Exploring the impact of novelty and objective-directed evolution in company with MAP-Elites and HyperNEAT

Author: Jeremy Breytenbach

Supervisor: Geoff Nitschke

March 2024

Version: 1.0

University of Cape Town

UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

Department of Computer Science

Computer Science

Exploring the impact of novelty and objective-directed evolution in company with MAP-Elites and HyperNEAT

Jeremy Breytenbach

Supervisor: Dr Geoff Nitschke

March 2024

Jeremy Breytenbach
Exploring the impact of novelty and objective-directed evolution in company with MAP-Elites and HyperNEAT
Computer Science, March 2024
University of Cape Town
Department of Computer Science
18 University Avenue
Rondebosch
Cape Town
7700

Abstract

Collective robotics refers to the field of robotics that focuses on the coordination and collaboration of multiple agents to perform a task or solve a problem. The ability to automatically design controllers for such agents in a collective system is an attractive proposition. In this thesis we investigate the impact on performance of combining MAP-Elites with HyperNEAT while varying the evolutionary search directive between an objective and non-objective search, and a hybrid approach. We make use of Keep-away, a simulated collective robotics task as a case study. HyperNEAT is an evolutionary method that makes use of indirect encoding to evolve agents. Whereas in typical evolutionary methods, only the fittest agents survive to future generations, the inclusion of MAP-Elites allows not only the fittest agents but also those that demonstrate unique behaviour to survive. By retaining these elite agents in the population, we expect to increase the chances of exploring novel, yet potentially high-performing regions of the search space. To evaluate these methods, we produced controllers for Keep-Away agents whose aim is to retain ball possession in the presence of opposing agents. This research report sheds light on how the combination of these methods affects the agents' performance and their ability explore the behaviour search space. The insights gained from this study will be valuable for researchers working to understand the value and applicability of combining illumination algorithms such as MAP-Elites with objective and non-objective search for gaining performance in Keep-away and similar tasks.

Keywords: *Keep-away soccer, MAP-Elites, HyperNEAT, hybrid search, performance evaluation, neuroevolutionary algorithms, evolutionary computation, novelty search, illumination algorithm, collective robotics.*

Acknowledgement

I extend my heartfelt gratitude to my wife whose unwavering support sustained me throughout this academic journey. My thanks also go to my employer for their support while conducting this research at the same time as attending to their ambitious goals. Special appreciation goes to my supervisor and the researchers whose work has inspired hundreds of interesting conversations with my colleagues, friends, acquaintances, and many countless creative thoughts while considering neuro evolution. This thesis reflects the collective efforts of those before me, and I am honoured to be part of the evolutionary computation community.

Table of Contents

List of Figures

List of Tables

1 Introduction

This research belongs to the fields of Artificial Intelligence and Evolutionary Robotics. Artificial Intelligence (AI) may be broadly described as capability of a machine or computer system to imitate human intelligence. It has been popularly described by Russell, et al. [4]. as "the study of agents that receive percepts from the environment and perform actions". Evolutionary Robotics (ER) is the field in which the principles of natural evolution [5] are applied to the design of robots with AI. A particular research focus in ER is neuroevolution, in which neural network-based controllers are evolved computationally. The overarching objective within the domain is to create agents with the ability to solve problems in a general context.

The neuroevolutionary training process we employ is based on the evolutionary principle of survival of the fittest [5]. The fittest agents in a large population survive and propagate to produce new agents with combinations of their traits, while the weakest agents are removed from the population.

Many benchmark tasks in robotics have been established as testbeds for AI research such as Pole-balancing [6], Knight-Joust [7], and the case study task selected for this research, RoboCup Keep-away Soccer (Keep-away) [3], a collective robotics task. These tasks provide domains for incremental learning because methods developed for solving these benchmark tasks are expected to be applicable in general.

The Keep-away task requires the implementation of an AI method that trains simulated soccer agents to maintain possession of a ball on a virtual soccer field in the presence of opposing soccer agents who aim to intercept the ball and thus end the game.

In this research, the agents will be controlled by an Artificial Neural Network and trained with a Neuroevolution method. Previous work has shown that HyperNEAT is an appropriate neuroevolutionary technique for Keep-away soccer [7, 8]. HyperNEAT can be directed by a host of different optimisation methods which may be broadly categorised as objective and non-objective based search. In this research, we employ Novelty Search [9] as our non-objective search directive. MAP-Elites falls into a class of methods known as illuminating algorithms which aim to illuminate the feature space for a problem by finding and mapping out the elite solutions [10].

This research builds upon a subset of previous work by Didi [8] in which the researchers conducted experiments on the same task with HyperNEAT directed by objective and non-objective-based search. Specifically, previous experiments investigated Objective-based search, Novelty search, Hybrid novelty-objective-based search, Genotypic novelty search (GNS) and Hybrid GNS-objective-based search [7].

This research aims to compare and analyse the difference in the performance of Objective-based search, Novelty-based search, and Hybrid novelty-objective-based search with and without MAP-Elites incorporated into the evolutionary process to evaluate the hypothesis that MAP-Elites will improve behaviour quality of evolved Keep-away agents for all mentioned search methods.

1.1 Primary Research Objectives

The primary objective of this research is to assess the value of incorporating an illumination algorithm into objective, non-objective and hybrid search in the context of multi-agent systems. We will evaluate the impact on task performance and search space exploration Keep-away soccer when combining MAP-Elites with Novelty search, objective search and hybrid novelty-objective search.

1.2 Importance of the research

This research aims to make an incremental contribution to this goal by providing the following benefits:

- Add to the understanding of MAP-Elites as it applies to the evolution of controllers for collective robotic tasks.
- Enable validation of future research by providing experimental data for researchers to make use of in their work.
- Enable researchers performing similar work to begin experimenting sooner by providing software to the research community to build upon in their work.

1.3 Existing research

Recent work in ER has demonstrated the applicability and performance of HyperNEAT in Keep-away soccer where Nitschke, et al. [7] used a variant of HyperNEAT called HyperNEAT Bird's Eye View (HyperNEAT-BEV) [11] to encode the geometric properties of the task.

Nitschke, et al. [7] compared various Evolutionary Algorithms including objective-based search (fitness function) and Novelty search [12] as well as a hybrid of the two (Hybridized novelty and objective-based search). Novelty search is the process of favouring the diversity of phenotypes during evolution. Novelty search has grown in popularity for use in directing evolution towards diverse phenotypes instead of only towards a better fitness value as is done in traditional objective-based search. Recent work has added to the growing weight of evidence that this process improves the quality of evolved behaviours [7].

In recent years the MAP-Elites algorithm has attracted attention and interest in the field of Evolutionary Computation for its potential in improving upon Novelty search [10]. Most notably, the algorithm has been demonstrated to be effective in generating widely diverse walking gaits for hexapod robots [13]. Currently, to the best of our knowledge, no research exists in which MAP-Elites has been applied to the Keep-away soccer task.

