Maze Navigation using Neural Networks Evolved with Novelty Search and Differential Evolution

Karl Mason*
National University of Ireland Galway
Galway, Ireland
k.mason2@nuigalway.ie

Jim Duggan
National University of Ireland Galway
Galway, Ireland
james.duggan@nuigalway.ie

Enda Howley
National University of Ireland Galway
Galway, Ireland
ehowley@nuigalway.ie

ABSTRACT

Novelty search is a recent approach to evolving neural networks that focuses on searching for networks with new and different behaviour rather than solely focusing on finding the network with the best objective fitness. In reality the concept of novelty is short lived in the sense that nothing stays new indefinitely. Algorithms that archive the best solutions to inform the search are therefore faced with the problem that the novelty scores of these archived solutions will change from generation to generation. This research aims to address this issue by proposing two methods of adjusting novelty scores of archived solutions: 1) Novelty Decay. 2) Recalculating Archived Novelty. Novelty decay enables novelty scores to decay overtime thus enabling the search algorithm to progress while recalculating novelty scores of the archived solutions updates the novelty of these solutions at each generation. When tested on the problem of maze navigation, it is observed that novelty decay and recalculating archived novelty converge faster than both objective search and novelty search alone. Recalculating archived novelty and novelty decay perform statistically equal to one another.

KEYWORDS

Neuroevolution; Novelty Search; Neural Networks; Evolutionary Algorithms; Differential Evolution

1 INTRODUCTION

The notion of evolving a neural network by selecting for behavioural novelty rather than fitness has received a lot of attention since it was first proposed [13]. The reason for the success of novelty search is that by ignoring the objective during the search process, a more diverse range of behaviours is explored. The process of searching for different behaviours gives the evolutionary algorithm a better ability to explore. This enhanced exploration give novelty search an advantage when problems are highly deceptive.

This research focuses on tailoring novelty search for evolutionary algorithms that keep a record of the best found solutions and use these previous solutions to inform the search process. A typical example of this would be evolutionary algorithms that utilize elitism, the process of passing solutions with the highest fitness directly onto the next generation unchanged. This practise has been shown to be highly effective for the standard objective based search [1].

Applying novelty search to algorithms that keep a record of the best found solutions to inform the search, e.g. by using elitism, raises the question: is a behaviour that was novel at the generation it was first observed still considered novel in future generations? If the solution with the highest novelty score is considered the best solution in novelty search, it will likely be used to create new solutions. These solutions are then likely to produce similar behaviour to the original highly novel solution. However if these new solutions have similar behaviour to the original highly novel solution, by definition the original solution that is passed through to subsequent generations is not novel anymore.

It should be noted here that this is not an issue for algorithms that are memory-less, as these algorithms do not consider previous solutions at future generations. Therefore there is no concern as to whether or not the behaviour of these solutions is still novel. There are however many examples of algorithms that do have a memory of best solutions: elitism used in genetic algorithms and genetic programming [1], the global best and personal best solutions are recorded and used to update particles in particle swarm optimisation [2] and the agents in differential evolution only update their solution if the new solution produced is better than their best previous one [26].

This research will explore two methods of addressing this issue: 1) To enable archived novelty scores to decay at each generation. 2) To recalculate the novelty of archived novelty scores at each generation. By simply allowing the novelty scores of recorded behaviours to decay over time, the novelty scores of previously novel behaviours will degrade and therefore be replaced. This novelty decay accounts for the fact that the concept of novelty is inherently temporary and allows algorithms that utilize recorded best solutions to find novel behaviors at subsequent generations. The second method of recalculating archived novelty scores is a more computationally expensive approach, but will have the same effect of accurately updating the novelty of previously observed behaviors.

In order to validate the proposed novelty decay and recalculating archived novelty approaches, this research will apply novelty search to differential evolution (DE) [26]. This is a state of the art evolutionary optimisation algorithm for real valued optimisation problems. This makes it well suited to optimising the weights of a neural network. DE also keeps a record of the best found solutions, which makes it ideal to test out the proposed novelty decay and recalculating archived novelty techniques. Finally novelty search has not yet been applied to DE, this paper aims to utilize novelty search to significantly improve the performance of DE like it has for other algorithms [8, 28]. In order to test the performance of novelty search with differential evolution, this research will consider the maze navigation problem domain which is routinely used to evaluate the performance of evolutionary neural networks and novelty search.

