE	1	1	1	1	1	3	2	2	3	4	4	4	4	4	4	4	4	4	4	4	*2.95*
GA	2	4	3	2	2	1	3	3	2	2	2	3	2	2	3	3	2	2	2	2	*2.35*
PSO	1	1	2	4	4	5	5	5	5	5	5	5	5	5	5	5	5	5	6	5	*4.4*
RMHC	5	5	6	5	5	4	4	4	4	3	3	2	3	3	2	1	1	1	1	1	*3.15*
SA	3	2	4	3	3	2	1	1	1	1	1	1	1	1	1	2	3	3	3	3	*2*

Table C.10: Algorithm Rankings for Levy Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean*
Sep-CMA-ES	1	1	1	1	1	1	1	1	1	1	3	4	4	2	3	2	2	2	2	2	*1.8*
DE	2	2	2	2	2	2	2	2	2	2	1	1	1	4	4	5	5	6	6	6	*2.95*
GA	4	4	4	4	5	4	4	4	4	4	4	3	3	3	2	3	3	3	3	3	*3.55*
PSO	2	2	2	2	3	5	5	5	6	6	6	6	6	6	6	6	6	5	5	5	*4.75*
RMHC	3	3	3	3	4	3	3	3	3	3	2	2	2	1	1	1	1	1	1	1	*2.2*
SA	5	5	5	5	6	6	6	6	5	5	5	5	5	5	5	4	4	4	4	4	*4.95*

Table C.11: Algorithm Rankings for Qing Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	***Mean***
Sep-CMA-ES	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	*1*
DE	2	2	2	2	2	3	2	2	2	2	4	4	4	6	6	6	6	6	6	6	*3.75*
GA	6	6	6	6	6	6	6	6	5	5	5	5	5	4	4	4	4	5	5	5	*5.2*
PSO	3	3	3	3	3	2	3	5	6	6	6	6	6	5	5	5	5	4	4	4	*4.35*
RMHC	4	4	4	4	4	4	4	3	3	3	2	2	2	3	2	3	3	2	2	3	*3.05*
SA	5	5	5	5	5	5	5	4	4	4	3	3	3	2	3	2	2	3	3	2	*3.65*

Table C.12: Algorithm Rankings for Rastrigin Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	***Mean***
Sep-CMA-ES	1	1	1	6	6	6	6	5	5	5	5	5	5	5	4	4	4	4	4	5	*4.35*
DE	1	1	1	1	1	1	1	1	1	1	2	4	4	4	5	5	5	6	6	6	*2.85*
GA	3	3	4	4	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	*3.1*
PSO	1	1	3	3	4	5	5	6	6	6	6	6	6	6	6	6	6	5	5	4	*4.8*
RMHC	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	*1.5*
SA	4	4	5	5	5	4	4	4	4	4	4	2	2	2	2	2	2	2	2	2	*3.25*

Table C.13: Algorithm Rankings for Rosenbrock Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	***Mean***
Sep-CMA-ES	1	3	3	3	3	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	*1.45*
DE	2	1	1	1	1	1	4	2	2	2	3	4	5	6	6	6	6	6	6	6	*3.55*
GA	5	5	6	6	6	6	5	6	6	6	6	6	6	5	5	5	5	5	5	5	*5.5*
PSO	3	2	2	2	2	4	6	4	4	4	4	2	3	3	4	4	4	4	4	4	*3.45*
RMHC	6	6	4	4	5	3	2	5	5	5	5	5	4	4	3	3	3	2	3	3	*4*
SA	4	4	5	5	4	5	3	3	3	3	2	3	2	2	2	2	2	3	2	2	*3.05*

Table C.14: Algorithm Rankings for Schwefel Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	***Mean***
Sep-CMA-ES	1	6	6	6	6	6	6	6	6	6	6	5	5	5	5	5	5	4	2	1	*4.9*
DE	2	1	1	1	1	1	1	1	1	2	2	2	2	2	2	3	4	5	5	5	*2.2*
GA	3	3	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	2	*1.6*
PSO	2	2	3	4	5	5	5	5	5	5	5	6	6	6	6	6	6	6	6	6	*5*
RMHC	5	5	5	5	4	4	4	4	4	4	4	4	4	4	4	4	3	3	4	3	*4.05*
SA	4	4	4	3	3	3	3	3	3	3	3	3	3	3	3	2	2	2	3	4	*3.05*

Table C.15: Algorithm Rankings for Sphere Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean*
Sep-CMA-ES	1	1	1	1	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	*1.25*
DE	1	1	1	1	1	1	1	1	1	2	2	2	3	3	3	5	6	6	6	6	*2.65*
GA	3	3	3	3	4	4	5	5	4	4	4	5	4	5	5	4	4	5	5	5	*4.2*
PSO	1	1	1	1	1	1	3	4	5	5	5	4	5	4	4	3	3	3	3	3	*3*
RMHC	2	2	2	2	3	3	4	3	3	3	3	3	2	2	2	2	2	2	2	2	*2.45*
SA	4	4	4	4	5	5	6	6	6	6	6	6	6	6	6	6	5	4	4	4	*5.15*

Table C.16: Algorithm Rankings for Sum of Different Powers Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean*
Sep-CMA-ES	1	1	1	2	2	2	2	2	2	3	4	3	4	4	4	4	4	3	4	6	*2.9*
DE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	*1*
GA	3	3	3	4	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	4	*4.55*
PSO	1	1	1	1	1	3	4	4	3	2	3	4	3	2	2	2	2	2	2	2	*2.25*
RMHC	2	2	2	3	3	4	3	3	4	4	2	2	2	3	3	3	3	4	3	3	*2.9*
SA	4	4	4	5	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	5	*5.55*

Table C.17: Algorithm Rankings for Sum Squares Function over 2 to 1448 Dimensions (1=Best, 5=Worst)

Alg. / Dim.	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean*
Sep-CMA-ES	1	1	1	1	2	2	2	2	2	1	1	1	1	1	1	2	3	3	4	3	*1.75*
DE	1	1	1	1	1	1	1	1	1	2	2	3	3	4	4	6	6	6	6	6	*2.85*
GA	3	3	3	4	5	4	5	5	4	4	5	5	5	6	5	5	5	5	5	5	*4.55*
PSO	1	1	1	1	1	1	3	4	6	6	6	6	6	5	6	4	4	4	3	4	*3.65*
RMHC	2	2	2	2	3	3	4	3	3	3	3	2	2	2	2	1	1	1	1	2	*2.2*
SA	4	4	4	3	4	5	6	6	5	5	4	4	4	3	3	3	2	2	2	1	*3.7*

C.3.2 *Mean Aggregated Algorithm Rank Ordering By All Problems and Problems Classified by Problem Feature Investigated*

Table C.18: Mean Aggregated Algorithm Rankings Over All Benchmark Functions

Alg. / Dim	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean Rank*
CMA-ES	1.41	1.76	1.76	2.12	2.24	2.29	2.35	2.41	2.35	2.24	2.29	***2.29***	***2.35***	***2.24***	***2.12***	***2.12***	***2.12***	***2.06***	***2.06***	2.47	***2.15***
DE	***1.35***	***1.24***	***1.24***	***1.24***	***1.29***	***1.65***	***1.53***	***1.35***	***1.47***	***1.88***	***2.18***	2.47	2.88	3.53	3.82	4.76	4.94	5.18	5.18	5.29	2.72
GA	3.53	3.88	3.71	3.76	3.88	3.94	4.35	4.47	4.24	4.00	4.18	4.24	4.12	4.00	4.35	3.94	3.88	4.00	4.00	3.88	4.02
PSO	1.47	1.71	2.06	2.18	2.47	3.18	3.88	4.24	4.53	5.00	4.94	4.88	4.88	4.71	4.53	4.18	4.18	3.94	3.82	3.35	3.71
RMHC	3.18	3.24	3.12	3.06	3.29	3.35	3.41	3.18	3.24	3.29	3.00	2.82	2.53	2.47	2.24	2.18	2.24	***2.06***	2.18	***2.29***	2.82
SA	4.00	4.18	4.41	4.18	4.47	4.88	4.88	4.76	4.59	4.59	4.41	4.29	4.24	4.06	3.94	3.82	3.65	3.76	3.76	3.71	4.23

Table C.19: Mean Aggregated Algorithm Rankings Over All Uni-modal Functions

Alg. / Dim	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean Rank*
CMA-ES	***1.00***	1.29	1.29	1.43	1.86	1.86	1.71	1.71	1.71	***1.43***	***1.43***	***1.29***	***1.43***	***1.43***	***1.43***	***1.57***	***1.71***	***1.57***	***2.00***	***2.43***	***1.58***
DE	1.29	***1.14***	***1.14***	***1.14***	***1.14***	***1.29***	***1.57***	***1.29***	***1.29***	1.71	2.00	2.43	3.14	3.71	3.86	5.00	5.14	5.14	5.14	5.14	2.69
GA	3.71	3.57	3.71	4.14	4.43	4.86	5.14	5.43	5.00	4.71	4.86	5.00	4.86	4.86	5.00	4.57	4.57	4.71	4.71	4.43	4.61
PSO	1.43	1.29	1.29	1.29	1.29	2.14	4.00	4.14	4.57	5.00	5.14	4.86	5.00	4.57	4.57	3.86	3.86	3.86	3.71	3.29	3.46
RMHC	2.71	2.71	2.43	2.57	3.14	3.43	3.29	3.29	3.43	3.43	3.14	2.86	2.29	2.43	2.43	2.29	2.43	2.29	2.29	2.57	2.77
SA	3.86	4.00	4.14	4.00	4.43	5.29	5.29	5.14	5.00	4.71	4.43	4.57	4.29	4.00	3.71	3.71	3.29	3.43	3.14	3.14	4.18

Table C.20: Mean Aggregated Algorithm Rankings Over All Multi-modal Functions

Alg. / Dim	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean Rank*
CMA-ES	1.70	2.10	2.10	2.60	2.50	2.60	2.80	2.90	2.80	2.80	2.90	3.00	3.00	2.80	2.60	2.50	2.40	2.40	***2.10***	2.50	***2.56***
DE	***1.40***	***1.30***	***1.30***	***1.30***	***1.40***	***1.90***	***1.50***	***1.40***	***1.60***	***2.00***	***2.30***	***2.50***	***2.70***	3.40	3.80	4.60	4.80	5.20	5.20	5.40	2.75
GA	3.40	4.10	3.70	3.50	3.50	3.30	3.80	3.80	3.70	3.50	3.70	3.70	3.60	3.40	3.90	3.50	3.40	3.50	3.50	3.50	3.60
PSO	1.50	2.00	2.60	2.80	3.30	3.90	3.80	4.30	4.50	5.00	4.80	4.90	4.80	4.80	4.50	4.40	4.40	4.00	3.90	3.40	3.88
RMHC	3.50	3.60	3.60	3.40	3.40	3.30	3.50	3.10	3.10	3.20	2.90	2.80	***2.70***	***2.50***	***2.10***	***2.10***	***2.10***	***1.90***	***2.10***	***2.10***	2.85
SA	4.10	4.30	4.60	4.30	4.50	4.60	4.60	4.50	4.30	4.50	4.40	4.10	4.20	4.10	4.10	3.90	3.90	4.00	4.20	4.10	4.27

Table C.21: Mean Aggregated Algorithm Rankings Over All Separable Functions

Alg. / Dim	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean Rank*
CMA-ES	***1.00***	1.63	1.63	1.75	2.13	2.25	2.25	2.50	2.50	2.25	2.25	***2.13***	2.25	***2.25***	***2.00***	***2.13***	***2.13***	***2.00***	***2.00***	***2.38***	***2.07***
DE	1.38	***1.25***	***1.25***	***1.25***	***1.25***	***1.38***	***1.25***	***1.13***	***1.13***	***1.63***	***2.00***	***2.13***	2.75	3.25	3.38	4.63	5.00	5.25	5.25	5.25	2.59
GA	3.75	3.75	3.63	4.00	4.13	4.38	4.75	4.88	4.50	4.13	4.25	4.25	4.13	4.25	4.50	4.00	4.00	4.25	4.25	4.25	4.20
PSO	1.50	1.63	1.75	1.88	2.25	2.63	4.00	4.50	5.00	5.25	5.38	5.50	5.50	5.00	4.88	4.25	4.25	4.00	3.75	3.13	3.80
RMHC	2.75	2.88	2.88	3.00	3.13	3.50	3.50	3.00	3.13	3.13	2.75	2.63	***2.13***	2.38	2.38	2.38	2.50	2.25	2.38	2.63	2.76
SA	4.00	4.25	4.25	4.00	4.50	5.00	5.25	5.00	4.75	4.63	4.38	4.38	4.25	3.88	3.88	3.63	3.13	3.25	3.38	3.38	4.16

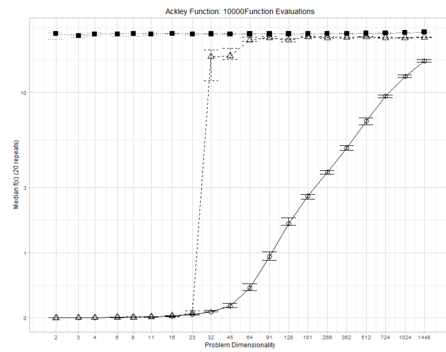
Table C.22: Mean Aggregated Algorithm Rankings Over All Non-separable Functions

Alg. / Dim	**2**	**3**	**4**	**6**	**8**	**11**	**16**	**23**	**32**	**45**	**64**	**91**	**128**	**181**	**256**	**362**	**512**	**724**	**1024**	**1448**	*Mean Rank*
CMA-ES	1.78	1.89	1.89	2.44	2.33	2.33	2.44	2.33	2.22	2.22	***2.33***	***2.44***	***2.44***	***2.22***	2.22	2.11	2.11	2.11	2.11	2.56	***2.23***
DE	***1.33***	***1.22***	***1.22***	***1.22***	***1.33***	***1.89***	***1.78***	***1.56***	***1.78***	***2.11***	***2.33***	2.78	3.00	3.78	4.22	4.89	4.89	5.11	5.11	5.33	2.84
GA	3.33	4.00	3.78	3.56	3.67	3.56	4.00	4.11	4.00	3.89	4.11	4.22	4.11	3.78	4.22	3.89	3.78	3.78	3.78	3.56	3.86
PSO	1.44	1.78	2.33	2.44	2.67	3.67	3.78	4.00	4.11	4.78	4.56	4.33	4.33	4.44	4.22	4.11	4.11	3.89	3.89	3.56	3.62
RMHC	3.56	3.56	3.33	3.11	3.44	3.22	3.33	3.33	3.33	3.44	3.22	3.00	2.89	2.56	***2.11***	***2.00***	***2.00***	***1.89***	***2.00***	***2.00***	2.87
SA	4.00	4.11	4.56	4.33	4.44	4.78	4.56	4.56	4.44	4.56	4.44	4.22	4.22	4.22	4.00	4.00	4.11	4.22	4.11	4.00	4.29

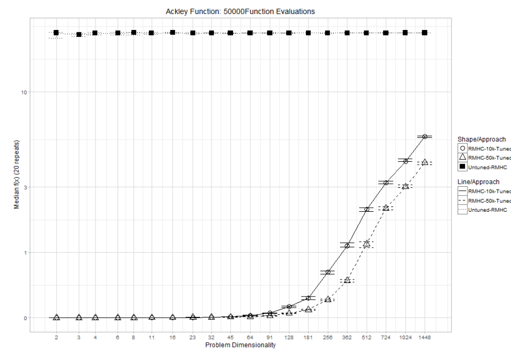
Appendix D: Plots showing the results of tuning each algorithm at a 10,000 and 50,000 evaluation budget compared to the corresponding untuned algorithm

D.1 SMAC parameter tuning results

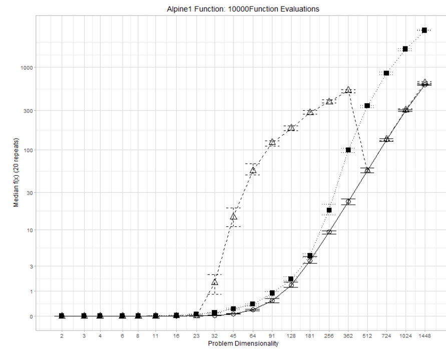
Random Mutation Hill Climbing (RMHC)



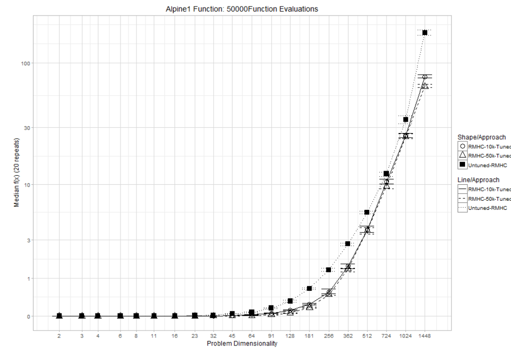
Ackley Function at 10,000 Evaluations



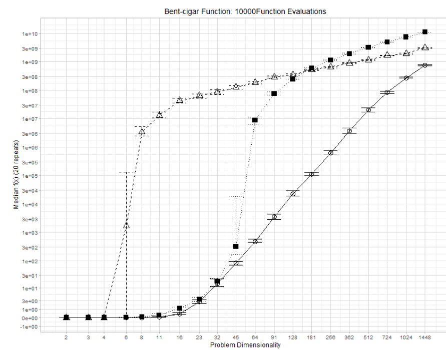
Ackley Function at 50,000 Evaluations



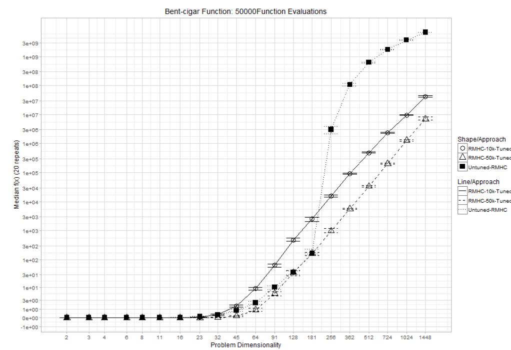
Alpine no.1 Function at 10,000 Evaluations



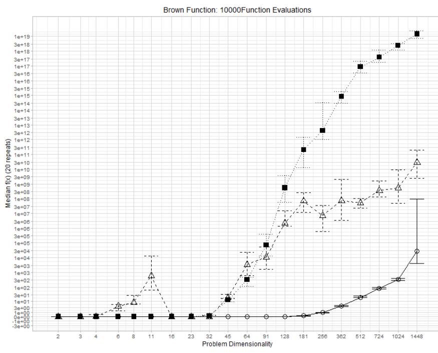
Alpine no.1 Function at 50,000 Evaluations



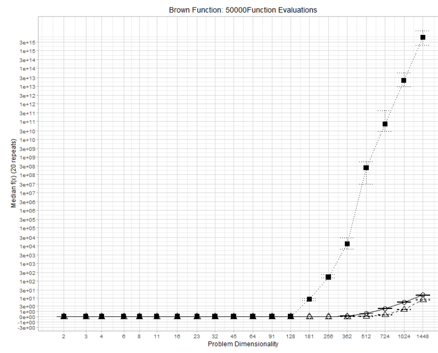
Bent Cigar Function at 10,000 Evaluations



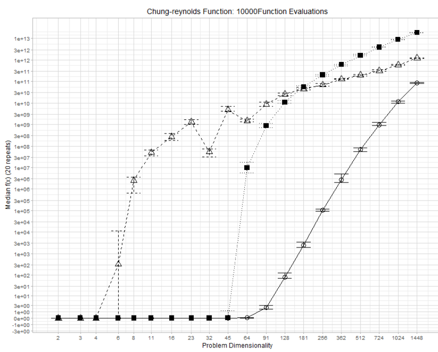
Bent Cigar Function at 50,000 Evaluations



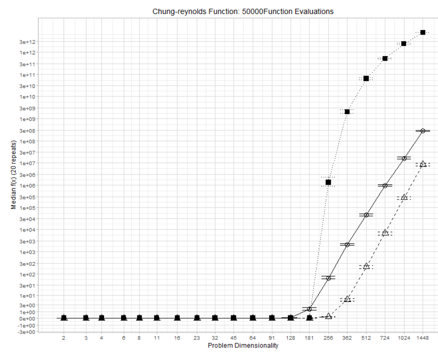
Brown Function at 10,000 Evaluations



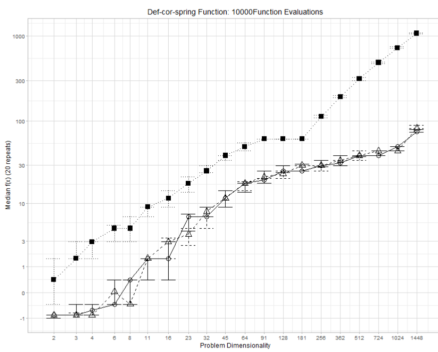
Brown Function at 50,000 Evaluations



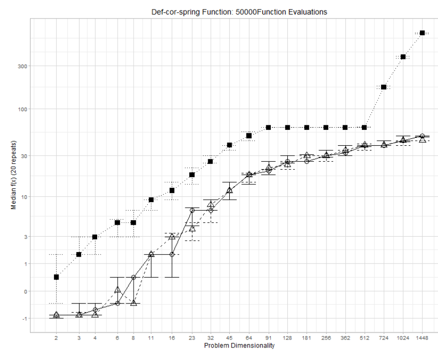
Chung-Reynolds Function at 10,000 Evaluations



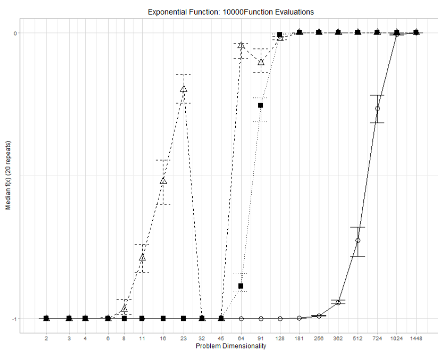
Chung-Reynolds Function at 50,000 Evaluations



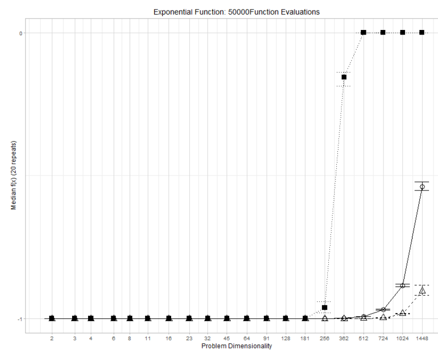
Deflected Corrugated Spring Function at 10,000 Evaluations



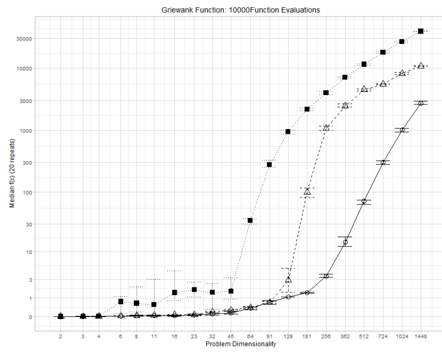
Deflected Corrugated Spring Function at 50,000 Evaluations



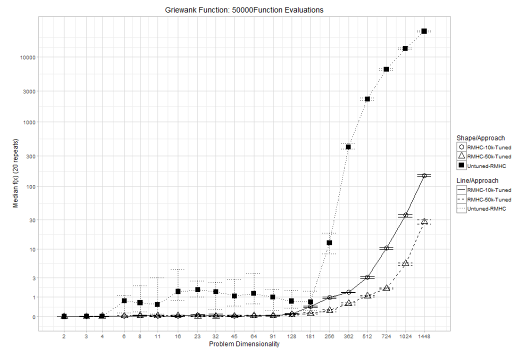
Exponential Function at 10,000 Evaluations



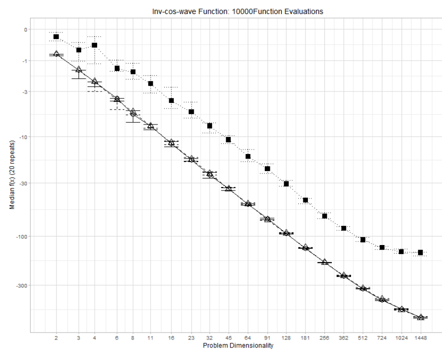
Exponential Function at 50,000 Evaluations



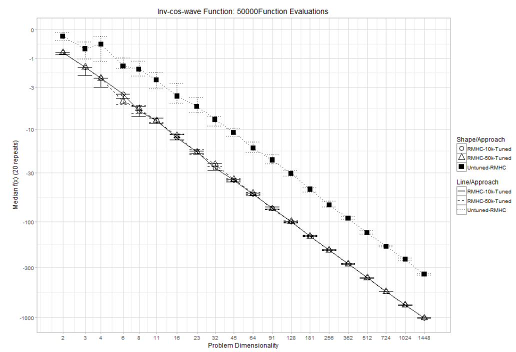
Griewank Function at 10,000 Evaluations



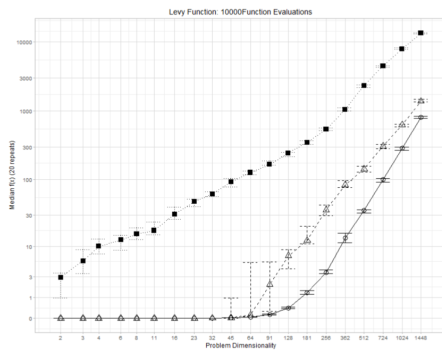
Griewank Function at 50,000 Evaluations



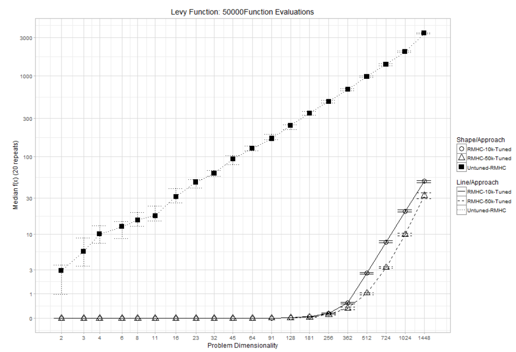
Inverted Cosine Wave Function at 10,000 Evaluations



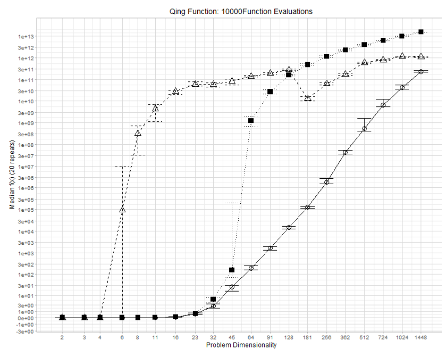
Inverted Cosine Wave Function at 50,000 Evaluations



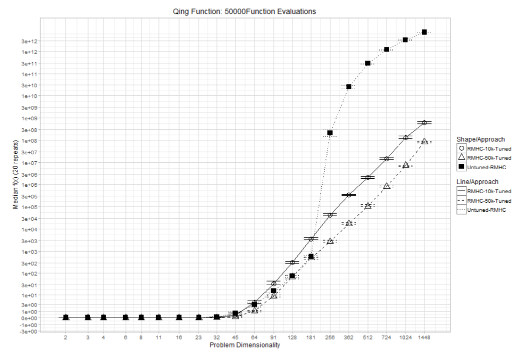
Levy Function at 10,000 Evaluations



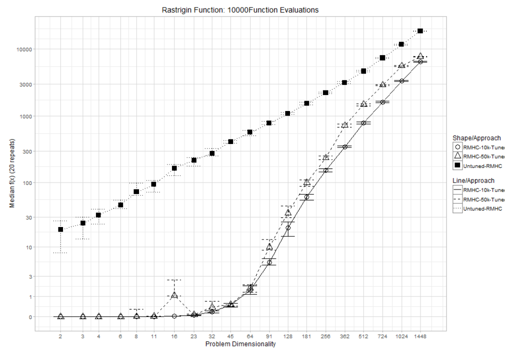
Levy Function at 50,000 Evaluations



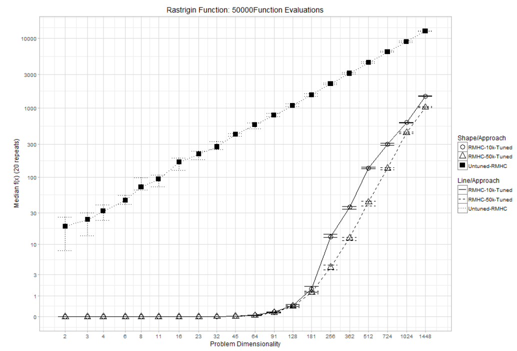
Qing Function at 10,000 Evaluations



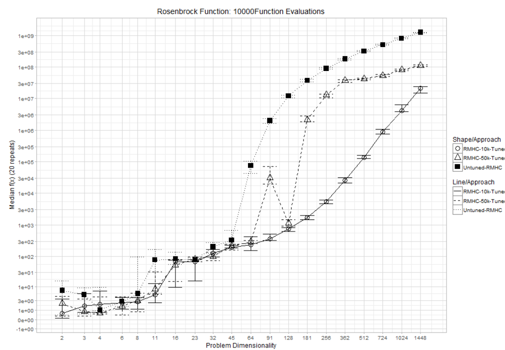
Qing Function at 50,000 Evaluations



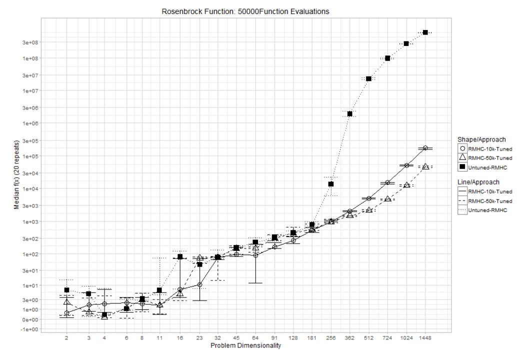
Rastrigin Function at 10,000 Evaluations



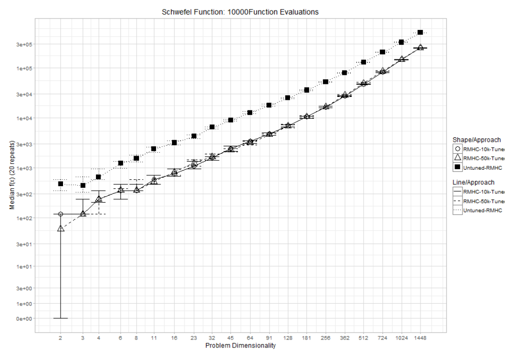
Rastrigin Function at 50,000 Evaluations



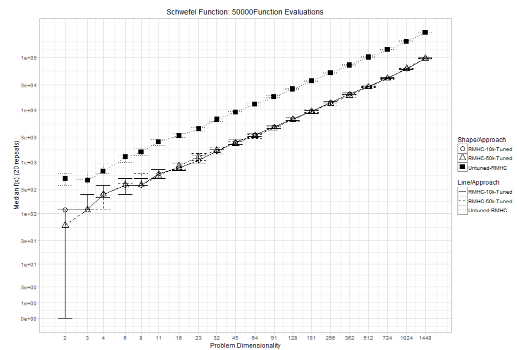
Rosenbrock Function at 10,000 Evaluations



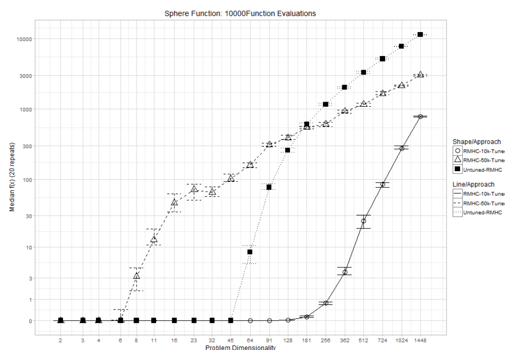
Rosenbrock Function at 50,000 Evaluations



