SN Computer Science (2022) 3:185
https://doi.org/10.1007/s42979-022-01064-6

ORIGINAL RESEARCH

Recombination and Novelty in Neuroevolution: A Visual Analysis

Stefano Sarti[1] · Jason Adair[1] · Gabriela Ochoa[1]

Received: 12 August 2021 / Accepted: 10 February 2022

Abstract
Neuroevolution has re-emerged as an active topic in the last few years. However, there is a lack of accessible tools to analyse, contrast and visualise the behaviour of neuroevolution systems. A variety of search strategies have been proposed such as Novelty search and Quality-Diversity search, but their impact on the evolutionary dynamics is not well understood. We propose using a data-driven, graph-based model, search trajectory networks (STNs) to analyse, visualise and directly contrast the behaviour of different neuroevolution search methods. Our analysis uses NEAT for solving maze problems with two search strategies: novelty-based and fitness-based, and including and excluding the crossover operator. We model and visualise the trajectories, contrasting and illuminating the behaviour of the studied neuroevolution variants. Our results confirm the advantages of novelty search in this setting, but challenge the usefulness of recombination.

Keywords Neuroevolution · NEAT · Algorithm analysis · Complex networks · Search trajectory networks · Novelty search · Recombination

Introduction

NeuroEvolution of Augmenting Topologies (NEAT) is one of the most influential algorithms for evolving the topology and weights of neural networks. When proposed in 2002 [32], NEAT provided solutions to the existing challenges of evolving complex topologies by facilitating the crossover between individuals of different sizes, adding new structure incrementally, and protecting innovations by speciation. NEAT is, therefore, a complex algorithmic system. Successful applications of NEAT include evolving neural network controllers for robots [29, 37], evolving both controllers and morphology [3, 11], and evolving innovative video game content [12, 34]. Neuroevolution has also been used in biochemistry [9], geosciences [36], and to address open questions in natural evolution [5].

Apart from standard statistical analysis and comparisons, there is a lack of accessible tools to analyse and visualise the dynamic behaviour of neuroevolution systems. Since neuroevolution traverses complex search spaces and solves complex tasks, we argue that analytical tools can help to improve our understanding and inform the design of better systems. In a recent study, Sarti and Ochoa [28] applied, for the first time, search trajectory networks (STNs) [20] to study neuroevolution. In [28], the behaviour of the classic NEAT algorithm with and without recombination was analysed on two simple benchmark functions: XOR and double-pole balancing. Contrary to what is reported in the original NEAT article [32], the analysis in [28] reveals that NEAT without crossover performs significantly better on the studied domains. The advantage of using recombination within NEAT is, therefore, not clear. Several studies report contrasting views on the role of recombination in NEAT, as discussed in detail in "Related Work". Moreover, a recent systematic review of NEAT [25] urges for revisiting the roles of its various components and operators. Such studies are particularly relevant as NEAT-specific operators render it incompatible with many other evolutionary algorithms, and hence NEAT cannot always benefit from advancements in the field [14].

This article is part of the topical collection "Applications of bioinspired computing (to real world problems)" guest edited by Aniko Ekart, Pedro Castillo and Juanlu Jiménez-Laredo.

✉ Stefano Sarti
stefano.sarti@stir.ac.uk

Jason Adair
jason.adair@stir.ac.uk

Gabriela Ochoa
gabriela.ochoa@stir.ac.uk

[1] University of Stirling, Stirling, Scotland

Published online: 08 March 2022

The main goal of the present article is to extend the analysis reported in [28] by incorporating a more challenging domain: maze navigation, and a successful neuroevolution strategy: novelty search. Our main contributions are to:

– Extend and generalise the application of STNs to model the dynamic NEAT algorithm variants on more complex domains.
– Explore the interplay between crossover and novelty as mechanisms for exploration and diversity in neuroevolution.
– Shed new light on the role of crossover in neuroevolution systems.

The rest of the article is organised as follows. The next section overviews related work covering both the role of crossover in NEAT and search trajectory networks. The third section describes the methodology, including the benchmark domain, the NEAT variants, and parameter values used. An analysis contrasting the performance of the NEAT variants is presented in the fourth section. The fifth section describes the STNs model and how it was adapted to deal with NEAT genomes, followed by a discussion of results derived from the STNs analysis. Finally, the last section summarises our main findings and suggest directions for future work.

Related Work

Neuroevolution of Augmenting Topologies (NEAT)

NEAT [32] is one of the earliest successful neuroevolution algorithms of the category known as Topology and Weight Evolving Artificial Neural Networks (TWEANN). To date, NEAT has been used successfully in diverse applications, spanning from dynamically evolving agents and content for video games [12, 34], generating complex musical compositions [13], evolving reaction networks in synthetic biochemical systems [9], prediction in geosciences [36], generating trading signals for financial markets [18], and estimating the measurement of the top quark from the Tevatron particle collider [1].

The objective of this system is to simultaneously discover effective weight values and topologies for neural networks through evolutionary computation. When first introduced [32], NEAT outperformed the best fixed-topology neuroevolution methods. It has since seeded many different variants [25], but the characteristics of the underlying algorithm remain.

These characteristics include *complexification*, which allows the system to begin from a minimal topology, increasing in complexity based on the problem domain, while retaining the simplest solution that can solve the task. *Speciation* discretises the population into separate niches according to genetic distance (similarity between genomes) and the use of *explicit fitness sharing*. This protects innovation and maintains diversity, avoiding the best performing solution to pervade the entire population, giving the opportunity for under-performing, yet interesting ones to evolve. *Historical markings* were introduced to overcome the problem of crossing over neural networks with similar topologies that compute the same function, but are differently organised [26].

Crossover (also known as genomes recombination) enables two highly performing genomes to produce an offspring which inherits important traits derived from both its parents. This is where historical markings allow the identification of matching and excess or disjoint genes. Matching genes are those that both parents have, whereas disjoint/excess are those that belong to just one of the parents. Matching genes are taken from the highest performing parent, the non-matching ones are chosen at random.

This operator, although innovative in some ways, has often been acknowledged as a primary source for debate on its usefulness in the evolution of ANNs [2, 38]. The following section provides an overview of the research specifically related to this operator.

The Role of Crossover in NEAT

NEAT consists of an encoding for neural networks with specialised operators to address the challenges of evolving increasingly complex topologies. The inclusion of historical markings in NEAT allowed the crossover operator to create valid offspring by identifying regions of each genome that were compatible. Each of these components was examined in a series of ablation experiments in Stanley and Miikkulaine [32] and each was determined to not only increase the performance of NEAT, but are independent and necessary for its application [33]. It was noted, however, that the non-mating NEAT, that is, NEAT without crossover, converged on the target fitness threshold significantly faster than the other ablation studies. This suggests that of all the operators, crossover contributes comparatively less.

These ablation experiments were performed again on a derivative of NEAT known as odNEAT [30], a decentralised version of NEAT designed to facilitate online learning within groups of autonomous robots. When crossover was ablated, a small reduction in the number of successful runs was observed (in comparison to the other operator ablation experiments) but the average number of evaluations of each robot increased by 18.9%, further supporting the contribution of crossover in neuroevolution. This finding is further supported by the same authors in [31] where ablation of the crossover in odNEAT was found to have a larger detrimental effect than that of the ablation of speciation.

These finding are not universal. NEAT was further compared against EANT2 in Siebel and Sommer [29], where the authors use an evolution strategy to develop the topology of neural networks through reinforcement learning, and apply it to a task which controls robots in a visual surveying scenario. While EANT2 is not a direct extension of NEAT, its operators are closely inspired by it: the authors note that they were not able to develop a crossover variant which contributed to the success of the algorithm.

Real et al. [27] proposed an algorithm to optimise the architectures of convolutional neural networks (CNNs). The architecture of each CNN was encoded as a graph with each of the vertices representing rank-3 tensors. Two of which encode the spatial coordinates of an underlying image, and the third is the number of channels. The operators applied in this evolutionary algorithm are again inspired by NEAT: while the mutation operators were demonstrated to be highly successful in the application, three variations on crossover failed to improve upon the results and were consequently discarded.

More recently, however, NEAT has been extended for use in deep neural networks. Costa et al. [6] proposed COEGAN which was based of the NEAT extension DeepNEAT [16]. In this work, they combine neuroevolution and coevolution to assist in the training of Generative Adversarial Networks. To create new offspring, they initially attempted to utilise both mutation and crossover operators. Their preliminary investigations discovered that the crossover operator led to the rapid saturation of the number of layers in the neural network, suggesting premature convergence.