1.4 Research contributions

This research makes several notable contributions. Firstly, it replicates key experiments conducted by Didi [8] and Nitschke, et al. [7] specifically focusing on their investigations into Novelty Search (NS), Objective Search (OS), and Hybrid Novelty-Objective-Based Search (ONS) within the context of Keep-away Soccer. Additionally, this research implements MAP-Elites as a method of search space illumination for HyperNEAT evolved controllers. The performance of HyperNEAT is evaluated, considering both scenarios with and without the integration of MAP-Elites, across OS, NS, and ONS-based searches. Moreover, the findings of this research are disseminated to add valuable insights to the existing body of evidence regarding the practicality and effectiveness of neuroevolutionary techniques, with a specific emphasis on the role of MAP-Elites. Lastly, the research contributes to the academic community by sharing the developed software through a published codebase.

2 Background

This chapter introduces and describes the key concepts related to the research topic. The first key concept introduced is RoboCup and the derived subtask known as Keep-away, as the benchmark testbed utilised for generating research data. The next piece of background concerns key concepts in Artificial Intelligence that are applied to collective robotic controller tasks such as Keep-away. Following this, evolutionary algorithms are introduced, and this is followed by more detail regarding HyperNEAT, MAP-Elites, and various search directive approaches to be utilised in the research described in later chapters.

2.1 RoboCup

In recent years, various challenges have emerged as leading areas for practical research in Artificial Intelligence (AI) and Evolutionary Robotics (ER). The idea of a robot playing soccer was first mentioned by Mackworth [14] in 1993. At around the same time, a group of Japanese researchers decided to launch a robot competition which was eventually named the Robot World Cup Initiative, or RoboCup for short [15]. The first official RoboCup games and conferences were held in 1997 [3, 16], and since then, the task has been adopted by many top research groups as a benchmarking testbed for their research [3, 7, 15-24]. The final goal of the initiative remains to beat a World Champion soccer team by the year 2050 [3]. However, this goal remains far off as the community has many challenges to face including hardware, and AI to control the hardware before such an accomplishment would be possible.

2.1.1 RoboCup as an AI benchmark

The game of soccer is characterised by a massive state space [16, 22], and partial observability [25]. The field of play in its most trivial form is a grid of $n \times n$ positions [17] wherein agents must track the distance and angle from themselves to the ball, the boundaries, the cooperating agents, and their opponent agents. RoboCup has drawn comparisons to other benchmarks such as Chess wherein IBM's Deep Blue defeated Kasparov, a human grandmaster [26]. The two games are compared in Table 1 according to their Environment, State Change, Information Accessibility, Sensor Readings and Control. While Deep Blue was able to defeat Kasparov via extensive search techniques (Brute Force methods), it is clear from the comparison that RoboCup is a vastly more challenging task.

2.1.2 Keep-away

To reduce the RoboCup task from the full game of soccer to something approachable via incremental learning, researchers have reduced the game to certain subtasks such as "Keep-away" [24] where two sets of agents compete for possession of a ball. The first set of agents called the keepers aim to keep possession of the ball by either holding or passing it to another keeper, while the second set of agents called the takers aim to intercept a pass or force the ball out of play. The Keep-away field with keeper and taker players is illustrated in Figure 3. In the Keep-away subtask performance is usually measured as the average amount of time that a keeper team can retain possession (measured in seconds) during a benchmark session [24] which we will refer to as Average Hold Time. A second measure of performance is related to the time taken to train an agent to perform at a preselected threshold level [22].

2.1.3 Features of the Keep-away task

Stone et al [17] who first set up the Keep-away task defined it with the injection of random noise to emulate the noisy environment expected in reality. This feature of the task presents a particular challenge in that methods to develop controllers must be capable of dealing with noise. Di Pietro, et. al [21] have pointed out that Evolutionary Algorithms (EA) are applicable to the noisy nature of the Keep-away environment. Further support is found in Darwen's [27] discussion on how evolutionary algorithms are known to work well in noisy and unknown domains.

Another feature of the Keep-away task is that it is a complex collective behaviour task [7] requiring multiple keepers to cooperate in completing their goal of maximising the time they collaboratively retain ball possession. There are several features of collective behaviour tasks that contribute to the challenge of solving them. These reasons have been summarised by Didi [8] as the curse of dimensionality of the state

	Chess	**RoboCup**
Environment	Static	Dynamic
State Change	Turn taking	Real time
Information Accessibility	Complete	Incomplete
Sensor Readings	Symbolic	Nonsymbolic
Control	Central	Distributed

Table 1: Contrasting features of Chess and RoboCup. While Deep Blue was able to defeat Kasparov via extensive search techniques (Brute Force methods), it is clear from the comparison that RoboCup is a vastly more challenging task where brute force methods will not be effective. Table adapted from Kitano, et al. [3].

space, a non-stationary environment, partial environment observability and complex collective behaviour interaction dynamics.

The curse of dimensionality results from the tendency of the solution state space to exponentially increase when multiple agents are involved due to the interdependence between each of their properties and actions on each other. For example, two agents with two variables each acting on each other would result in a state space of order 2^2, while three agents would result in a state space of order 2^3, etc. The non-stationary environment is a result of other active agents being part of the environment. Thus, when they take action, the environment changes. Partial environment observability results from each agent being only able to observe some of the environment. For example, in Keep-away, keepers can only partially observe their position, the other players' positions, and the position of the ball due to noise in the system, and they cannot observe the internal states and decision-making process of takers.

The complex collective behaviour interaction dynamics are a result of each action taken by an agent impacting the environment and hence the decision-making of other agents in the environment, with a continuous knock-on effect.

A further feature of the Keep-away task is the apparent presence of a fractured decision space [2, 8], meaning that the best action an agent can take will vary discontinuously as it moves from state to state. Figure 1 illustrates the challenge of a fractured decision space, given that significantly more complex network structures would be required to deal with more than one discontinuous region of fitness in the search space.

2.1.4 Rules of Engagement

The Keep-away field is initialised with keepers and takers arranged as in Figure 3 and the system is reset to this condition whenever the ball leaves the boundary of the field, or a successful intercept is made. An interception occurs when an attacking agent moves within a predefined distance of the ball.

Researchers increase the task complexity by changing the size of the field of play as well as the number of keeper and taker agents, typically beginning with 3 vs. 2 (keepers vs. takers) and increasing gradually to 6 vs. 5.

Keeper agents can make high-level decisions including holding or passing the ball, as well as moving to the ball, or moving to an open space on the field. The lower-level actions that enable these moves (such as kicking the ball or turning) are handled by the simulation framework. These actions are selected for execution every 100ms long simulation cycle. During each simulation cycle, an agent receives inputs and acts independently in an asynchronous fashion since there is no communication between agents. The takers make use of a heuristic control strategy as given in Algorithm 1 [8] in which two agents are selected arbitrarily to rush towards the ball's estimated position. The rest of the takers estimate open positions between the keepers and attempt to block passes in this space.