The contributions of this paper are:

(1) To apply novelty search to differential evolution.

*Corresponding Author

(2) The evaluation of two techniques, novelty decay and recalculating archived novelty, to better facilitate the application of novelty search to algorithms with a memory of the best found solutions.

2 BACKGROUND

This section will give brief overview of the areas of: novelty search, neuroevolution and differential evolution.

2.1 Novelty Search

Traditionally in machine learning and optimisation, a solution is judged based on its ability to satisfy an objective. This measure of quality, or fitness, is also what is used to inform and guide the algorithm in its search for better solutions. Novelty search on the other hand, uses the novelty or uniqueness of a solution or behaviour as a means to guide the search process. Novelty search was first proposed by Lehman and Stanley in 2011 [13]. It has since had many successful applications including: controlling underwater soft robots [4], airfoil design [23], classification [21] and computer games [15].

Novelty can be defined as how different a behaviour is with respect to other behaviours. Equation 1 shows how novelty is calculated in novelty search. Where nov is the novelty score given to the current behaviour x, k is the number of nearest neighbours and μ_i is the behaviour observed by the i^{th} nearest neighbour.

$$nov(x) = \frac{1}{k}\sum_{i=0}^{k} dist(x, \mu_i) \tag{1}$$

For the maze problem domain considered in this research, the distance measure between two behaviours is the Euclidean distance between the final positions of the robot for each behaviour. There have been other proposed measures of behavioural distance, e.g. hamming distance, normalized compression distance, entropy, etc [6, 10]. This paper will utilize the Euclidean distance, as it is the natural choice for the maze problem domain.

This research will implement the normalized novelty score proposed by Cuccu and Gomez as it is a more general measure of novelty at each generation [5]. This normalized novelty is calculated using Equation 2.

$$\overline{nov}(x) = \frac{nov(x) - nov_{min}}{nov_{max} - nov_{min}} \tag{2}$$

A sample of some of the current research in novelty search include combining novelty search with deep neural networks to identify interesting differences in images [22], applying novelty search to deep reinforcement learning [3] and applying novelty search to cooperative coevolutionary algorithms [9].

2.2 Neuroevolution

A simple definition of neuroevolution would be the use evolutionary algorithms to search for the optimal neural network configuration to approximate some function [29]. Neuroevolution algorithms have a wide range of applications including Atari games [27], pattern recognition [11] and robotics [12].

Neuroevolution methods generally consist of a population of network configuration. The population of networks iteratively improves over time as a result of the application of evolutionary operators, i.e. selection, crossover, mutation, etc. In neuroevolution, aspects of the neural network design are encoded into genotype. These typically contain network information such as synaptic weight values, number of neurons, connectivity, etc. These network traits form the genotype. These genotypes are evolved over a series of generations. The phenotype is the expression of the genotype. In neuroevolution, the phenotype is the actual neural network.

2.3 Differential Evolution

The Differential Evolution (DE) algorithm is an evolutionary optimisation algorithm [26]. Differential evolution has been successfully applied to evolve neural networks for many different problems, including: control [17], tire contact prediction [7], power generation [16, 19], air conditioning load forecasting [14], CPU forecasting [20] and wind power forecasting [18]. The algorithm uses the principles of evolutionary computing to solve global real valued optimisation problems. The algorithm creates N agents with random solutions to the optimisation problem. At each generation, each agent selects three other distinct agents A, B and C. For each dimension i of the problem, if $rand[0,1] < CR$ or $i = indexR$ the new solution variable $y_i = a_i + F \times (b_i - c_i)$ otherwise $y_i = x_i$. Where CR is the crossover rate, D is the number of dimensions, R is a random index $\in D$, F is the differential weight, $\vec{x}$ and $\vec{y}$ are the current and new solutions of the current agent and $\vec{a}$, $\vec{b}$ and $\vec{c}$ are the current solutions of agents A, B and C. If the fitness of the new solution $\vec{y}$ is better than the previous solution $\vec{x}$, $\vec{y}$ replaces $\vec{x}$. This is repeated until a predetermined number of evaluations has been completed.

One of the contributions of this research is the application of novelty search to differential evolution. Novelty search has already been successfully applied to many other algorithms such as particle swarm optimisation [8] and grammatical evolution [28]. It is hypothesized that differential evolution can also be enhanced by combining it with novelty search. There are currently no examples of this in the literature.