The contributions of the crossover operator must be weighed up against the advantages of integration with more recent advancements in the field of search. Mouret and Doncieux [17] introduced a new variant of NSGA-II in which the encodings developed in NEAT are adapted for use with evolving neural network topologies. To achieve this, they were able to forego the crossover operator and achieve state-of-the art results. In light of the recent advances in NEAT implementations that do not include a crossover operator, Papavasileiou et al. [25] suggest that it is prominent to begin revisiting the original ablation studies.

Search Trajectory Networks (STNs)

Search trajectory networks (STNs) are a data-driven, graph-based model of search trajectories where nodes represent a given state of the search process and edges represent search progression between consecutive states. The STNs model was inspired by local optima networks (LONs) [22], which are a graph model of fitness landscapes where nodes are local optima and edges are transitions among optima with a given search operator. STNs differ from LONs in that the nodes represent states of the search process, not

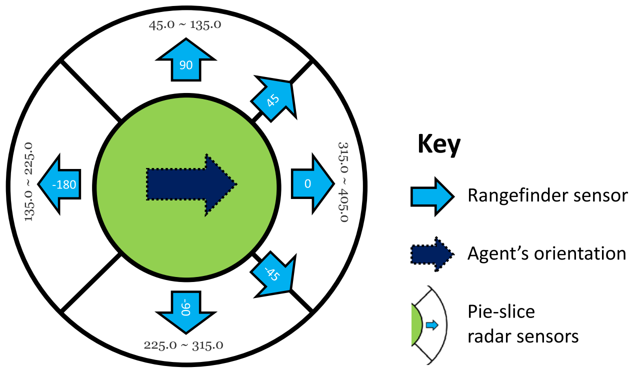


Fig. 1 Architecture of the maze-navigating agent. The agent is comprised of six rangefinder sensors for obstacles detection and four pie-slice radar sensors acting as a compass to detect the goal orientation. Pie-slice labels indicate the degree range of the compass, and arrow labels indicate the rangefinder sensors positions, both in reference to the agent's orientation. Illustration adapted from [24]

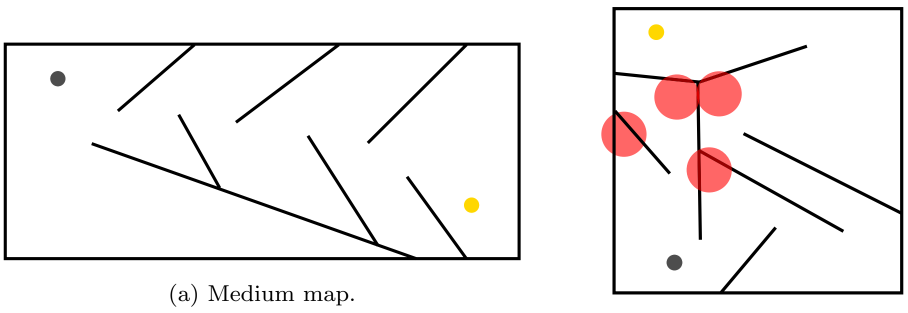
necessarily local optima, which generalises and extends the use of this graph-based model of search dynamics. Once a system is modelled as a graph (network) it can be visualised and analysed with the plethora of powerful analytical and visualisation tools provided by the science of complex networks [19]. STNs were initially proposed to characterise differential evolution and particle swarm optimisation for several classical continuous optimisation benchmark functions [21]. STN analysis was later extended to cover not only population-based algorithms but also stochastic local search methods, and both continuous and combinatorial optimisation problems [20].

Methodology

Benchmark Domain

We use the classic 2-D maze navigation domain outlined in [15]. The task involves an agent (robot) controlled by a neural network navigating a maze from a starting point to and end point, for a fixed number of time steps.

Agent's sensors and actuators The agent architecture is presented in Fig. 1. The agent has six rangefinder sensors that indicate the distance to the nearest obstacle. These are rays (represented as arrows in Fig. 1) which originate from the centre of the agent and detect obstacles that are in close proximity, returning the distance to such obstacles. The four pie-slice radar sensors are known as the field of view (FOV) that orient the agent towards the goal (maze exit point). When the line from the goal location to the centre of the robot falls within these (FOV degrees are specified in Fig. 1), the specific radar sensor becomes activated.



(a) Medium map.

(b) Hard map.

Fig. 2 Maze navigation maps. In both maps, the dark-grey dot represents the starting position of the agent and the yellow dot represents the goal. In the hard maze, the landscape local optima are highlighted in red. Image adapted from [24]

The activation of the sensors are returned as inputs for the maze-navigating agent to compute behaviours and stored to represent the state of the agent at each simulation time steps.

The outputs computed by the ANN are relative to the available actions that the agent can take. There are two actuators (actions) which relate to forces that either rotate and/or propel the agent's body. These correspond to changes in linear and/or angular velocity.

The maze domain is relevant for testing novelty search as it has a deceptive fitness landscape. The fitness function used in [15] is how close the agent is to the goal at the end of the maze navigation simulation. The navigation is made difficult as the maze has walls that form "culs-de-sac". These dead ends that lead close to the goal are local optima to which an objective-based algorithm may converge. This is especially accentuated in the hard maze map (see Fig. 2b), where the local optima that can trap the search progress are highlighted in red. We used the two maps designed in [15], described as follows.

Medium Maze (low deception). Figure 2a shows the map for the medium maze. This configuration is of low to medium difficulty. The map presents areas of low deception that can be circumvented by the agent without major difficulty. The path from the starting point (dark-grey dot) to the goal (yellow dot) is reasonably linear with a lower chance, as compared to the hard map, for the agent to get trapped in between walls.

Hard Maze (high deception). Figure 2b illustrates the hard maze configuration. This map is harder as the placement of the walls generate local minima (red-shaded circles) capable of trapping the search progress of agents traversing the maze. These areas of high deception are what most challenges the neuroevolution search strategies.

NEAT Variants

We contrast two NEAT variants, the standard fitness-based NEAT [32] versus Novelty search as proposed in [15]. For these two strategies, we consider the algorithms with and without the crossover operator. Therefore, our study considers four NEAT variants that we name: *Novelty_X*, *Novelty_NoX*, *Fitness_X* and *Fitness_NoX*. Representing novelty search with and without crossover, and fitness-based search with and without crossover, respectively.

Fitness-Based Search

Standard evolutionary algorithms use a fitness function to guide the search process. The original NEAT variant was guided by a fitness function. In the maze domain, the fitness function measures the quality of an agent based on its proximity to the goal at the end of the navigation task evaluation process:

$$\mathcal{L} = \sqrt{\sum_{i=1}^{2} \left(a_i - b_i\right)^2}. \tag{1}$$

Equation 1 is used to calculate the Euclidean distance between the agent's simulated location with respect to the goal (exit point of the maze). $\mathcal{L}$ represents the specific root-mean-squared error function used for the proximity evaluation, where a denotes the position of the agent at the end of the simulation and b the location of the maze exit (expressed as 2-dimensional coordinates):

$$\mathcal{F} = \begin{cases} 1.0 & \mathcal{L} <= R_{\text{exit}} \\ \mathcal{F}_n & \text{otherwise} \end{cases}. \tag{2}$$

Using the above equation, it is possible to define the fitness function as illustrated in Eq. 2. R_{exit} refers to the radius (0.05) of the exit circumference, this is defined as the solution threshold, which in set to 0.95. Any resulting scores equal or exceeding this value will be returned as 1.0. In Eq. 3, the fitness function is normalised. D_{init}denotes the initial distance of the agent from the maze goal:

$$\mathcal{F}_n = \frac{\mathcal{L} - D_{\text{init}}}{D_{\text{init}}}. \tag{3}$$

This way values are normalised to be in the rage of (0, 1]. In the case that the values negatively exceed this range, these will be adjusted and returned as 0.01. In summary, the closer an agent can approach the radius of the exit point, the higher will its scaled fitness be.

Novelty Search

This search strategy has been introduced in [15], an early publication which described the benefits of abandoning objectives in the pursuit of novelty. The authors specifically detailed the performance improvement of NEAT using this counter-intuitive strategy. Essentially, the idea is to define an objective function which uses the *novelty* of the agents' behaviours as a metric of performance. Novelty, specific to NEAT, can either be considered *structural* (novelty in the ANNs topologies) or, as it is in our scenario, *behavioural* novelty (novelty of the ANNs explorative behaviours).