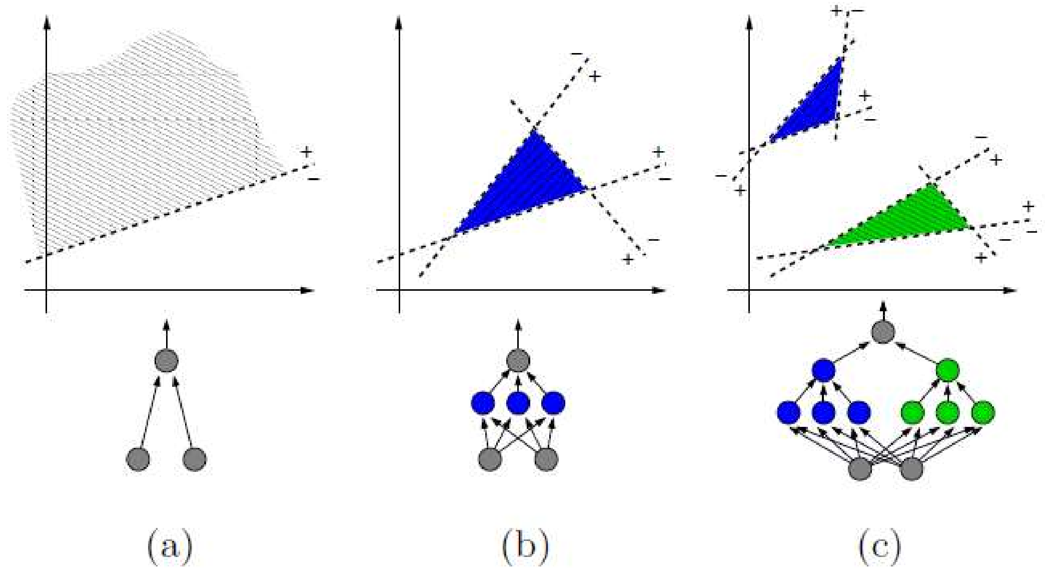


Figure 1: Illustrations depicting the amount of network structure required to bound isolated sections of a 2D decisions space. Figure (a) depicts how a single layer divides the decision space in half. Figure (b) depicts how a triangular section of the decision space can be bounded with the addition of three hidden nodes. Figure (c) depicts the significantly more complex network structure with two hidden layers and 8 nodes that would be required to bound two separate triangular regions. Figure taken from Kohl, et al. [2].

Algorithm 1: Taker agent heuristic control strategy

```
1: Initialize the position of agents; assigns IDs
2: read the taker agent's ID
3: repeat
4:     foreach time_step ϵ episode_duration do
5:       if agent_ID <= 2 then
6:           next_position ← predict(next_ball_position)
7:           policy ←moveTo(next_ball_position)
8:       else agent_Id > 2
9:           next_position ← predict(most_Open_Space)
10:          policy ← moveTo(most_Open_Space) + Intercept(Ball )
11: until Terminal_State
```

2.2 Artificial Neural Networks

Artificial Neural Networks (ANN) have existed since the 1980s [28] and have grown in popularity in recent years thanks to advances in affordable processing power [29]. ANNs are vaguely inspired by our understanding of the human brain but are essentially nonlinear statistical models [29]. Figure 2 depicts an ANN in its simplest form as a single hidden layer feed-forward neural network with an input layer (X_n), a hidden layer (Z_n), and an output layer (Y_n). ANN are characterised by their topology or architecture of nodes, their activation functions, and their weights and biases.

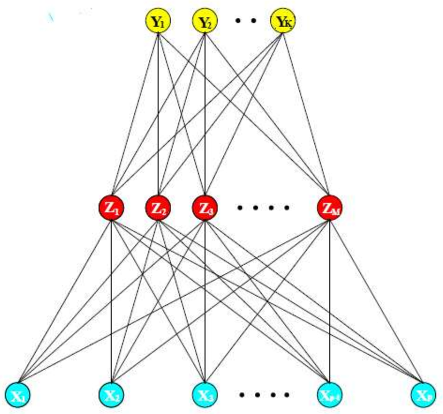


Figure 2: Diagram illustrating a feed-forward neural network with a solitary hidden layer. Figure taken from Friedman, et al. [37]

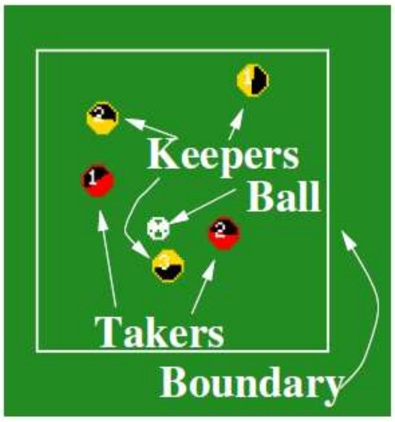


Figure 3: RoboCup Keep-away field illustrating takers, keepers, ball and field. Figure taken from Stone, et al. [22]

Several types of ANNs have been developed to solve different problems, notable are the Recurrent Neural Network (RNN) and the Convolutional Neural Network (CNN). RNNs key feature is that outputs are fed back into earlier layers of the ANN to capture certain dynamics such as those found in cyclical systems [30]. CNNs make use of convolution operations which attempt to mimic the biological visual cortex and a major area of application for CNNs is in image recognition and computer vision due to the way they operate.

An ANN is typically created by first designing its topology and initialising it with random starting weights and biases, and then training the ANN with the use of back-propagation which is a mathematical technique for propagating modelling errors backwards through the network iteratively, usually with the use of gradient descent until a satisfactory error threshold is reached or no more improvement is being made [29].

2.2.1 Artificial Neural Networks in RoboCup

Deep Imitation Learning with the use of Long-Short Term Memory (LSTM) ANNs have been used to tackle the RoboCup Half-Field-Offence subtask [18]. Here, agents are trained by watching teachers perform the task. With the use of this technique, researchers were able to match teacher performance. Similar work by Raza, et al. [25] showed positive results for a defensive subtask with the use of imitation and a classification ANN model, however, they found that a Decision Tree was considerably faster to train while providing comparatively good performance.

2.3 Reinforcement Learning

A growing area of interest that has gained significant momentum in recent years is Reinforcement Learning (RL) in which agents are rewarded or penalised based on certain behaviour characteristics [31]. RL typically makes use of Q-functions in the form $Q(s, a)$ where a reward is calculated based on taking an action (a) while in state (s) [31].

One feature of RL that sets it apart from traditional supervised learning approaches is that it does not require a dataset of correct answers. In supervised learning, an error term can be calculated based on the provided answer set. In RL the agent (for example, an ANN) will typically be initialised with a random topology and parameter set. The first stage of training then comprises the agent taking random actions which in return produce a reward score. RL typically requires a high number of epochs (training iterations) before it stumbles upon some actions that result in a positive reward. The reward function must be carefully designed to enable the agent to continue to improve its reward, but not to prevent it from exploring new possibilities [22, 32].