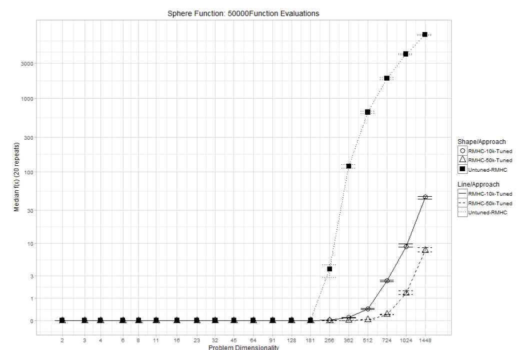
Schwefel Function at 10,000 Evaluations



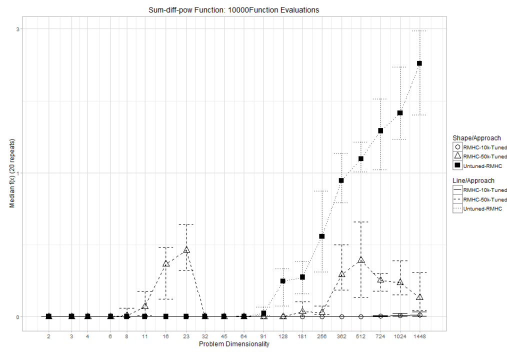
Schwefel Function at 50,000 Evaluations



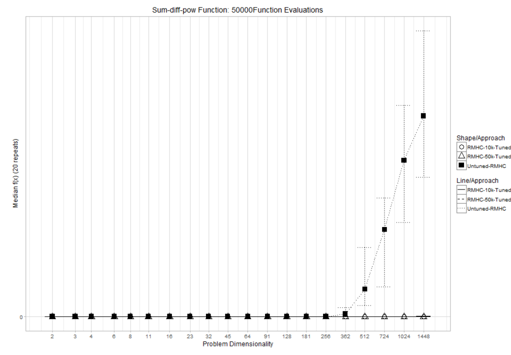
Sphere Function at 10,000 Evaluations



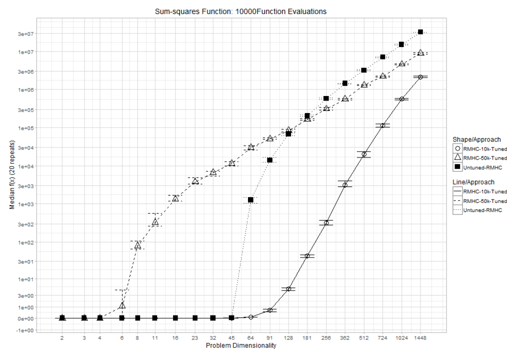
Sphere Function at 50,000 Evaluations



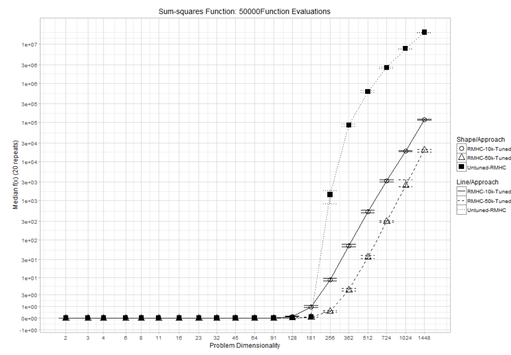
Sum of Different Powers Function at 10,000 Evaluations



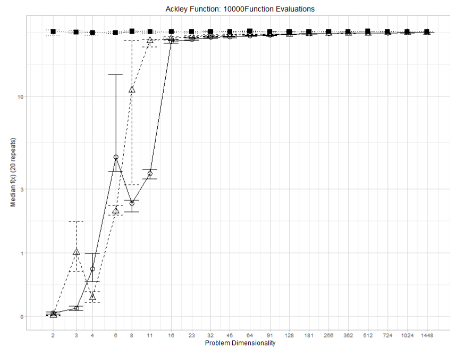
Sum of Different Powers Function at 50,000 Evaluations



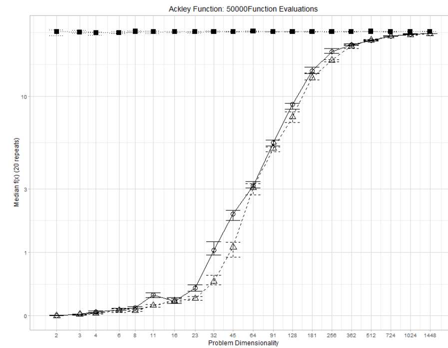
Sum Squares Function at 10,000 Evaluations



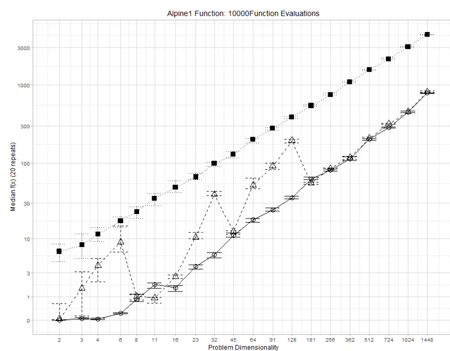
Sum Squares Function at 50,000 Evaluations



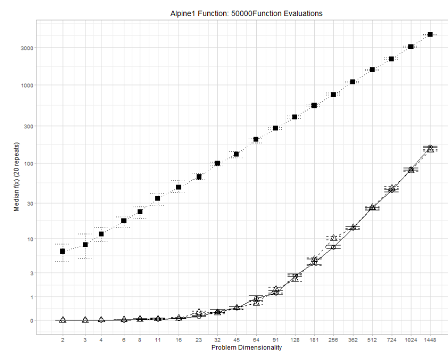
Ackley Function at 10,000 Evaluations



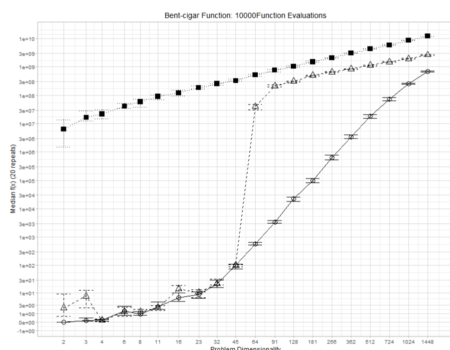
Ackley Function at 50,000 Evaluations



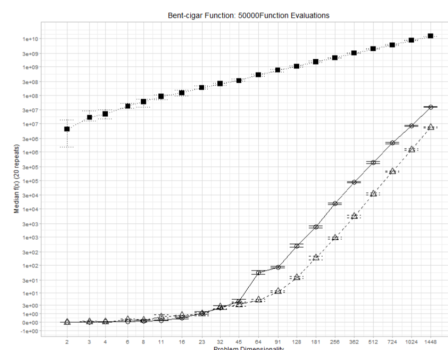
Alpine no.1 Function at 10,000 Evaluations



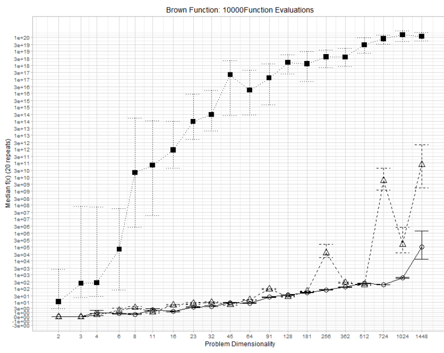
Alpine no.1 Function at 50,000 Evaluations



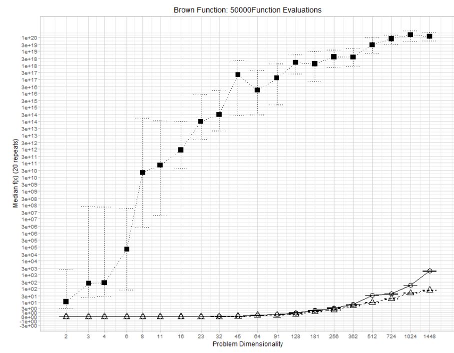
Bent Cigar Function at 10,000 Evaluations



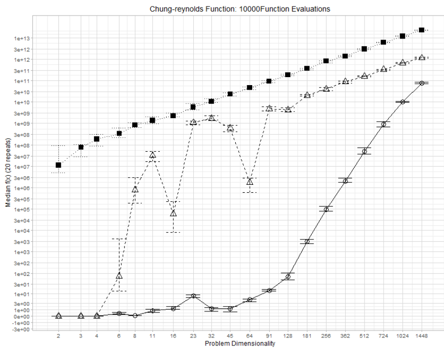
Bent Cigar Function at 50,000 Evaluations



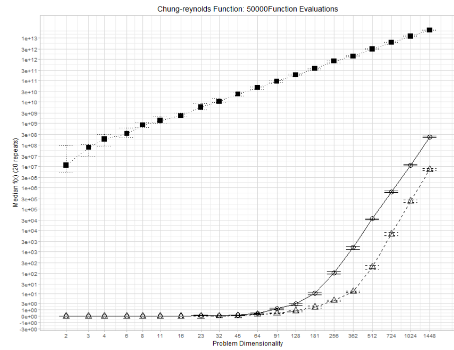
Brown Function at 10,000 Evaluations



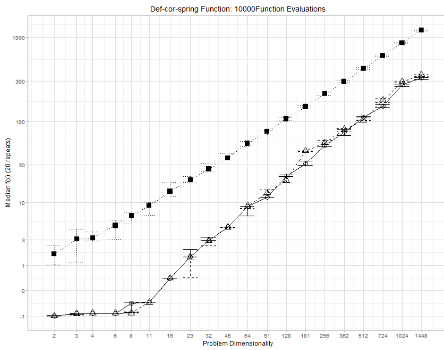
Brown Function at 50,000 Evaluations



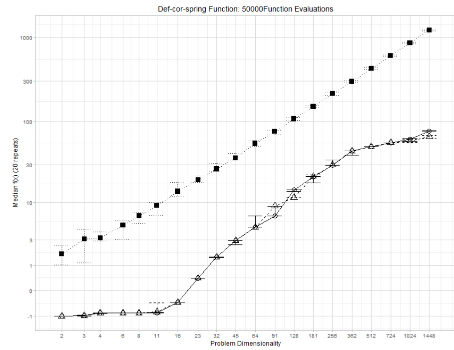
Chung-Reynolds Function at 10,000 Evaluations



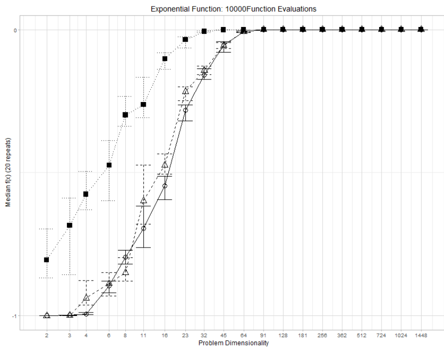
Chung-Reynolds Function at 50,000 Evaluations



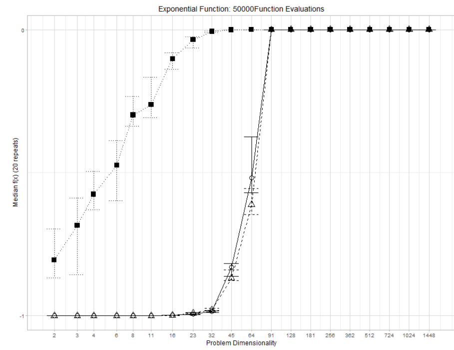
Deflected Corrugated Spring Function at 10,000 Evaluations



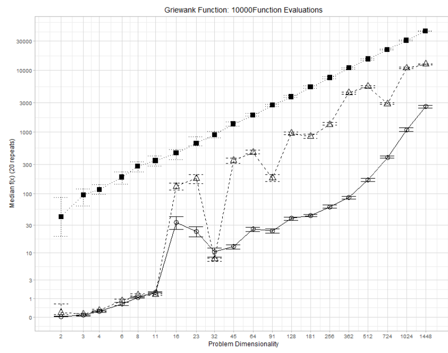
Deflected Corrugated Spring Function at 50,000 Evaluations



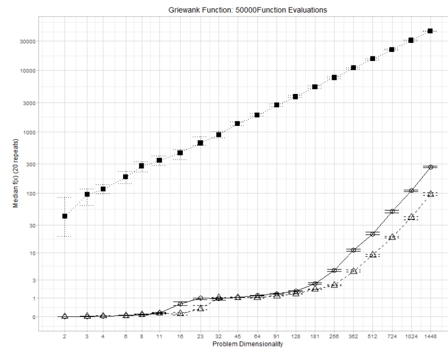
Exponential Function at 10,000 Evaluations



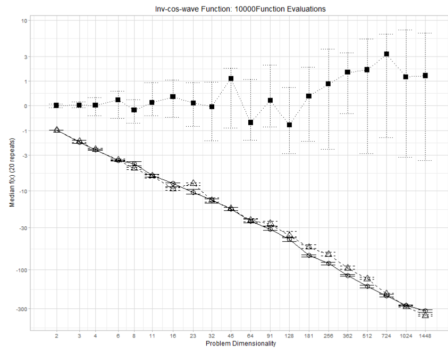
Exponential Function at 50,000 Evaluations



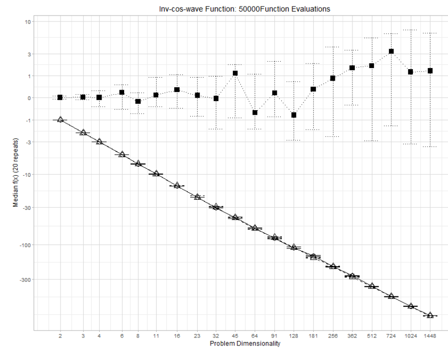
Griewank Function at 10,000 Evaluations



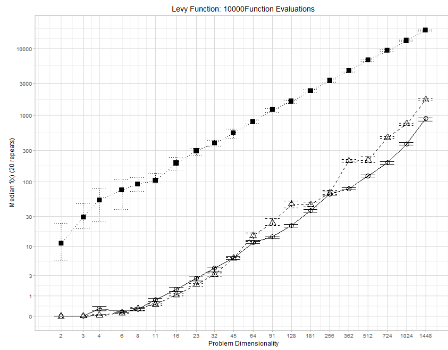
Griewank Function at 50,000 Evaluations



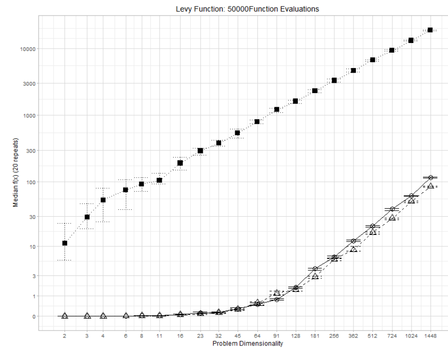
Inverted Cosine Wave Function at 10,000 Evaluations



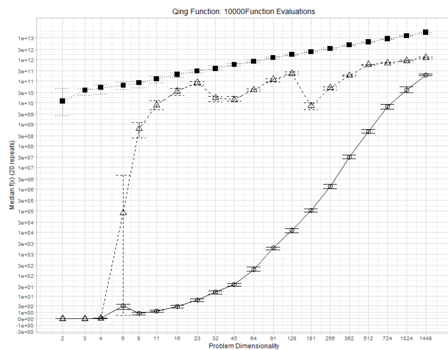
Inverted Cosine Wave Function at 50,000 Evaluations



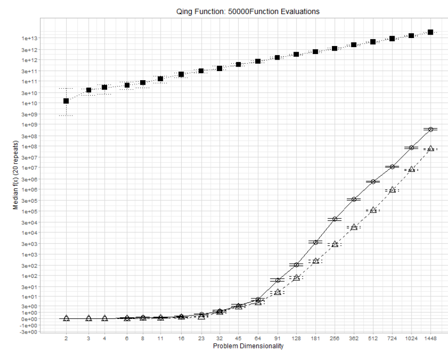
Levy Function at 10,000 Evaluations



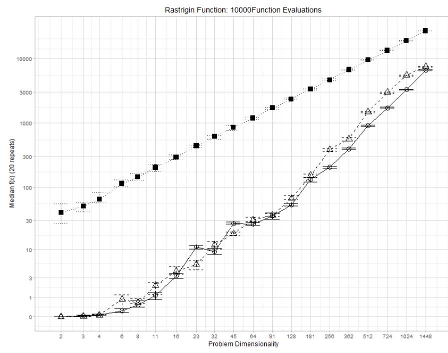
Levy Function at 50,000 Evaluations



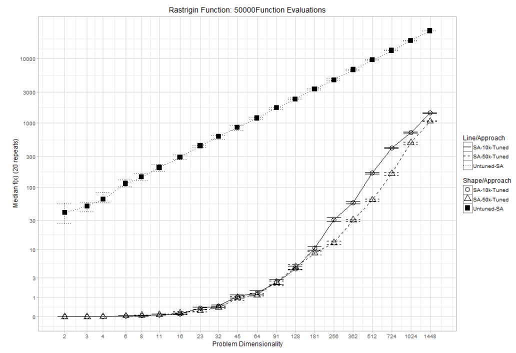
Qing Function at 10,000 Evaluations



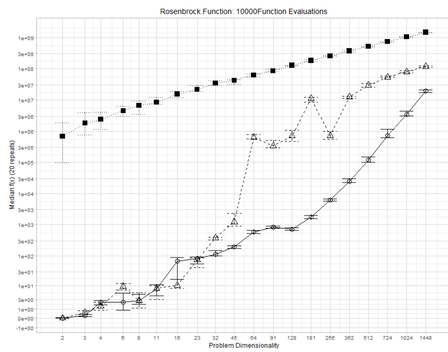
Qing Function at 50,000 Evaluations



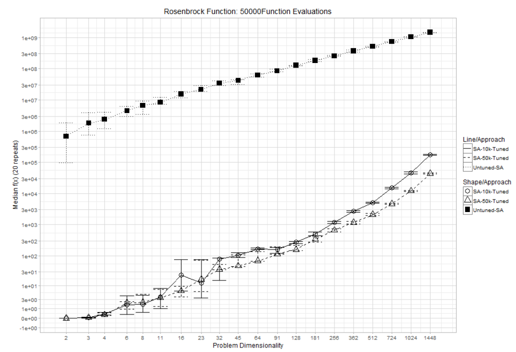
Rastrigin Function at 10,000 Evaluations



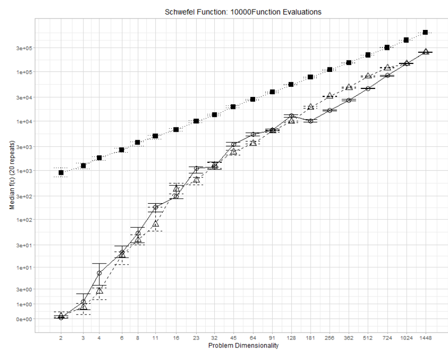
Rastrigin Function at 50,000 Evaluations



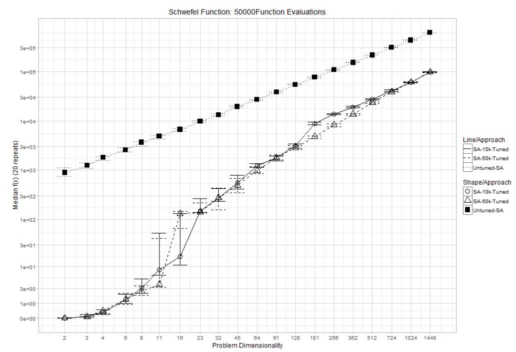
Rosenbrock Function at 10,000 Evaluations



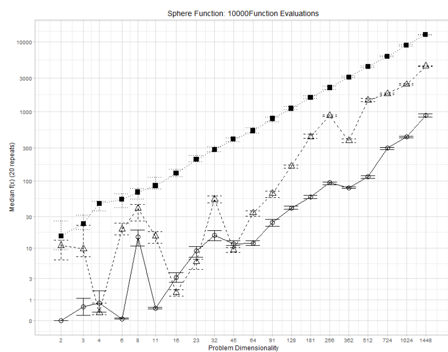
Rosenbrock Function at 50,000 Evaluations



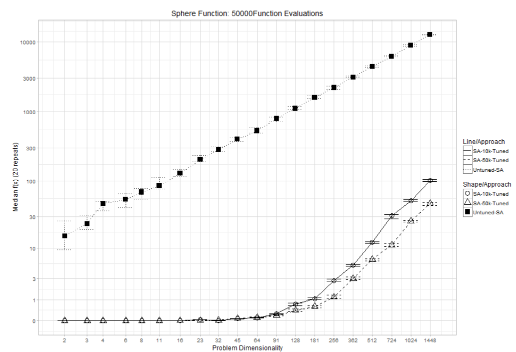
Schwefel Function at 10,000 Evaluations



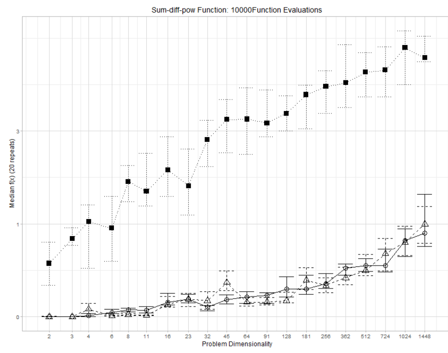
Schwefel Function at 50,000 Evaluations



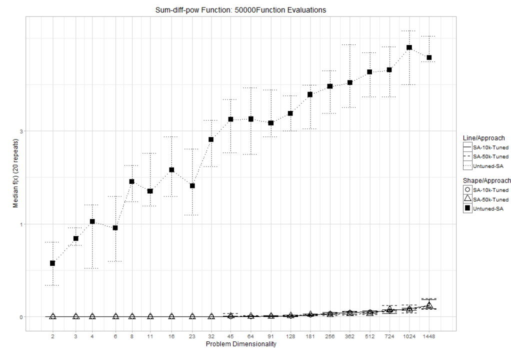
Sphere Function at 10,000 Evaluations



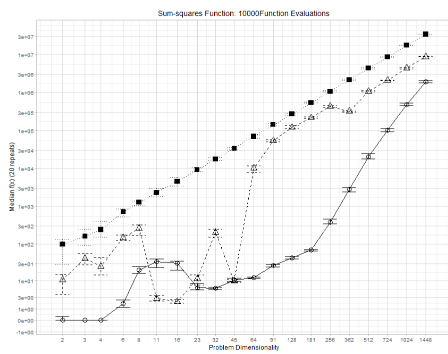
Sphere Function at 50,000 Evaluations



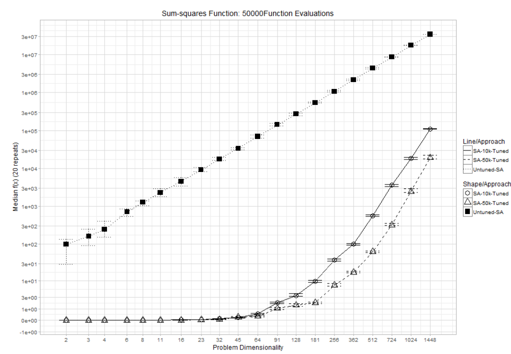
Sum of Different Powers Function at 10,000 Evaluations



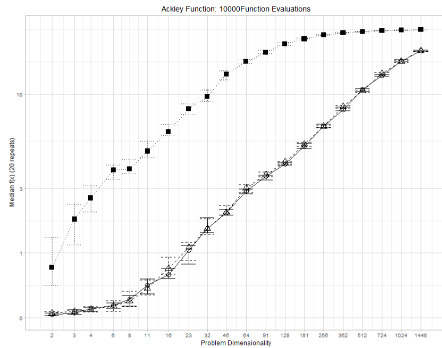
Sum of Different Powers Function at 50,000 Evaluations



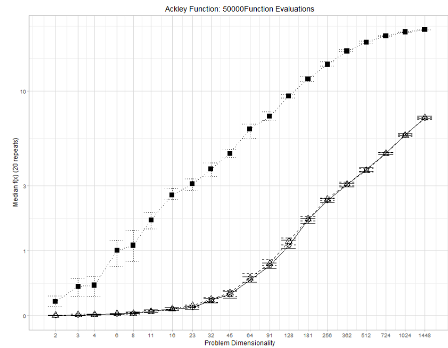
Sum Squares Function at 10,000 Evaluations



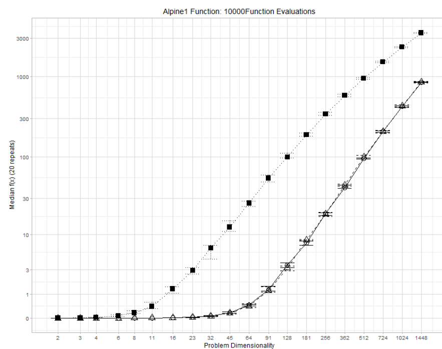
Sum Squares Function at 50,000 Evaluations



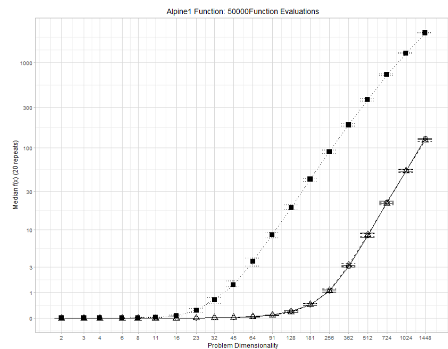
Ackley Function at 10,000 Evaluations



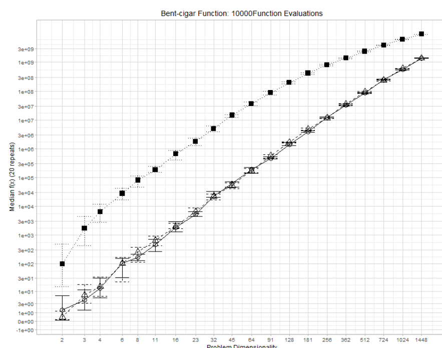
Ackley Function at 50,000 Evaluations



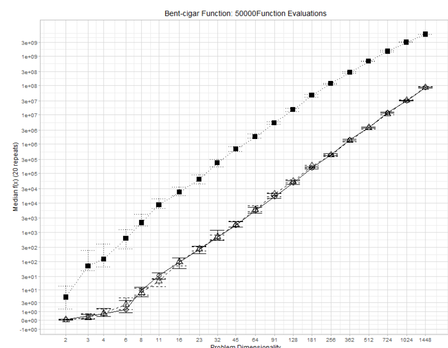
Alpine no.1 Function at 10,000 Evaluations



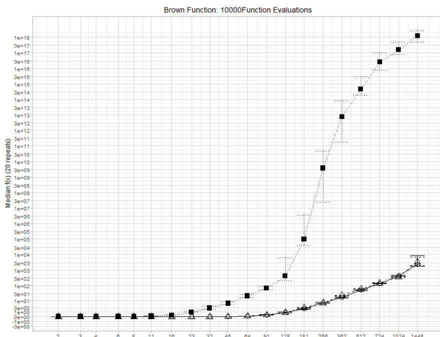
Alpine no.1 Function at 50,000 Evaluations



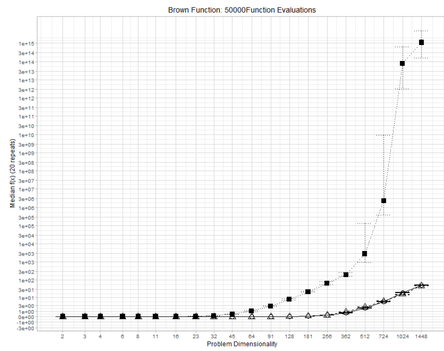
Bent Cigar Function at 10,000 Evaluations



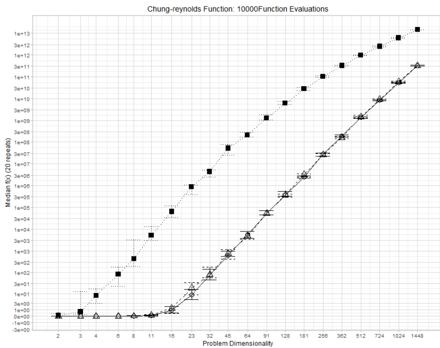
Bent Cigar Function at 50,000 Evaluations



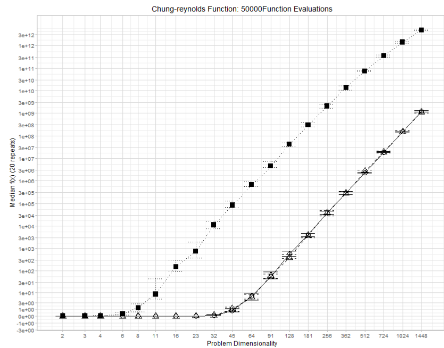
Brown Function at 10,000 Evaluations



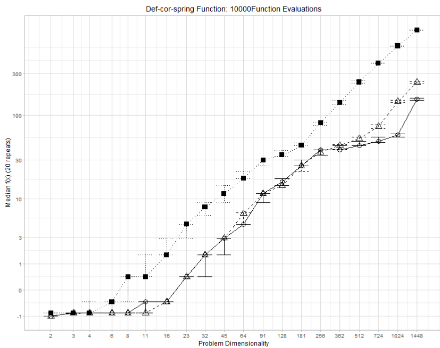
Brown Function at 50,000 Evaluations



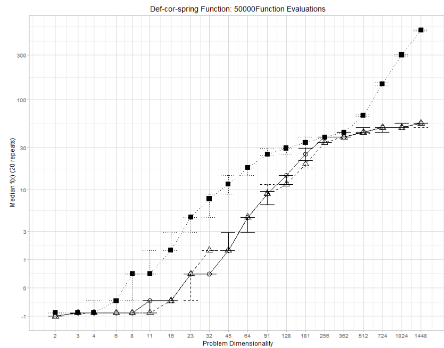
Chung-Reynolds Function at 10,000 Evaluations



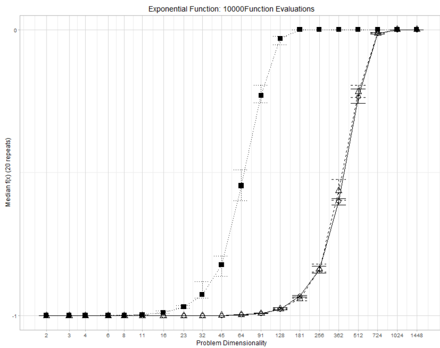
Chung-Reynolds Function at 50,000 Evaluations



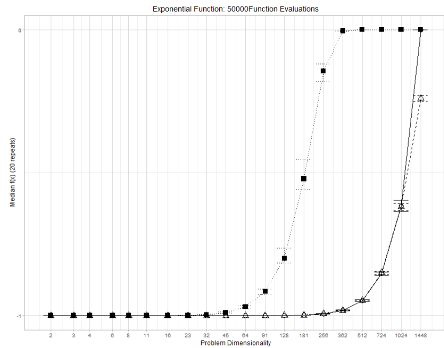
Deflected Corrugated Spring Function at 10,000 Evaluations



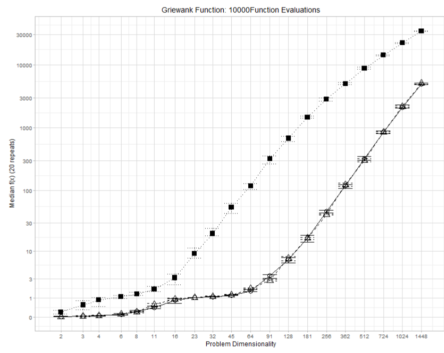
Deflected Corrugated Spring Function at 50,000 Evaluations



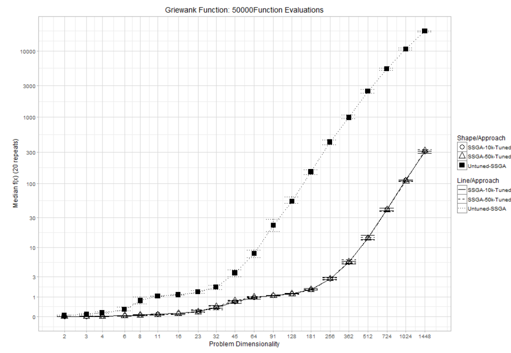
Exponential Function at 10,000 Evaluations