Differently from fitness search, the interest diverges from seeking the highest proximity to the exit point, for the objective of achieving solvers which exhibit diverse explorative behaviours. The aim is to drive the evolutionary process towards diversity and reward those actions that yield path towards new and unexplored locations of the domain. The hope is to evolve neurocontrollers capable of finding unforeseen tactics to escape the maze's basins of attractions, ultimately to identify the goal:

$$\mathrm{dist}(x, \mu) = \frac{1}{n} \sum_{j=n}^{n} \left| x_j - \mu_j \right|. \tag{4}$$

In this specific scenario, the performance of the genomes producing the neurocontrollers are calculated using the metric of sparseness. To do so, the implementation, derived from [24], similarly to [15], uses the *k-nearest neighbours* algorithm outlined in Eq. 4.

Increased sparseness is obtained by neurocontrollers capable of tracing exploratory trajectories (Cartesian coordinates) towards least visited locations of the maze. Therefore, the novelty metric is determined by the distance between the two trajectory vectors (one for each compared agent). This sparseness assessment is achieved by comparing historical novelty items that are logged in a novelty archive and items generated from the current population. The resulting metric is used to assign the fitness of a given agent based on the novelty of its behaviours.

In Eq. 4, $\mathrm{dist}(x, \mu)$ is the novelty score denoting the behavioural difference between two agents, computed as the distance between the two trajectory vectors (one vector per agent; x and μ). Trajectory vectors, which are traced by agents, are comprised of bi-dimensional maze coordinates of size n. x_j and μ_j are the values of the compared vectors (x and μ) at position j. To simplify the calculation, in this implementation, only the agent's trial end coordinates ($j = n$) are considered as the coordinates of interest. This way we can determine the final position of the agent and, therefore, the distance to the goal.

Table 1 NEAT parameter values used

Parameter	Value
Population size	250
Maximum generations	1000
Solver time steps (medium maze)	400
Solver time steps (hard maze)	600
Solution fitness value	1.00
Fitness threshold	0.95
Bias range	[− 30, 30]
Weight range	[− 30, 30]
$c1$	1
$c2$	1
$c3$	3
Probability add link	0.1
Probability add node	0.005
k (k-nearest neighbours)	16

The k parameter (k-nearest neighbours) is relevant only for the novelty search variants

Parameters

Table 1 outlines the parameters values used in our experiments, we emulate the values used in [15]. All parameters, with the exception of the solver time steps, are identical for both maze maps. Similarly, the parameter values are the same for the four algorithm variants. The only difference between the crossover and no-crossover variants is the ablation or removal of the crossover operator. The k parameter in the k-nearest neighbours algorithm is required for the sparseness calculation, necessary only for novelty search. The solver time steps had to be increased for the hard maze, as for this specific implementation [24], our tests have shown that 400 time steps were not a sufficient allowance to reach the goal in this map.

The coefficients $c1$, $c2$ and $c3$ are all NEAT-specific parameters. The first two relating to the *excess* and *disjoint* genes, and the last relating to the average *weight* difference of matching genes. This is a fundamental step in the algorithm to generate species; this similarity check reduces compatible genomes from the entire population into niches.

Table 2 Performance metrics

	Medium map		Hard map	
	Crossover			
	Novelty_X	Fitness_X	Novelty_X	Fitness_X
Success rate	86.67% (26 runs)	20.0% (6 runs)	3.33% (1 runs)	0.0% (0 runs)
Best fitness	0.9842 (0.0422)	0.9264 (0.0387)	0.77 (0.0437)	0.7629 (0.0)
Generations	422.54 (309.89)	417.67 (361.71)	623.0 (0.0)	-
	No Crossover			
	Novelty_NoX	Fitness_NoX	Novelty_NoX	Fitness_NoX
Success rate	86.67% (26 runs)	6.67% (2 runs)	13.33% (4 runs)	0.0% (0 runs)
Best fitness	0.9857 (0.0376)	0.9109 (0.0324)	0.7933 (0.0855)	0.7629 (0.0)
Generations	265.38 (243.63)	583.0 (265.0)	719.75 (165.6)	-

Best fitness and generations are mean values with standard deviations in parenthesis. Mean generations are computed for the successful runs only

Performance Analysis

Experiments' Setup

For each algorithm variant and maze map, 30 runs were executed with the parameter settings outlined in Table 1. For both maps, a genome with fitness 1.0 is considered to solve the maze. In other words, a run is successful if the best fitness achieved reaches a value of 1.00. To measure algorithm performance, we consider three metrics: (i) the success rate, which computes the ratio of runs reaching a solution, (ii) the best fitness attained at the end of the run, averaged over 30 runs and (iii) the number of generations to reach a solution for the successful runs, averaged over the number of successful runs for each variant. We also studied the distribution of values for the best fitness and the number of generations across the 30 runs.

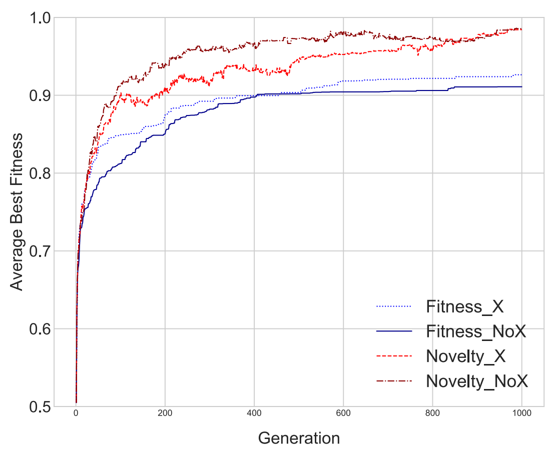
Results and Discussion

Table 2 shows the performance metrics for the four algorithm variants on the two maze maps. On both maps, novelty search produces higher success rates and average best fitness than fitness-based search, confirming the findings reported in [15]. The differences in the best fitness attained are statistically significant ($p < 0.001$) according to the Mann–Whitney test. The variants contrasted in [15] all used crossover. Our comparison instead includes variants without crossover, so we can contrast the usefulness of this operator within both novelty search and fitness-based search.

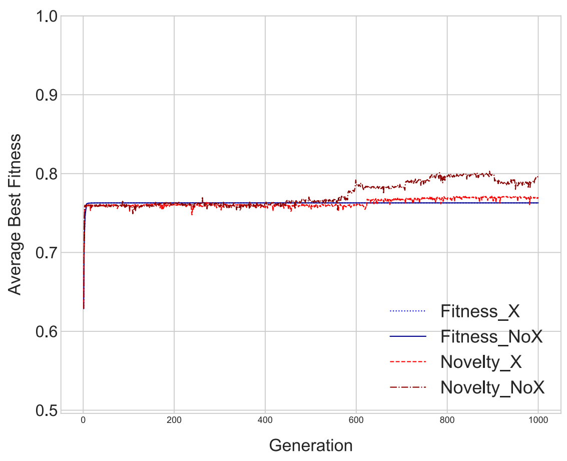
On the medium map (left part of Table 2), we can see that Novelty_X and Novelty_NoX have the same success rate. However, the successful runs without crossover reach a solution much faster (fewer evaluations on average) than the crossover variant. This result is statistically significant ($p < 0.01$) according to the Mann–Whitney test. This suggest that crossover hinders the search progress for novelty search. In contrast, fitness-based search on the medium map produces a higher success rate, higher best average fitness, and a lower number of evaluations to reach a solution when recombination is used. Therefore, crossover seems to be helpful for fitness-based NEAT, in this specific case. However, the differences in performance between Fitness_X and Fitness_NoX, are not statistically significant.

On the hard map (right part of Table 2), we can see that Novelty_NoX has a higher success rate than Novelty_X and reaches a higher fitness on average. This supports the suggestion that crossover is not useful for novelty search and it hinders the performance. The difference in best fitness between Novelty_X and Novelty_NoX are statistically significant ($p < 0.001$). Fitness-based search on the hard maps fails to reach a solution for both variants (X and NoX). The success rate is zero, and the best fitness reached by all runs is consistently 0.7629, indicating that the search gets trapped in a local optimum. The number of generations each variant took to find this local optimum are not significantly different, indicating no tangible effect to the inclusion of crossover in this scenario. These results are, therefore, not useful for us to assess whether recombination can help fitness-based NEAT.