Common techniques employed by RL are Markov Decision Processes, Monte Carlo methods, Dynamic Programming, linear SARSA(λ) and most recently Temporal Differences [22, 31] which have shown impressive results in the RoboCup problem.

2.3.1 Reinforcement Learning in RoboCup

One such example of RL in the RoboCup problem is described in Stone, et al. [22] where they used the Semi Markov Decision Process (SMDP) version of the SARSA(λ) algorithm [31] to find and maximise the cumulative reward. Temporal Differences were used by Taylor, et al. [32] where they achieved a mean hold time of between 12 and 40 seconds for non-deterministic and deterministic tasks respectively. Taylor, et al. [32] compared this method to the use of the NEAT algorithm (See Section 2.6) which they found performed significantly better.

One of the major weaknesses of RL is its initial training performance [33]. This weakness has been addressed in part by the application of Transfer learning.

2.4 Transfer Learning

Transfer learning (TL) [11, 34], otherwise referred to as policy transfer or behaviour transfer, builds on the intuition that many tasks have similarities, and certain agent skills learned in a source task may be transferable to a similar target task [7]. Typically, in TL, the final few layers of an already trained ANN will be removed and retrained with the new more complex task data.

2.4.1 Transfer Learning in RoboCup

This concept is often used to transfer learning from a simple to a more complex version of a task as has been the case with Nitschke, et al. [7], Taylor, et al. [34] who have applied transfer learning to the RoboCup Keep-away subtask. Nitschke, et al. [7] trained an agent in a simple 3v2 game of Keep-away and then successfully transferred that learning to increasingly complex versions of the same game in which player numbers increased to 4v3, 5v4 and 6v5.

2.5 Evolutionary Algorithms

Back-propagation is the most widely used training technique for ANNs [29] and RL methods have shown great promise in application [31], but recent work by Buk, et al. [35] Stanley, et al. [36]; Doncieux, et al. [37]; Nitschke, et al. [7]; Pietro, et al. [21]; Stanley, et al. [38]; Taylor, et al. [32] has shown the benefits of Genetic Algorithms and Evolutionary Techniques.

Evolutionary Algorithms are modelled loosely on Darwin's theory of evolution [5] in which the most successful genotypes survive and the weakest are filtered out of the population. In EA, a problem is solved by starting with a random population of genotypes (solutions). From this population, selection takes place in which only the best genotypes survive. After this, crossover (reproduction) occurs in which genes from different pairs of genotypes are combined into offspring. Finally, mutation takes place when a random gene from certain offspring is changed randomly. This process repeats iteratively until an optimal solution is found [21]. This process is described algorithmically in Algorithm 2.

Algorithm 2: Computational evolution

```
1. Initialise a random population of solutions
2. Evaluate solutions task performance through simulation
3. Select solutions to reproduce based on fitness criteria
4. Reproduce new solutions by combining features of surviving solutions
5. Mutate genes in solutions (usually randomly)
6. Repeat 2-6
7. Conclude evolution once pre-determined conditions are satisfied, such as a time or fitness limit.
```

2.5.1 Evolutionary Algorithms Applicability in Keep-away

Pietro, et al. [21] have demonstrated the use of EA for training an agent for the Keep-away task and exceeded results from previous studies. They chose the methods due to the knowledge that EA work well in noisy and unknown domains [27], a characteristic of the Keep-away task. They used the same high-level actions that were originally designed by Stone, et al. [17] who introduced Keep-away as a benchmark testbed for machine learning research. In this research, Pietro, et al. [21] were able to demonstrate better performance than Stone's, improving hold time to 24.8 seconds, compared to 12.8 seconds achieved by random decision-making in the agents.

2.6 Neuro-Evolution of Augmented Topologies

The popularity of ANNs and EA have given rise to Neuro Evolutionary (NE) techniques in which ANN weights and synapse connections are determined by EA instead of traditional methods such as back-propagation or RL techniques. A breakthrough that helped to create progress in this research came by way of the algorithm known as Neuro-Evolution for Augmented Topologies (NEAT) [38] developed by Stanley et. al. [38] in 2002.

NEAT is a method for developing and training ANNs with the use of Evolutionary Algorithms (EA) while solving the competing conventions problem which was a long-standing challenge in Neuro Evolution (NE) (the evolution of ANNs). Before NEAT was developed, Neuro-Evolution was typically used to evolve weights and synapses for predetermined ANN topologies. Angeline, et al. [39], Gruau, et al. [40] Yao [41] showed however that the topology of an ANN is significant to its performance. NEAT addresses this by not only evolving the weights and synapses but also providing a way for augmenting the network topology through evolution by adding new neurons and synaptic connections through complexification [38]. The NEAT method achieves complexification by starting with small, simple networks which it evolves into larger, more complex networks while maintaining features of both parents in the topology.

This capability is a powerful advantage of the NEAT method because it provides a way to solve the Competing Conventions Problem, also known as the Permutations Problem commonly faced in NE [38]. This problem is characterised by damaged offspring due to duplicate genes with different encodings being propagated as illustrated in Figure 4.

NEAT solves this problem through the implementation of speciation in which incompatible phenotypes are grouped into separate species. This is achieved through NEAT's Historical Marking mechanism which enables the algorithm to keep track of evolving phenotypes using a global Innovation Number (GIN) assigned to each new synapse and neuron that is added to a network. Figure 5 illustrates how the algorithm uses this GIN to track network lineage and the order in which networks are evolved to create more diverse networks and protect innovation. The GINs are used during the crossover phase of evolution where the NEAT algorithm uses this information to ensure that offspring inherit only one of each unique synapse and neuron from its parent's combined genetics. [38]

One possible weakness and an open problem in the field is whether complexification can go too far and result in overly complex networks. However Stanley, et al. [38] argue that this is avoided through justified, incremental structural mutation in which only the useful structural additions survive.