3 NOVELTY DECAY AND RECALCULATING ARCHIVED NOVELTY

The fundamental driving force behind novelty search is that the most novel behaviours survive and move onto the next generation rather than the fittest. This has been shown to be an effective strategy [13].

This research builds on this philosophy by introducing the idea of novelty decay, i.e. enabling novelty scores to degrade as generations pass. Novelty decay accounts for the fact that the concept of novelty itself is a temporary phenomenon. What is meant by this is that something that is "new" now will not remain new indefinitely. Many algorithms keep a record of the best solutions or behaviours in order to inform their search. For example genetic algorithms use elitism and differential evolution uses the agent's position to keep a record of the best solutions and aid the search process. For novelty search, the best solutions are those with the highest novelty score.

Novelty search is commonly applied to the NEAT algorithm [25]. Novelty search with novelty decay may not be appropriate for the

NEAT algorithm however. This is because novelty decay relies on an algorithm keeping a record of the best (most novel) solutions. The original implementation of NEAT does not use elitism as the entire population is replaced by the offspring in the next generation. This is partly why differential evolution was selected to evolve the network weights in this research. DE only replaces the current solutions if they are improved upon. This means that network configurations can be remembered by the DE for many generations if the new solutions do not improve upon the existing solutions. The DE algorithm was also selected because as previously stated, it is a state of the art evolutionary algorithm and there are no examples in the literature applying novelty search to DE.

At each generation of the DE algorithm, each agent generates a new solution (detailed in the previous section). If this solution is more novel than the agent's current solution, the current solution is replaced. Consider however the situation where an agent discovers a very novel (but still sub optimal) behaviour. This solution will have a very high novelty score. This will cause the algorithm to produce new solutions based on this solution with high novelty. This high novelty score creates a problem however when calculated using Equation 1. If the original behaviour has an exceptionally high novelty score, new solutions in subsequent generations with behaviours that have high novelty but not as high as the original behaviour will be ignored as they are not as high as the original behaviour. This is problematic as the algorithm will not consider these the new behaviours. This can be mitigated to some extent by utilizing the normalized novelty score in Equation 2. This places an upper bound on the novelty score so that the maximum novelty score is 1. This normalized novelty score does not fix this problem however as the DE agents will still face the issue of finding novel solutions with a normalized novelty score of 1 that do not improve upon their previous novelty score of 1 and are therefore discarded.

It is this problem that motivates novelty decay. Novelty decay is applied to DE by introducing a δ value that is applied to the novelty scores from the previous generation. This is described in Equation 3.

$$\overline{nov}(x)_t = \overline{nov}(x)_{t-1} \times \delta \tag{3}$$

By enabling the novelty scores to decay over time, the DE algorithm acknowledges that behaviours that were novel in previous generations are not necessarily novel in the future. This novelty δ value stops the algorithm from stagnating on previous novel solutions that were novel in the generation they were first observed but not in subsequent generations. The advantage of novelty decay is that it is computationally cheap and straightforward to implement.

The second method that will be evaluated as a means of addressing the problem of changing archived novelty is the Recalculating Archived Novelty scores (RAN) method. This method simply involves recalculating the novelty scores of all archived behaviors at each generation to check if they are still novel. This method has the advantage of ensuring that novel behaviours will remain in the archive if they remain novel in subsequent generations. The disadvantage of this approach is that it is more computationally expensive. In the case of evolving a neural network with differential evolution, this will result in doubling the number of novelty score calculations.

4 EXPERIMENTS

Novelty search was first proposed and evaluated using a maze environment. The evolutionary neural network must learn to navigate its way through a maze to a target location in a finite number of steps. The nature of the maze environment also lends itself to calculating behavioural novelty, i.e. the coordinates of the robot within the maze. For these reasons, the maze environment seems like a natural choice of problem to test the proposed novelty decay.