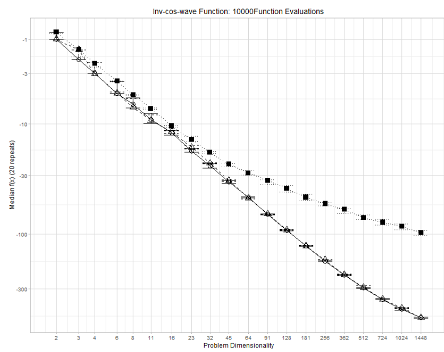
Exponential Function at 50,000 Evaluations



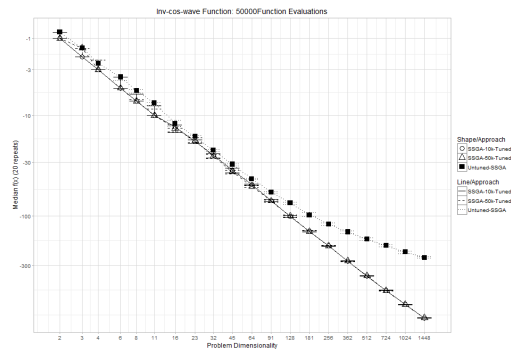
Griewank Function at 10,000 Evaluations



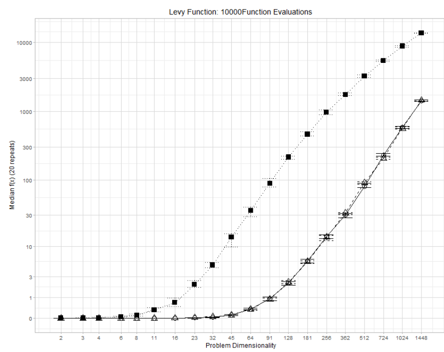
Griewank Function at 50,000 Evaluations



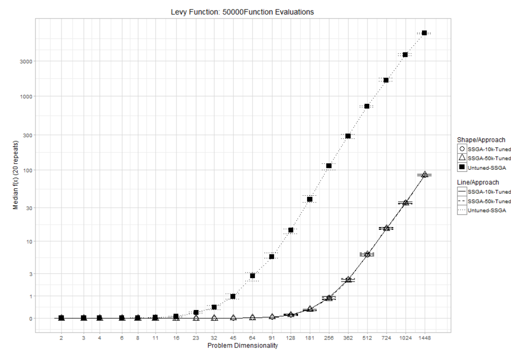
Inverted Cosine Wave Function at 10,000 Evaluations



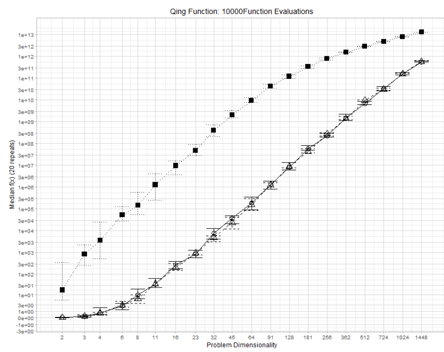
Inverted Cosine Wave Function at 50,000 Evaluations



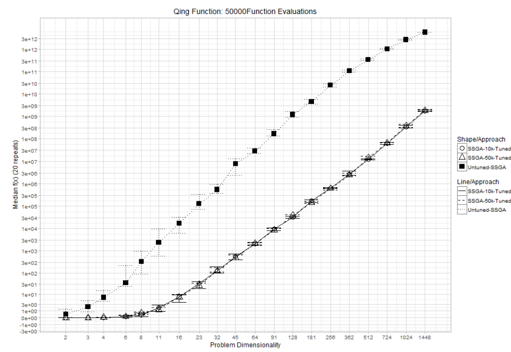
Levy Function at 10,000 Evaluations



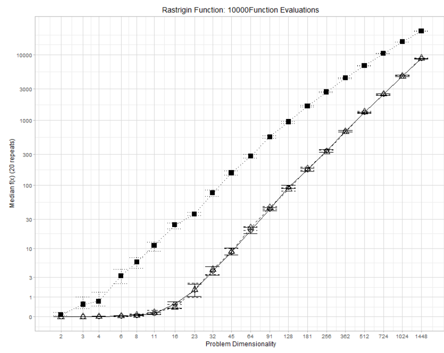
Levy Function at 50,000 Evaluations



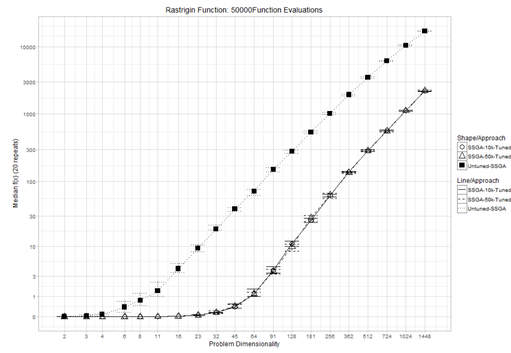
Qing Function at 10,000 Evaluations



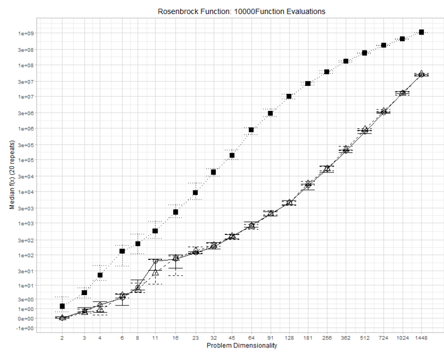
Qing Function at 50,000 Evaluations



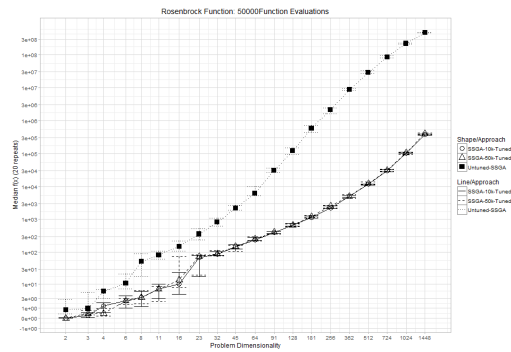
Rastrigin Function at 10,000 Evaluations



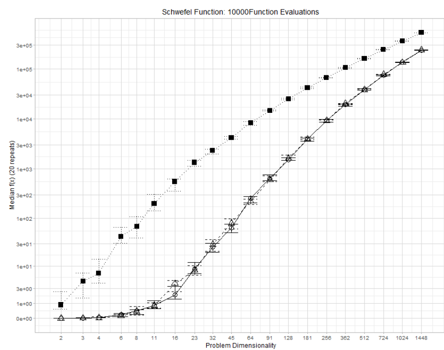
Rastrigin Function at 50,000 Evaluations



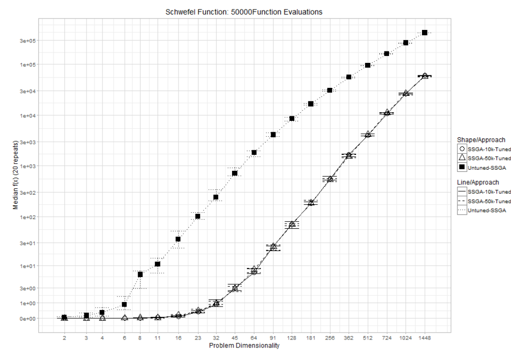
Rosenbrock Function at 10,000 Evaluations



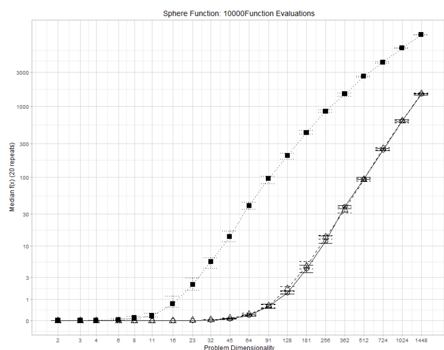
Rosenbrock Function at 50,000 Evaluations



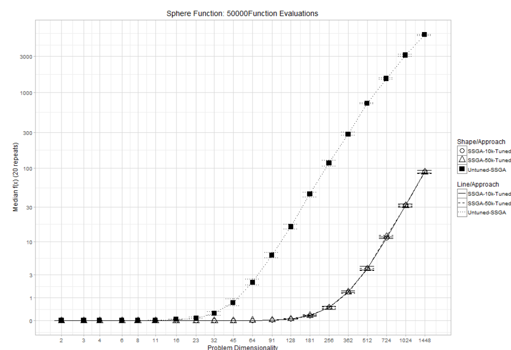
Schwefel Function at 10,000 Evaluations



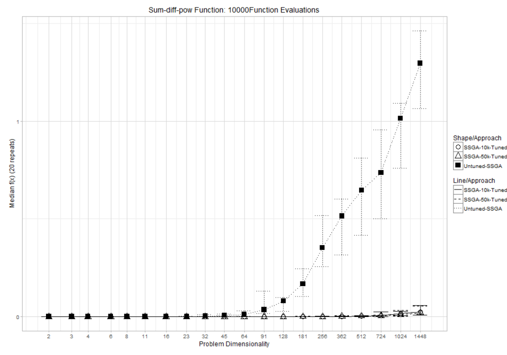
Schwefel Function at 50,000 Evaluations



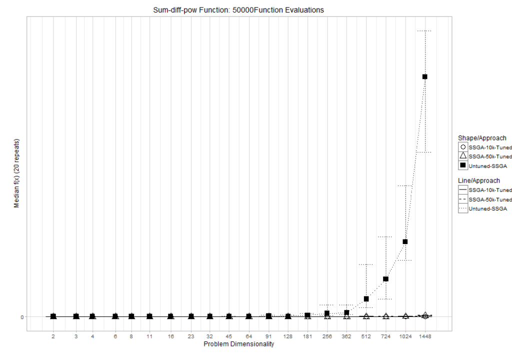
Sphere Function at 10,000 Evaluations



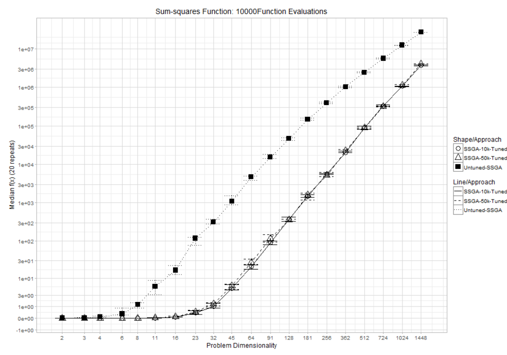
Sphere Function at 50,000 Evaluations



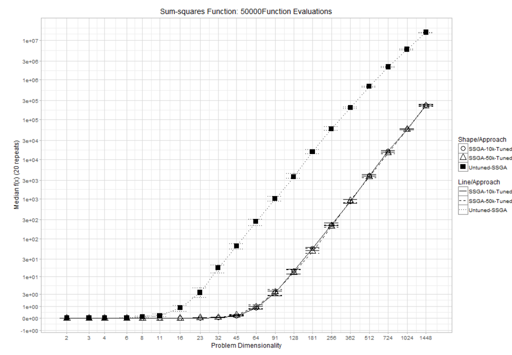
Sum of Different Powers Function at 10,000 Evaluations



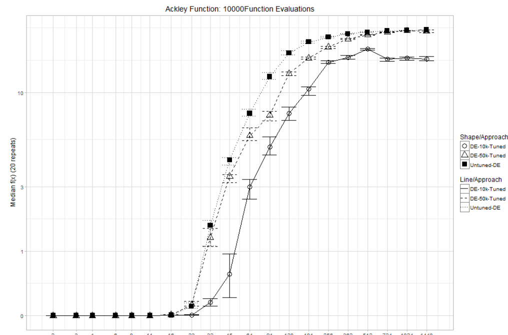
Sum of Different Powers Function at 50,000 Evaluations



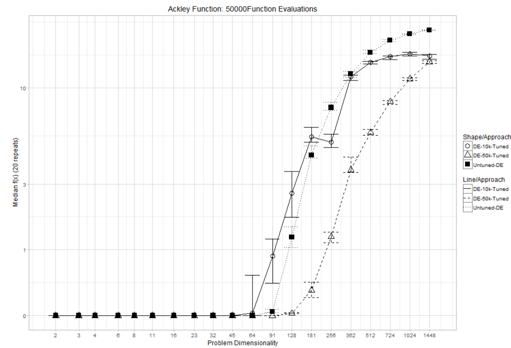
Sum Squares Function at 10,000 Evaluations



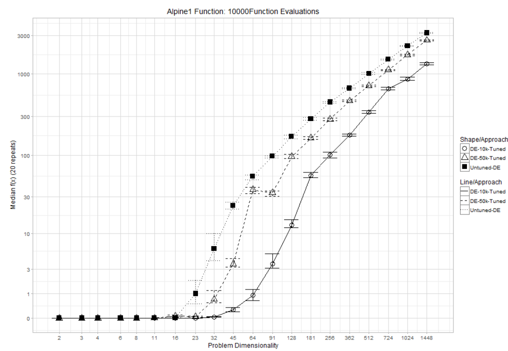
Sum Squares Function at 50,000 Evaluations



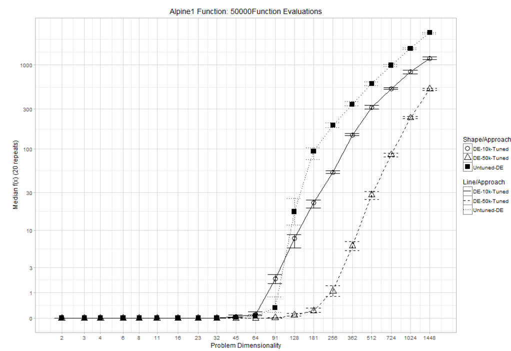
Ackley Function at 10,000 Evaluations



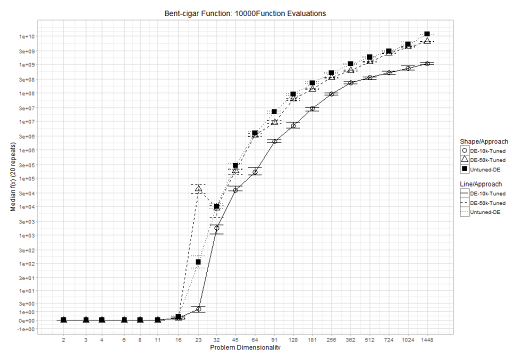
Ackley Function at 50,000 Evaluations



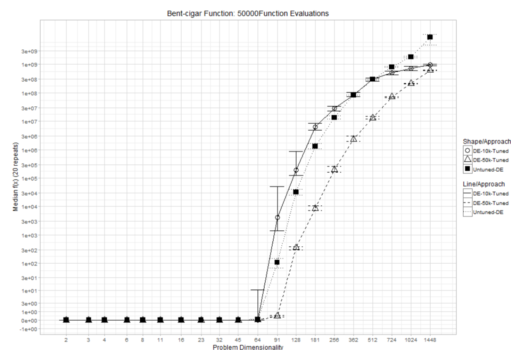
Alpine no.1 Function at 10,000 Evaluations



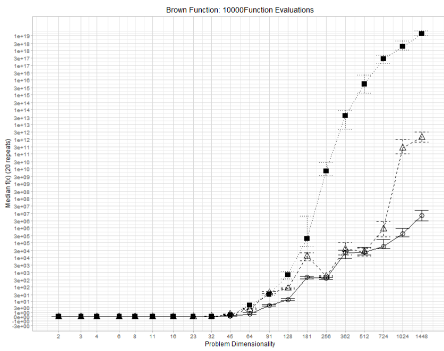
Alpine no.1 Function at 50,000 Evaluations



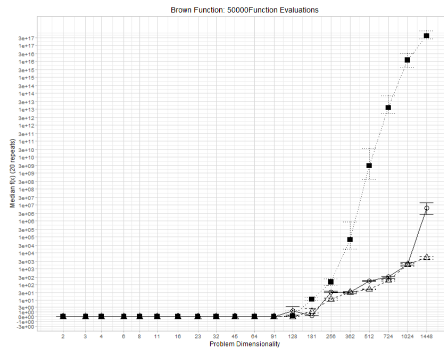
Bent Cigar Function at 10,000 Evaluations



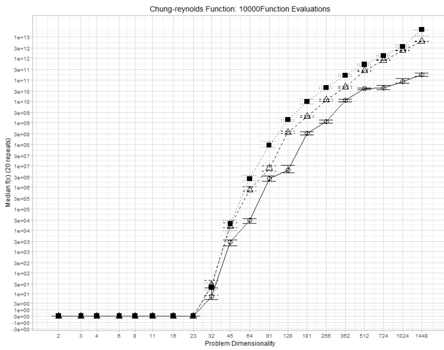
Bent Cigar Function at 50,000 Evaluations



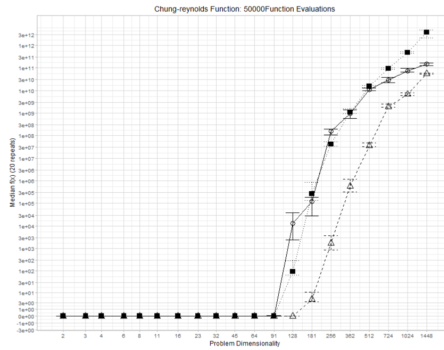
Brown Function at 10,000 Evaluations



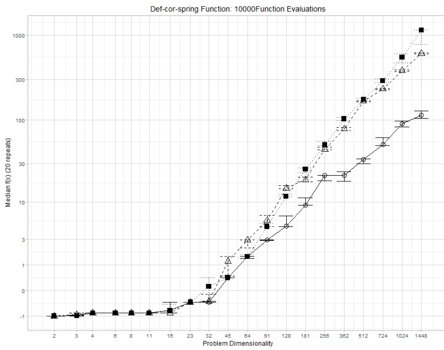
Brown Function at 50,000 Evaluations



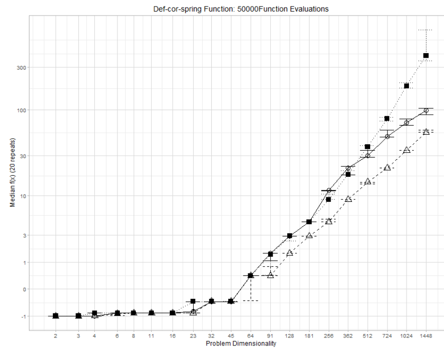
Chung-Reynolds Function at 10,000 Evaluations



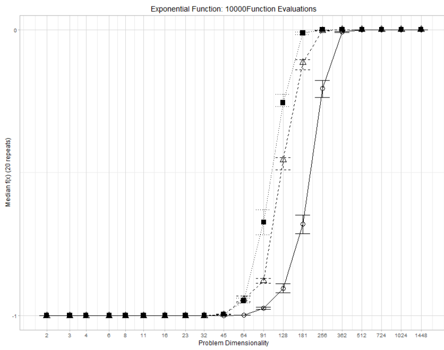
Chung-Reynolds Function at 50,000 Evaluations



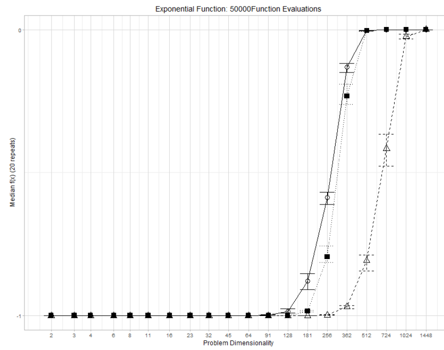
Deflected Corrugated Spring Function at 10,000 Evaluations



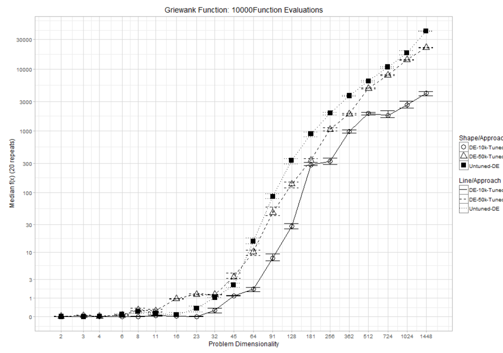
Deflected Corrugated Spring Function at 50,000 Evaluations



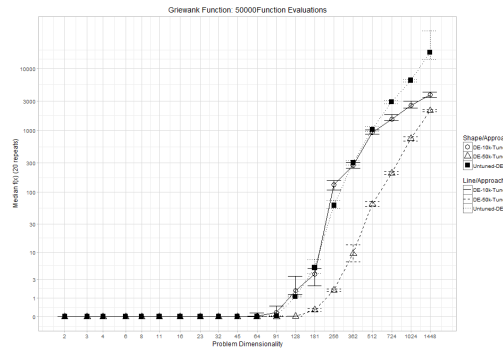
Exponential Function at 10,000 Evaluations



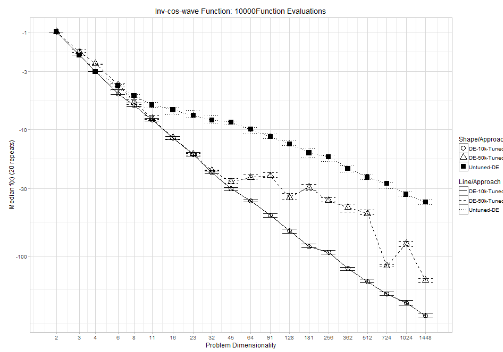
Exponential Function at 50,000 Evaluations



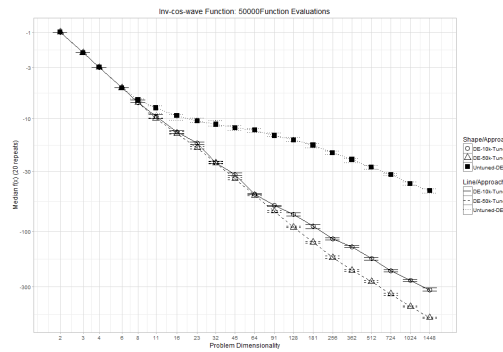
Griewank Function at 10,000 Evaluations



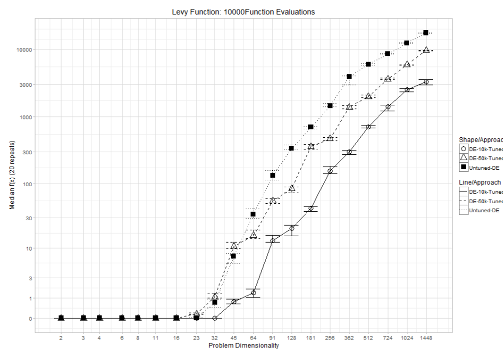
Griewank Function at 50,000 Evaluations



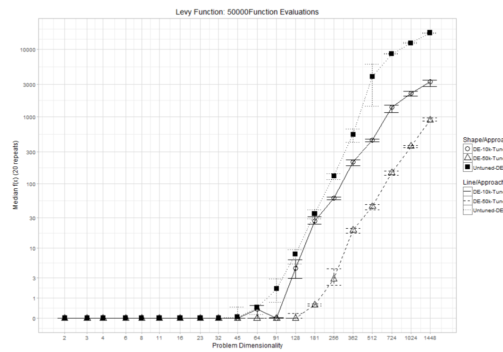
Inverted Cosine Wave Function at 10,000 Evaluations



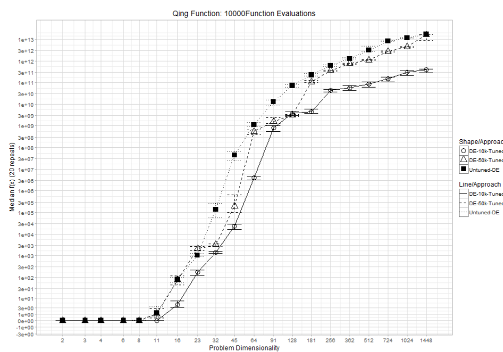
Inverted Cosine Wave Function at 50,000 Evaluations



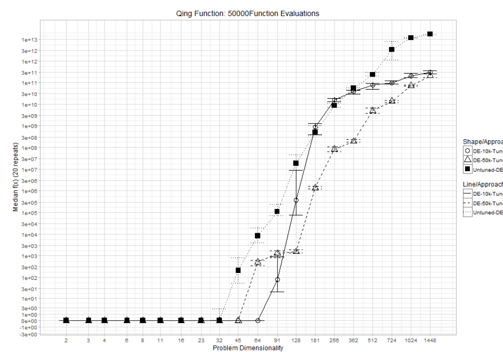
Levy Function at 10,000 Evaluations



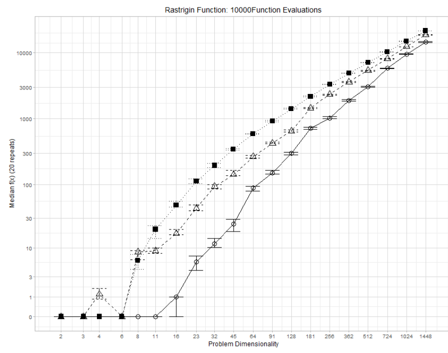
Levy Function at 50,000 Evaluations



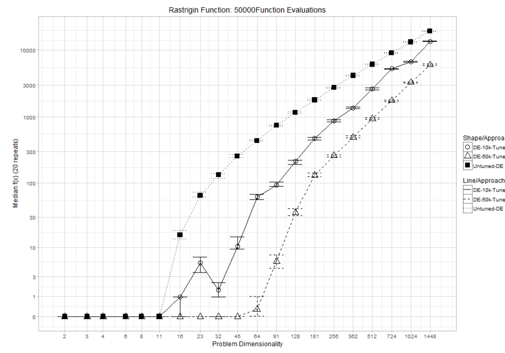
Qing Function at 10,000 Evaluations



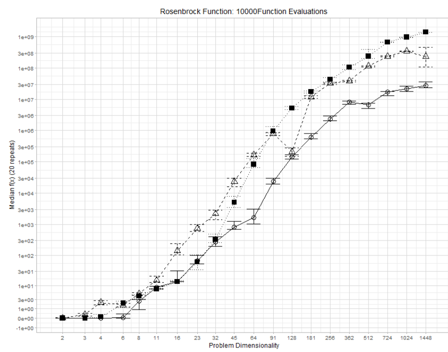
Qing Function at 50,000 Evaluations



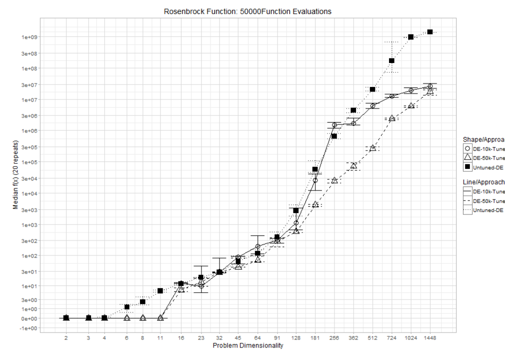
Rastrigin Function at 10,000 Evaluations



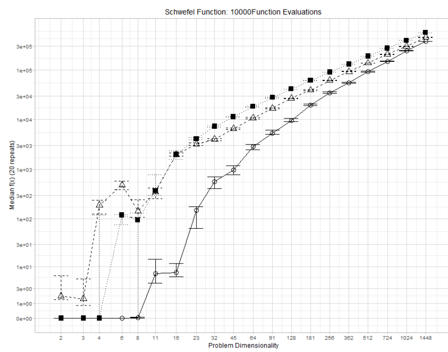
Rastrigin Function at 50,000 Evaluations



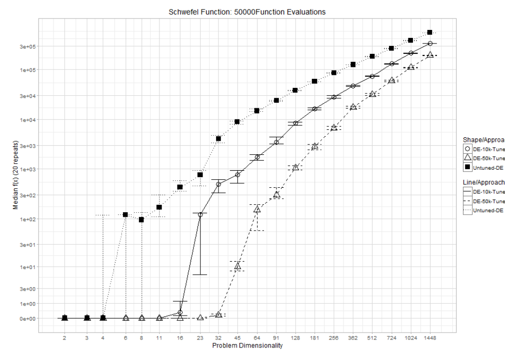
Rosenbrock Function at 10,000 Evaluations



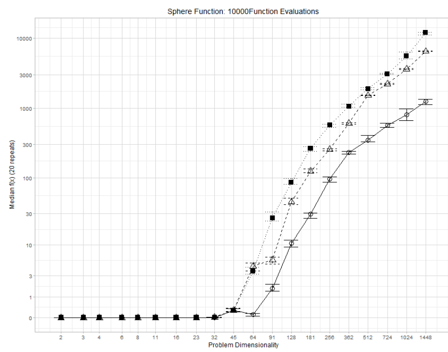
Rosenbrock Function at 50,000 Evaluations



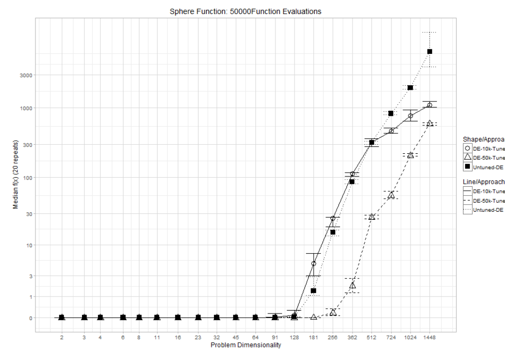
Schwefel Function at 10,000 Evaluations



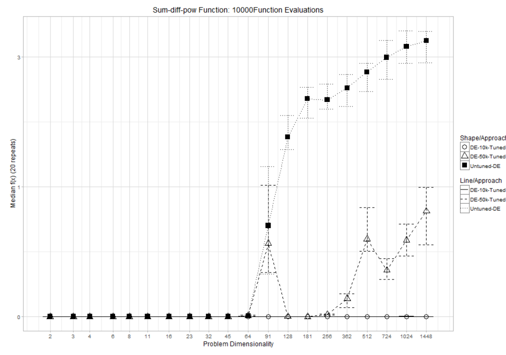
Schwefel Function at 50,000 Evaluations



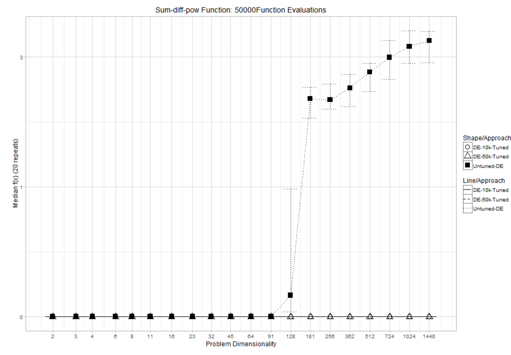
Sphere Function at 10,000 Evaluations



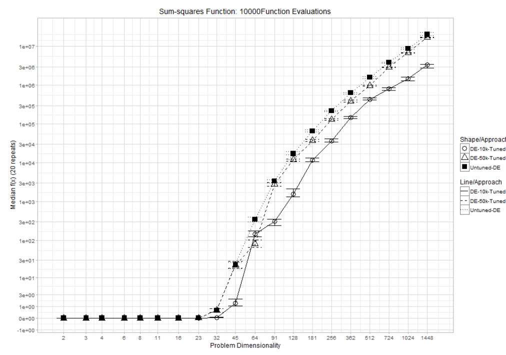
Sphere Function at 50,000 Evaluations



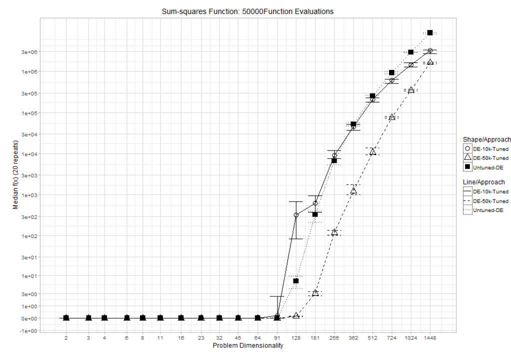
Sum of Different Powers Function at 10,000 Evaluations