2.7 HyperNEAT and Compositional Pattern-Producing Networks

Hypercube-based Neuro-Evolution of Augmenting Topologies (HyperNEAT) [36] extends NEAT by evolving Compositional Pattern-Producing Networks (CPPN)[1] with NEAT to determine the connection weights of a network substrate with a fixed topology that is in turn used to control agents or complete a task. The word "Hypercube" represents the typical multidimensionality of CPPNs. The advantage of this design is that the connectivity pattern of the substrate can directly map to the geometry of the solution space (e.g. the sensors and actuators of a robot) and remain unchanged, while evolution takes place on the CPPN with the freedom to complexify and update its topology. CPPNs are analogous to traditional ANNs in that they have similar topologies composed of neurons and synaptic connections with their associated

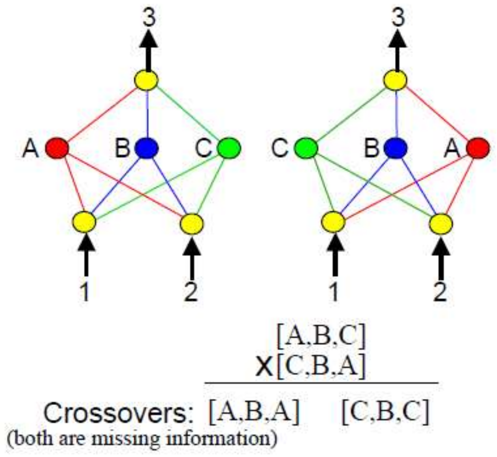


Figure 4: The competing conventions problem. Figure taken from Stanley, et al. [33] The figure shows two networks that result in the same output even though they are composed in different arrangements of the same three neurons (A, B, C) and their synaptic connections. Performing single-point crossover would inevitably result in the loss of one of the three hidden neurons and its corresponding synaptic connections, leading to the emergence of a network that is either ABA without C or CBC without A.

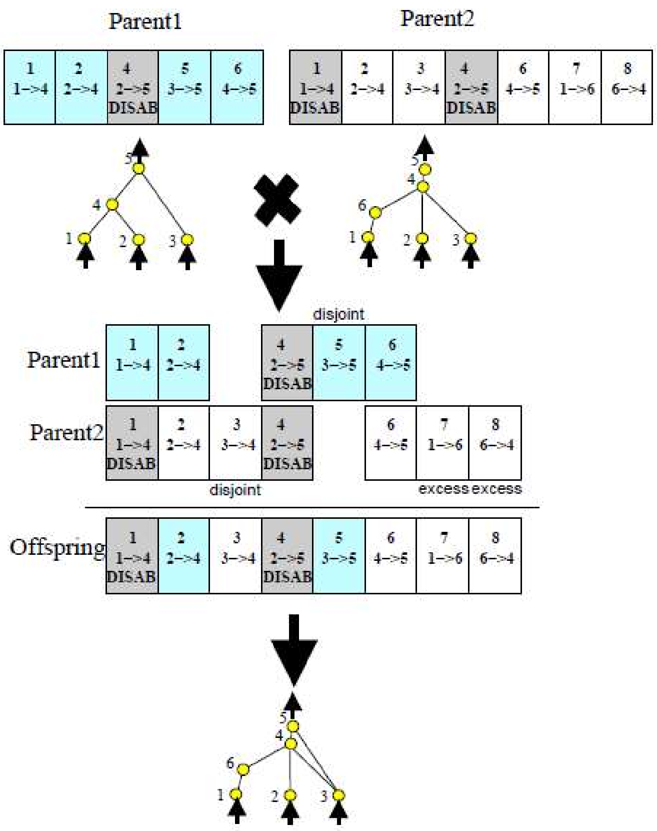


Figure 5: Aligning genomes with distinct network structures through the use of innovation numbers. Figure taken from Stanley, et al. [33]. Each block represents a genotype neuron with its GIN and an indication of which neurons it links to via synaptic connections (each of which has its own GIN not shown in the figure). Offspring are generated by aligning each GIN and randomly selecting from one of the parents if the GIN is represented in both, thus resulting in a more complex network which represents both parent's innovations (neuron 6 from Parent 2, and synapse 3->5 from Parent 1).

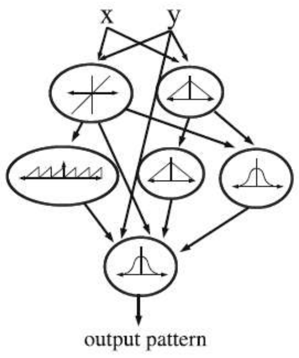


Figure 6: Composition of functions as a graph. Figure taken from Stanley [1] Each node takes on the form of a potentially different activation function.

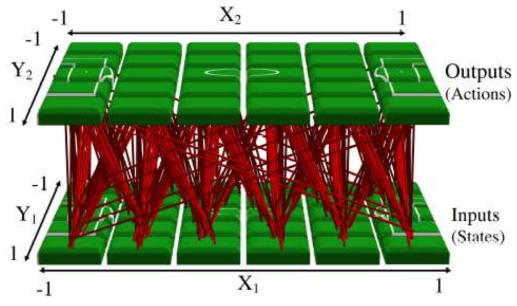


Figure 7: Birds Eye View of the Keep-away field implemented for use with HyperNEAT where the input and output layer form a one-to-one mapping of the field of play as a fully connected ANN with no hidden layers. Figure taken from Verbancsics et. al. [9].

weights. However, CPPNs differ in that typical ANN neurons all employ a single activation function (such as sigmoid or Gaussian functions), whereas CPPNs may use any variety of different activation functions in each node as illustrated in Figure 6. Patterns result in the output of CPPNs due to the function composition in which the output of each node is transformed by the function in the next node. The use of different functions can result in useful patterns of behaviour. For example, a sine function can introduce regular repetition, and a Gaussian function can introduce bilateral symmetry [8].

With the use of CPPNs, HyperNEAT can map the high dimensionality in a hypercube to a low dimensional substrate in the ANN topology. This characteristic makes the technique highly applicable to tasks such as Keep-away [7] because the field of play can remain the same as evolution runs its course and the CPPN is complexified in parallel to an unchanging substrate.

2.8 Static Representation in Evolution

Verbancsics and Stanley [11] have noted the value of being able to evolve the 'intelligence' in agents whilst leaving the domain in which the intelligence is applied unaltered. For example, in robotics, it is typical that a controller ANN will have nodes that map directly to the physical robot's joints, actuators, etc. This means that the controller is limited in application to robots of the same physical makeup. It would be useful to be able to use the same controller with little adaption in a different robot that might have a similar objective but a different physical design.

This concept applies strongly to the Keep-away and Robot Soccer tasks in general in that the physical makeup of the task changes when more players are added to the field, and when the size and dimensions of the field change.

Thus, it is useful to be able to maintain a distinction between the properties of the task, and the evolving network. In the case of Keep-away, this principle has been applied as described in HyperNEAT in Keep-away.

Recent work in Evolutionary Robotics (ER) has demonstrated the applicability and performance of HyperNEAT in Keep-away where Nitschke, et al. [7] used a variant of HyperNEAT called HyperNEAT Bird's Eye View (HyperNEAT-BEV) [11] to encode the geometric properties of the Keep-away task. The mapping of inputs to outputs by way of this approach is illustrated in Figure 7. This figure shows a Birds Eye View of the Keep-away field implemented for use with HyperNEAT where the input and output layer form a one-to-one mapping of the field of play as a fully connected ANN with no hidden layers.