Figure 3a and b provides a visual comparison of the performance achieved for all variants in each domain, that is the best performance averaged over the 30 runs executed. As outlined above, in the medium maze (Fig. 3a), the Novelty_NoX variant visibly outperforms all other search strategies in terms of speed to solution. As for the fitness quality, both novelty with and without recombination reach similar fitness values at the end of the iterations. Both variants of the fitness search strategy, on the other hand, reach sub-optimal fitness on average, with the variant with active recombination outperforming its counterpart both in terms of speed



(a) Medium Maze Domain



(b) Hard Maze Domain

Fig. 3 Convergence plots representing the averaged fitness performance over 30 runs for all variants tested in the *medium* maze (**a**) and in the *hard* maze (**b**) domains

and quality of solution. This finding highlights the benefit of active recombination in this specific scenario.

In Fig. 3b, we observe that predominantly, all variants and search strategies rapidly achieve a sub-optimal fitness in the region of 0.76. This rapid increase is followed by immediate stagnation around these values. The stagnation of fitness is due to the search process getting trapped in the first local minimum (second red area in the upper left-hand side of Fig. 2b).

Both variants of the objective function reach true stagnation, without any visible oscillating behaviours as opposed to the novelty search variants. If we were to magnify the visualisation enough, we would be able to visibly perceive a very marginal difference in the two lines in favour of the Fitness_NoX variant. This being not enough to assume superiority of this variant. From Fig. 3b, we can observe that the novelty variants start to diverge around halfway point. Novelty_NoX increases in performance while its counterpart does not vary. Eventually, Novelty_X begins to increase in average performance but never reaches the levels obtained by its ablated counterpart.

In Fig. 4, we provide a magnified version of the convergence plot discussed above. In this visualisation we clearly appreciate the specific differences of the novelty search variants. We specifically observe that the beginning of the divergence in best average performance between the two variants happens approximately at generation 400 of 1000. Novelty_NoX increases steadily until generation 900 to a value of 0.80 of 1.00, to then proceed to adjust to lower levels (concluding approx. at 0.789). On the other hand, Novelty_X climbs slightly, to higher values and progresses steadily until generations endpoint.

To further corroborate our analysis of the solutions achieved in the hard map, we visualised the navigation paths of the best genome evolved by each NEAT variant out of the 30 runs. Figure 5 illustrates the exploratory paths in the Cartesian behavioural space, traced by the four NEAT variants. The illustration shows that both fitness-based NEAT variants (in blue) get trapped in the left dead-end; this is the local optimum that caused the stagnation seen in the convergence plots. In contrast, the navigation paths of the two novelty search variants (in red) reach the goal (yellow dot). If we visually contrast the paths of Novelty_NoX (dark red) and Novelty_X (bright red), it appears that Novelty_NoX is

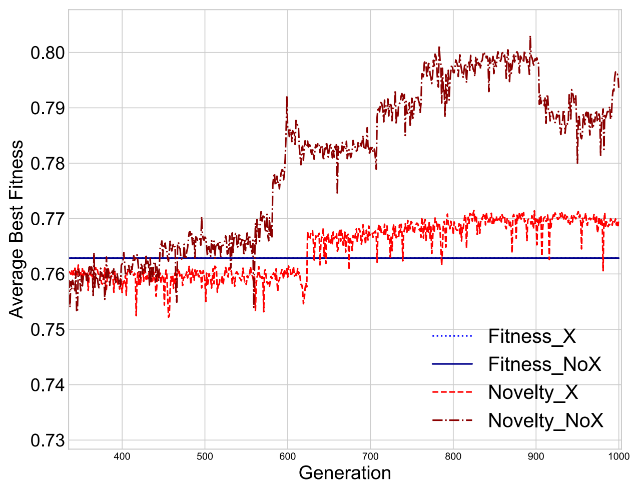


Fig. 4 Convergence plot representing the averaged fitness performance over 30 runs for all variants tested in the hard maze domain. Specifically magnified to highlight salient differences in the novelty search strategy

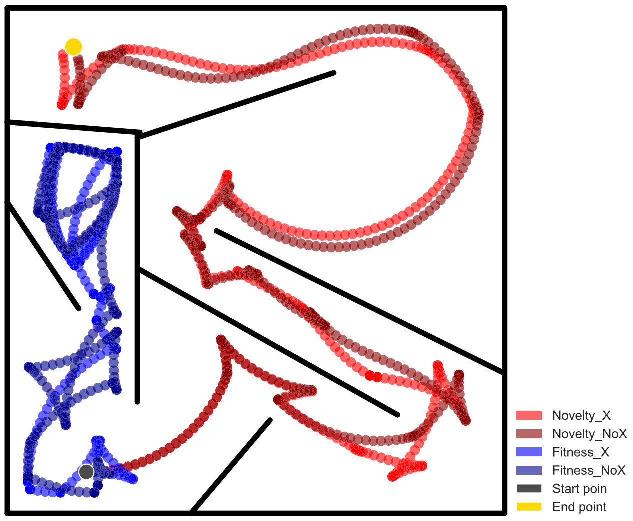


Fig. 5 Navigation paths of the best genomes evolved by each NEAT variants

faster (more direct) in identifying the left turn required to reach the diagonal channel of the maze, and in identifying the goal (maze exit location).

To further analyse the distribution of the performance metrics for all NEAT variants, we produced violin plots for both the best fitness at the end of the run, and the number of generations to reach a solution. These results are discussed as follows first for the medium maze and then for the hard maze.

Medium Map

Figure 6 shows violin plots with the distribution of best fitness values on the medium map and the four NEAT variants. Novelty search is shown at the left in two shades of red for the variants with and without crossover, while fitness-based search is shown on the right with two shades of blue. The individual 30 data points for each variant are also overlaid as black dots. The distributions clearly confirm that the final best fitness values for novelty search are considerably higher than those found by fitness-based search.

The two novelty search variants (X and NoX) (left plots in red, Fig. 6), show similar distributions, with Novelty_NoX slightly skewed towards higher values and a tighter distribution. For fitness-based search, a higher concentration resides nearer higher values for the crossover variant crossover (bright blue).

Figure 7 shows violin plots with the distribution of the number of generations to reach a solution for all variants. The individual 30 data points for each variant are also overlaid as black dots; when a solution is not reached within the maximum of 1000 generations the black dots appear above the dotted line. Here, lower values indicate better performance, as they are indicative of a faster convergence. The

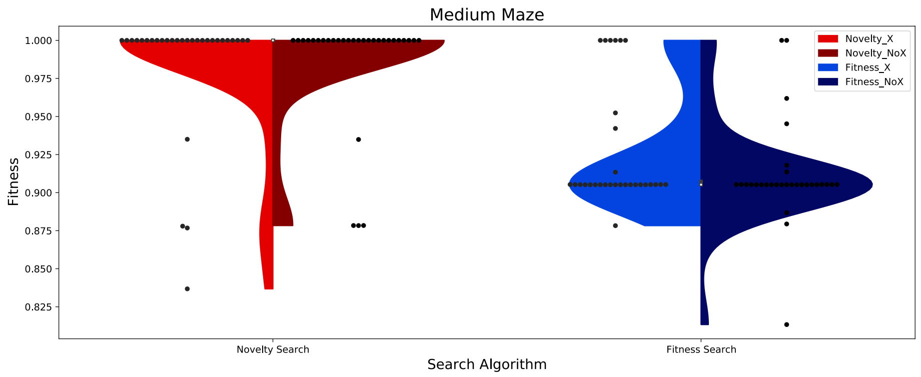


Fig. 6 Violin plots denoting the distribution of the best fitness values on the medium map for all NEAT variants. Overlaid, are swarm plots demonstrating the individual data points for each NEAT variant

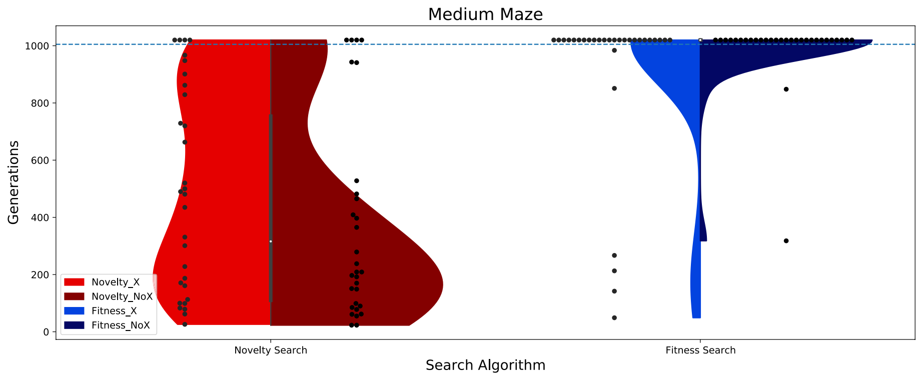


Fig. 7 Violin plots denoting the distribution of the number of generations taken by each variant to reach a solution on the medium map. Overlaid are swarm plots showing the individual data points for each variant—points that lie above the dotted line represent runs that failed to reach a solution