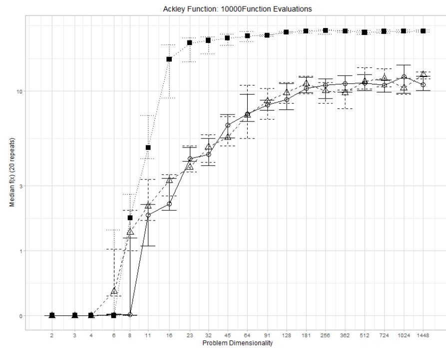
Sum of Different Powers Function at 50,000 Evaluations



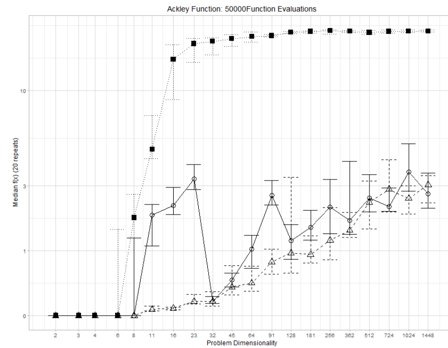
Sum Squares Function at 10,000 Evaluations



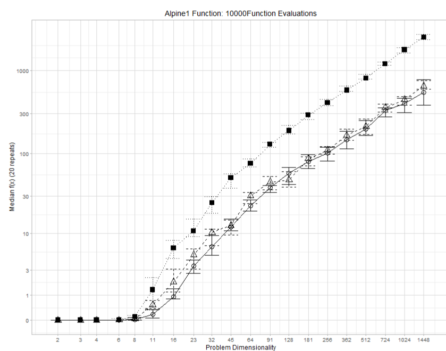
Sum Squares Function at 50,000 Evaluations



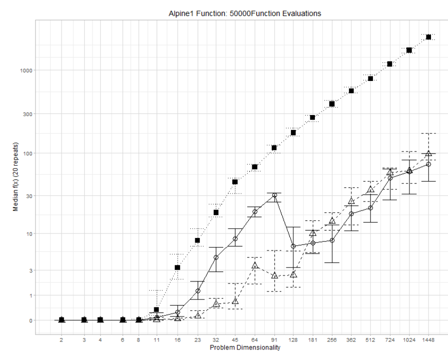
Ackley Function at 10,000 Evaluations



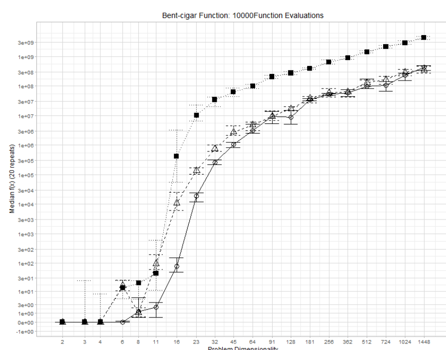
Ackley Function at 50,000 Evaluations



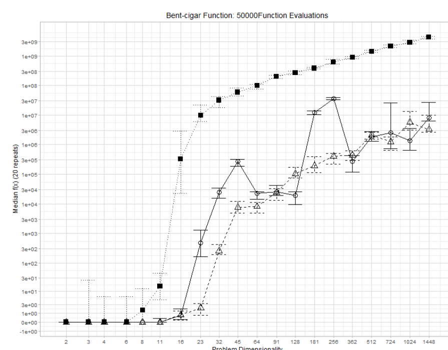
Alpine no.1 Function at 10,000 Evaluations



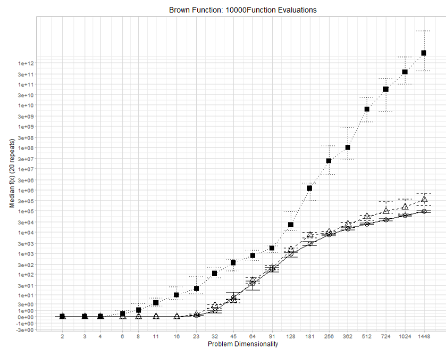
Alpine no.1 Function at 50,000 Evaluations



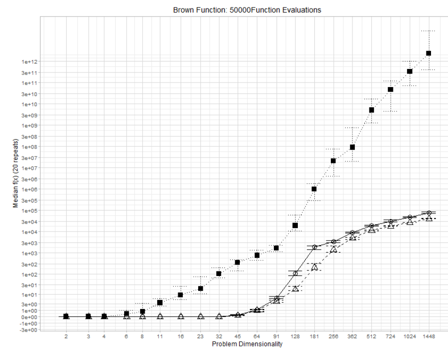
Bent Cigar Function at 10,000 Evaluations



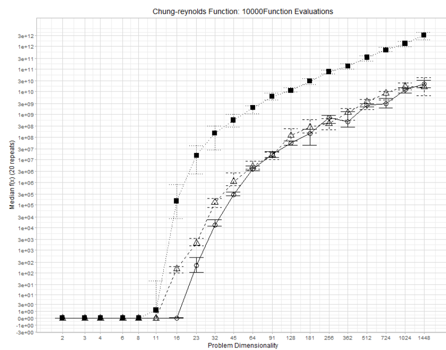
Bent Cigar Function at 50,000 Evaluations



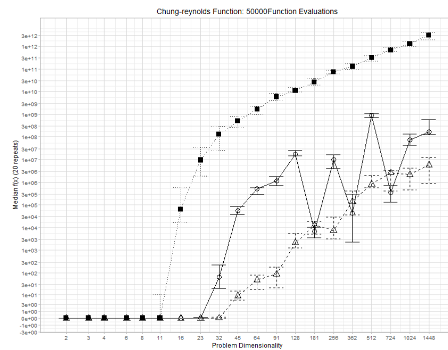
Brown Function at 10,000 Evaluations



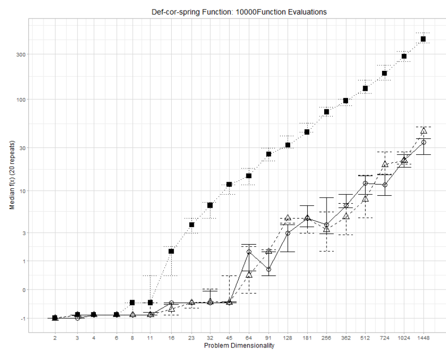
Brown Function at 50,000 Evaluations



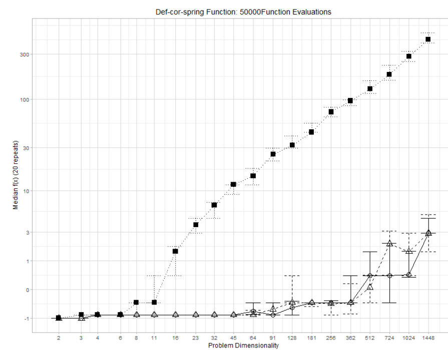
Chung-Reynolds Function at 10,000 Evaluations



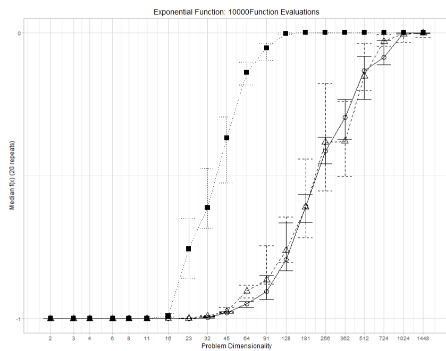
Chung-Reynolds Function at 50,000 Evaluations



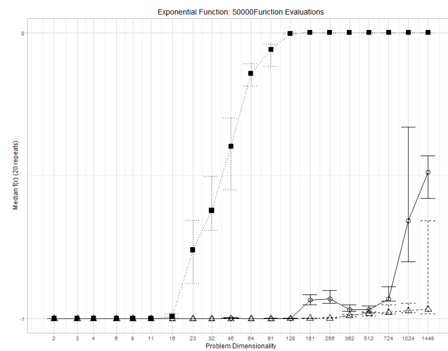
Deflected Corrugated Spring Function at 10,000 Evaluations



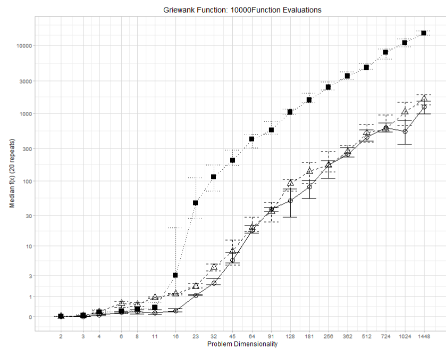
Deflected Corrugated Spring Function at 50,000 Evaluations



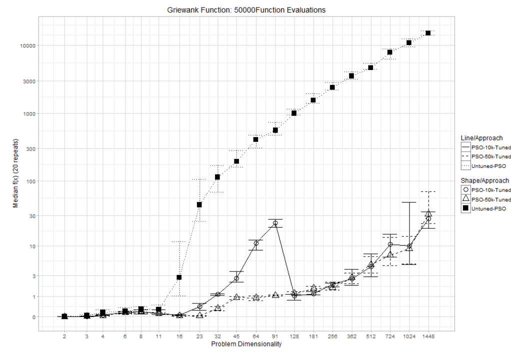
Exponential Function at 10,000 Evaluations



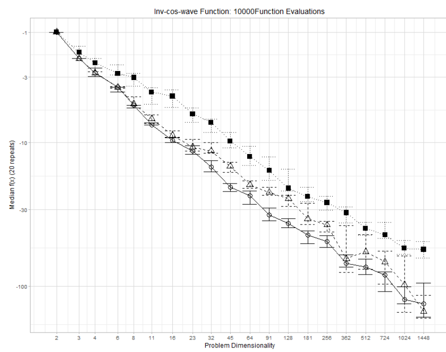
Exponential Function at 50,000 Evaluations



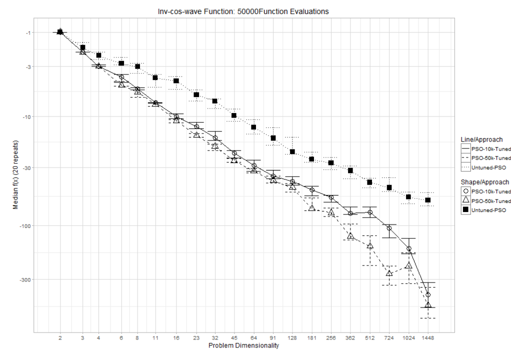
Griewank Function at 10,000 Evaluations



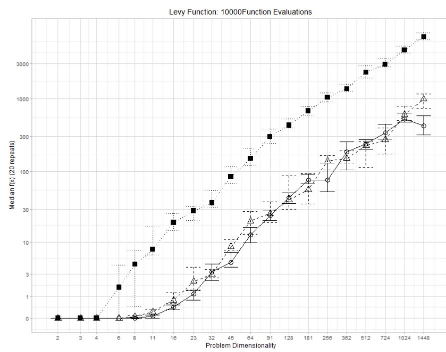
Griewank Function at 50,000 Evaluations



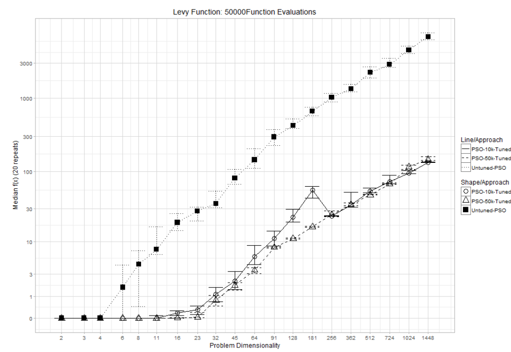
Inverted Cosine Wave Function at 10,000 Evaluations



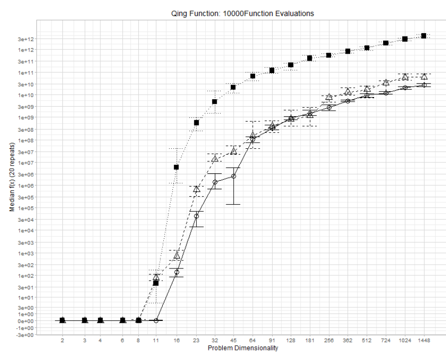
Inverted Cosine Wave Function at 50,000 Evaluations



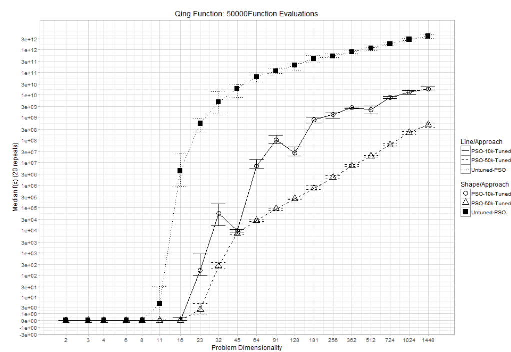
Levy Function at 10,000 Evaluations



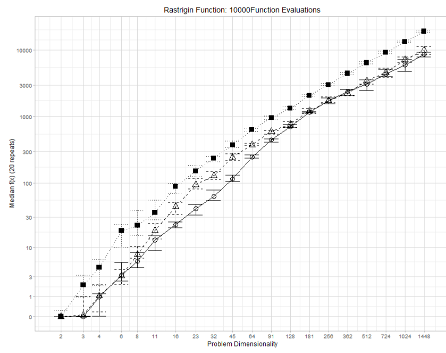
Levy Function at 50,000 Evaluations



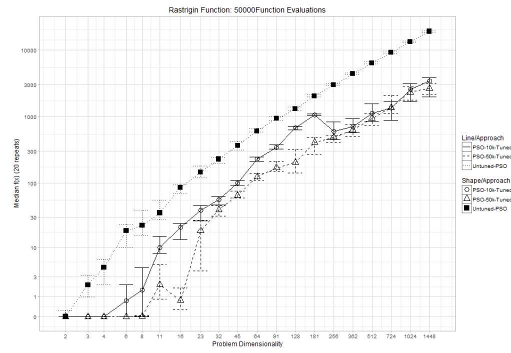
Qing Function at 10,000 Evaluations



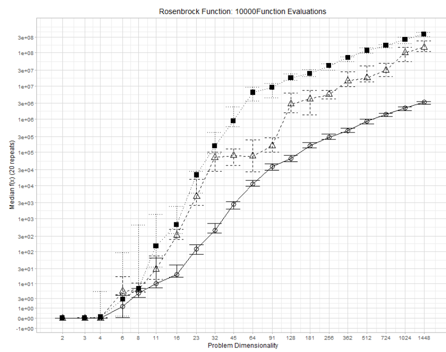
Qing Function at 50,000 Evaluations



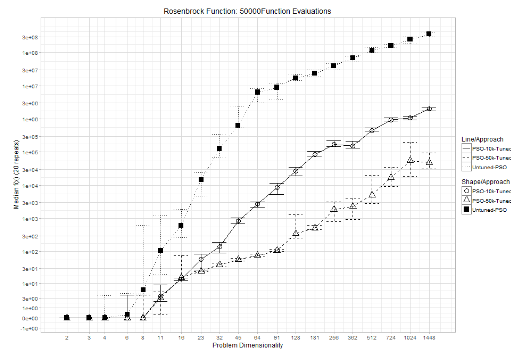
Rastrigin Function at 10,000 Evaluations



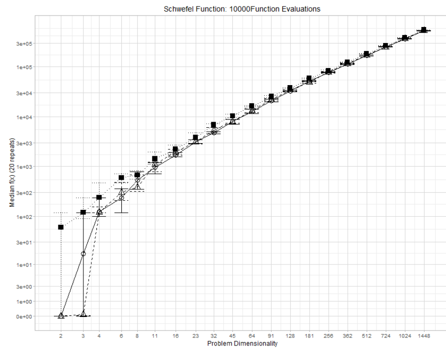
Rastrigin Function at 50,000 Evaluations



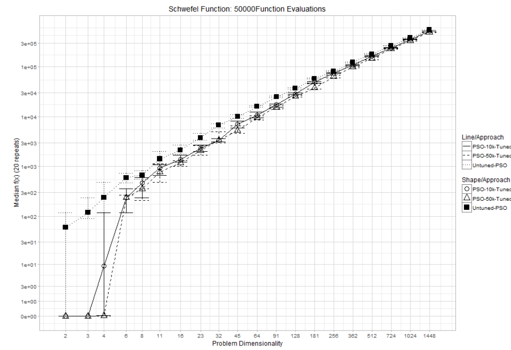
Rosenbrock Function at 10,000 Evaluations



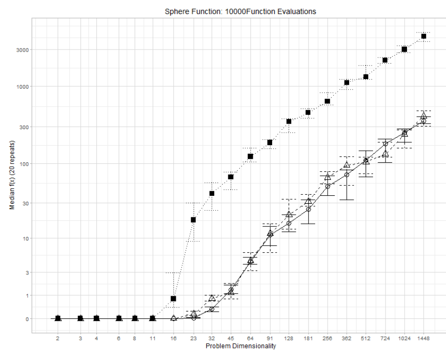
Rosenbrock Function at 50,000 Evaluations



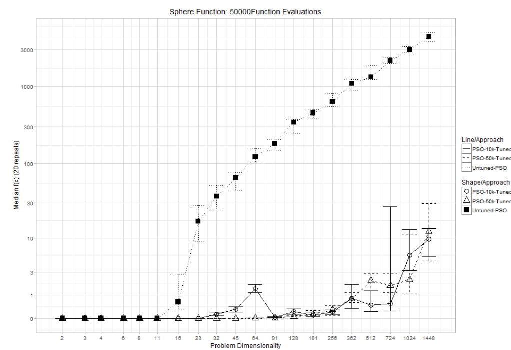
Schwefel Function at 10,000 Evaluations



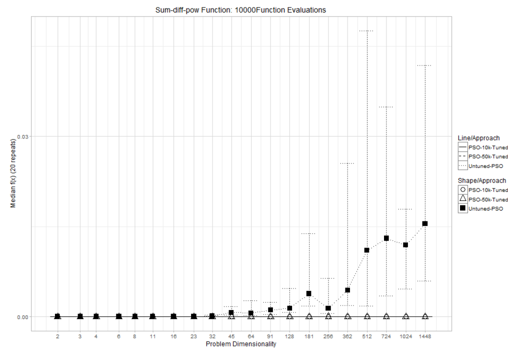
Schwefel Function at 50,000 Evaluations



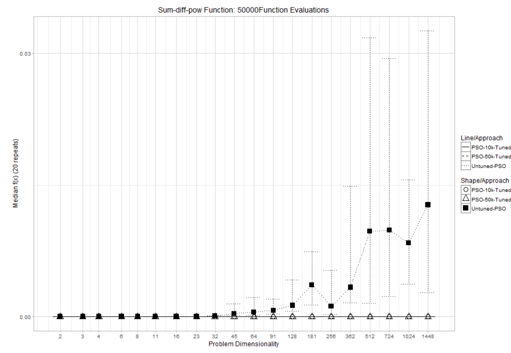
Sphere Function at 10,000 Evaluations



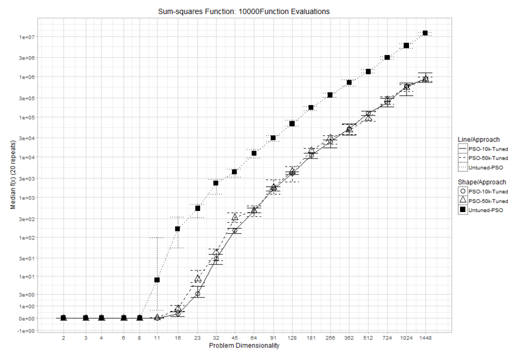
Sphere Function at 50,000 Evaluations



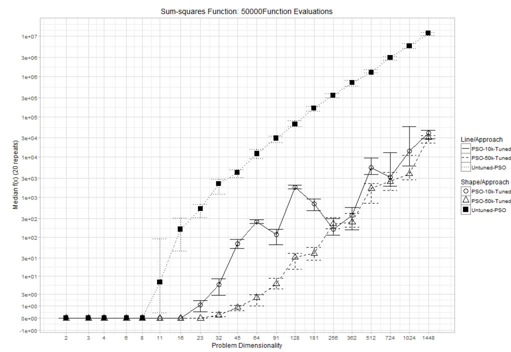
Sum of Different Powers Function at 10,000 Evaluations



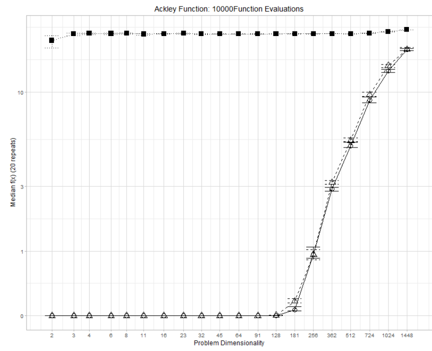
Sum of Different Powers Function at 50,000 Evaluations



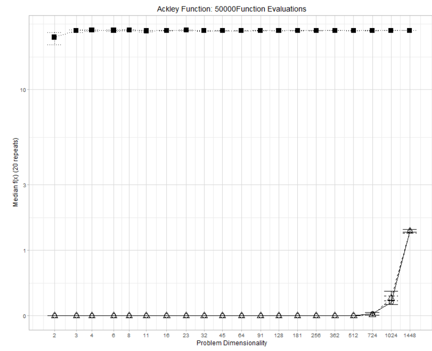
Sum Squares Function at 10,000 Evaluations



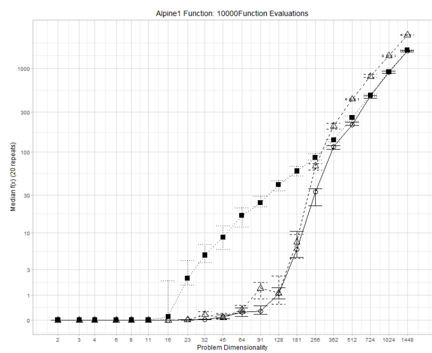
Sum Squares Function at 50,000 Evaluations



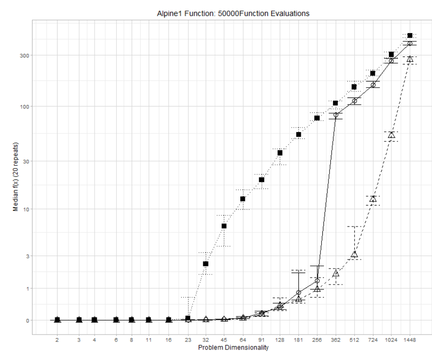
Ackley Function at 10,000 Evaluations



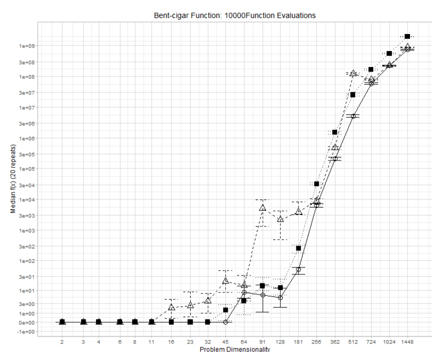
Ackley Function at 50,000 Evaluations



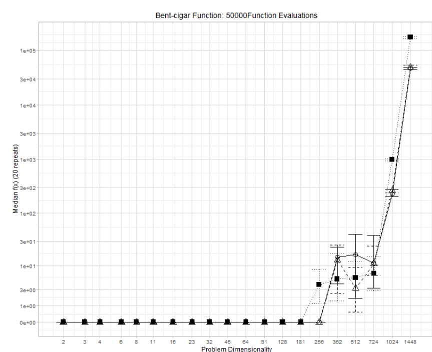
Alpine no.1 Function at 10,000 Evaluations



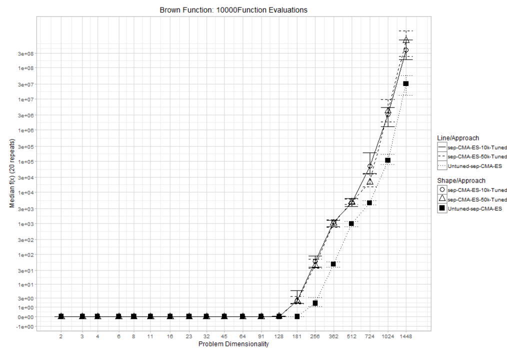
Alpine no.1 Function at 50,000 Evaluations



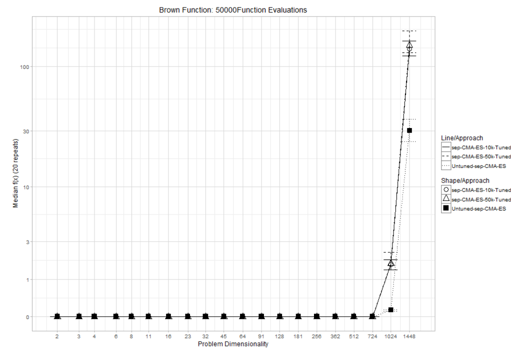
Bent Cigar Function at 10,000 Evaluations



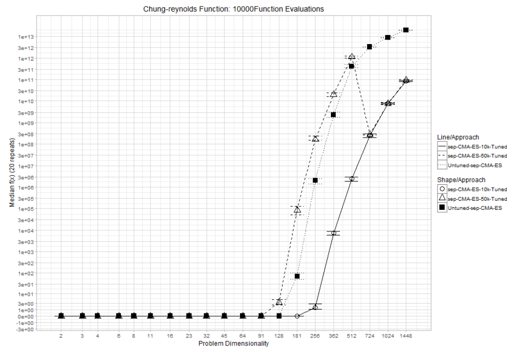
Bent Cigar Function at 50,000 Evaluations



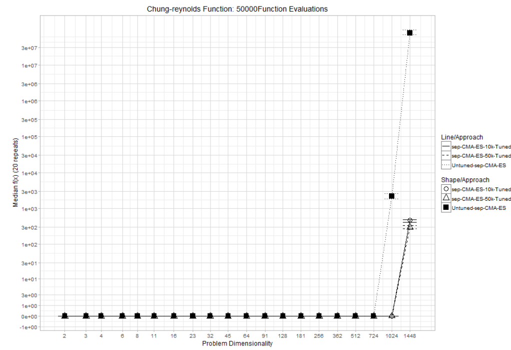
Brown Function at 10,000 Evaluations



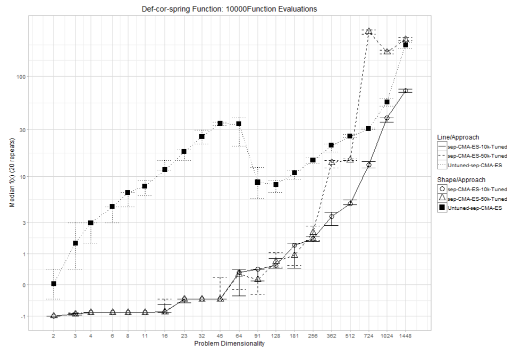
Brown Function at 50,000 Evaluations



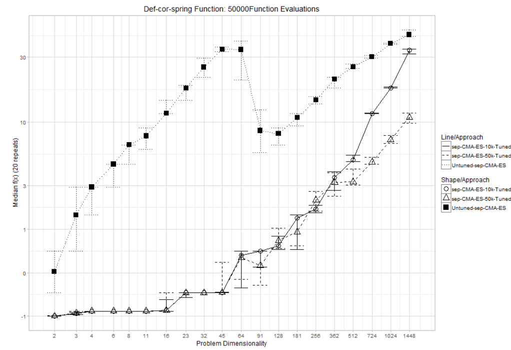
Chung-Reynolds Function at 10,000 Evaluations



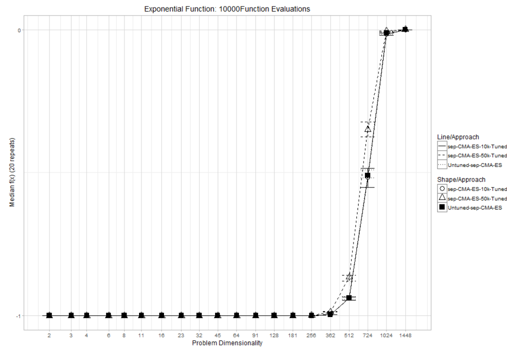
Chung-Reynolds Function at 50,000 Evaluations



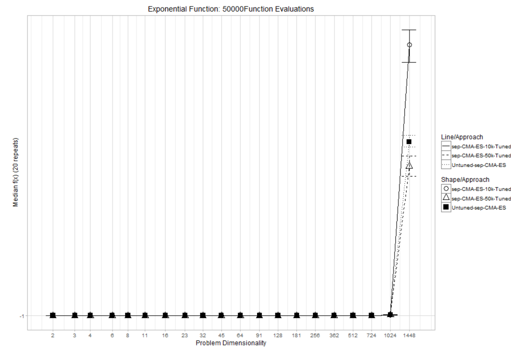
Deflected Corrugated Spring Function at 10,000 Evaluations



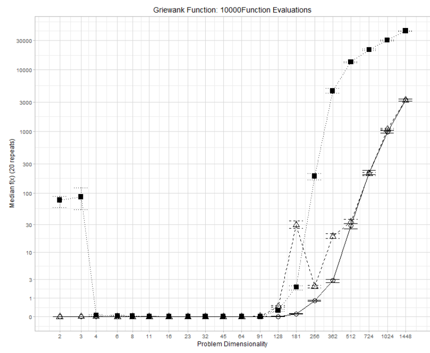
Deflected Corrugated Spring Function at 50,000 Evaluations



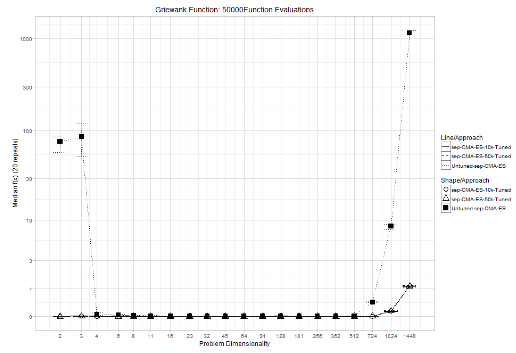
Exponential Function at 10,000 Evaluations



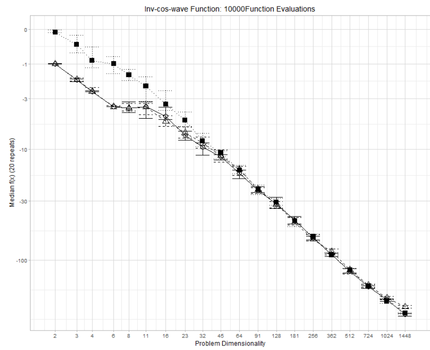
Exponential Function at 50,000 Evaluations



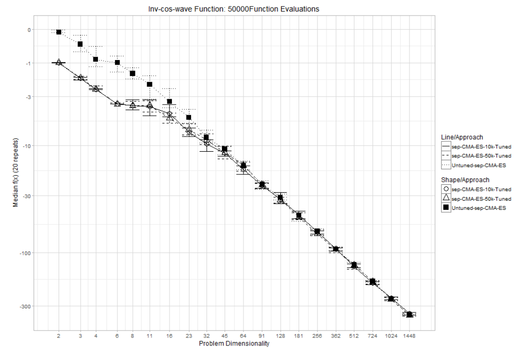
Griewank Function at 10,000 Evaluations



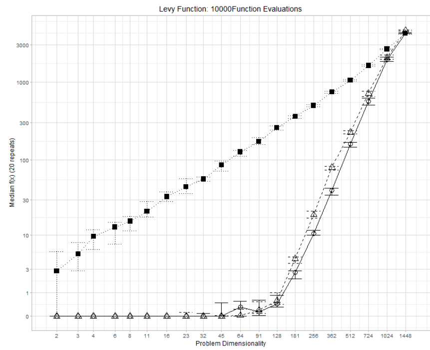
Griewank Function at 50,000 Evaluations



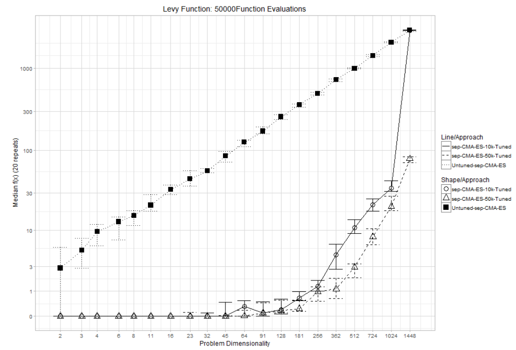
Inverted Cosine Wave Function at 10,000 Evaluations