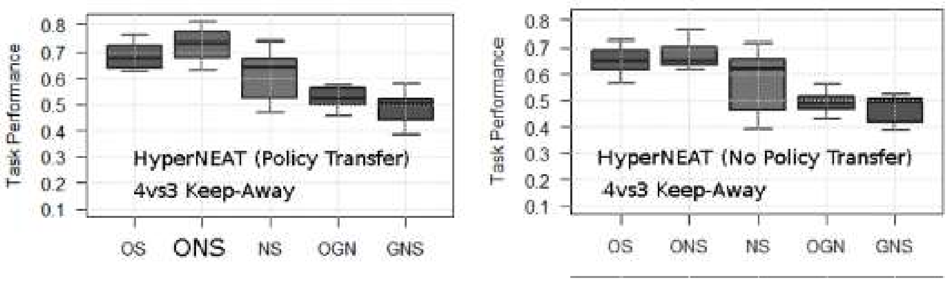


Figure 8: Average normalized maximum task performance for HyperNEAT variants. Figure taken from Nitschke, et al. [3]. The left column shows task performance with transfer learning (Policy Transfer) applied. Task performance is compared for Objective-Based (OS), Novelty Search (NS), Objective-Novelty hybrid (ONS), Genotypic Novelty Search (GNS), and Objective-GNS hybrid (GNS).

Nitschke, et al. [7] compared various EA methods including objective-based search (fitness function) and Novelty Search (NE) [12] as well as a hybrid of the two (ONS). NS is a special case of the Genotypic Diversity Method (GDM) [42] which is the process of favouring the diversity of phenotypes during evolution. NS has grown in popularity for use in directing evolution towards diverse phenotypes instead of only towards a better fitness value as is done in traditional objective-based search.

Recent work has added to the growing weight of evidence that this process improves the quality of evolved behaviours [7]. A further contribution of the work by Didi [8] and Nitschke, et al. [7] was the application of TL to NE while TL had traditionally only been applied to RL. Didi [8] and Nitschke, et al. [7] found that *"Hybridized novelty and objective-based evolutionary search used in company with policy transfer across increasingly complex collective behaviour tasks, results in significantly higher behaviour quality compared to other evolutionary search methods."*. Figure 8 shows the distribution of task performance for each of the various approaches used to train an agent for increasingly complex Keep-away tasks over 20 runs. The figure shows that the hybridized novelty and objective-based evolutionary search (ONS) consistently outperformed the competing approaches and that TL improved Task performance for ONS in all cases.

2.9 Multidimensional Array of Phenotypic Elites (MAP-Elites)

Multidimensional Array of Phenotypic Elites (MAP-Elites) is an illumination algorithm or quality diversity algorithm [43] that creates a map that shows the relationship between behaviour and performance along predefined behaviour-space dimensions. This algorithm's core function is to create a map of high-performing solutions (the elites) at each point in a behavioural space defined by behavioural dimensions [44]. This concept is illustrated in Figure 9 in which each elite is mapped to a discretised feature landscape.

An early application of MAP-Elites was hypothesised for use in the control of robots that would benefit from "knowing" how to achieve optimal performance through varied behaviours. For example, if a hexapod robot were to lose two legs – there is considerable benefit in already knowing what the best (highest performing) movement strategy would be to still complete its mission [44].

The MAP-Elites algorithm is described in pseudocode in Algorithm 3 which has been adapted from Mouret, et al. [10] and Tarapore, et al. [44]. In this algorithm, x represents a candidate solution and thus performance(x) is its performance, each N behaviour_descriptor(x) is chosen by the user and represents.

the chosen dimensions to be mapped into the feature-performance map. Each of these chosen dimensions are discretised based on the user preference, thus creating an N-dimensional grid where each cell represents a particular niche where elites may be recorded and preserved through evolution.

There is significant overlap philosophically in the thinking behind MAP-Elites and Novelty search (2.10.2) which seeks novel behaviours. The difference with MAP-Elites is that it not only rewards novelty in evolution (albeit implicitly [44]), but maps out the relationship between objective performance and pre-defined behavioural dimensions of interest, thus allowing users of this algorithm to find the best of each type of behaviour they are interested in (for example, the fastest, or most reliable).

Algorithm 3: A pseudocode description MAP-Elites

```
procedure MAP-Elites
1.  $(P \leftarrow \theta, X \leftarrow \theta)$
2.  for $iter = 1 \rightarrow I$ do
3.      if $iter < G$ then
4.          $x' \leftarrow random\_solution()$
5.      else
6.          $x \leftarrow random\_selection(X)$
7.          $x' \leftarrow random\_variation(x)$
8.      $b' \leftarrow behavior\_descriptor(x')$
9.      $p' \leftarrow performance(x')$
10.     if $P(b') = \theta$; or $P(b') < p'$ then
11.         $P(b') \leftarrow p'$
12.         $X(b') \leftarrow x'$
13. return feature-performance map (P and X)
```

1. *Create an empty, N-dimensional map of elites: (solutions X and their performances P)*
2. *Repeat for I iterations*
3. *Initialise by generating G ...*
4. *... random solutions*
5. *All subsequent solutions are generated from elites in the map*
6. *Randomly select an elite x from the map X*
7. *Create x', a randomly modified copy of x (via mutation and/or crossover)*
8. *Simulate the candidate solution x' and record its feature descriptor b'*
9. *Record the performance p' of x'*
10. *If the appropriate cell is empty or its occupants' performance is ≤ p', then,*
11. *store the performance of x' in the map of elites according to its feature descriptor b'*
12. *Store the solution x' in the map of elites according to its feature descriptor b'*
13. *Return the feature-performance map (P and X)*

2.9.1 Quality Metrics for MAP-Elites

Four quality metrics were introduced by Mouret, et al. [10] and are summarised below.

1. **Global performance** represents the proximity to the highest-performing solution across all runs. This is analogous to the performance of each solution compared to the highest possible performance discovered so far.
2. **Global reliability** is the average performance of the filled cells or niches in the feature-performance map (0 where cells are empty) across all runs for the same task.
3. **Precision (opt-in reliability)** is the average performance of the filled cells or niches in the feature-performance map of a particular run.
4. **Coverage** measures how many cells or niches of the feature-performance map a particular run of MAP-Elites can fill of the total possible.

In addition to these metrics, Mailer, et al. [45] also considered the maximum performance, average performance and coverage of the maps during generation. In this research, we can consider maximum performance to yield similar results and give rise to similar conclusions as those gained from global performance, and we can consider precision to be an alternative version of global reliability that does not add value to our particular analysis.

2.10 Evolutionary Search Variants

Evolutionary algorithms are directed by an objective function that defines agent fitness; this fitness is then used in the selection process where only some agents survive to influence future generations. There are various ways to direct an evolutionary algorithm, and each approach has a major effect on the performance and behaviour of the agents produced. Eiben, et al. [46] have shown that the two main properties of evolutionary search approaches that drive the search process are exploration of the search space and exploitation of highly fit regions.

In this research, we consider three approaches to direction in order to evaluate the effect of exploration and exploitation and a hybrid of the two.