4.1 Problem Mazes

In order to validate the proposed novelty decay approach, two mazes will be considered in this paper. Figures 1 and 2 illustrate the normal and hard maze design respectively. The normal maze is more straightforward. The robot's evolved network must navigate from its starting point in the north west corner of the map to the target location in the south east corner of the map. There are various dead ends along the way that the robot must navigate however the robot for the most part is moving in a south easterly direction while meandering around the maze boundaries. The hard map is much more deceptive. The robot is initialized in the south west corner of the map and must navigate to the north west corner of the map where its target is located. As Figure 2 shows however, the robot must first head to the south east corner of the map where there is a narrow opening and then to the north east corner of the map where it can pass through to the target location. The robot must move in a direction away from the target location at many points in the maze in order to finally make it to the target. These maps are similar to those previously used to test real-time NEAT [24] and novelty search [13].

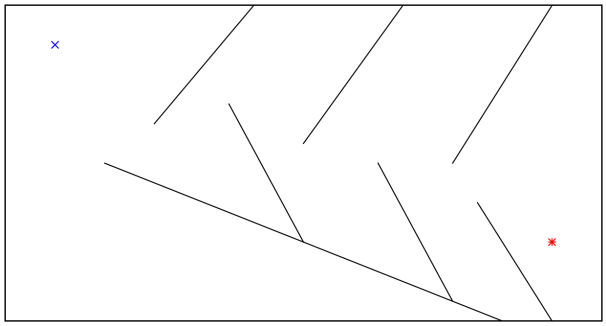

Figure 1: Normal Problem Maze. The robot must navigate its way from the north west of the map to the south east.

4.2 Maze Navigation

The simulated robot is equipped with a number of sensors in order to detect its environment. Figure 3 illustrates these sensors in relation to the robot. These include 6 rangefinder sensors facing: straight ahead, behind, left, right and also diagonally to the front left and right. The purpose of these rangefinders is to detect how far away the robot is from the maze boundaries. The robot also has a pie-slice radar sensor. This allows the robot to orientate its self in relation to the target. One of the 4 radar sensors will activate if the target falls within its detection region, thus telling the robot if

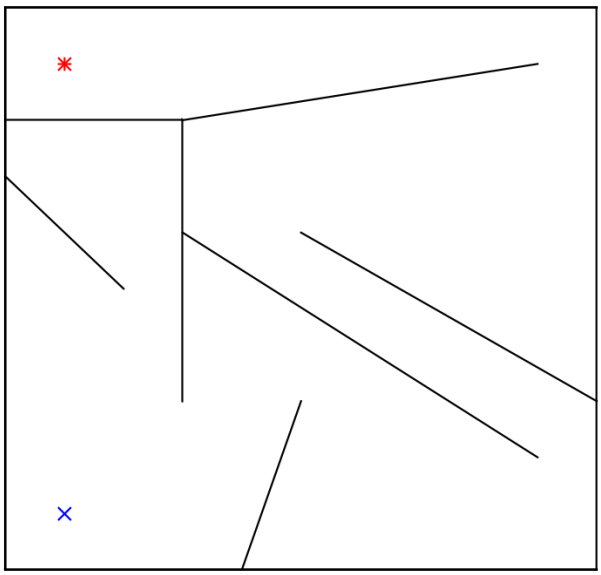

Figure 2: Hard Problem Maze. The robot must navigate its way from the south west of the map to the north west.

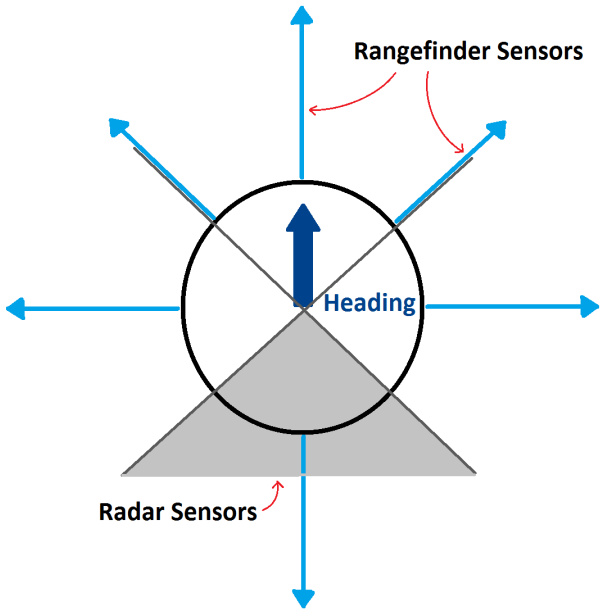

Figure 3: Robot Sensors. The robot has 6 rangefinder sensors and 4 radar sensors.

the target is to the left, in front, etc. Figure 4 depicts the network that must be evolved to navigate the maze. It consists of 10 inputs relating to the 6 range finding sensors and the 4 radar sensors. The network has 2 outputs. These correspond to instructions telling the robot to go left/right and forward/backward. It was found that a recurrent network with 2 hidden neurons was sufficient for the robot to navigate the maze. This corresponds to 28 connection weights that must be optimised.