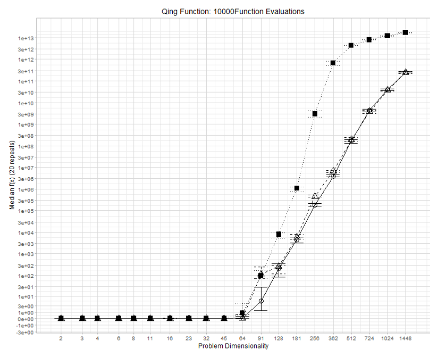
Inverted Cosine Wave Function at 50,000 Evaluations



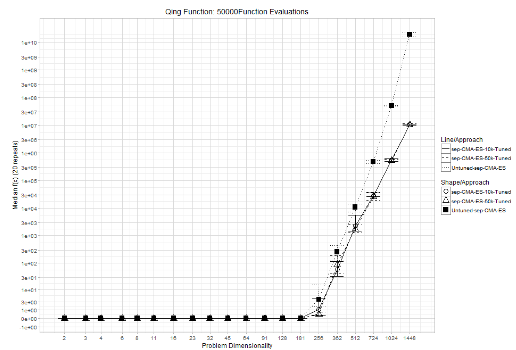
Levy Function at 10,000 Evaluations



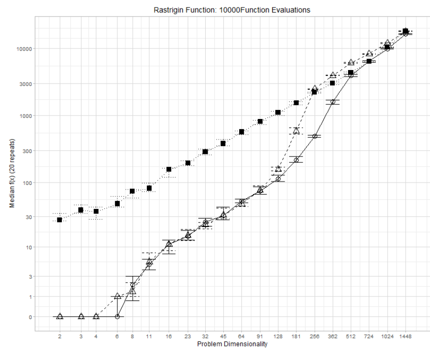
Levy Function at 50,000 Evaluations



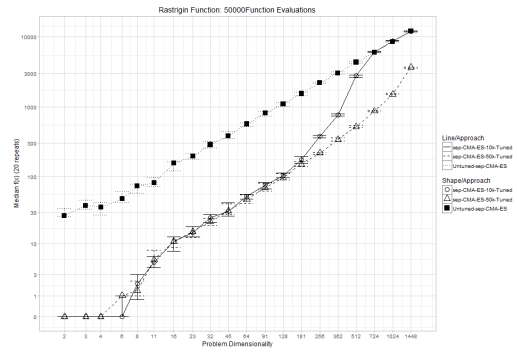
Qing Function at 10,000 Evaluations



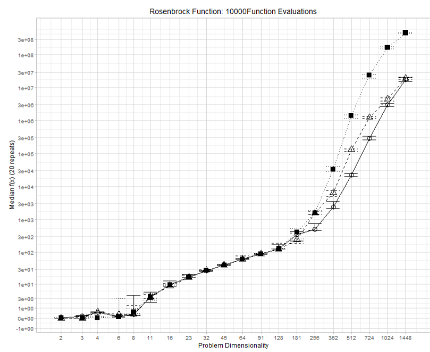
Qing Function at 50,000 Evaluations



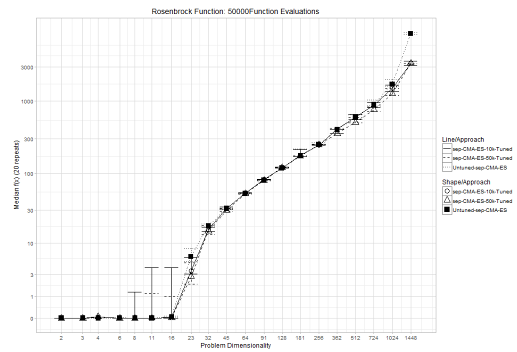
Rastrigin Function at 10,000 Evaluations



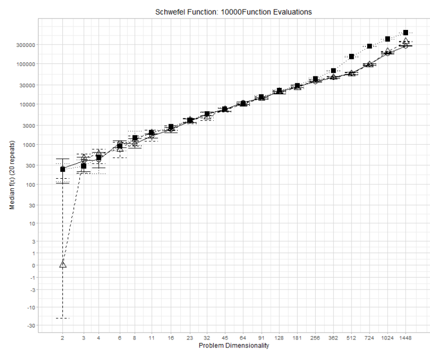
Rastrigin Function at 50,000 Evaluations



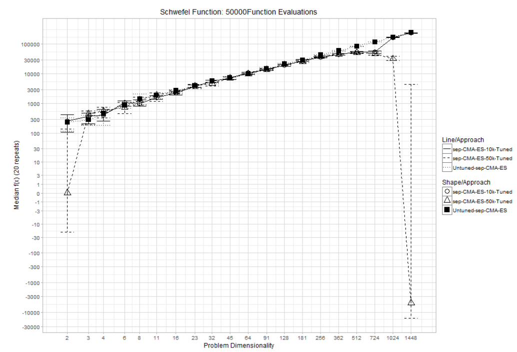
Rosenbrock Function at 10,000 Evaluations



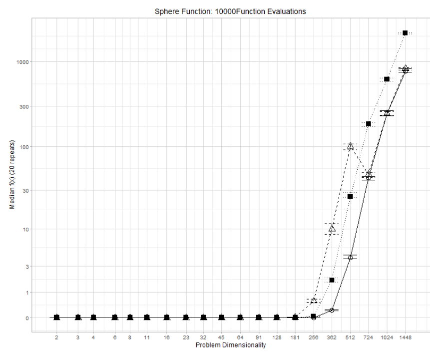
Rosenbrock Function at 50,000 Evaluations



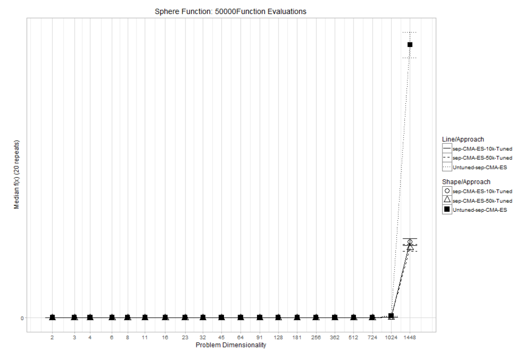
Schwefel Function at 10,000 Evaluations



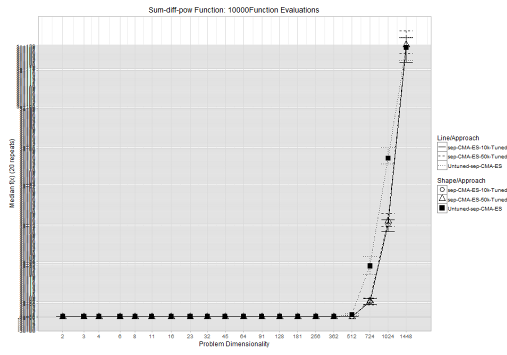
Schwefel Function at 50,000 Evaluations



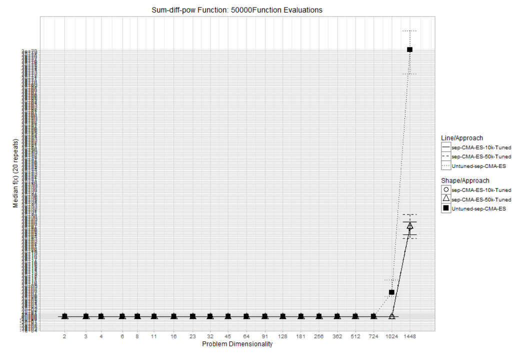
Sphere Function at 10,000 Evaluations



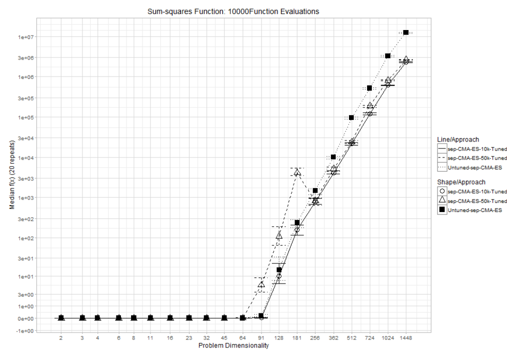
Sphere Function at 50,000 Evaluations



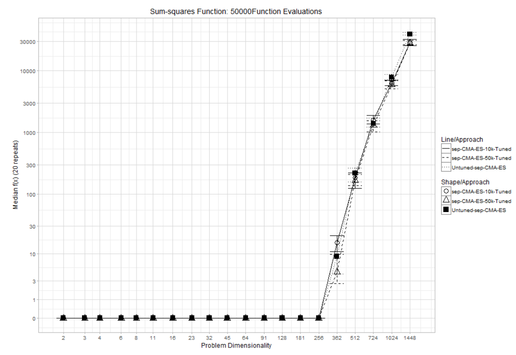
Sum of Different Powers Function at 10,000 Evaluations



Sum of Different Powers Function at 50,000 Evaluations

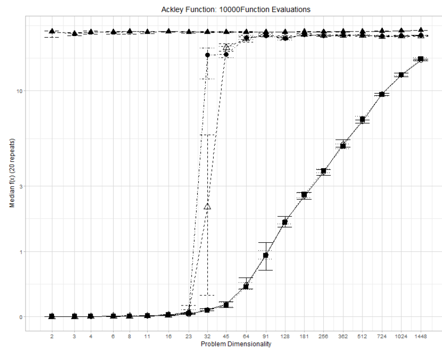


Sum Squares Function at 10,000 Evaluations

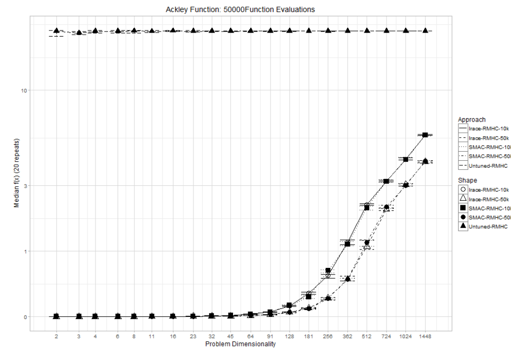


Sum Squares Function at 50,000 Evaluations

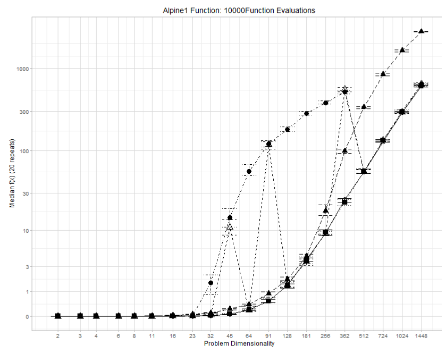
Random Mutation Hill Climbing (RMHC)



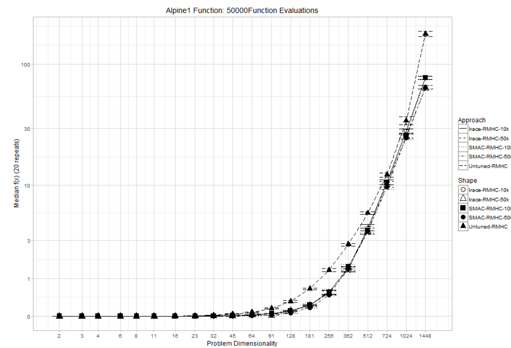
Ackley Function at 10,000 Evaluations



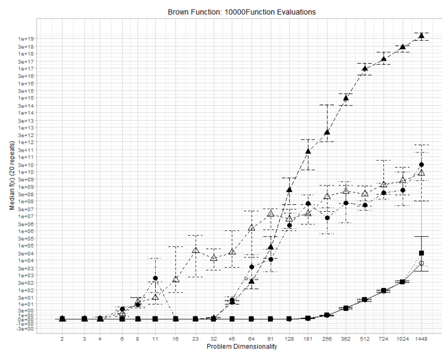
Ackley Function at 50,000 Evaluations



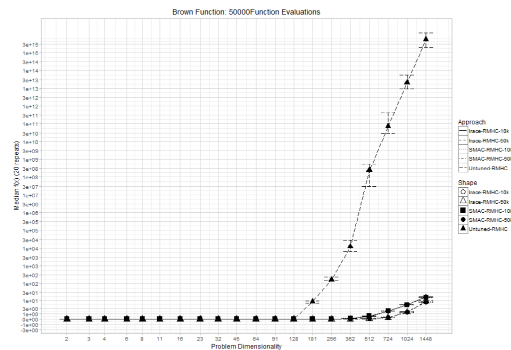
Alpine no.1 Function at 10,000 Evaluations



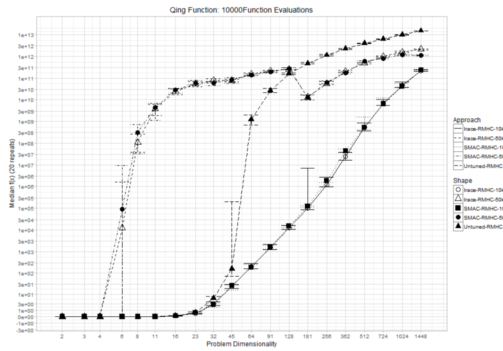
Alpine no.1 Function at 50,000 Evaluations



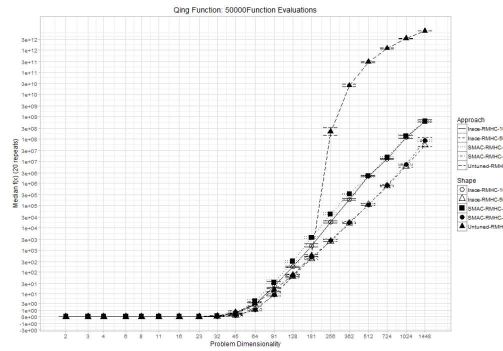
Brown Function at 10,000 Evaluations



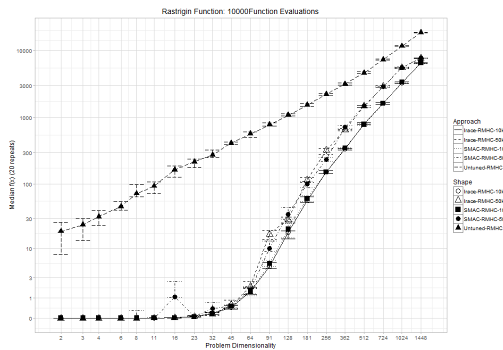
Brown Function at 50,000 Evaluations



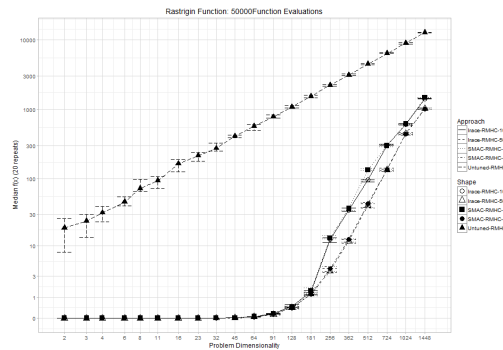
Qing Function at 10,000 Evaluations



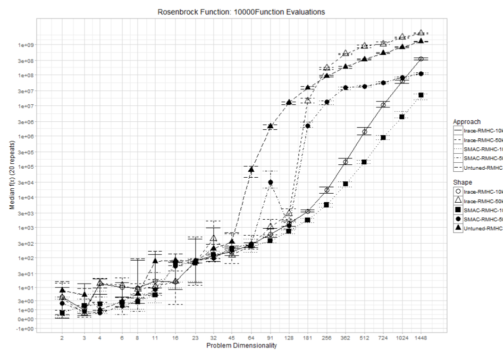
Qing Function at 50,000 Evaluations



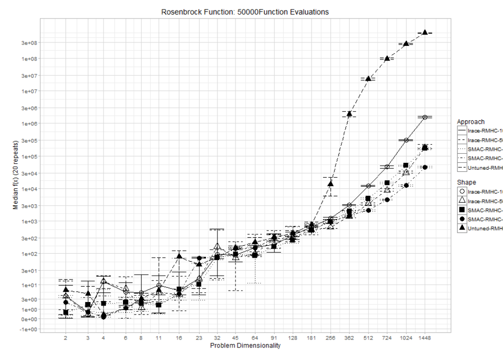
Rastrigin Function at 10,000 Evaluations



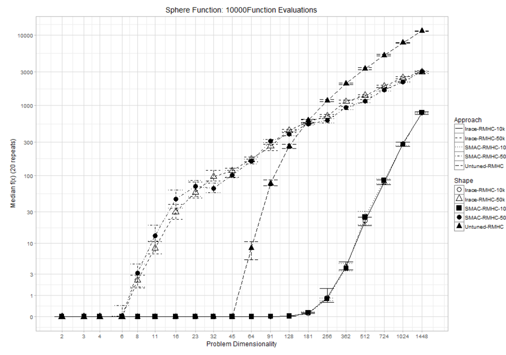
Rastrigin Function at 50,000 Evaluations



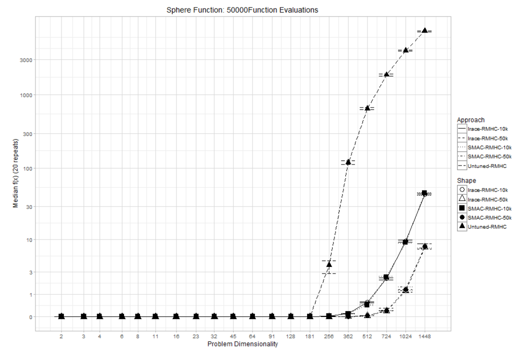
Rosenbrock Function at 10,000 Evaluations



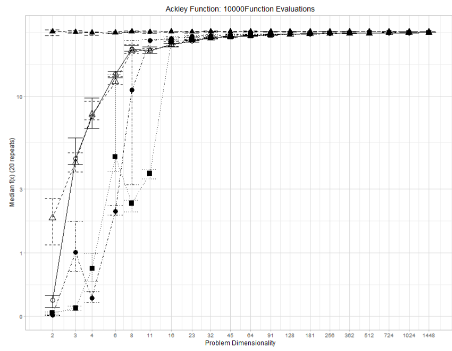
Rosenbrock Function at 50,000 Evaluations



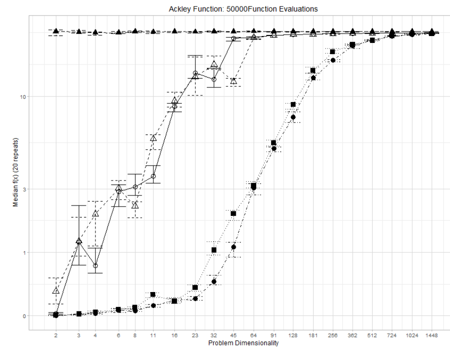
Sphere Function at 10,000 Evaluations



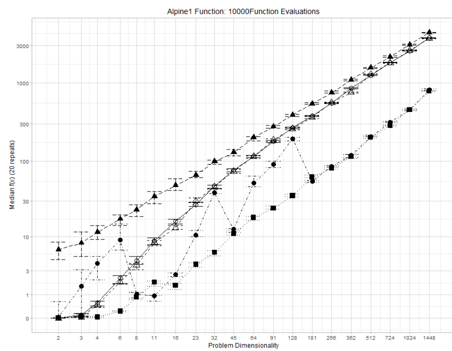
Sphere Function at 50,000 Evaluations



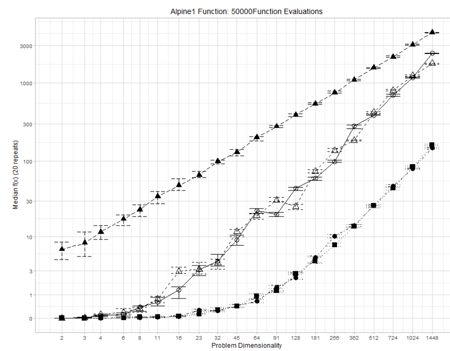
Ackley Function at 10,000 Evaluations



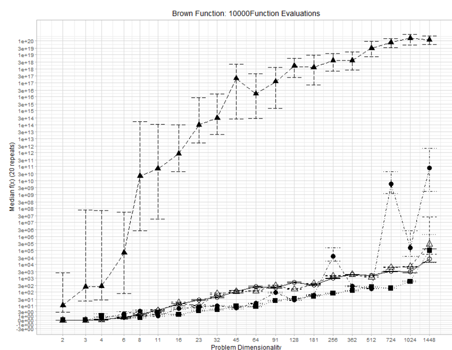
Ackley Function at 50,000 Evaluations



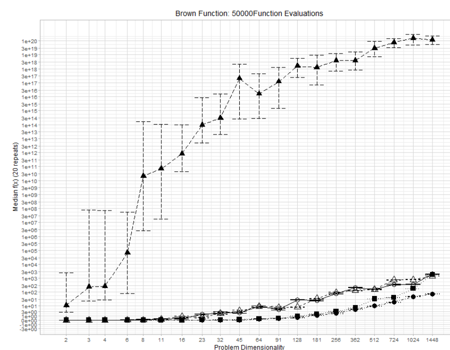
Alpine no.1 Function at 10,000 Evaluations



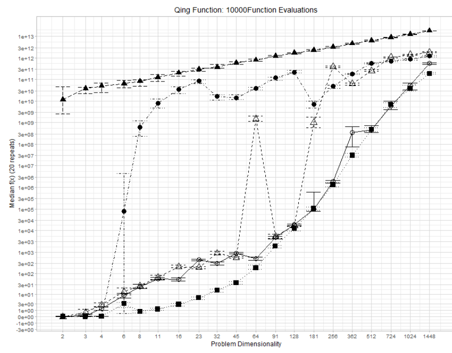
Alpine no.1 Function at 50,000 Evaluations



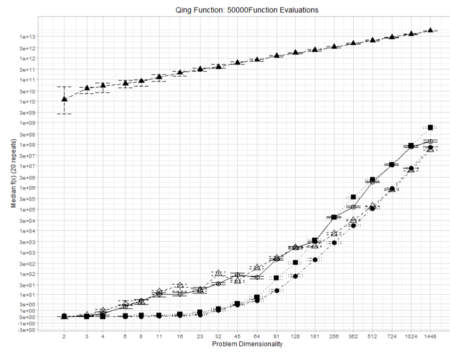
Brown Function at 10,000 Evaluations



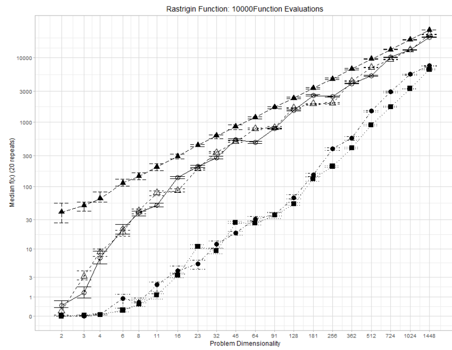
Brown Function at 50,000 Evaluations



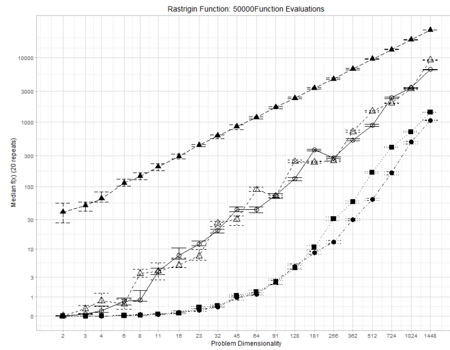
Qing Function at 10,000 Evaluations



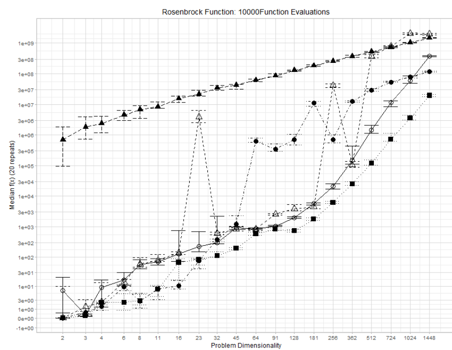
Qing Function at 50,000 Evaluations



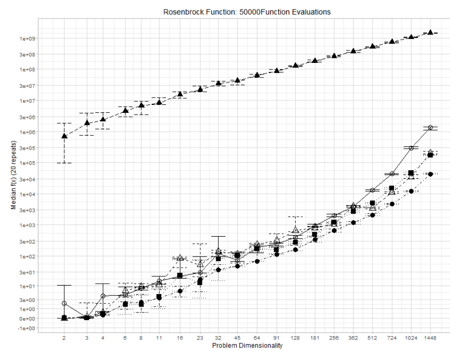
Rastrigin Function at 10,000 Evaluations



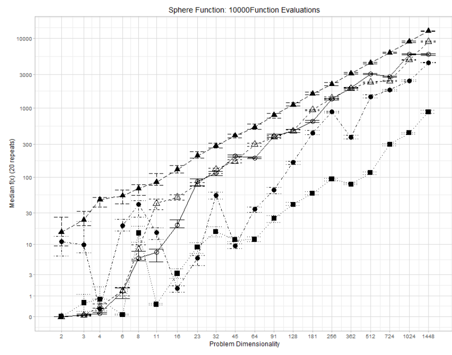
Rastrigin Function at 50,000 Evaluations



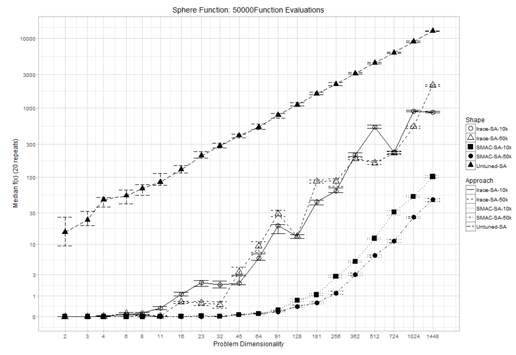
Rosenbrock Function at 10,000 Evaluations



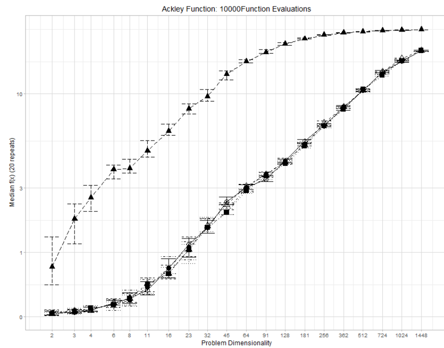
Rosenbrock Function at 50,000 Evaluations



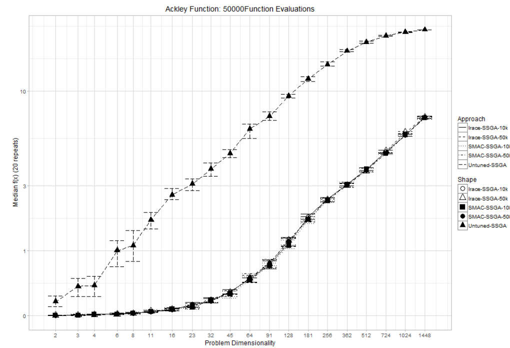
Sphere Function at 10,000 Evaluations



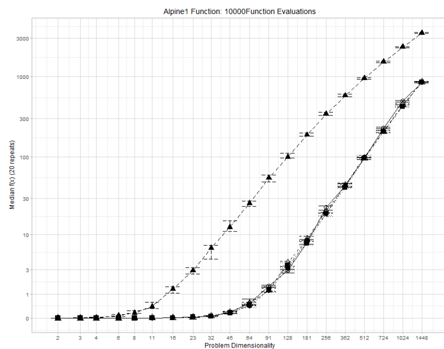
Sphere Function at 50,000 Evaluations



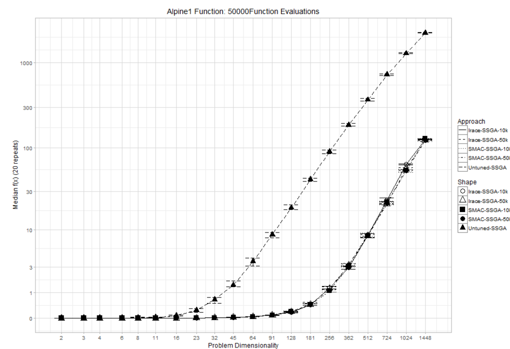
Ackley Function at 10,000 Evaluations



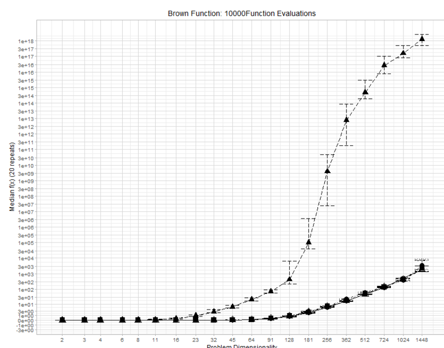
Ackley Function at 50,000 Evaluations



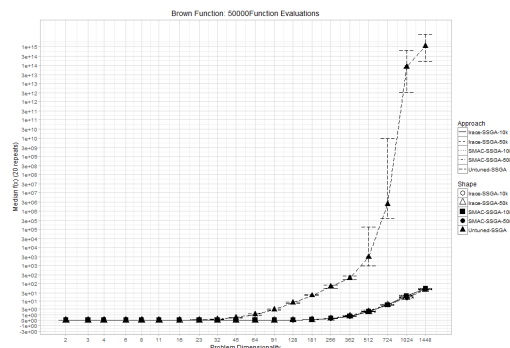
Alpine no.1 Function at 10,000 Evaluations



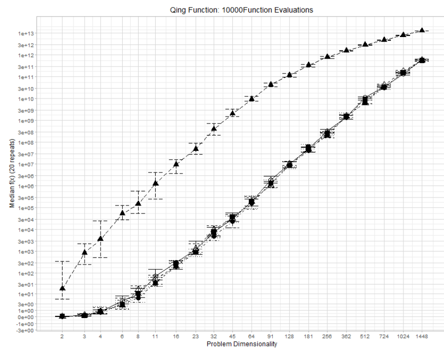
Alpine no.1 Function at 50,000 Evaluations



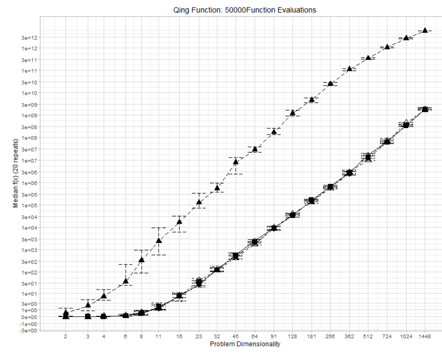
Brown Function at 10,000 Evaluations



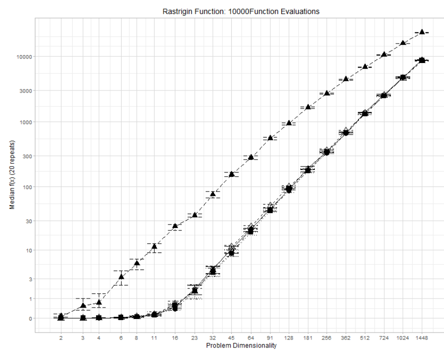
Brown Function at 50,000 Evaluations



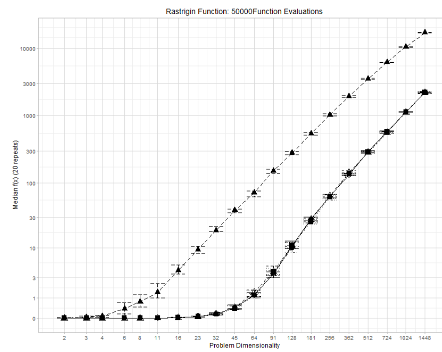
Qing Function at 10,000 Evaluations



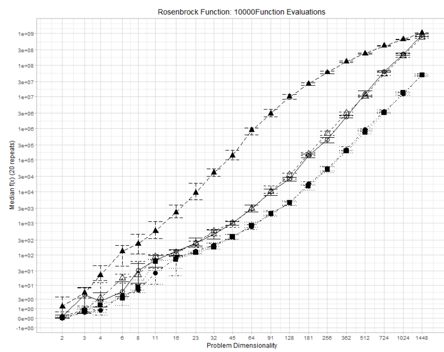
Qing Function at 50,000 Evaluations



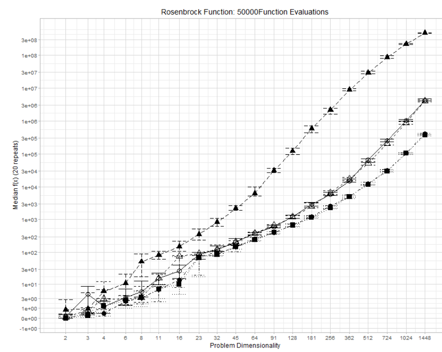
Rastrigin Function at 10,000 Evaluations