1. Objective-based search (OS)
2. Novelty search (NS)
3. Hybrid Objective-Novelty search (ONS)

Objective-based search is comparatively strong in the exploitation of highly fit regions. Novelty search is comparatively strong in the exploration of the search space, and in some sense, these two methods represent the two extremes in the spectrum of search direction approaches. Hybrid Objective-Novelty

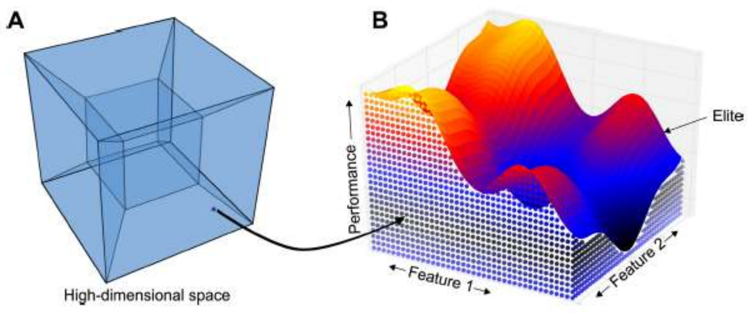


Figure 9: Figure taken from Mouret, et al. [8]. The figure depicts how MAP-Elites maps a high dimensional space in A to a lower dimensional 3D space in B by mapping task performance to one dimension, and various features of interest (Feature 1 and Feature 2 in this example) to the 2nd and 3rd dimensions.

search then represents an attempt to balance these approaches in a single evolutionary process. Each of these approaches are described in the following sections.

2.10.1 Objective-based Search

Evolutionary algorithms typically use goal, or objective-based fitness functions that aim to direct evolution as linearly as possible towards fitter solutions (those solutions that are closer to the pre-determined objective). In the case of Keep-away, objective performance is defined as the hold time of the keepers. Thus, with the objective search approach, the fittest agents are those with the greatest average hold time. The fitness function can thus be described as mean episodic length, as in equation (1):

$$fit_x = \frac{1}{N}\sum_{j=1}^{N} T_j \tag{1}$$

In this equation, the length of an episode x is denoted by T_x, and N is the number of task trials, T_j is the length of the task trial *j*. The nature of objective search is that it will drive the solution as directly as possible towards the highest performance. As we will see, however, when discussing Novelty search, this direct approach can sometimes fail to find novel solutions that may lead to better performance, given the chance.

Lehman, et al. [9] have described a typical challenge in evolution with objective search which is that it may fail to select agents that underperform in the current population but have traits that could result in high-performing agents in future generations if given the opportunity to pass these traits along to their children.

Consider that certain traits exhibited by underperforming agents could undergo evolutionary development over successive generations, leading to the emergence of higher-performing proficient agents. These intermediate generations of relatively poor performance that are important for future leaps in performance can be considered intermediate stepping stones.

By way of example consider the sporting discipline of high jump. Traditionally, athletes would approach the task by leaping over the bar akin to hurdling. The adoption of any alternative technique would have hindered athletes' progression in their athletic career as this would have been seen as a senseless divergence from what was known to be effective by their coaches.

However, we now know that there is a superior technique to reach higher scores, namely the Fosbury flop, characterised by a backwards leap. Although initially unconventional, this novel approach has proven to be remarkably effective, surpassing its predecessors.

As in the example, it is plausible that within the population of agents, there exists an individual executing a novel strategy that currently yields unsatisfactory performance in comparison to the best-performing solutions but has the potential to be a stepping stone to greater performance. It is conceivable that a specific trait inherent in this agent possesses latent potential, capable of allowing future generations to achieve unprecedented levels of performance.

2.10.2 Novelty Search

Novelty search was first described by Stanley et al [9] as an alternative approach to objective search in which novelty becomes a proxy for the intermediate steppingstones beneficial in evolution that lead to greater objective performance.

With this non-objective evolutionary directive, novelty is rewarded, sometimes at the expense of linear increases in objective performance, but often resulting in higher objective performance because of the hidden performance features that may be behind a series of evolutions that objective search would not discover.

Novelty search ascribes innate value to new behaviour, irrespective of whether they are the best solutions. Instead, solutions are evaluated based on their divergence from the existing behaviours already discovered throughout evolution.

When employed as an objective function for use in HyperNEAT, Novelty search takes the form given in equations (2) and (3).

$$Nov = \frac{1}{3k}\sum_{i=1}^{k}\sum_{j=1}^{3}\delta(x_j, y_{ij}) \qquad (2)$$

$$\text{where,} \quad \delta_i(x,y) = \left\| x_i - y_{ij} \right\| \qquad (3)$$

In this equation x_j represents the j^{th} behavioural property of a genotype, similarly y_{ij} is the j^{th} behavioural property of the i^{th} nearest neighbour of genotype *x*. δ is the behavioural distance between two genotypes *x* and *y*. k is the number of nearest neighbours and is user specified. Thus Nov_x is the Euclidean distance between a genotype's chosen behaviour properties and k of its nearest neighbour's behaviour properties. Hence, the further away that a genotype is from other genotypes in the search space with regard to their behavioural properties, the higher the Novelty of that genotype.

One of the challenges associated with novelty search could be the potential delay in generating solutions of superior performance despite its capacity to produce a wide array of diverse and interesting (novel) solutions. It may take a comparatively long time to produce comparatively high-performing solutions to reach a global optimum.

Thinking about this using the analogy of high jump once again, novelty search might result in jumpers who jump backwards, dive over forwards, try to walk on their hands (rather than feet), those that roll towards the jump, and so on. These diverging novel behaviours may over time result in high-performing solutions, but the majority will tend to be bad solutions, thus simply distracting the process, and causing a 'wild goose chase'.

2.10.3 Hybrid Objective-Novelty Search

Studies have shown that controller evolution to solve complex collective behaviour does not perform well when either Novelty or when Objective search is employed [7]. Instead, various studies have found that a hybrid approach can result in the evolution of controllers capable of good performance [47-49]. Hybrid search is a linear combination of objective and other search methods such as Novelty, which will be the case in this research contribution. This process is intuitively a powerful combination, allowing the linear performance-biased objective search to drive evolution towards continuously stronger solutions while allowing the Novelty search to incorporate interesting new (novel) behaviours that could result in new evolutionary leaps forward.

Hybrid search when employed as an objective function takes the form given in equation(4).

$$score_i = \rho \cdot \overline{fit_i} + (1-\rho) \cdot \overline{nov_i} \qquad (4)$$

In this equation, fit_i is the objective-based search fitness, and nov_i is the novelty score (as per equation (2) of the i^{th} genotype. $\rho \in [0,1]$ is a user-defined hybrid scalar value that is used to control the relative contribution of normalised objective and novelty-based fitness in the hybrid score ($score_i$).

2.11 Discussion

This research aims to build upon the work reviewed in this literature survey which has highlighted the capabilities and potential weak aspects of various approaches to evolving high performing agents in collective behaviour tasks such as Keep-away.