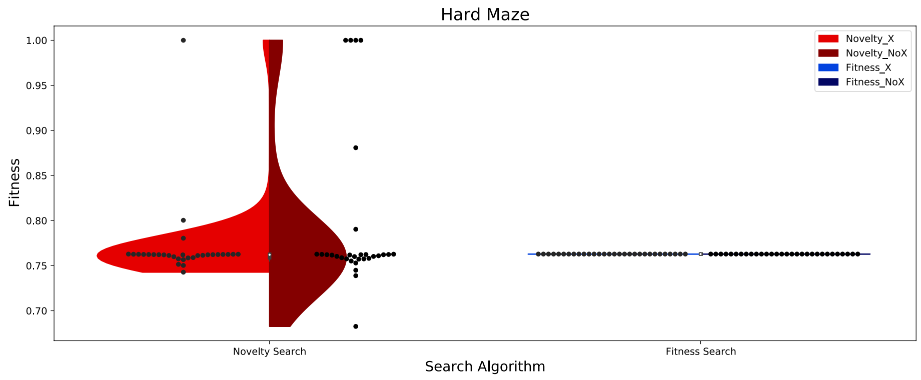


Fig. 8 Violin plots denoting the distribution of the best fitness values on the hard map for all NEAT variants. Overlaid are swarm plots demonstrating the individual data points for each NEAT variant

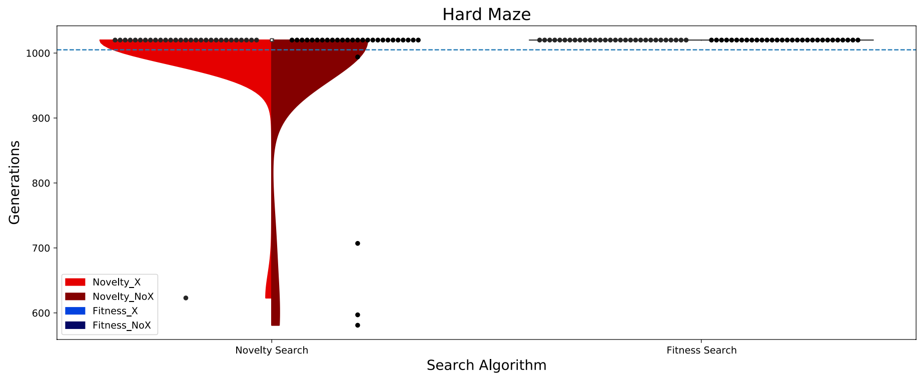


Fig. 9 Violin plots denoting the distribution of the number of generations taken by each variant to reach a solution on the hard map. Overlaid are swarm plots showing the individual data points for each variant—points that lie above the dotted line represent runs that failed to reach a solution

plot clearly indicates a faster convergence of the novelty variants as compared to the fitness-based variants.

For novelty search, the variant with no crossover (dark red, Fig. 7) shows a tighter distribution towards lower generation values than the variant with crossover (bright red), supporting that crossover slows down the progress for novelty search. The situation is reversed for fitness-based search, where the variant with crossover (light blue, Fig. 7) shows a tendency towards lower values. However, for the fitness variants, most runs are unsuccessful, as indicated by the majority of data points above the dotted line. Therefore, crossover seems to be of some use for fitness-based search on this maze, but this strategy is not competitive against novelty search.

Hard Map

Figure 8 shows violin plots with the distribution of best fitness values on the medium map and the four NEAT variants. The distributions confirm again for this map that the final best fitness values for novelty search are higher than those found by fitness-based search. However, for both novelty and fitness-based search, the bulk of the distribution is around the local optimum with a fitness value of approximately 0.76. Novelty search has a few points above (and below) this value, whereas for fitness-based search all the data points reach a fixed fitness corresponding the local optimum trapping the search process. Novelty search without crossover has a larger number of successful runs for this map, as compared to the variant with crossover.

In terms of the number of generations to reach a solution, Fig. 9 shows violin plots with the distributions for all NEAT variants. The individual 30 data points for each variant are also overlaid as black dots. We can see that all the runs were unsuccessful for the fitness-based variants on the right, as all the dots appear above the dotted line. For Novelty_X, only one run out of 30 was successful, whereas four Novelty_NoX runs out 30 reached a solution, with a number of generations that varied from over 550 to almost 1000. These results seem to indicate that crossover can be detrimental for novelty search.

Search Trajectory Networks (STNs) Analysis

The original STN model definitions can be found in [20], we reproduce them here for completeness, and also introduce a model variation that we named compressed STNs (CSTNs). This compressed model is inspired by a similar idea applied to local optima networks [23], to deal with search spaces with a large amount of neutrality, that is, adjacent portions of the search space with the same fitness. Modelling neutrality is relevant for NEAT, as it is well known that there are many ways to set neural network weights that instantiate the same behaviour, owing to function-preserving re-scaling of weights, permuting units or redundant mappings [35].

Definitions

To define a network model, we need to specify their nodes and edges. The relevant definitions are given as follows.

Representative solution is a solution to the problem (in this study, an evolved neural network) at a given iteration that represents the status of the search process. For population-based algorithms such as NEAT, the solution with best fitness in the population at a given iteration is chosen as the representative solution.

Location is a non-empty subset of solutions that results from a predefined mapping process. Each solution in the search space is mapped to one location. Several similar solutions are generally mapped to the same location, as the locations represent a partition of the search space. We use the procedure for mapping NEAT genotypes to locations proposed in [28], summarised as follows "Mapping NEAT Genotypes to Locations" for completeness.

Search trajectory Given a sequence of representative solutions in the order in which they are encountered during the search process, a search trajectory is defined as a sequence of locations formed by replacing each solution with its corresponding location.

Node is a location in a search trajectory of the search process being modelled. The set of nodes is denoted by N.

Edges Edges are directed and connect two consecutive locations in the search trajectory. Edges are weighted with the number of times a transition between two given nodes occurred during the process of sampling and constructing the STN. The set of edges is denoted by E.

Search Trajectory Network (STN) is a directed graph $\text{STN} = (N, E)$, with node set N, and edge set E as defined above.

Compressed Node is a node that aggregates a set of connected nodes (a connected component) in the STN with the same fitness value. The set of compressed nodes is denoted by CN.

Compressed edges The set of edges is defined as above for the STN model. However, after compression, there are no edges between nodes with the same fitness, as connected components with the same fitness become a single node. The set of edges among compressed nodes are also aggregated and their weights summed. We call this set compressed edges, CE.

Compressed STN is the directed graph $\text{CSTN} = (CN, CE)$, where nodes are the compressed nodes CN and edges the compressed edge set CE.

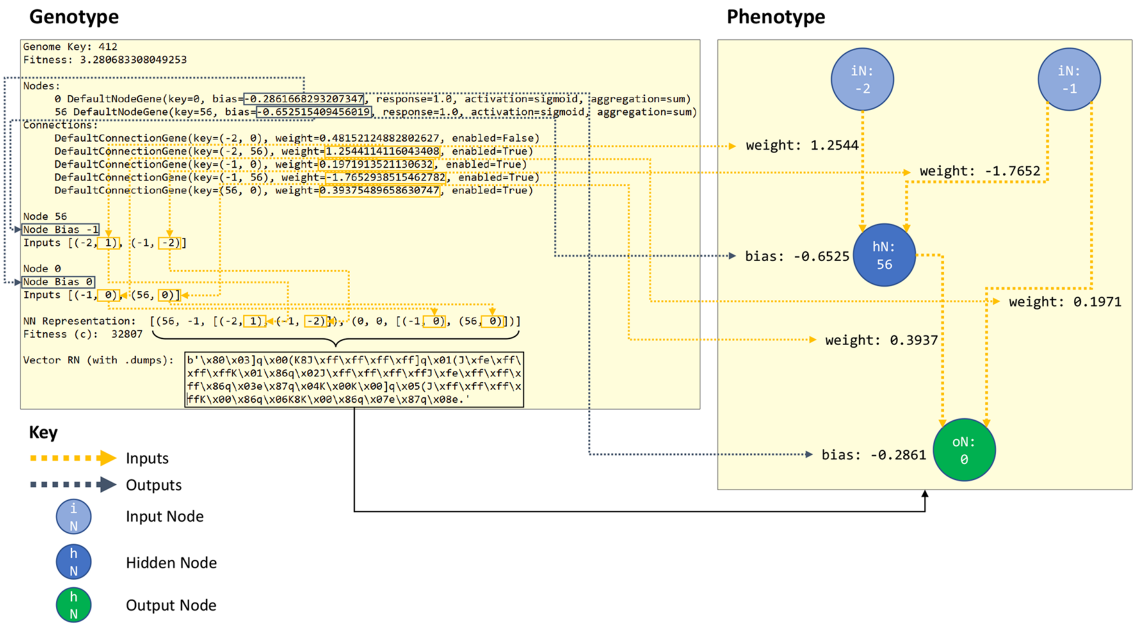
Merged CSTN Once the CSTN models for a set of algorithm–problem pairs are constructed, we can proceed to merge the CSTNs of different algorithms for a given problem. Let us assume we have two algorithms. The merged CSTN model of the two algorithms for a given problem is obtained by the graph union of the two individual graphs for that problem. The merged graph contains the nodes and edges that are present in at least one of the algorithm graphs. Attributes are kept for the nodes and edges indicating whether they were visited by both algorithms or by one of them only.