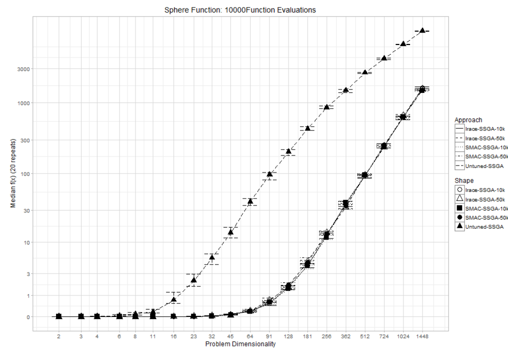
Rastrigin Function at 50,000 Evaluations



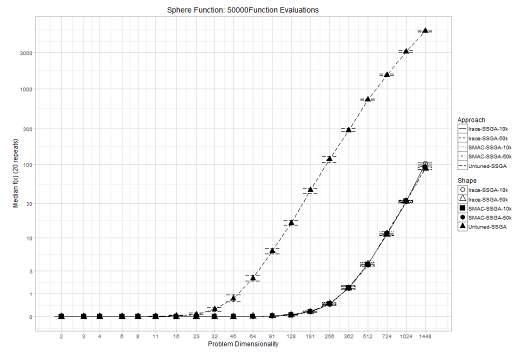
Rosenbrock Function at 10,000 Evaluations



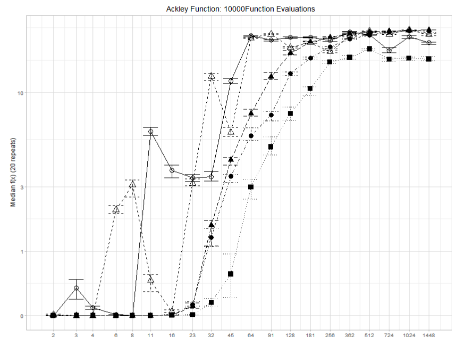
Rosenbrock Function at 50,000 Evaluations



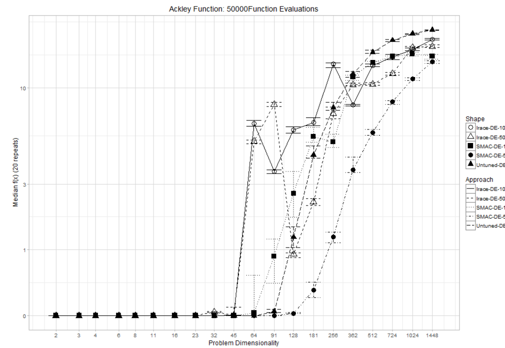
Sphere Function at 10,000 Evaluations



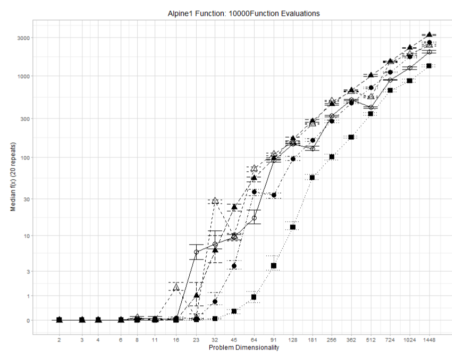
Sphere Function at 50,000 Evaluations



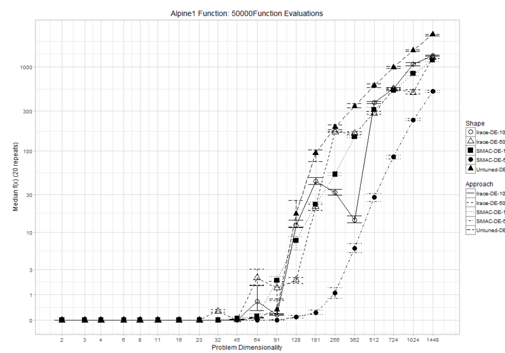
Ackley Function at 10,000 Evaluations



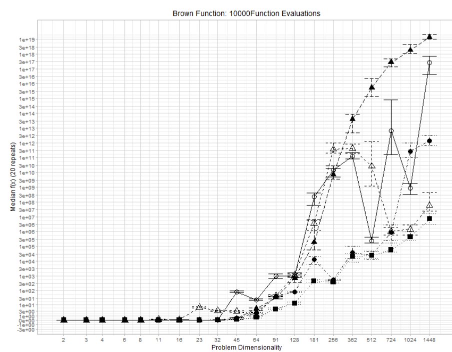
Ackley Function at 50,000 Evaluations



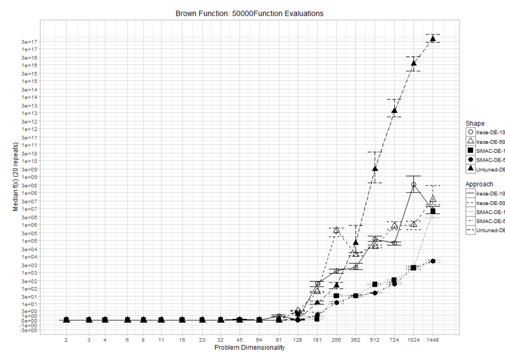
Alpine no.1 Function at 10,000 Evaluations



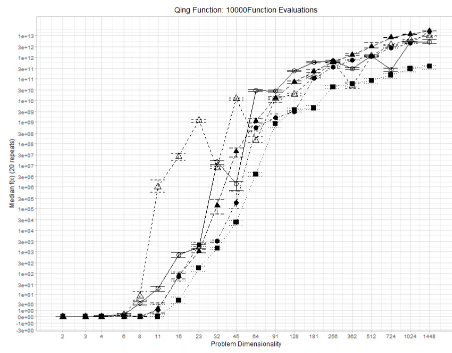
Alpine no.1 Function at 50,000 Evaluations



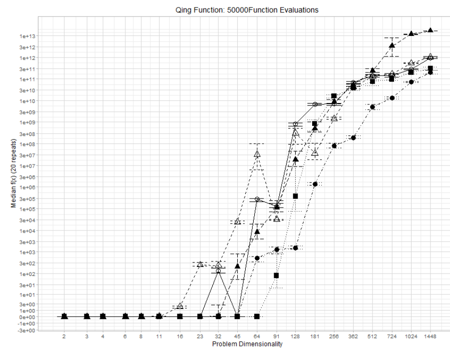
Brown Function at 10,000 Evaluations



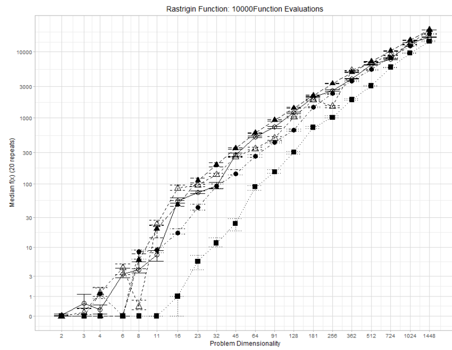
Brown Function at 50,000 Evaluations



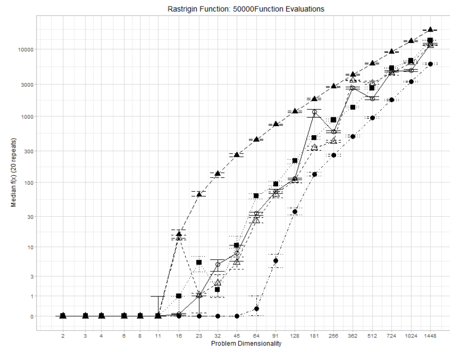
Qing Function at 10,000 Evaluations



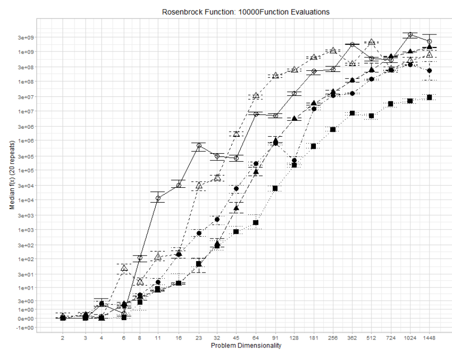
Qing Function at 50,000 Evaluations



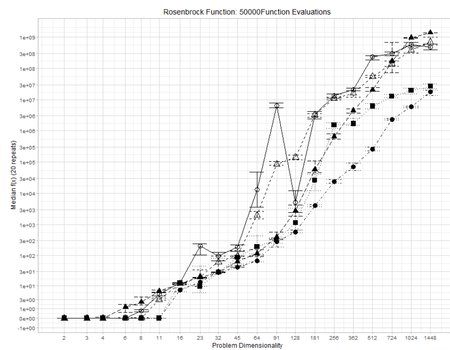
Rastrigin Function at 10,000 Evaluations



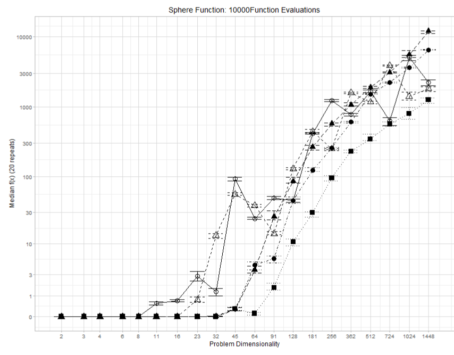
Rastrigin Function at 50,000 Evaluations



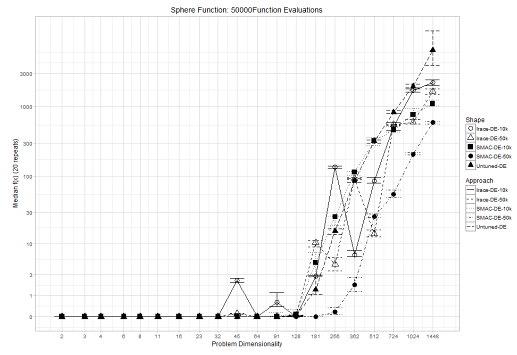
Rosenbrock Function at 10,000 Evaluations



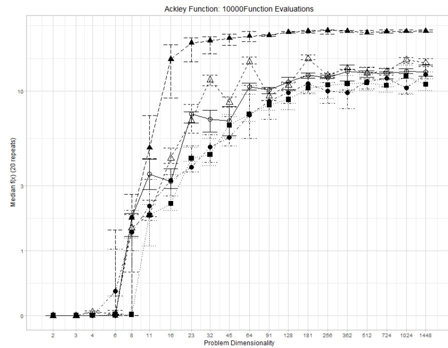
Rosenbrock Function at 50,000 Evaluations



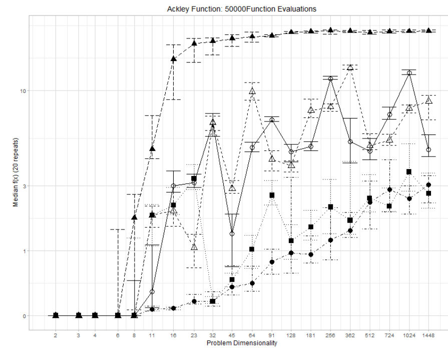
Sphere Function at 10,000 Evaluations



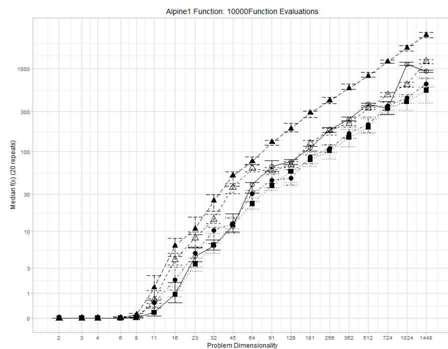
Sphere Function at 50,000 Evaluations



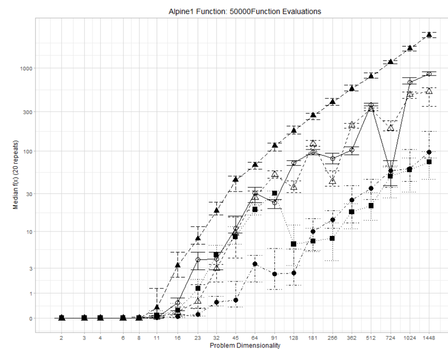
Ackley Function at 10,000 Evaluations



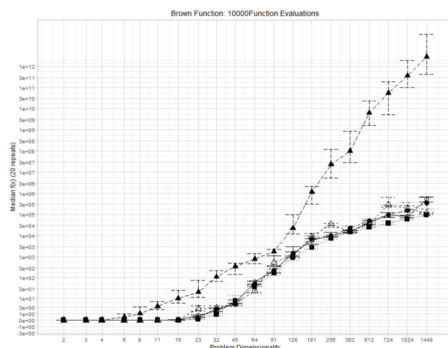
Ackley Function at 50,000 Evaluations



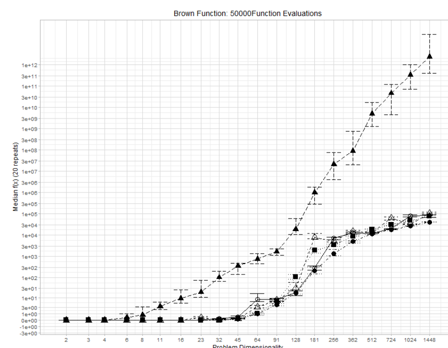
Alpine no.1 Function at 10,000 Evaluations



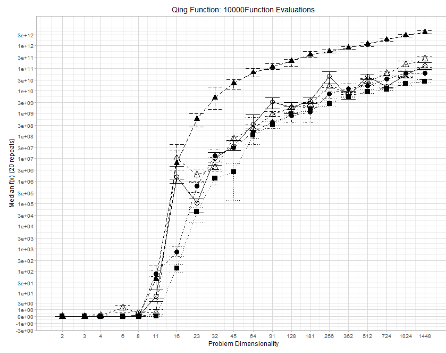
Alpine no.1 Function at 50,000 Evaluations



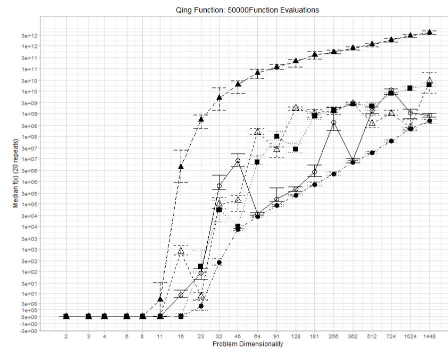
Brown Function at 10,000 Evaluations



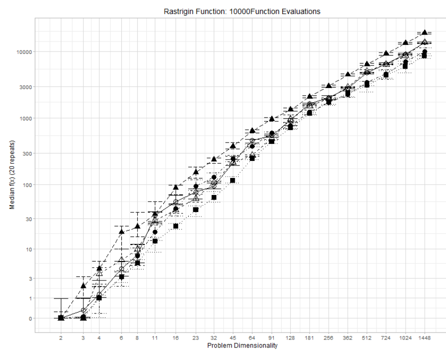
Brown Function at 50,000 Evaluations



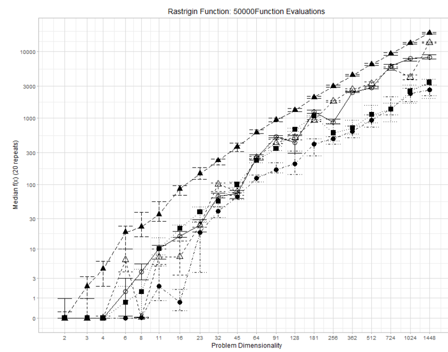
Qing Function at 10,000 Evaluations



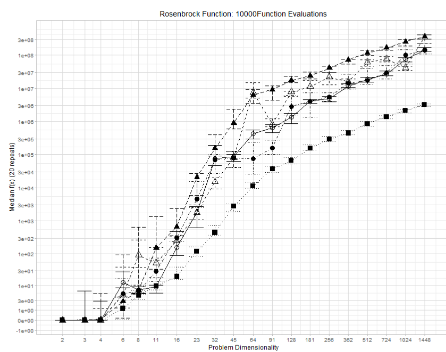
Qing Function at 50,000 Evaluations



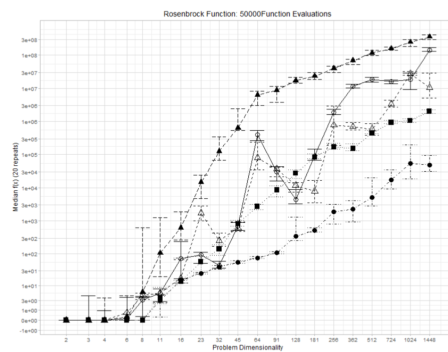
Rastrigin Function at 10,000 Evaluations



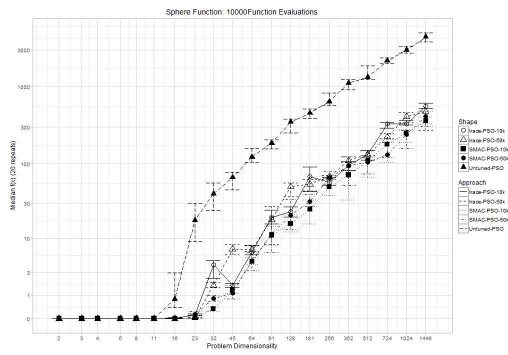
Rastrigin Function at 50,000 Evaluations



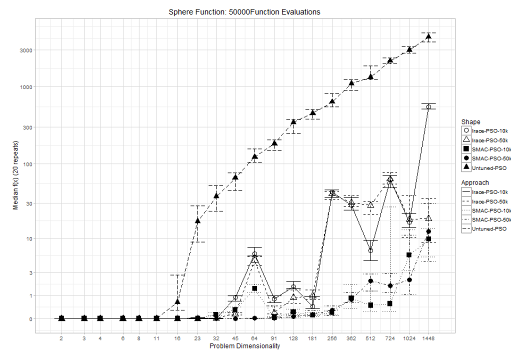
Rosenbrock Function at 10,000 Evaluations



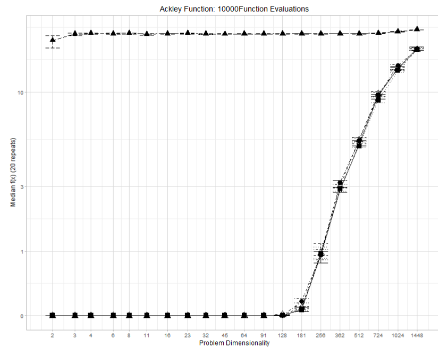
Rosenbrock Function at 50,000 Evaluations



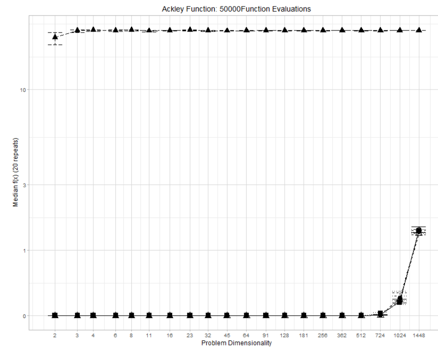
Sphere Function at 10,000 Evaluations



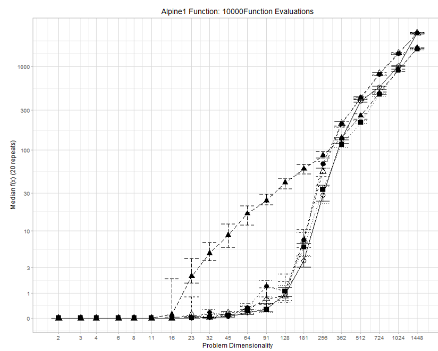
Sphere Function at 50,000 Evaluations



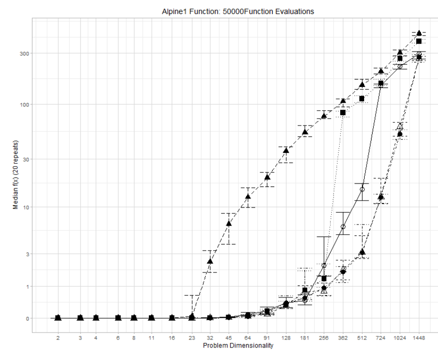
Ackley Function at 10,000 Evaluations



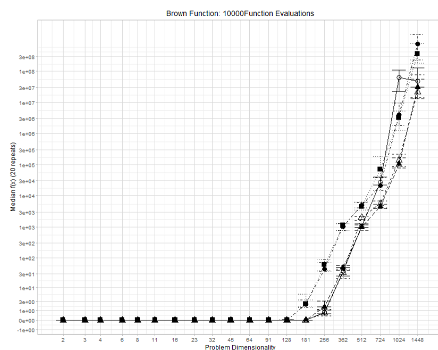
Ackley Function at 50,000 Evaluations



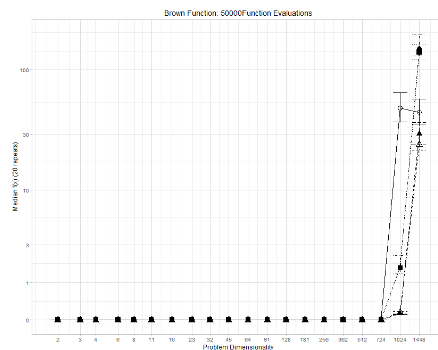
Alpine no.1 Function at 10,000 Evaluations



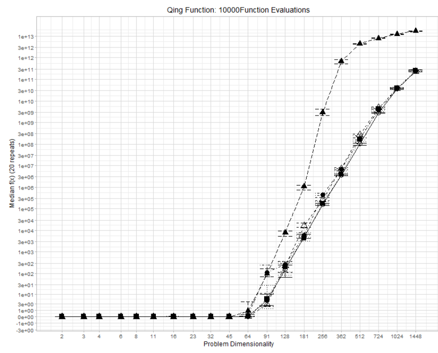
Alpine no.1 Function at 50,000 Evaluations



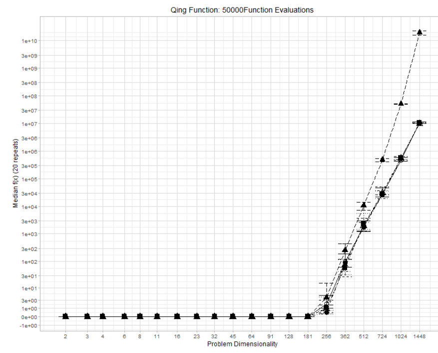
Brown Function at 10,000 Evaluations



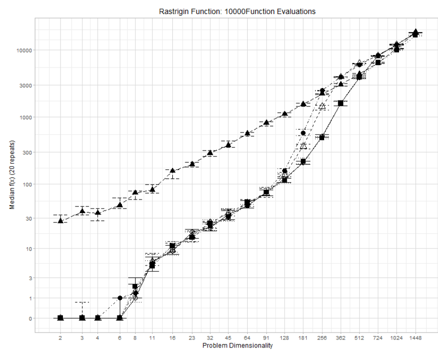
Brown Function at 50,000 Evaluations



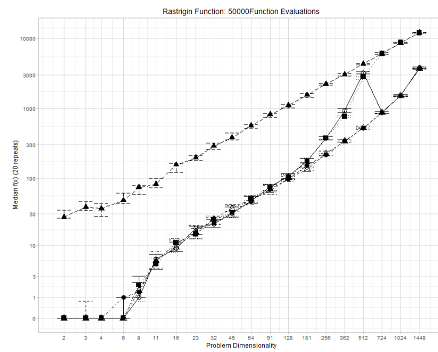
Qing Function at 10,000 Evaluations



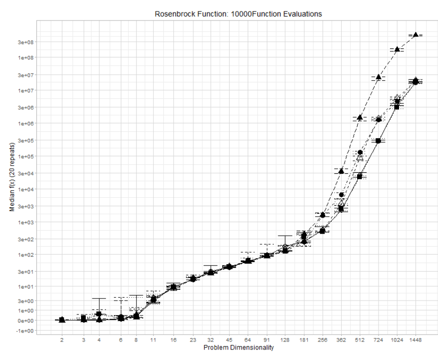
Qing Function at 50,000 Evaluations



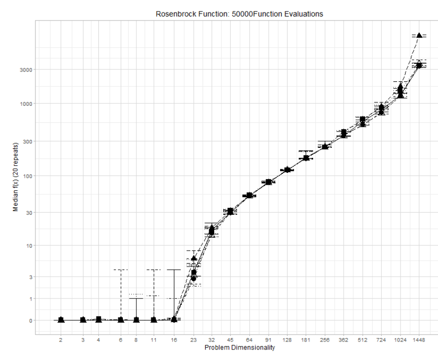
Rastrigin Function at 10,000 Evaluations



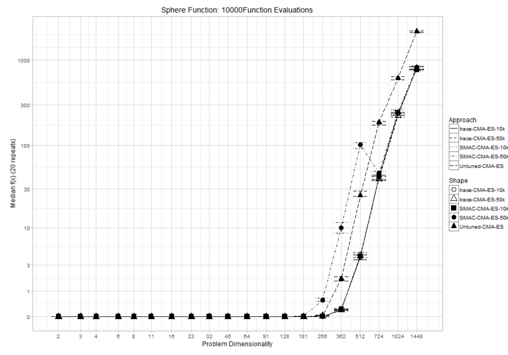
Rastrigin Function at 50,000 Evaluations



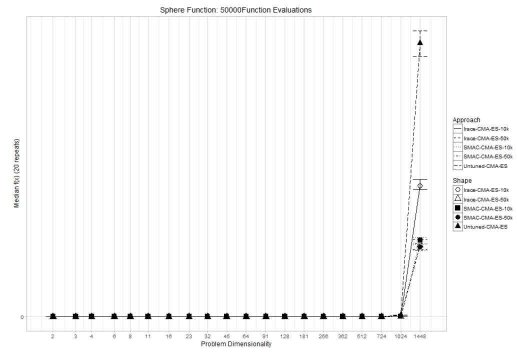
Rosenbrock Function at 10,000 Evaluations



Rosenbrock Function at 50,000 Evaluations



Sphere Function at 10,000 Evaluations



Sphere Function at 50,000 Evaluations

E

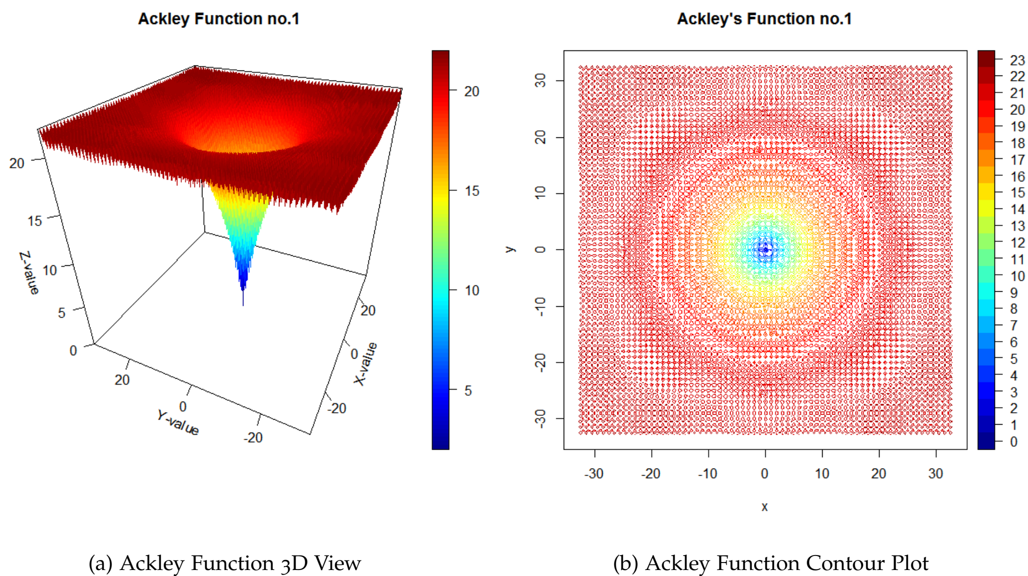
APPENDIX E: DEFINITIONS OF BENCHMARK FUNCTIONS IN THE TEST SUITE

E.1 ACKLEY FUNCTION NO.1

$$f(x) = -20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + \exp(1) \tag{E.1}$$

Subject to domain, $-32.768 \leqslant x_i \leqslant 32.768$ for all $i\{1,2,...,n\}$. The global minima is found at the origin $x_i^* = 0$ with value $f(x^*) = 0$.

Figure E.1a and E.1b shows a 3D and contour plot representing Ackley function no.1 in 2-dimensional space.



(a) Ackley Function 3D View

(b) Ackley Function Contour Plot

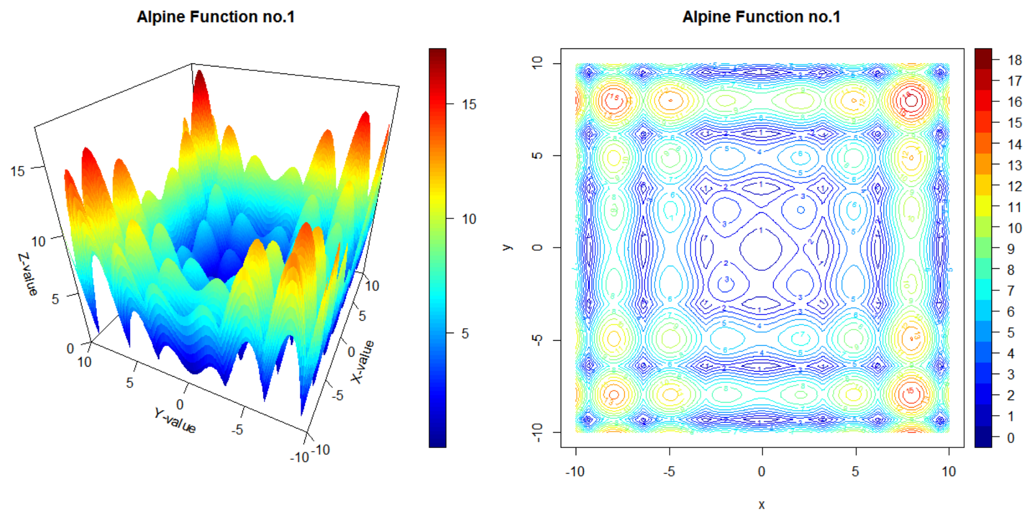
Figure E.1: Ackley Function no.1 in its 2-dimensional form

E.2 ALPINE FUNCTION NO.1

$$f(x) = \sum_{i=1}^{n} |x_i \sin(x_i) + 0.1x_i| \tag{E.2}$$

Subject to domain, $-10 \leqslant x_i \leqslant 10$ for all $i\{1, 2, ..., n\}$. The global minima is found at the origin $x_i^* = 0$ with value 0.

Figure E.2a and E.2b shows a 3D and contour plot representing Alpine function no.1 in 2-dimensional space.