4.3 Experimental Settings

The fitness of any given network is simply defined as Euclidean distance between the robot's final position and the target location, which the robot must minimize. The robot is determined to have reached the target if its Euclidean distance to the target is less than a predefined amount. The maximum number of network evaluations is 200,000. The DE parameters are: CR = 0.9, F = 0.5 and No. agents = 28.

5 RESULTS

Novelty search with novelty decay performs significantly better on both maps when compared to novelty search without decay. Table 1 highlights the average fitness and standard deviation of each δ value. This table reveals that novelty search with $\delta = 0.8$ performs best on the normal map, while novelty search with $\delta = 0.99$ performs best on the hard map. In terms of statistical significance, $\delta = 0.8$ is the only statistically better decay value for the normal map (P = 0.0079). Other δs have higher average fitness values but are not significantly better. With regards to the hard map, the following δ values perform statistically better than novelty search: $\delta = 0.99$ (P = 0.0006), $\delta = 0.9$ (P = 0.0004), $\delta = 0.8$ (P = 0.0232), $\delta = 0.7$ (P = 0.0054) and $\delta = 0.6$ (P = 0.0111). It is observed that many more δ values perform statistically better on the hard map than on the

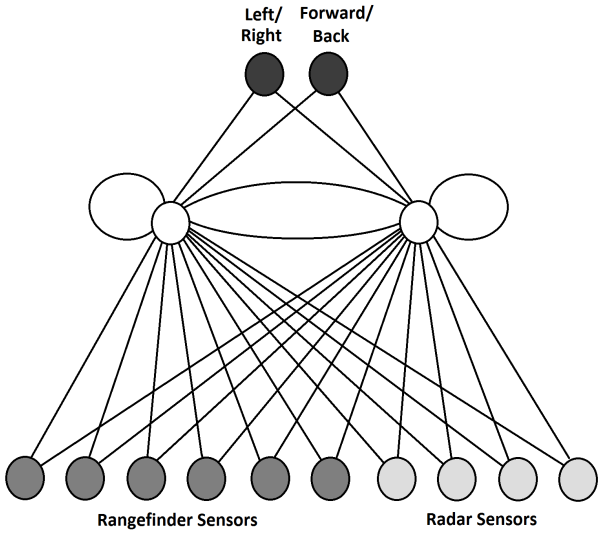

Figure 4: Neural Network Configuration. A fully connected recurrent neural network with 10 inputs, 2 hidden neurons and 2 outputs.

normal map. This implies that novelty decay is more effective as the problem difficulty increases.

The fitness function which has the highest fitness on both mazes is the novelty search with recalculated archived novelty (RAN) scores. Similar to novelty decay, RAN performs statistically better than both novelty search alone and objective based search. When comparing RAN with novelty decay, the average target to distance

achieved by RAN is statistically equal to that of the best performing novelty decay.

Table 1: Average Target Distance of each Fitness Function

Fitness Function	Normal Map Avg	Normal Map StDev	Hard Map Avg	Hard Map StDev
Objective	0.040	0.085	0.107	0.000
Novelty	0.022	0.036	0.041	0.050
RAN	0.005	0.007	0.005	0.013
$\delta = 0.99$	0.017	0.027	0.010	0.023
$\delta = 0.9$	0.020	0.053	0.011	0.009
$\delta = 0.8$	0.006	0.009	0.021	0.022
$\delta = 0.7$	0.011	0.038	0.018	0.009
$\delta = 0.6$	0.028	0.064	0.019	0.019
$\delta = 0.5$	0.017	0.013	0.024	0.026
$\delta = 0.2$	0.035	0.042	0.077	0.041

Figures 5 and 6 illustrate the fitness convergence of novelty search, objective search novelty seach with RAN and finally novelty search with the best performing δ value as determined from Table 1 ($\delta = 0.8$ and $\delta = 0.99$ for normal and hard maps respectively). For both maps it is observed that objective based search converges the slowest, followed by novelty search alone. RAN and novelty decay converge at very similar rates on the normal maze (Figure 5). When comparing convergence on the hard maze (Figure 6), RAN converges faster than novelty decay. Each method converges to a statistically equal distance to target. All variants of novelty search were able to surpass the threshold target distance 0.05 (and therefore complete the maze) for both mazes. Objective based search surpassed the target distance threshold for the normal map but could not escape the cul-de-sac in the more deceptive hard map.