Mapping NEAT Genotypes to Locations

NEAT genotypes encode both topologies and connection weights and can grow or shrink through generations. To map NEAT genotypes to locations that serve as STNs nodes, in [28], we proposed using the Python object serialisation facilities. The idea is to serialise NEAT genomes and use the resulting byte streams as location signatures. Since the signatures are unique and contain all the genotypic information, they provide a faithful representation for the STNs nodes.

Figure 10 details the mapping process. NEAT genotypes encode both neural network units and connections. Each unit has an identifying id (key) and a bias value. Each connection can be either enabled or disabled, and has a weight. This information is extracted and used to construct a pseudo-phenotypical vector representation (NN representation in Fig. 10). The mapping is completed by passing this vector representation to the `pickle.dumps` function, which produces a flattened, compressed representation of the genotype as a byte stream.

Before the data are mapped, the numerical precision of the weights and bias values needs to be reduced. The goal is to partition the search space, and thus reduce the number of possible locations. This search space partitioning is fundamental for STNs modelling [20, 21] and allows manageable visual representations. In the experiments reported in this paper, the partition is achieved by rounding off to $1e-0$ the numeric values in the genotype (weights and biases, as they are bounded in the range [– 30, 30], see Table 1), and to $1e-2$ for the fitness values.



```
Genome Key: 412
Fitness: 3.280683308049253

Nodes:
     0 DefaultNodeGene(key=0, bias=-0.2861668293207347, response=1.0, activation=sigmoid, aggregation=sum)
     56 DefaultNodeGene(key=56, bias=-0.652515409456019, response=1.0, activation=sigmoid, aggregation=sum)
Connections:
     DefaultConnectionGene(key=(-2, 0), weight=0.48152124882802627, enabled=False)
     DefaultConnectionGene(key=(-2, 56), weight=1.2544114116043408, enabled=True)
     DefaultConnectionGene(key=(-1, 0), weight=0.1971913521130632, enabled=True)
     DefaultConnectionGene(key=(-1, 56), weight=-1.7652938515462782, enabled=True)
     DefaultConnectionGene(key=(56, 0), weight=0.39375489658630747, enabled=True)

Node 56
Node Bias -1
Inputs [(-2, 1), (-1, -2)]

Node 0
Node Bias 0
Inputs [(-1, 0), (56, 0)]

NN Representation:  [(56, -1, [(-2, 1), (-1, -2)]), (0, 0, [(-1, 0), (56, 0)])]
Fitness (c):  32807

Vector RN (with .dumps):  b'\x80\x03]q\x00(K8J\xff\xff\xff\xff]q\x01(J\xfe\xff\
                          xff\xffK\x01\x86q\x02J\xff\xff\xff\xffJ\xfe\xff\xff\x
                          ff\x86q\x03e\x87q\x04K\x00K\x00]q\x05(J\xff\xff\xff\x
                          ffK\x00\x86q\x06K8K\x00\x86q\x07e\x87q\x08e.'
```

Fig. 10 Mapping NEAT genotypes to locations using Python object serialisation (`pickle.dumps`)

Table 3 Description of network metrics

Nodes	Number of nodes
Solutions	Number of nodes that reach the desired fitness target
c-ratio	Measures the relationship between the total number of nodes and the number of compressed nodes
w-edges	Number of worsening edges
n-paths	Number of shortest paths from start nodes to solution nodes in the CSTN
p-length	Average lengths of the shortest paths from start nodes to solution nodes in the CSTN

Sampling and Model Construction

The data to construct the models are gathered while the algorithm variants are running. Specifically, the required output from a run is a list of steps connecting two adjacent representative solutions in the search process. Each search step is stored as an entry in a log file containing the two consecutive representative solutions being linked with the step; these transitions become the edges of the network model. Once the data logs of a predefined number of runs of a given variant–problem pair are gathered, a post-processing maps' solution to locations, aggregates all the locations and transitions, and constructs a network object.

To extract the models, 9 out of the 30 independent runs were selected for each algorithm variant on the 2 maze problems. Specifically, for each variant and problem, we ranked the 30 runs (seeds) according to fitness and then took the top 3, the bottom 3, and 3 intermediate runs. The idea was to select a representative sample of the 30 runs. These runs were then repeated, now keeping logs of the search process.

Network Metrics and Visualisation

Once a system is modelled as a graph, many structural properties can be computed. The most basic metrics are the number of nodes and edges, but a variety of other metrics could be calculated; such as the degree distribution, length of paths, community structure, and centrality of nodes to name a few [19]. To keep things simple, in our approach, we use six network metrics to assess the structure of the trajectories, and thus bring insight into the behaviour of the search variants studied. These metrics are summarised in Table 3. It is worth noting that additional metrics could also be considered.

The justification of this selection of metrics is as follows. The total number of nodes, *nodes*, gives an idea of the amount of the search space that was explored. The number of nodes that reach the fitness target, *solutions*, indicates how many different locations solve the target problem. The ratio of the number of compressed nodes to the total number of nodes reflects the amount of neutrality in the explored space, that is, the proportion of adjacent solutions with the

Table 4 Network metrics

	Medium maze				Hard maze			
	Novelty		Fitness		Novelty		Fitness	
	X	NoX	X	NoX	X	NoX	X	NoX
Nodes	322	798	185	190	1523	1065	138	146
Solutions	48	520	39	2	1	12	0	0
c-ratio	0.20	0.66	0.48	0.50	0.09	0.09	0.81	0.77
w-edges	155	163	2	7	826	555	0	0
n-path	8	9	6	1	1	2	0	0
p-length	21.25	19.67	11.33	11.00	43	91	NA	NA

same fitness. The higher this value, the higher is the neutrality. This metric is computed as follows: $c\text{-}ratio = 1.0 - \frac{|CN|}{|N|}$. The number of worsening edges, i.e. edges that link a node with higher fitness to a node with lower fitness (*w-edges*), is indicative of the amount of non-greedy exploration during the search process. The last two metrics capture the number and length of the shortest paths from start nodes to solutions in the CSTNs, these metrics are indicative of the reachability of solutions. If no solution is achieved, these last two metrics are not defined.

Visualisation is a powerful tool that may allow us to appreciate structural features which can be difficult to infer from the network metrics alone. Node-edge diagrams, used here, are the most common visual representation of a network. Node-edge diagrams assign nodes to points in the 2-dimensional Euclidean space, and connect adjacent nodes by lines. For directed graphs, arrowheads are used to indicate the direction of connections. Nodes are then drawn on top of the edges using simple geometric shapes (such as circles or squares). Typically, the most important attributes of nodes and edges are assigned to visual properties (such as size and colour) of the shapes and lines. The graph visualisations in this paper were produced with the igraph library [7] of the R programming language. We considered *force-directed* layout algorithms [10], which strive to satisfy some generally accepted criteria, such as distributing the nodes evenly on the plane, minimising the number of edges crossing and keeping edges lengths approximately uniform.