(a) Alpine Function no.1 3D View

(b) Alpine Function no.1 Contour Plot

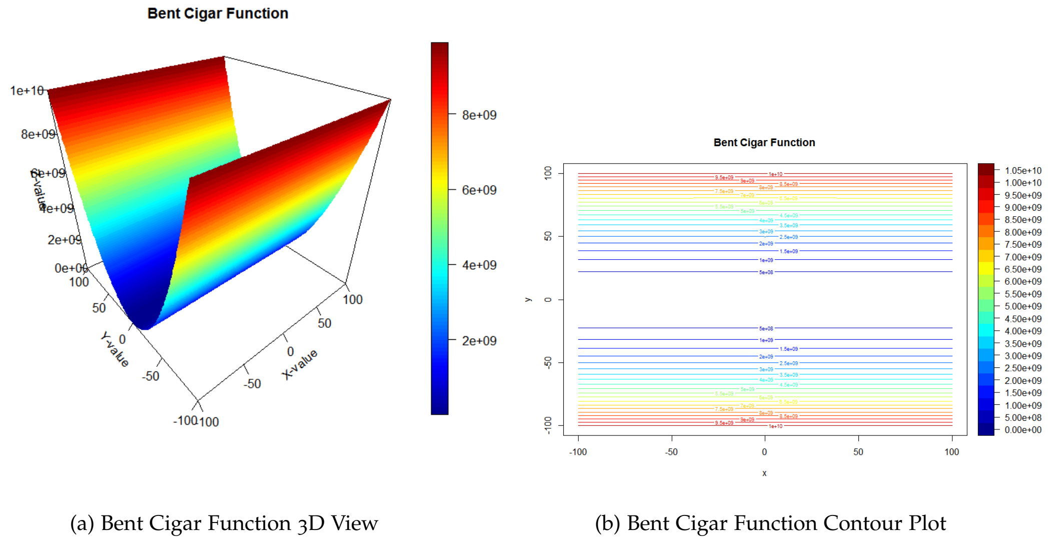
Figure E.2: Alpine Function no.1 in its 2-dimensional form

E.3 BENT CIGAR FUNCTION

$$f(x) = x_1^2 + 10^6 \sum_{i=2}^{n} x_i^2 \tag{E.3}$$

Subject to domain, $-100 \leqslant x_i \leqslant 100$ for all $i\{1, 2, ..., n\}$. The global minima is found at the origin $x_i^* = 0$ with value $f(x^*) = 0$.

Figure E.3a and E.3b shows a 3D and contour plot representing Bent Cigar function in 2-dimensional space.



(a) Bent Cigar Function 3D View

(b) Bent Cigar Function Contour Plot

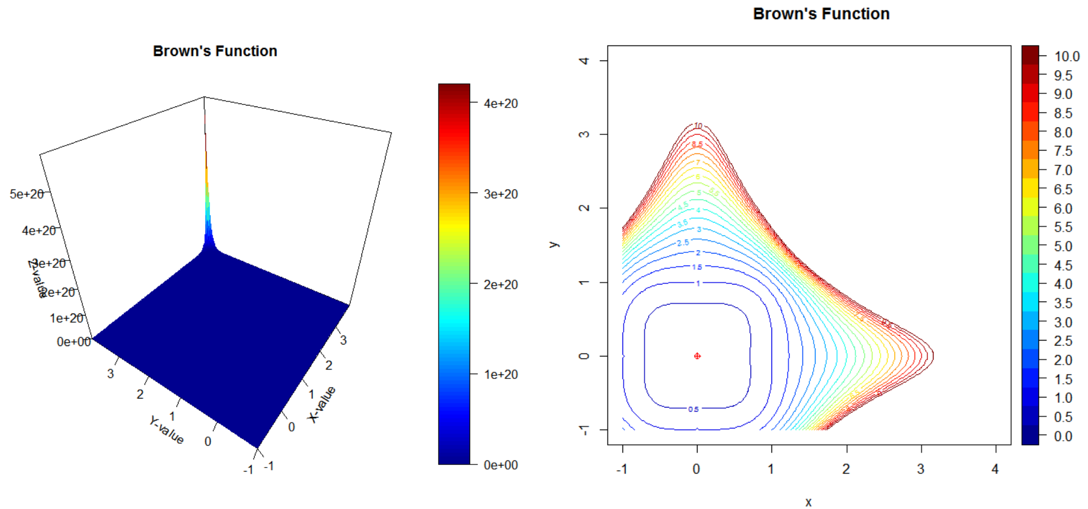
Figure E.3: Bent Cigar Function in its 2-dimensional form

E.4 BROWN FUNCTION

$$f(x) = \sum_{i=1}^{n-1} (x_i^2)^{x_{i+1}^2+1} + (x_{i+1}^2)^{x_i^2+1} \tag{E.4}$$

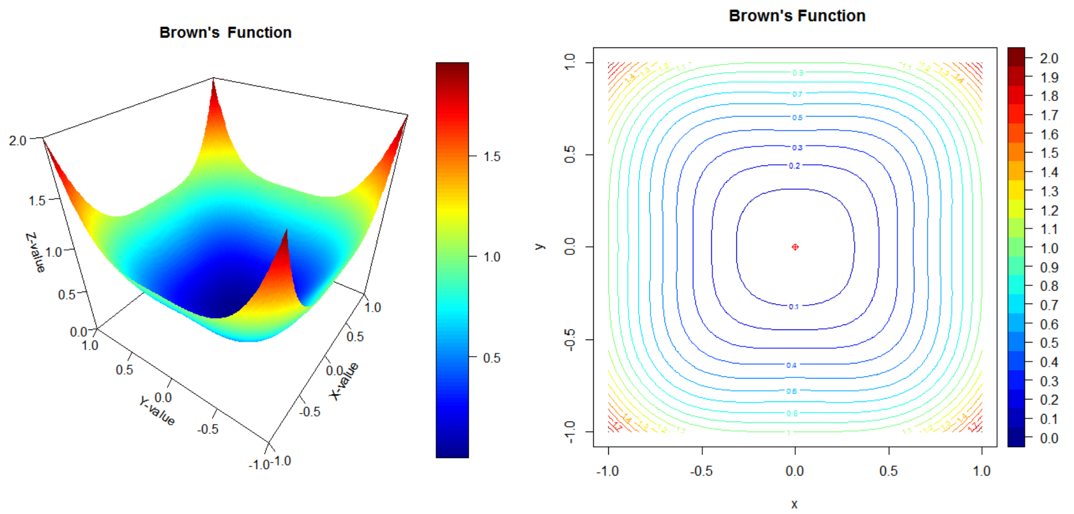
Subject to domain, $-1 \leqslant x_i \leqslant 4$ for all $i\{1, 2, ..., n\}$. The global minima is found at the origin $x_i^* = 0$ with value $f(x^*) = 0$.

Figures E.4a and E.4b shows a 3D and contour plot representing the Brown function in 2-dimensional space, subject to the full sized domain. To clearly show the function structure near the global optimum, Figures E.4c and E.4d represents the function with domain restricted to $-1 \leqslant x_i \leqslant 1$



(a) Brown Function 3D View (Full Domain)

(b) Brown Function Contour Plot (Full Domain)



(c) Brown Function Contour Plot (Limited Domain)

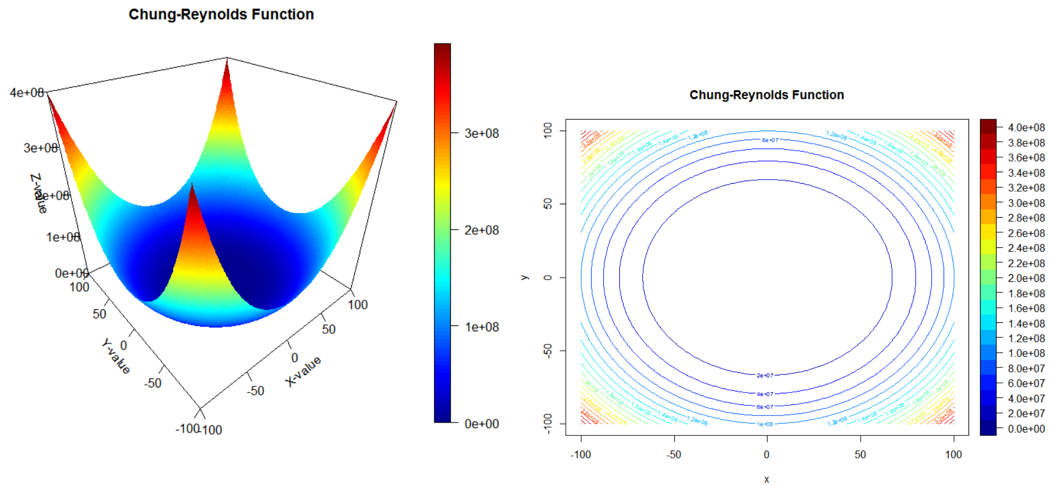
(d) Brown Function Contour Plot (Limited Domain)

Figure E.4: Brown Function in its 2-dimensional form: (a) and (b) shows the function's full domain, and (c) and (d) shows the function with domain limited to $-1 \leqslant x_i \leqslant 1$

E.5 CHUNG-REYNOLDS FUNCTION

$$f(x) = \left(\sum_{i=1}^{n} x_i^2 \right)^2 \qquad \text{(E.5)}$$

Subject to domain, $-100 \leqslant x_i \leqslant 100$ for all $i\{1, 2, ..., n\}$. The global minima is found at the origin $x_i^* = 0$ with value $f(x^*) = 0$.



(a) Chung-Reynolds Function 3D View

(b) Chung-Reynolds Function Contour Plot

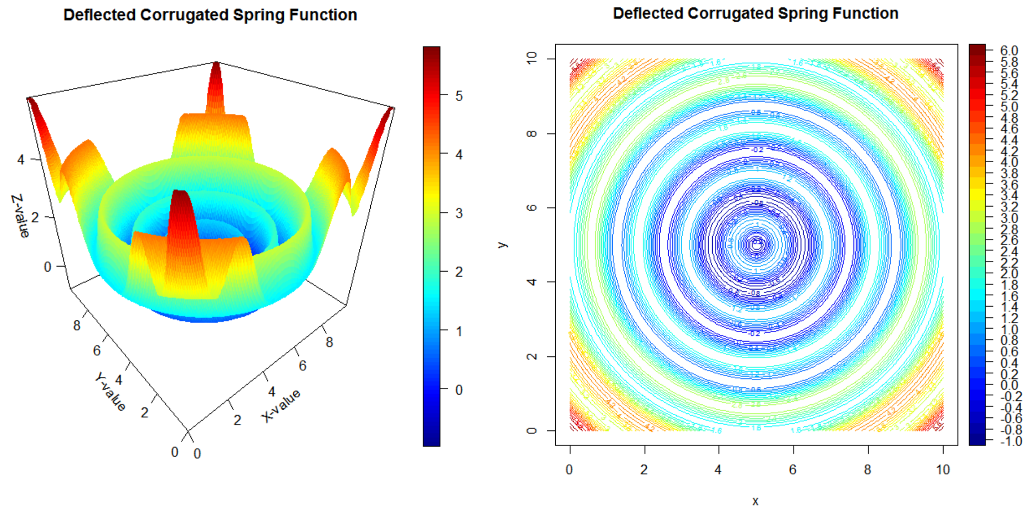
Figure E.5: Chung-Reynolds Function in its 2-dimensional form

E.6 DEFLECTED CORRUGATED SPRING FUNCTION

$$f(x) = 0.1 \sum_{i=1}^{n} \left[(x_i - \alpha)^2 - \cos\left(K \sqrt{\sum_{i=1}^{n} (x_i - \alpha)^2} \right) \right] \tag{E.6}$$

The function is subject to domain, $0 \leqslant x_i \leqslant 2\alpha$ for all $i\{1, 2, ..., n\}$. and the global minima is found at the point $x_i^* = \alpha$ with value $f(x^*) = -1$. The variables α and K are both set equal to 5.

Figures E.6a and E.6b shows a 3D and contour plot representing the Deflected Corrugated Spring function in 2-dimensional space.



(a) Deflected Corrugated Spring Function 3D View (b) Deflected Corrugated Spring Function Contour Plot

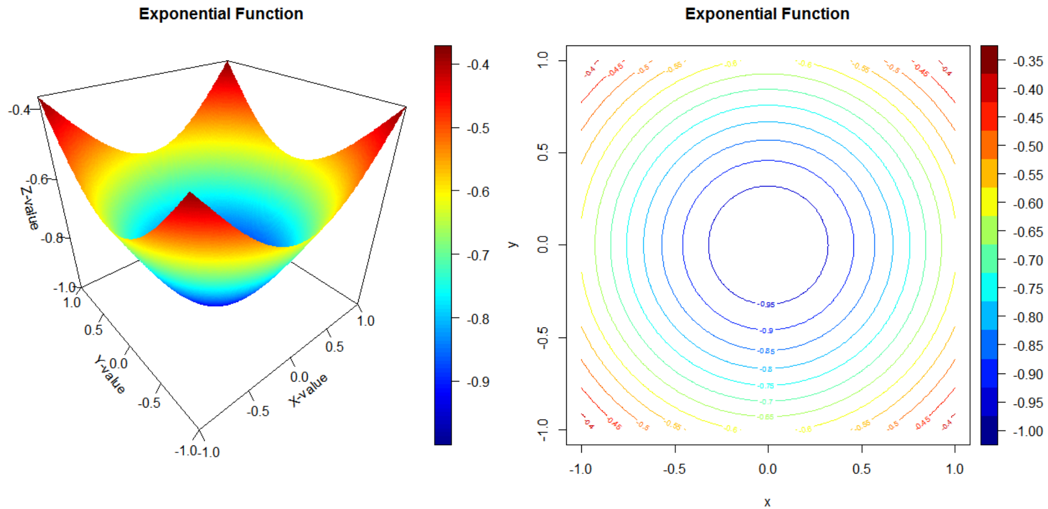
Figure E.6: Deflected Corrugated Spring Function in its 2-dimensional form

E.7 EXPONENTIAL FUNCTION

$$f(x) = -\exp\left(-0.5 \sum_{i=1}^{n} x_i^2\right) \tag{E.7}$$

Subject to domain, $-1 \leqslant x_i \leqslant 1$ for all $i\{1, 2, ..., n\}$. The global minima is found at the origin $x_i^* = 0$ with value $f(x^*) = -1$. This optimum of this function is it's maxima not it's minima - to be used for minimisation purposed the result of the function is negated - i.e., the minima becomes the negative maxima.

Figures E.7a and E.7b shows a 3D and contour plot representing the Exponential function in 2-dimensional space.



(a) Exponential Function 3D View

(b) Exponential Function Contour Plot

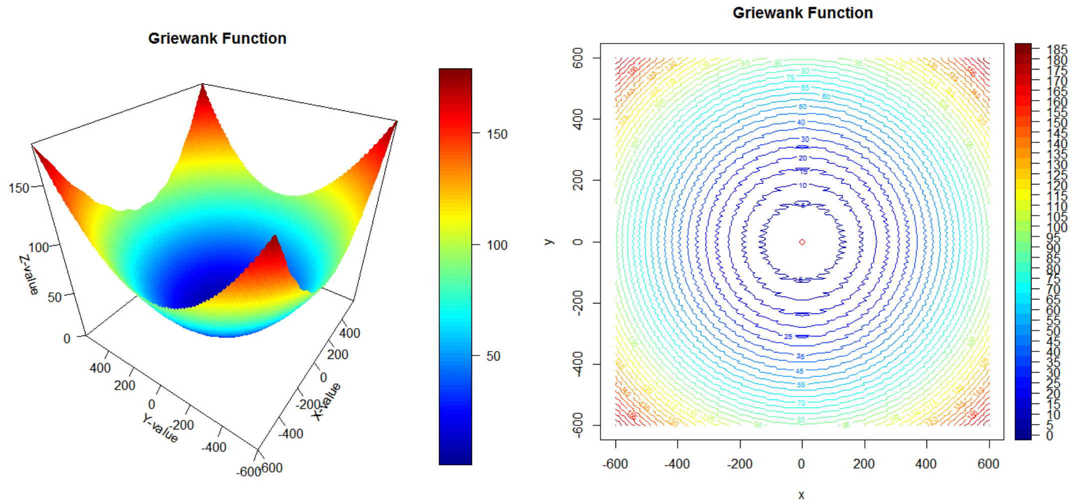
Figure E.7: Exponential Function in its 2-dimensional form

E.8 GRIEWANK FUNCTION

$$f(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{E.8}$$

Subject to domain, $-600 \leqslant x_i \leqslant 600$ for all $i\{1, 2, ..., n\}$. The global minima is found at the origin $x_i^* = 0$ with value $f(x^*) = 0$.

Figures E.8a and E.8b shows a 3D and contour plot representing the Griewank function in 2-dimensional space.



(a) Griewank Function 3D View

(b) Griewank Function Contour Plot

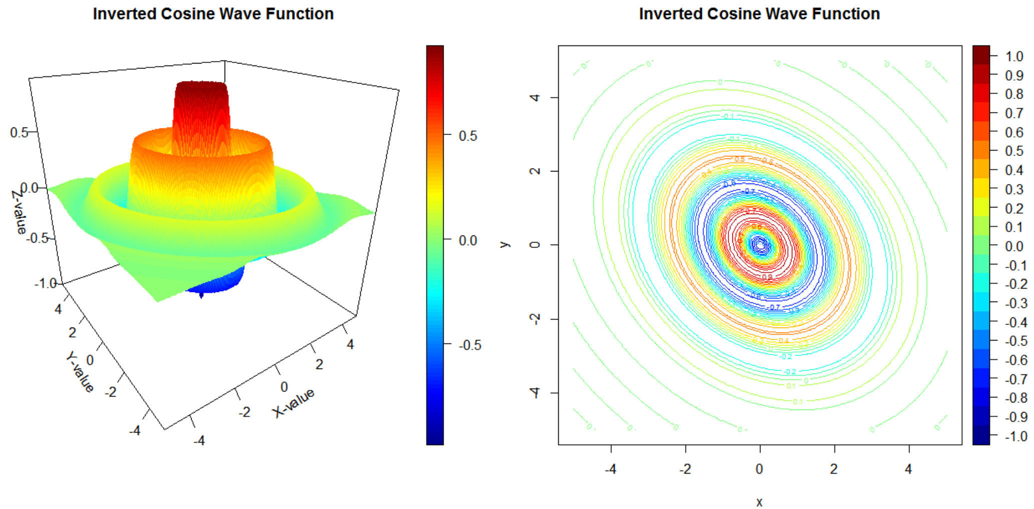
Figure E.8: Griewank Function in its 2-dimensional form

E.9 INVERTED COSINE WAVE FUNCTION

$$f(x) = \sum_{i=1}^{n-1} \left\{ e^{\left[\frac{-\left(x_i^2 + x_{i+1}^2 + 0.5 x_i x_{i+1}\right)}{8} \right]} \cos\left(4 \times \sqrt{x_i^2 + x_{i+1}^2 + 0.5 x_i x_{i+1}} \right) \right\} \tag{E.9}$$

Subject to domain, $-5 \leqslant x_i \leqslant 5$ for all $i\{1, 2, ..., n\}$. The global minima is found at the origin $x_i^* = 0$ with value $f(x^*) = -n + 1$.

Figures E.9a and E.9b shows a 3D and contour plot representing the Inverted Cosine Wave function in 2-dimensional space.



(a) Inverted Cosine Wave Function 3D View

(b) Inverted Cosine Wave Function Contour Plot

Figure E.9: Inverted Cosine Wave Function in its 2-dimensional form

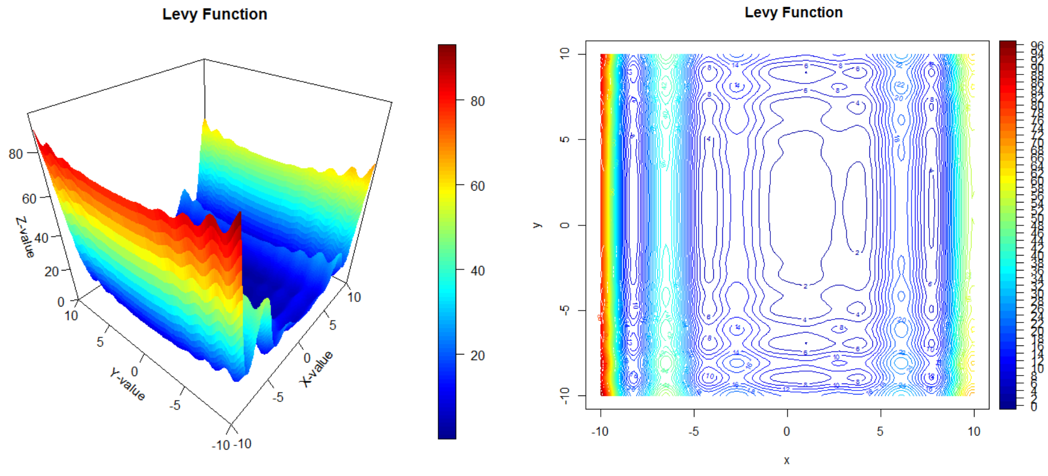
E.10 LEVY FUNCTION

$$f(x) = \sin^2(\pi w_1) + \sum_{i=1}^{n-1}(w_i - 1)^2\left[1 + 10\sin^2(\pi w_i + 1)\right] + (w_n - 1)^2\left[1 + \sin^2(2\pi w_n)\right], \tag{E.10}$$

$$\text{where } w_i = 1 + \frac{x_i - 1}{4} \text{ for all } i\{1, 2, ..., n\}$$

Subject to domain, $-10 \leqslant x_i \leqslant 10$ for all $i\{1, 2, ..., n\}$. The global minima is found at the point $x_i^* = 1$ with value $f(x^*) = 0$.

Figures E.10a and E.10b shows a 3D and contour plot representing the Levy function in 2-dimensional space.



(a) Levy Function 3D View

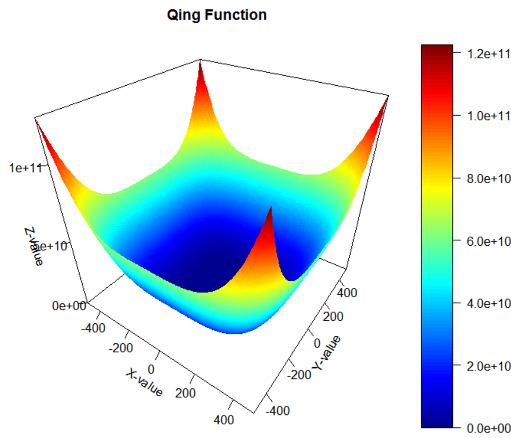
(b) Levy Function Contour Plot

Figure E.10: Levy Function in its 2-dimensional form

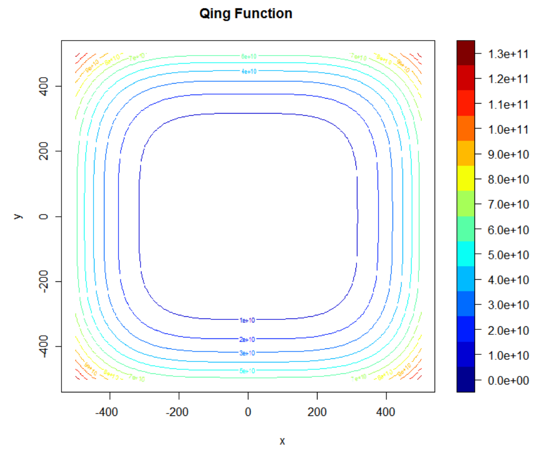
$$f(x) = \sum_{i=1}^{n} \left(x_i^2 - i\right)^2 \tag{E.11}$$

Subject to domain, $-500 \leqslant x_i \leqslant 500$ for all $i\{1, 2, ..., n\}$. There are 2^n global minima found at points $x_i^* = \pm\sqrt{i}, where i = 1...n$, with value $f(x^*) = 0$.

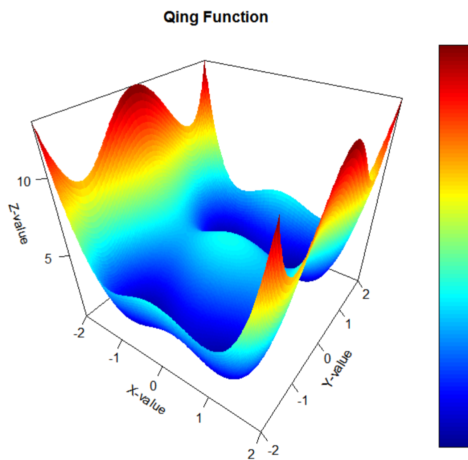
Figures E.11a and E.11b shows a 3D and contour plot representing the Qing function in 2-dimensional space, subject to the full sized domain. To clearly show the function structure near the global optima, Figures E.11c and E.11d represents the function with domain restricted to $-2 \leqslant x_i \leqslant 2$



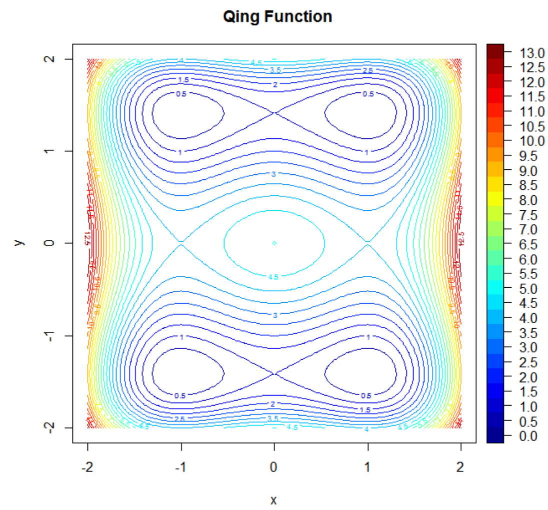
(a) Qing Function 3D View (Full Domain)



(b) Qing Function Contour Plot (Full Domain)



(c) Qing Function Contour Plot (Limited Domain)



(d) Qing Function Contour Plot (Limited Domain)

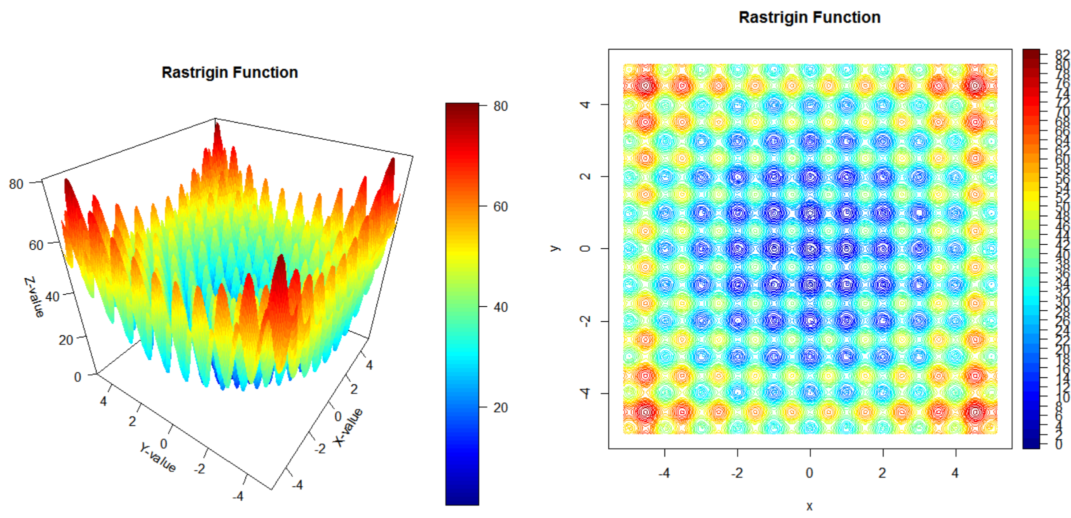
Figure E.11: Qing Function in its 2-dimensional form: (a) and (b) shows the function's full domain, and (c) and (d) shows the function with domain limited to $-2 \leqslant x_i \leqslant 2$

E.12 RASTRIGIN FUNCTION

$$f(x) = 10n \sum_{i=1}^{n} \left[x_i^2 - 10 \cos \left(2\pi x_i \right) \right] \tag{E.12}$$

Subject to domain, $-5.12 \leqslant x_i \leqslant 5.12$ for all $i\{1, 2, ..., n\}$. The global minima is found at the origin $x_i^* = 0$ with value $f(x^*) = 0$.

Figures E.12a and E.12b shows a 3D and contour plot representing the Rastrigin function in 2-dimensional space.



(a) Rastrigin Function 3D View

(b) Rastrigin Function Contour Plot

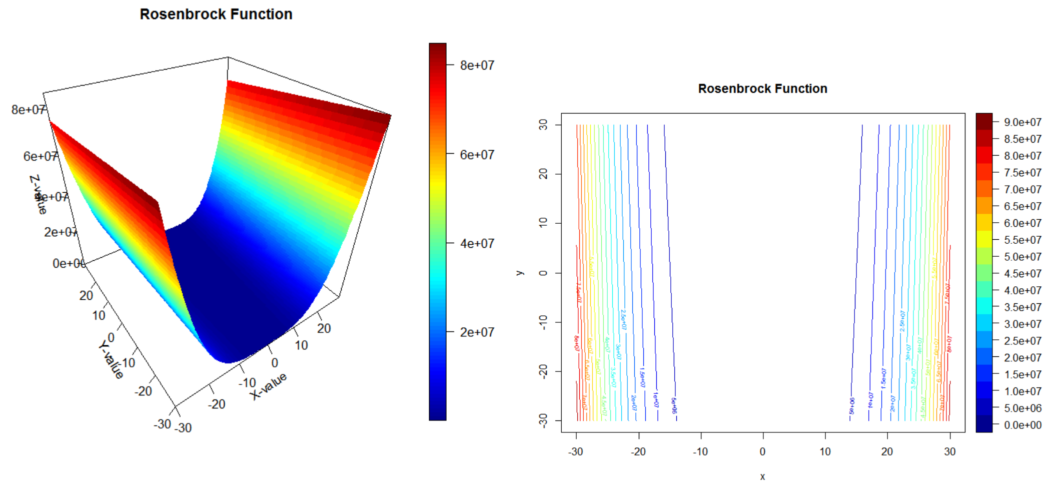
Figure E.12: Rastrigin Function in its 2-dimensional form

E.13 ROSENBROCK FUNCTION

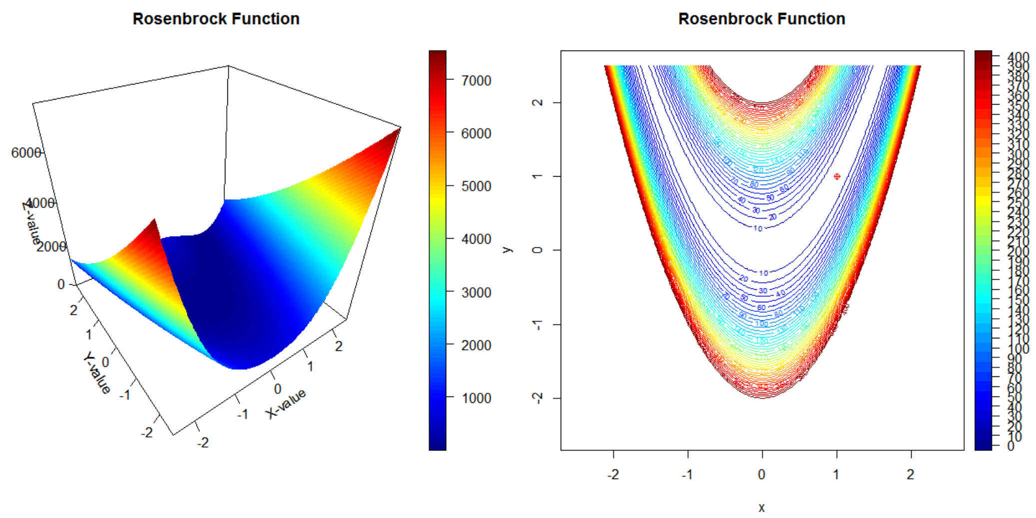
$$f(x) = \sum_{i=1}^{n-1} \left[100 \left(x_{i+1} - x_i^2 \right)^2 + (x_i - 1)^2 \right] \tag{E.13}$$

Subject to domain, $-30 \leqslant x_i \leqslant 30$ for all $i\{1, 2, ..., n\}$. The global minima is found at the point $x_i^* = 1$ with value $f(x^*) = 0$.