When evaluated on the normal map, novelty search with decay value $\delta = 0.8$ took 24,967 evaluations on average to reach the target distance while novelty search with RAN took and average of 23,762 evaluations to reach the target. Novelty search alone required 44,352 evaluations, nearly double the amount of problem evaluations to reach the target that novelty decay and RAN required. When tested on the hard map, novelty search with decay value $\delta = 0.99$ reached the target distance in 54,517 evaluations. Novelty search with RAN required far fewer evaluations than novelty decay, with a target distance achieved after 28,179 evaluations. Both of these methods are significantly faster at achieving the target distance than novelty search alone, which required 138,801 evaluations.

The number of failed runs for each method is highlighted in Table 2. This is out of a total number of 40 runs. It can be seen that both novelty search with novelty decay and RAN fail to solve the maze on fewer runs than both novelty search and objective search. This table also demonstrates that there is no single fitness function that doesn't fail at least once.

In general, it is observed that a high δ value closer to 1 gives the best performance on both maps. This is to be expected since lower δ values shorten the lifetime of novelty scores more. This leads to the algorithm forgetting very novel behaviours too quickly. An insightful way to view the effect that the δ value has on the novelty score is to look at the resulting novelty score half life, i.e. how many

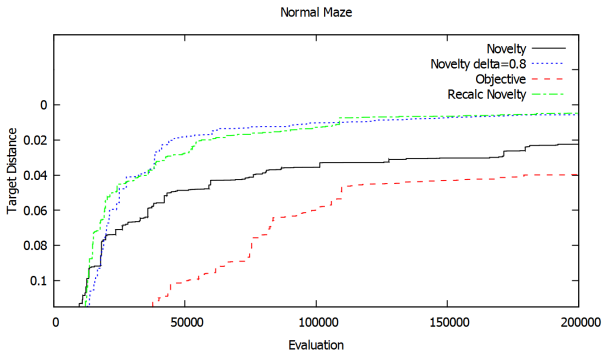

Figure 5: Normal Maze Convergence.

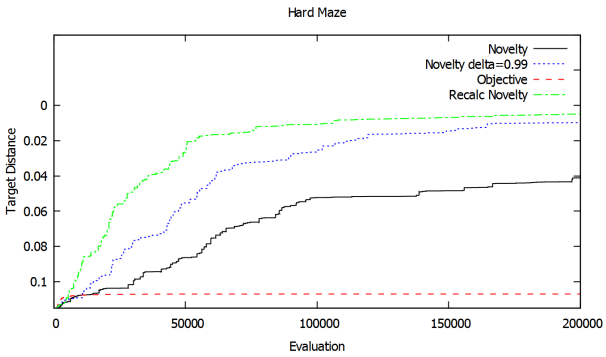

Figure 6: Hard Maze Convergence.

Table 2: Number of Maze Failures for each Fitness Function

Fitness Function	Normal Map	Hard Map
Objective	6	40
Novelty	4	14
RAN	0	1
$\delta = 0.99$	1	2
$\delta = 0.9$	1	0
$\delta = 0.8$	0	2
$\delta = 0.7$	1	0
$\delta = 0.6$	3	2
$\delta = 0.5$	1	3
$\delta = 0.2$	4	25

generations it takes for half of the novelty score to decay. This is the value $T_{0.5}$ where $\delta^{T_{0.5}} = 0.5$. A $\delta = 0.99$ gives the novelty score a half life of approximately $T_{0.5} \approx 69$ generations. While a $\delta = 0.8$ gives the novelty score a half life of approximately $T_{0.5} \approx 3$ generations. A $\delta = 0.99$ gives the novelty score a significantly longer than the half life for the $\delta = 0.8$ value. The values $\delta = 0.8$ and $\delta = 0.99$ gave the best performance on the normal and hard maps respectively. This is a memory in the order of 20 times shorter for the normal map. This implies that a longer memory is more advantageous for harder maps. This raises the question of what δ

value should be selected for problems where the problem difficulty is not known? It is suggested in these cases to select larger δ values as these value always result in statistically better or equal performance to standard novelty search, as Tables 1 and 2 show.