Figure 11 illustrates the merged CSTNs for the two maze maps. Nodes and edges are decorated to highlight relevant features. Compressed node sizes are proportional to number of individual nodes (locations) they contain. The meaning of the node colours is indicated in the plots legend; the start of trajectories are represented as dark-grey squares, the end nodes reaching a solution as yellow circles, and the sub-optimal end nodes as dark-grey triangles. The intermediate nodes and edges visited by each algorithm variant are coloured circles with the colour convention followed in "Performance Analysis". When a node in the merged CSTN is visited by the two variants, it is visualised in light grey. Finally, bright green lines are used to highlight worsening edges, that is, edges that go from a location of higher fitness to a location of lower fitness. This is relevant to appreciate the explorative (non-greedy) dynamics of novelty search.

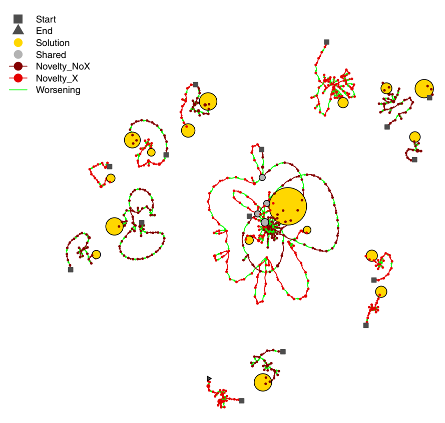
Note that the R scripts used for creating, visualising and analysing the STN models presented in this paper follow those provided on the STN GitHub repository.[1]

Results and Discussion

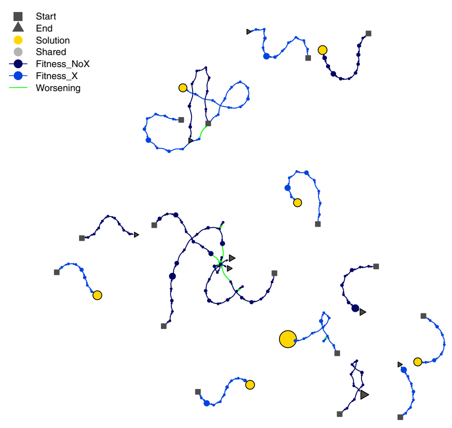
Table 4 shows the values of the network metrics described in Table 3 for the two mazes and the four algorithm variants. The following main observations can be gathered from these metrics:

- The number of nodes is consistently larger for novelty search compared to fitness search. This confirms the stronger explorative power of novelty search.
- Novelty search is also the strategy reaching a larger number of different solutions. Confirming the superiority of this approach for maze domains.
- Novelty search has a much larger number of worsening edges as compared to fitness search. This is consistent with the non-greedy exploration of novelty search.
- In novelty search, the no-crossover variant achieves a larger number of solutions on both mazes. Indeed, novelty *without* crossover is the only strategy achieving multiple solutions in the hard maze, and in the medium maze, it has a larger number of paths to solutions and these paths are shorter on average. These findings challenge the usefulness of crossover in novelty search.
- In fitness search and the medium maze, the crossover variant produced a larger number of solutions and a larger number of paths to solutions than the variant *without* crossover. Potentially suggesting that crossover can be useful as an exploration mechanism in fitness-based

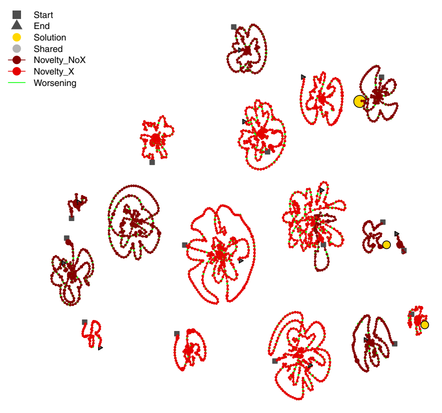
[1] https://github.com/gabro8a/STNs.git.



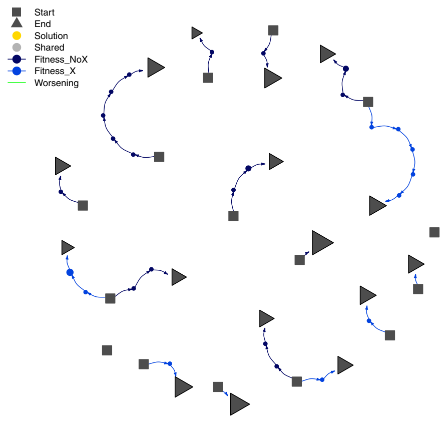
(a) Novelty, medium map



(b) Fitness, medium map



(c) Novelty, hard map



(d) Fitness, hard map

Fig. 11 Merged CSTNs for fitness and novelty search on the two maze maps. Each subfigure merges the trajectories with and without crossover. The size of nodes is proportional to the number of locations in the compressed nodes

search. More research is required as these results could not be confirmed on the hard maze, as fitness search failed to solve the most deceptive domain.

– In the large maze, the compressed ratio is very large for fitness search, indicating that the search traverses many different sub-optimal solutions with the same fitness. In contrast, this metric is rather low for novelty search, indicating that a wide diversity of candidate solutions is explored.

Figure 11 shows the merged CSTNs for the two maze maps. We merge the trajectories *with* and *without* crossover for both novelty-based search and fitness-based search. These illustrations reveal the following main observations, which complement what was observed in the metrics.

– Novelty search, Fig. 11a and c, shows longer trajectories than fitness search (in terms of number of nodes and edges) and the novelty search trajectories reveal a larger number of worsening (bright green) edges.
– Novelty search trajectories, Fig. 11a and c, reach a larger number of solution (yellow) nodes, which are of larger size.
– Novelty_NoX, dark-red trajectories in Figs. 11a and c, reach a higher number solutions (yellow nodes) which are of larger size, as compared to the crossover variant in bright red. The opposite happens with fitness search in the medium maze, Fig. 11b with the crossover variant in bright blue reaches a higher number of solutions than the no-crossover variant.
– For the hard maze, fitness search trajectories (Fig. 11d) are rather short, featuring from zero to a handful of edges. Moreover, all trajectories end in a sub-optimal location of large size (dark-grey triangles). This indicates that the trajectories quickly reach a local optimum, but many different neural networks produce the same sub-optimal fitness.

Overarching Discussion

In this research, we examined the role of recombination in neuroevolution, with specific focus on the evolutionary dynamics between search gradients based on novelty versus fitness. In contrast to our previous research [28] where the problem domains were simple reinforcement learning benchmarks, here we wanted to assess the applicability of STNs to different search strategies by assessing the usefulness of crossover in harder domains.

Due to the deceptive characteristics of the maze problem, reliance on the fitness gradient for search can be demonstrably insufficient; requiring the use of novelty-based fitness, and providing an ideal use-case for the advanced visual analysis and modelling technique: STNs. In [28], the performance analysis and STNs analysis demonstrated similar, but complementary and independent findings. This suggested that STNs are applicable to NEAT and can offer an insight into its inner workings. This paper builds upon these findings by demonstrating the value of the complementary, and different, information provided via STN analysis in neuroevolution. Both analyses are useful and can be performed in isolation.

While statistical analysis paints a comprehensive picture of the performance behaviour of the evaluated variants, performing just this type of analysis could have suggested that fundamentally different solutions were similar, simply because they achieved similar results. Contrary to this, the STN analysis drills down into the genomes' characteristics, presenting an accurate picture of the inherent optimisation search process—highlighting salient differences that are otherwise missed by the statistics lens.

In summary, performance analysis is useful to highlight which algorithm or variant performs best in a particular setting or whether the performance is comparably equal; STNs help further to identify "why" a specific solution is different (or similar) from the perspective of the search space dynamics.

Conclusions

We analysed the role of crossover in the behaviour and performance of fitness-based NEAT and novelty NEAT in the maze domain. We conducted both a standard comparative analysis and a search trajectory networks (STNs) analysis. To use STNs, we adapted the tool to incorporate complex neural genomes, and search spaces with large amounts of neutrality, as it is well known that there are many ways to set a neural network that instantiate the same behaviour.

Our results confirm the advantage of novelty search over fitness-based search in a deceptive domain such as evolving neural controllers for maze navigation. As for the role of crossover, it seems that it can help the search process for fitness-based search. The advantage of using of recombination is less clear when novelty search is used. Our findings on the maze navigation domain indicate that Novelty search without crossover is more effective at reaching good solutions. Without recombination, a larger number of trajectories reach a successful neural network design, and they do so with shorter trajectories (shortest paths in the trajectory network). However, the trajectories with recombination seem to explore more widely the search space, as indicated by the longer trajectories. Therefore, recombination brings additional population diversity which can potentially be useful in yet more complex domains. Recombination in evolutionary computation is generally useful as it can bring diversity without destroying good solution components. We argue, therefore, that the role of recombination within Novelty search deserves further investigation. In particular, the interplay between the diversity brought by the two mechanisms, novelty and recombination, is not yet well understood.