Figures E.13a and E.13b shows a 3D and contour plot representing the Rosenbrock function in 2-dimensional space, subject to the full sized domain. To clearly show the function structure near the global optima, Figures E.13c and E.13d represents the function with domain restricted to $-2.5 \leqslant x_i \leqslant 2.5$



(a) Rosenbrock Function 3D View (Full Domain) (b) Rosenbrock Function Contour Plot (Full Domain)



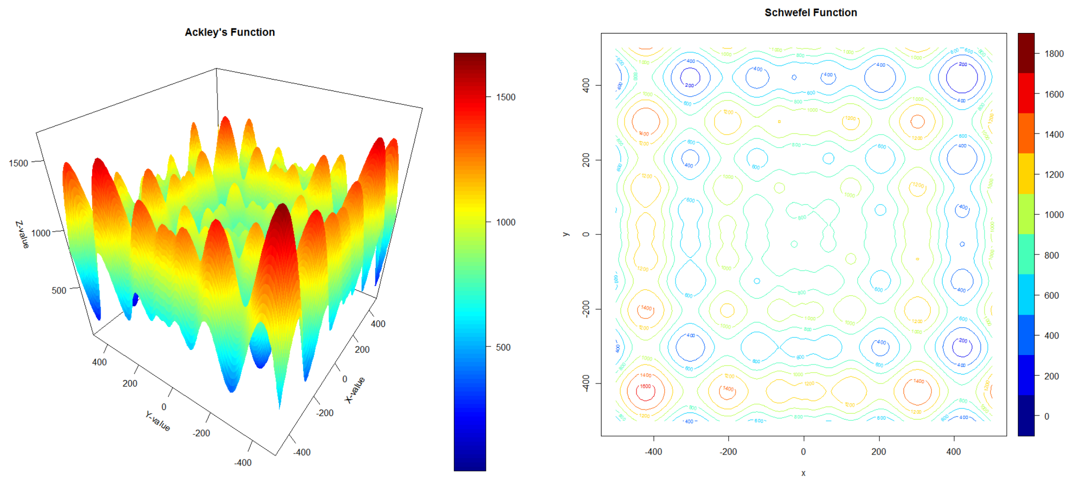
(c) Rosenbrock Function Contour Plot (Limited Domain) (d) Rosenbrock Function Contour Plot (Limited Domain)

Figure E.13: Rosenbrock Function in its 2-dimensional form: (a) and (b) shows the function's full domain, and (c) and (d) shows the function with domain limited to $-2.5 \leqslant x_i \leqslant 2.5$

$$f(x) = 418.9829n - \sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right) \tag{E.14}$$

Subject to domain, $-500 \leqslant x_i \leqslant 500$ for all $i\{1, 2, ..., n\}$. The global minima is found at the point $x_i^* = 420.9687$ with value $f(x^*) = 0$.

Figures E.14a and E.14b shows a 3D and contour plot representing the Schwefel function in 2-dimensional space.



(a) Schwefel Function 3D View

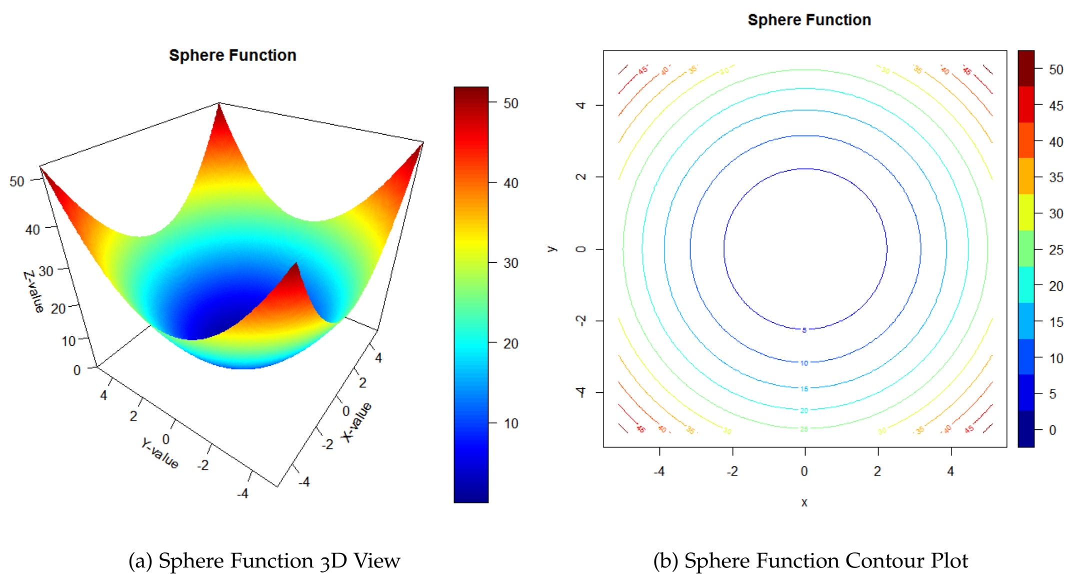
(b) Schwefel Function Contour Plot

Figure E.14: Schwefel Function in its 2-dimensional form

$$f(x) = \sum_{i=1}^{n} x_i^2 \tag{E.15}$$

Subject to domain, $-5.12 \leqslant x_i \leqslant 5.12$ for all $i\{1, 2, ..., n\}$. The global minima is found at the origin $x_i^* = 0$ with value $f(x^*) = 0$.

Figures E.15a and E.15b shows a 3D and contour plot representing the Sphere function in 2-dimensional space.



(a) Sphere Function 3D View

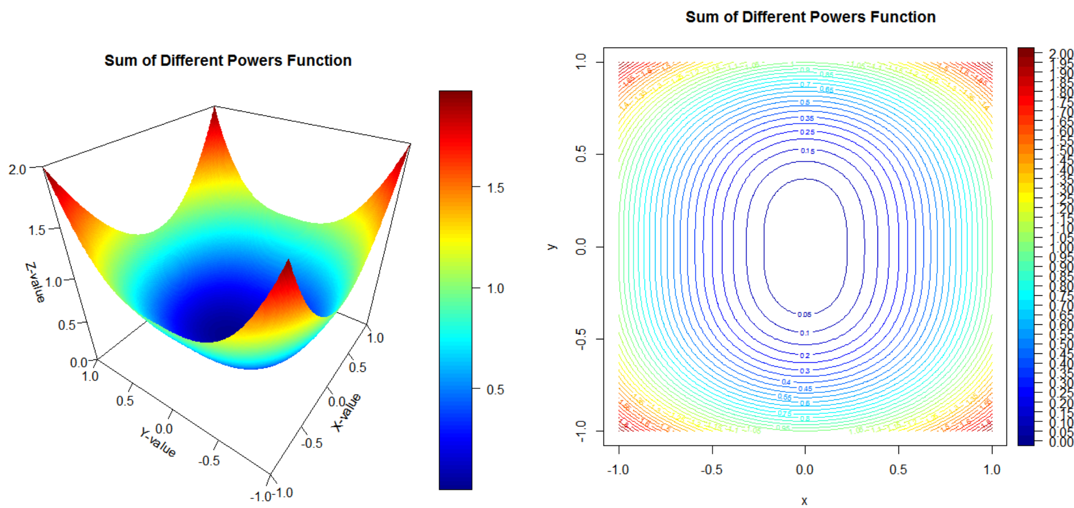
(b) Sphere Function Contour Plot

Figure E.15: Sphere Function in its 2-dimensional form

$$f(x) = \sum_{i=1}^{n} |x_i|^{i+1} \tag{E.16}$$

Subject to domain, $-1 \leqslant x_i \leqslant 1$ for all $i\{1, 2, ..., n\}$. The global minima is found at the origin $x_i^* = 0$ with value $f(x^*) = 0$.

Figures E.16a and E.16b shows a 3D and contour plot representing the Sum of Different Powers function in 2-dimensional space.



(a) Sum of Different Powers Function 3D View

(b) Sum of Different Powers Function Contour Plot

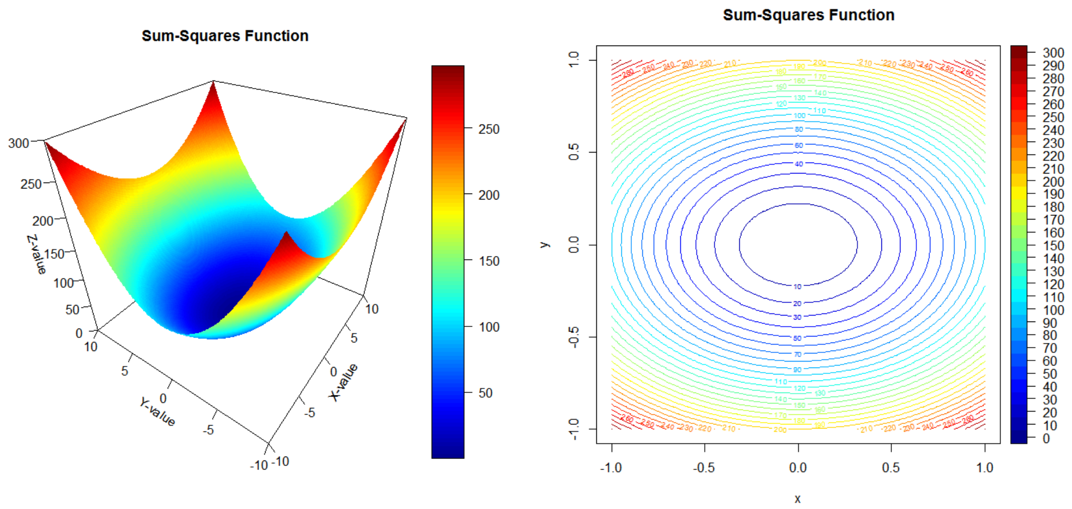
Figure E.16: Sum of Different Powers Function in its 2-dimensional form

E.17 SUM SQUARES FUNCTION

$$f(x) = \sum_{i=1}^{n} i x_i^2 \tag{E.17}$$

Subject to domain, $-10 \leqslant x_i \leqslant 10$ for all $i\{1, 2, ..., n\}$. The global minima is found at the origin $x_i^* = 0$ with value $f(x^*) = 0$.

Figures E.17a and E.17b shows a 3D and contour plot representing the Sum Squares function in 2-dimensional space.



(a) Sum Squares Function 3D View

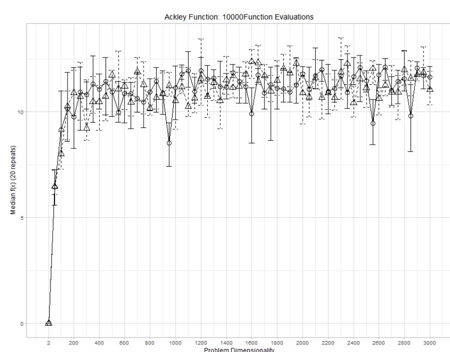
(b) Sum Squares Function Contour Plot

Figure E.17: Sum Squares Function in its 2-dimensional form

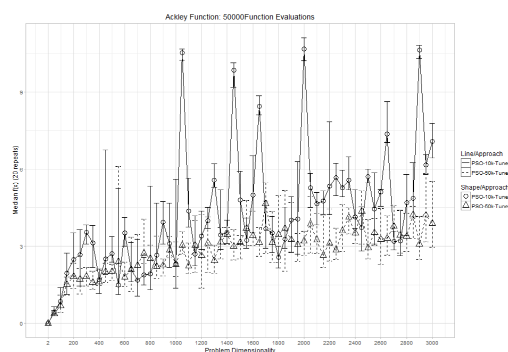
F

SUPPLEMENTARY EXPERIMENT MATERIAL

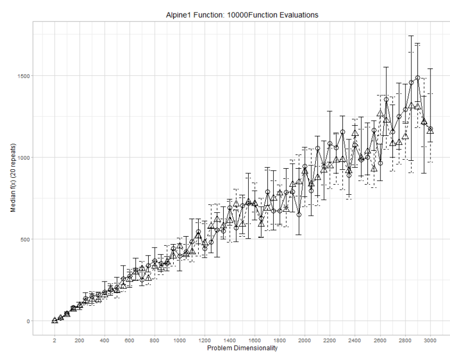
F.1 EXTENDED PSO EXPERIMENT RESULTS



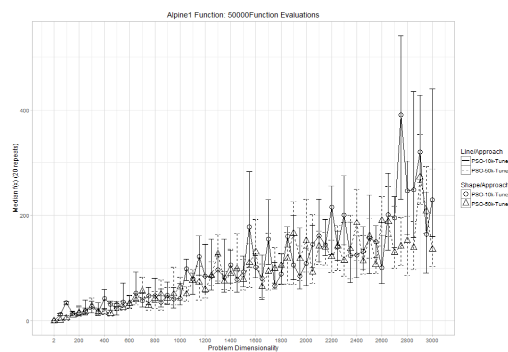
Ackley Function at 10,000 Evaluations



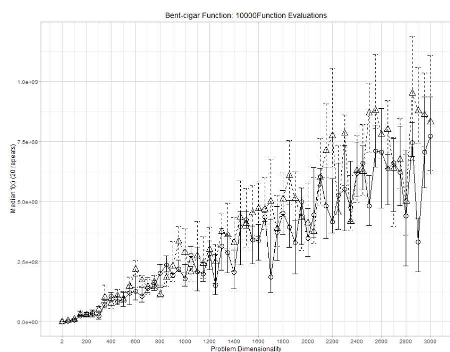
Ackley Function at 50,000 Evaluations



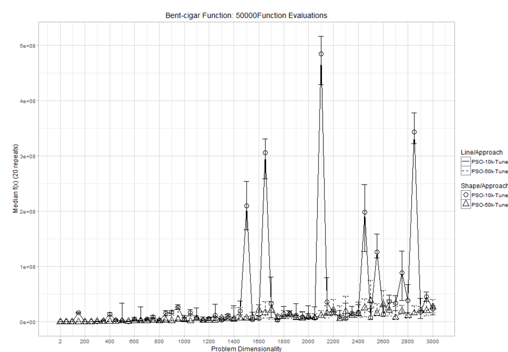
Alpine no.1 Function at 10,000 Evaluations



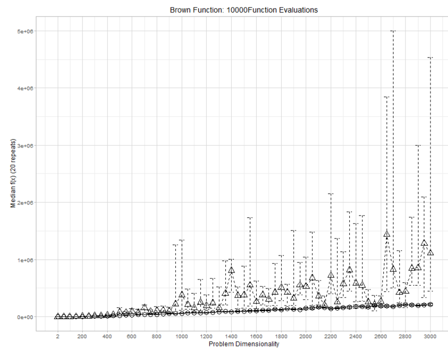
Alpine no.1 Function at 50,000 Evaluations



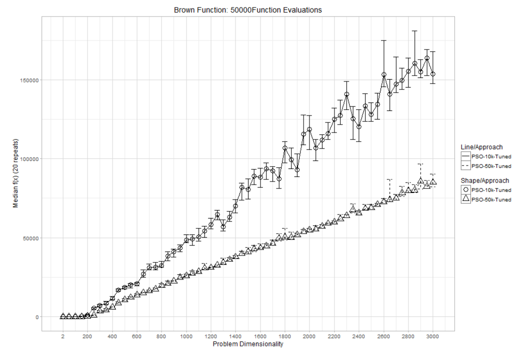
Bent Cigar Function at 10,000 Evaluations



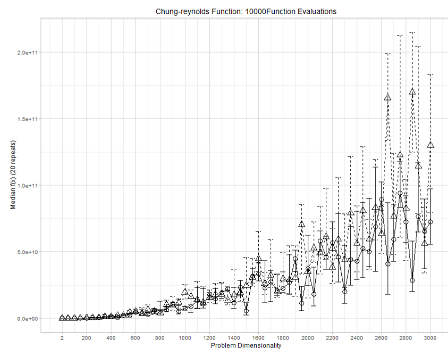
Bent Cigar Function at 50,000 Evaluations



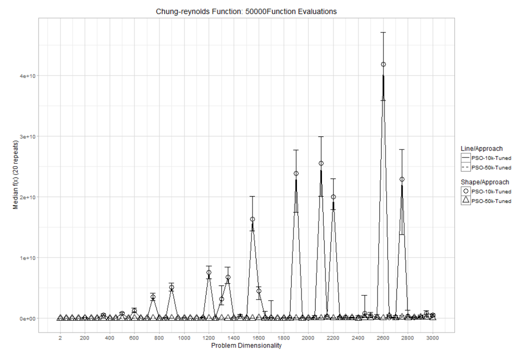
Brown Function at 10,000 Evaluations



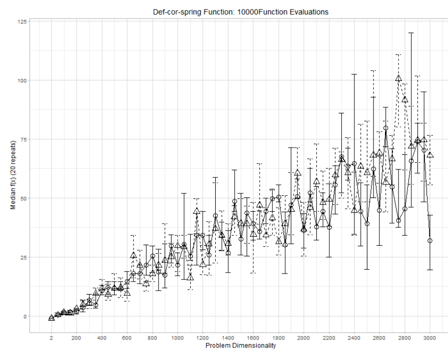
Brown Function at 50,000 Evaluations



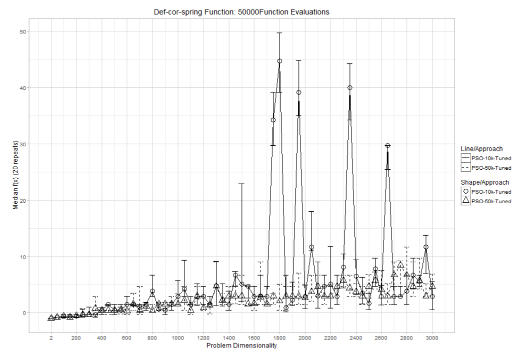
Chung-Reynolds Function at 10,000 Evaluations



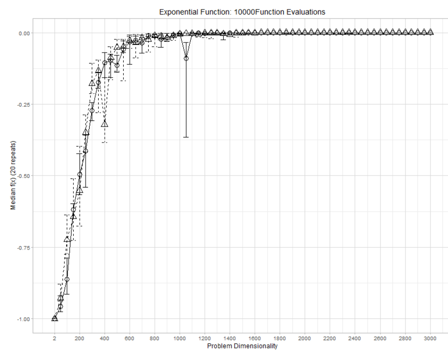
Chung-Reynolds Function at 50,000 Evaluations



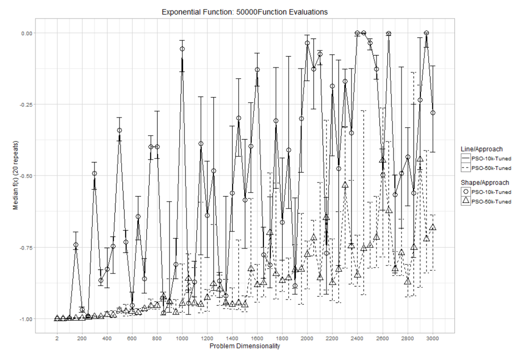
Deflected Corrugated Spring Function at 10,000 Evaluations



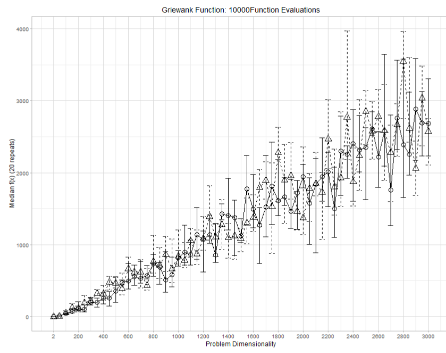
Deflected Corrugated Spring Function at 50,000 Evaluations



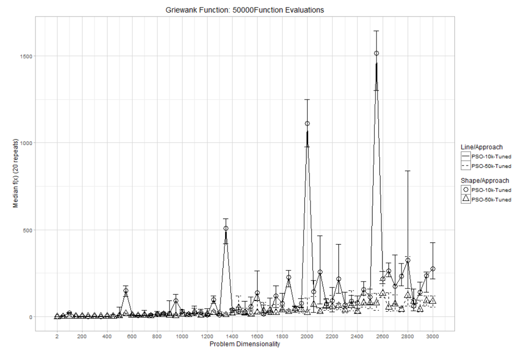
Exponential Function at 10,000 Evaluations



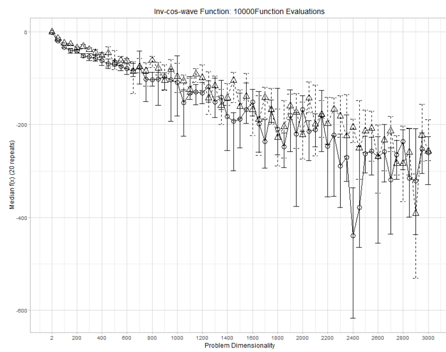
Exponential Function at 50,000 Evaluations



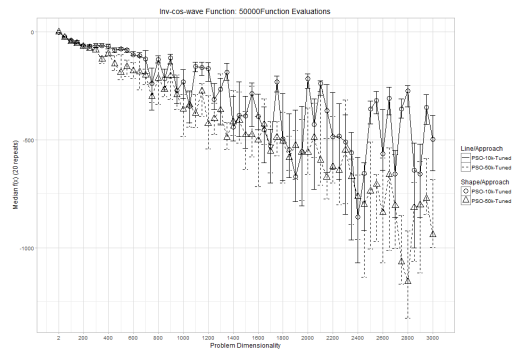
Griewank Function at 10,000 Evaluations



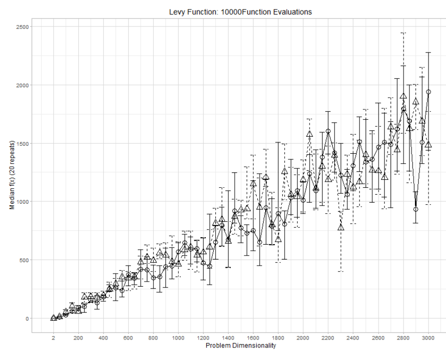
Griewank Function at 50,000 Evaluations



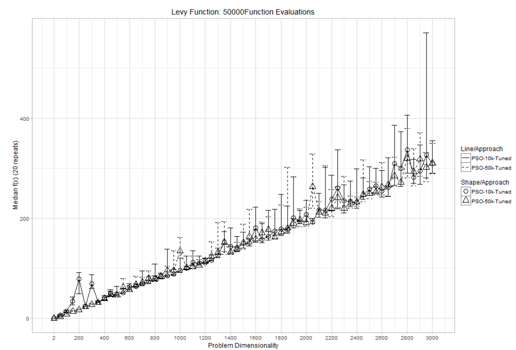
Inverted Cosine Wave Function at 10,000 Evaluations



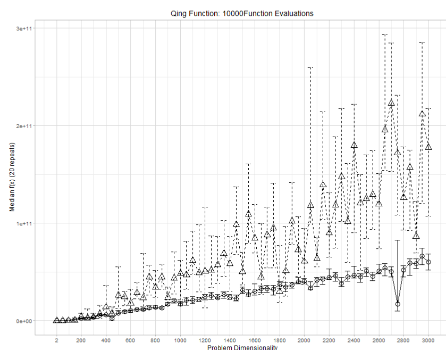
Inverted Cosine Wave Function at 50,000 Evaluations



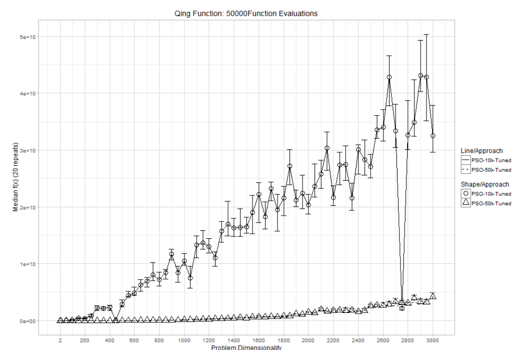
Levy Function at 10,000 Evaluations



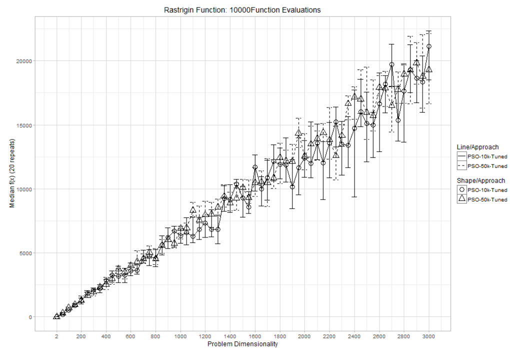
Levy Function at 50,000 Evaluations



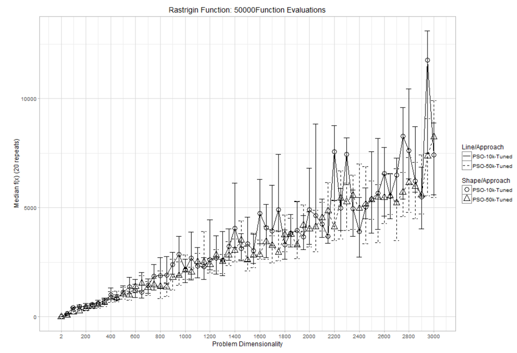
Qing Function at 10,000 Evaluations



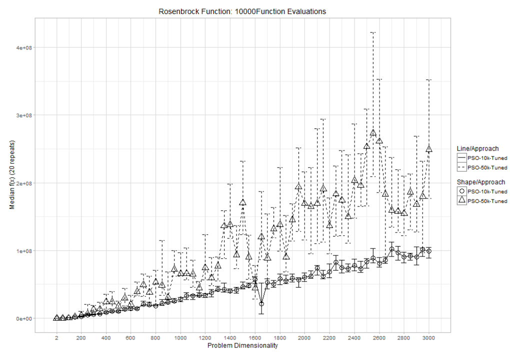
Qing Function at 50,000 Evaluations



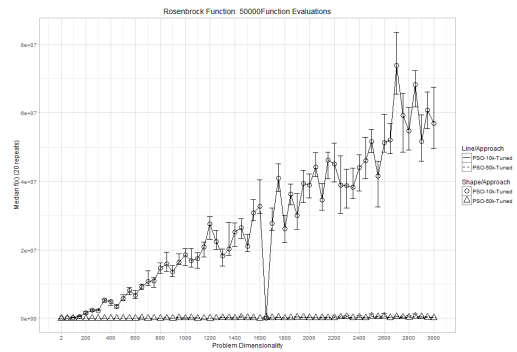
Rastrigin Function at 10,000 Evaluations



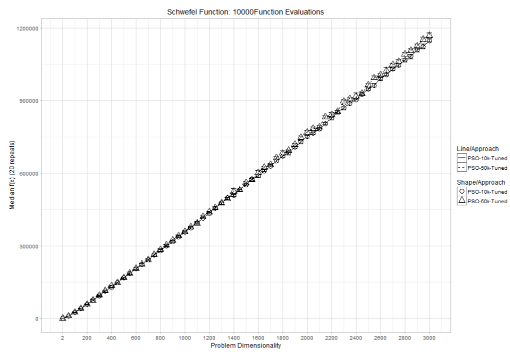
Rastrigin Function at 50,000 Evaluations



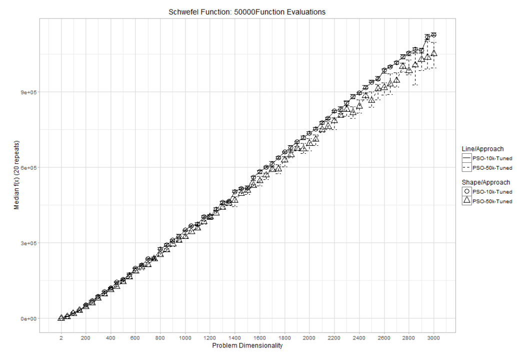
Rosenbrock Function at 10,000 Evaluations



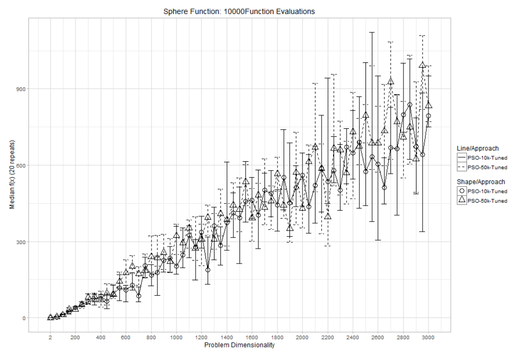
Rosenbrock Function at 50,000 Evaluations



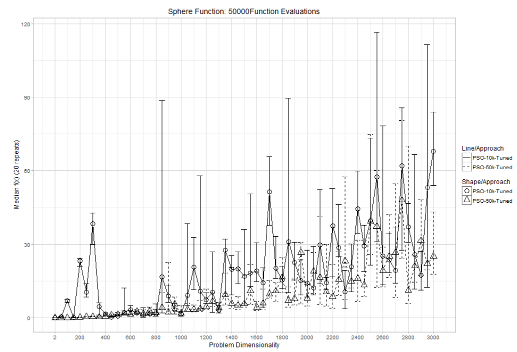
Schwefel Function at 10,000 Evaluations



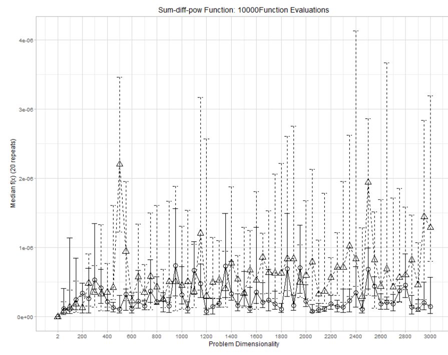
Schwefel Function at 50,000 Evaluations



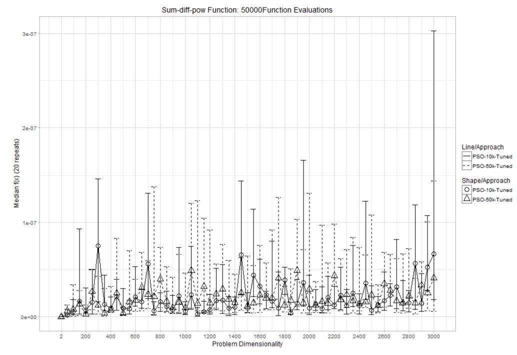
Sphere Function at 10,000 Evaluations



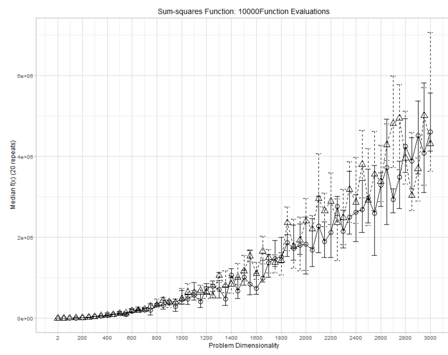
Sphere Function at 50,000 Evaluations



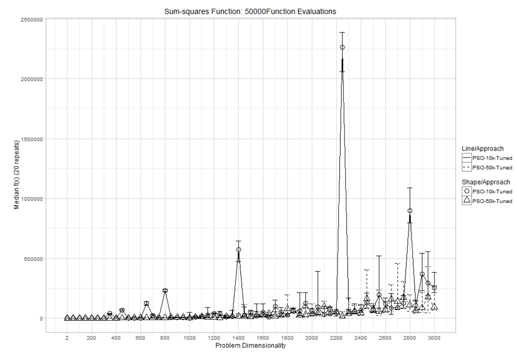
Sum of Different Powers Function at 10,000 Evaluations



Sum of Different Powers Function at 50,000 Evaluations



Sum Squares Function at 10,000 Evaluations



Sum Squares Function at 50,000 Evaluations