In terms of the computational cost of the proposed methods, RAN is more expensive than novelty decay. On the normal and hard maze, RAN took on average 71 and 136 seconds respectively to complete each run of 200,000 problem evaluations. Novelty decay required 64 and 118 seconds respectively. Standard novelty search required 62 and 119 seconds respectively while objective based search completed the 200,000 problem evaluations in 56 and 107 seconds respectively. RAN is significantly slower to run than all other fitness functions due to the extra computational cost of recalculating the archived novelty scores. Novelty decay adds no significant additional computational cost when compared to novelty search alone. Objective based search is significantly computationally cheaper than all other methods. As the results presented above demonstrate however, objective based search is significantly worse than all other fitness functions.

This research clearly demonstrates that novelty search is compatible with DE. All variants of novelty search outperform standard objective based DE. The results in this section demonstrate that simply implementing novelty search will improve upon objective based search but does suffer from not updating novelty scores of archived solutions. Both methods proposed to address this issue: novelty decay and recalculating archived novelty, significantly improve upon novelty search alone. Each method has advantages and disadvantages. Novelty decay is computationally cheaper and straightforward to implement while recalculating archived novelty scores converges faster. In terms of final distance to target achieved, these two methods perform statistically equal to one another when evaluated over 40 runs. These results are important as the problem of changing novelty scores for archived solutions would occur for other algorithms that incorporate a memory of previous good solutions to inform the search, by using elitism for example. The methods proposed in this paper would be applicable to other algorithms such as genetic algorithms, grammatical evolution, genetic programming, particle swarm optimisation, etc.

6 CONCLUSION

This paper has demonstrated that novelty search can augment the performance of a differential evolution trained neural network. It is demonstrated that simply applying novelty search without updating archived solutions will result in sub optimal performance. Both enabling novelty scores to decay over time and recalculating archived novelty scores at each generation will increase performance. It is observed that problems with different levels of difficulty require different levels of novelty decay for DE to perform optimally. Generally speaking higher δ values perform better as they give the system a longer memory. Novelty decay is computationally cheap and straight forward to implement. Recalculating archived novelty scores converges faster but is more computationally expensive. Each provide statistically equal performance.

The contributions of the paper can be summarized as:

(1) It is demonstrated that the performance of differential evolution is increased by combining it with any novelty search variant.
(2) Two methods of updating the novelty scores of archived solutions are proposed: novelty decay and recalculating archived novelty. Both of these methods significantly improve upon novelty search alone.

A potential route for future work would be to explore dynamically adjusting the δ value during the evolutionary process. Different problems require different δ values for novelty decay to perform optimally. It possible that dynamically adjusting the δ value, as the network is evolved, to adapt to the problem difficulty would remove the issue of selecting the most effective δ value.

It is also hoped to apply the methods proposed in this paper to other algorithms with memory of the best solution such as genetic algorithms, grammatical evolution, genetic programming, particle swarm optimisation, etc.