Future work will analyse search algorithms that hybridise novelty and fitness; it is now well known that both components are required for effective neuroevolution, as the growing body of work in quality-diversity optimisation [4, 8] indicates.

Declarations

Conflict of Interest On behalf of all the authors, the corresponding author states that there is no conflict of interest.

References

1. Aaltonen T, Adelman J, Akimoto T, Albrow e.a. Measurement of the top-quark mass with dilepton events selected using neuroevolution at CDF. Phys Rev Lett. 2009;102(15):1–7. https://doi.org/10.1103/PhysRevLett.102.152001.
2. Angeline PJ, Saunders GM, Pollack JB. An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans Neural Networks. 1994;5(1):54–65.
3. Buchanan E, Le Goff LK, Li W, Hart E, Eiben AE, De Carlo M, Winfield AF, Hale MF, Woolley R, Angus M, Timmis J, Tyrrell

AM. Bootstrapping artificial evolution to design robots for autonomous fabrication. Robotics. 2020;9(4):1–24. https://doi.org/10.3390/robotics9040106.
4. Chatzilygeroudis K, Cully A, Vassiliades V, Mouret JB. Quality-diversity optimization: a novel branch of stochastic optimization. In: Black box optimization, machine learning, and no-free lunch theorems. Springer; 2021. pp. 109–135.
5. Clune J, Mouret JB, Lipson H. The evolutionary origins of modularity. Proc R Soc B. 2013;280:20122863.
6. Costa V, Lourenço N, Machado P. Coevolution of generative adversarial networks. In: International conference on the applications of evolutionary computation (part of EvoStar). Springer; 2019. pp. 473–487.
7. Csardi G, Nepusz T. The igraph software package for complex network research. Inter J Complex Syst. 2006;1695(5):1–9.
8. Cully A, Demiris Y. Quality and diversity optimization: a unifying modular framework. IEEE Trans Evol Comput. 2018;22(2):245–59.
9. Dinh H, Aubert N, Noman N, Fujii T, Rondelez Y, Iba H. An effective method for evolving reaction networks in synthetic biochemical systems. IEEE Trans Evol Comput. 2015;19(3):374–86.
10. Fruchterman TMJ, Reingold EM. Graph drawing by force-directed placement. Softw Pract Exp. 1991;21(11):1129–64.
11. Goff LKL, Buchanan E, Hart E, Eiben AE, Li W, De Carlo M, Winfield AF, Hale MF, Woolley R, Angus M, Timmis J, Tyrrell AM. Morpho-evolution with learning using a controller archive as an inheritance mechanism. IEEE Trans Cogn Dev Syst. 2022. https://doi.org/10.1109/TCDS.2022.3148543.
12. Hastings E, Guha R, Stanley K. Automatic content generation in the galactic arms race video game. IEEE Trans Comput Intell AI Games. 2009;1(4):245–63.
13. Hoover A, Stanley K. Exploiting functional relationships in musical composition. Connect Sci. 2009;21(2–3):227–51.
14. Le Goff LK, Hart E, Coninx A, Doncieux S. On pros and cons of evolving topologies with novelty search. The 2020 conference on artificial life. 2020. https://doi.org/10.1162/isal_a_00291.
15. Lehman J, Stanley KO. Abandoning objectives: evolution through the search for novelty alone. Evol Comput. 2011;19(2):189–222. https://doi.org/10.1162/EVCO_a_00025.
16. Miikkulainen R, Liang J, Meyerson E, Rawal A, Fink D, Francon O, Raju B, Shahrzad H, Navruzyan A, Duffy N, et al. Evolving deep neural networks. In: Artificial intelligence in the age of neural networks and brain computing. Elsevier; 2019. pp. 293–312.
17. Mouret JB, Doncieux S. Encouraging behavioral diversity in evolutionary robotics: an empirical study. Evol Comput. 2012;20(1):91–133.
18. Nadkarni J, Ferreira Neves R. Combining neuroevolution and principal component analysis to trade in the financial markets. Expert Syst Appl. 2018;103:184–95.
19. Newman MEJ. Networks: an introduction. Oxford: Oxford University Press; 2010.
20. Ochoa G, Malan KM, Blum C. Search trajectory networks: a tool for analysing and visualising the behaviour of metaheuristics. Appl Soft Comput. 2021. https://doi.org/10.1016/j.asoc.2021.107492.
21. Ochoa G, Malan KM, Blum C. Search trajectory networks of population-based algorithms in continuous spaces. In: Castillo PA, Jiménez Laredo JL, Fernández de Vega F, editors. Applications of evolutionary computation. EvoApplications 2020. Lecture Notes in Computer Science, vol 12104. Cham: Springer. https://doi.org/10.1007/978-3-030-43722-0_5.
22. Ochoa G, Tomassini M, Verel S, Darabos C. A study of NK landscapes' basins and local optima networks. In: GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation. USA: Association for Computing Machinery. 2008. https://doi.org/10.1145/1389095.1389204
23. Ochoa G, Veerapen N, Daolio F, Tomassini M. Understanding phase transitions with local optima networks: Number partitioning as a case study. In: Evolutionary computation in combinatorial optimization, EvoCOP, Lecture Notes in Computer Science, 2017. vol. 10197, pp. 233–248. https://doi.org/10.1007/978-3-319-55453-2_16.
24. Omelianenko I. Hands-on neuroevolution with python. Birmingham: Packt Publishing Limited; 2019.
25. Papavasileiou E, Cornelis J, Jansen B. A systematic literature review of the successors of "neuroevolution of augmenting topologies''. Evol Comput. 2021;29(1):1–73.
26. Radcliffe NJ. Genetic set recombination and its application to neural network topology optimisation. Neural Comput Appl. 1993;1(1):67–90. https://doi.org/10.1007/BF01411376.
27. Real E, Moore S, Selle A, Saxena S, Suematsu YL, Tan J, Le QV, Kurakin A. Large-scale evolution of image classifiers. In: Precup D, Teh YW, editors. Proceedings of the 34th International Conference onMachine Learning. Proceedings of Machine Learning Research. PMLR; 2017. p. 2902–2911.
28. Sarti S, Ochoa G. A NEAT visualisation of neuroevolution trajectories. In: Applications of evolutionary computation, lecture notes in computer science, vol. 12694. Springer; 2021. pp. 714–728. https://doi.org/10.1007/978-3-030-72699-7_45.
29. Siebel NT, Sommer G. Evolutionary reinforcement learning of artificial neural networks. Int J Hybrid Intell Syst. 2007;4(3):171–83.
30. Silva F, Urbano P, Correia L, Christensen AL. odNEAT: an algorithm for decentralised online evolution of robotic controllers. Evol Comput. 2015;23(3):421–49. https://doi.org/10.1162/evco_a_00141.
31. Silva F, Correia L, Christensen AL. Evolutionary online behaviour learning and adaptation in real robots. Roy Soc Open Sci. 2017;https://doi.org/10.1098/rsos.160938.
32. Stanley KO, Miikkulainen R. Evolving neural networks through augmenting topologies. Evol Comput. 2002;10(2):99–127.
33. Stanley KO, Miikkulainen R. Competitive coevolution through evolutionary complexification. J Artif Intell Res. 2004;21:63–100.
34. Stanley K, Bryant B, Miikkulainen R. Real-time neuroevolution in the nero video game. IEEE Trans Evol Comput. 2005;9(6):653–68.
35. Stanley KO, Clune J, Lehman J, Miikkulainen R. Designing neural networks through neuroevolution. Nat Mach Intell. 2019;2:24–35.
36. Wang G, Cheng G, Carr T. The application of improved neuroevolution of augmenting topologies neural network in marcellus shale lithofacies prediction. Comput Geosci. 2013;54:50–65.
37. Wen R, Guo Z, Zhao T, Ma X, Wang Q, Wu Z. Neuroevolution of augmenting topologies based musculor-skeletal arm neurocontroller. In: 2017 IEEE international instrumentation and measurement technology conference (I2MTC), 2017; pp. 1–6. https://doi.org/10.1109/I2MTC.2017.7969727.
38. Yao X, Liu Y. Towards designing artificial neural networks by evolution. Appl Math Comput 1998;9(1):83–90. https://doi.org/10.1016/S0096-3003(97)10005-4.