REFERENCES

[1] Chang Wook Ahn and Rudrapatna S Ramakrishna. 2003. Elitism-based compact genetic algorithms. *IEEE Transactions on Evolutionary Computation* 7, 4 (2003), 367–385.
[2] Daniel Bratton and James Kennedy. 2007. Defining a standard for particle swarm optimization. In *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*. IEEE, 120–127.
[3] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2017. Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents. *arXiv preprint arXiv:1712.06560* (2017).
[4] Francesco Corucci, Marcello Calisti, Helmut Hauser, and Cecilia Laschi. 2015. Evolutionary discovery of self-stabilized dynamic gaits for a soft underwater legged robot. In *Advanced Robotics (ICAR), 2015 International Conference on*. IEEE, 337–344.
[5] Giuseppe Cuccu and Faustino Gomez. 2011. When novelty is not enough. *Applications of evolutionary computation* (2011), 234–243.
[6] Stephane Doncieux and Jean-Baptiste Mouret. 2013. Behavioral diversity with multiple behavioral distances. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, 1427–1434.
[7] Carlos A Duchanoy, Marco A Moreno-Armendáriz, Leopoldo Urbina, Carlos A Cruz-Villar, Hiram Calvo, and J de J Rubio. 2017. A novel recurrent neural network soft sensor via a differential evolution training algorithm for the tire contact patch. *Neurocomputing* 235 (2017), 71–82.
[8] Diana F Galvao, Joel Lehman, and Paulo Urbano. 2015. Novelty-Driven Particle Swarm Optimization. In *International Conference on Artificial Evolution (Evolution Artificielle)*. Springer, 177–190.
[9] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. 2017. Novelty-driven cooperative coevolution. *Evolutionary computation* 25, 2 (2017), 275–307.
[10] Faustino J Gomez. 2009. Sustaining diversity using behavioral information distance. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM, 113–120.
[11] Benjamin Inden, Yaochu Jin, Robert Haschke, and Helge Ritter. 2012. Evolving neural fields for problems with large input and output spaces. *Neural Networks* 28 (2012), 24–39.
[12] Matt Knudson and Kagan Tumer. 2011. Adaptive navigation for autonomous robots. *Robotics and Autonomous Systems* 59, 6 (2011), 410–420.
[13] Joel Lehman and Kenneth O Stanley. 2011. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation* 19, 2 (2011), 189–223.
[14] Gwo-Ching Liao. 2014. Hybrid improved differential evolution and wavelet neural network with load forecasting problem of air conditioning. *International Journal of Electrical Power & Energy Systems* 61 (2014), 673–682.
[15] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. 2013. Enhancements to constrained novelty search: Two-population novelty search for generating game content. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 343–350.
[16] Karl Mason, Jim Duggan, and Enda Howley. 2017. Evolving multi-objective neural networks using differential evolution for dynamic economic emission dispatch. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 1287–1294.

[17] Karl Mason, Jim Duggan, and Enda Howley. 2017. Neural network topology and weight optimization through neuro differential evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 213–214.

[18] Karl Mason, Jim Duggan, and Enda Howley. 2018. Forecasting energy demand, wind generation and carbon dioxide emissions in Ireland using evolutionary neural networks. *Energy* 155 (2018), 705 – 720. https://doi.org/10.1016/j.energy.2018.04.192

[19] Karl Mason, Jim Duggan, and Enda Howley. 2018. A multi-objective neural network trained with differential evolution for dynamic economic emission dispatch. *International Journal of Electrical Power & Energy Systems* 100 (2018), 201–221.

[20] Karl Mason, Martin Duggan, Enda Barrett, Jim Duggan, and Enda Howley. 2018. Predicting host CPU utilization in the cloud using evolutionary neural networks. *Future Generation Computer Systems* (2018).

[21] Enrique Naredo, Leonardo Trujillo, and Yuliana Martínez. 2013. Searching for novel classifiers. In *European Conference on Genetic Programming*. Springer, 145–156.

[22] Anh Nguyen, Jason Yosinski, and Jeff Clune. 2016. Understanding innovation engines: Automated creativity and improved stochastic optimization via deep learning. *Evolutionary Computation* 24, 3 (2016), 545–572.

[23] Edgar Reehuis, Markus Olhofer, Bernhard Sendhoff, and Thomas Bäck. 2013. Novelty-guided Restarts for Diverse Solutions in Optimization of Airfoils. *A Bridge between Probability, Set-Oriented Numerics, and Evolutionary Computation, EVOLVE* 2013 (2013).

[24] Kenneth O Stanley, Bobby D Bryant, and Risto Miikkulainen. 2005. Real-time neuroevolution in the NERO video game. *IEEE transactions on evolutionary computation* 9, 6 (2005), 653–668.

[25] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.

[26] Rainer Storn and Kenneth Price. 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization* 11, 4 (1997), 341–359.

[27] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2017. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. *arXiv preprint arXiv:1712.06567* (2017).

[28] Paulo Urbano, Enrique Naredo, and Leonardo Trujillo. 2014. Generalization in maze navigation using grammatical evolution and novelty search. In *International Conference on Theory and Practice of Natural Computing*. Springer, 35–46.

[29] Xin Yao. 1999. Evolving artificial neural networks. *Proc. IEEE* 87, 9 (1999), 1423–1447.