The Keep-away task presents difficulties due to the curse of dimensionality of the state space and its dynamic, real-time nature; coupled with the partial environment observability, and the complex collective behaviour interaction dynamics.

Another common challenge appears to be in effectively determining what the behaviour search space looks like in an intuitive way that could give rise to useful information about the task (such as which behaviour is most important for success). Didi [8] tackled this challenge with the use of Kohonen Self-Organising Maps (SOMS).

MAP-Elites presents a unique approach to illuminating the behaviour search space in a way that can be visualised effectively in 2 or 3 dimensions if no more than three behavioural descriptors are selected.

There appears to also be a gap in the literature regarding the effectiveness of combining MAP-Elites as an illumination algorithm with novelty search, objective search, and hybrid search. These gaps raise questions for us as to whether it would be useful to combine MAP-Elites and novelty search, given that there is a similar goal at the heart of both approaches to augmenting neuro evolution.

Having perused the literature, we wonder whether MAP-Elites improves, hinders, or impacts task performance in Keep-away agents evolved with HyperNEAT in unexpected ways. Further to this, we wonder what the impact of MAP-Elites is on HyperNEAT evolved Keep-away agents when evolution has been directed by objective-search, novelty-search, and hybrid search.

3 Methods and Experiments

This chapter describes the research methods used to obtain experimental data and understand it so as to draw robust statistical conclusions about the impact of HyperNEAT, MAP-Elites, and various search directives for evolution.

In the first stage of this study, MAP-Elites will be programmed into an existing codebase that contains an implementation of HyperNEAT with options for Novelty, Objective, and Hybrid novelty-objective-based search. In the second stage, keeper agents will be trained with the use of this implementation of ME-HyperNEAT, as well as with an existing implementation of HyperNEAT with Novelty, Objective, and Hybrid Novelty-objective based search. The agents will be trained on 2 vs. 3 Keep-away soccer only, so as to limit the scope of the study to manageable dimensions, although future work should consider how increasing task complexity impacts behaviour. Training is conducted with the use of the same simulation framework used in previous work that this research is based on and takers will be controlled by the same heuristic controller as has been used in previous research.

This simulation engine is described in section 3.1. We then made use of the evolution framework also used in previous work by Didi [8] to evolve our controllers as they attempt to perform the Keep-away task in the simulation engine. However, in order to make use of this framework for our purposes, some modifications were required. The most critical modification made was the addition of the MAP-Elites algorithm to the evolution framework. The software architecture and design of this addition is described in section 3.2.1.

Finally, once the effectiveness and efficiency of the two techniques have been measured and recorded, their performance will be compared and analysed so as to accept or reject the null hypothesis.

3.1 Keep-away Simulator

In this research we made use of the RoboCup simulator [1] developed by Phillip Verbancsics [50]which was based on and translated from the work by the RoboCup Soccer Server Maintenance Group[2].

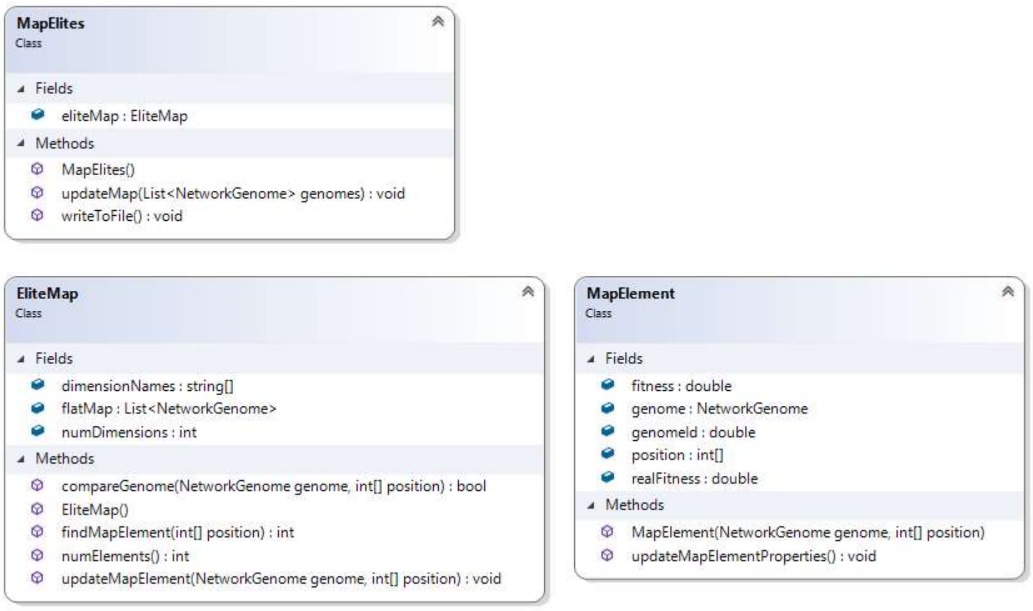
This simulation software is comprised of an object orientated architecture comprised of a Server class, and various other classes to which calls are made, including Ball, Field, Player, Referee, Stadium, etc. The Server drives the simulation synchronously from kick-off until the Referee decides that the game is over (typically because the ball has been intercepted by a Taker). In each simulation cycle players on the field make a move, which may include moving towards the ball, moving to a new position, or kicking the ball in an attempt to pass the ball to another player. During each cycle, the balls movement is simulated, and the referee analyses the state of play in order to enforce the rules of play and decide when the game is over. Takers are guided by a heuristic controller, while Keepers are guided by an evolved controller.

The evolution framework reads the state of the game including player locations, ball location, etc. from the simulation server and uses this together with the evolved agent to determine player actions. These player actions are then fed back into the simulation by the evolution framework and a new cycle of simulation takes place.

3.2 Evolution Framework

In this research we made use of the existing HyperNEAT implementation also developed by Phillip Verbancsics and modified by Sabre Didi for his thesis work entitled *Neuro-Evolution Behavior Transfer for Collective Behavior Tasks [8].* This evolution framework is incorporated into the same software project as the RoboCup simulator as an object orientated architecture and is comprised of classes for all key aspects of the HyperNEAT algorithm. These classes include the base Program, and supporting classes such as ComputeNovelty, Diversity, Utilities, etc.

[1] https://github.com/jeremybreytenbach/KeepawaySim/tree/devJeremy
[2] https://github.com/rcsoccersim/rcssserver



```
MapElites
Class
Fields
    eliteMap : EliteMap
Methods
    MapElites()
    updateMap(List<NetworkGenome> genomes) : void
    writeToFile() : void

EliteMap
Class
Fields
    dimensionNames : string[]
    flatMap : List<NetworkGenome>
    numDimensions : int
Methods
    compareGenome(NetworkGenome genome, int[] position) : bool
    EliteMap()
    findMapElement(int[] position) : int
    numElements() : int
    updateMapElement(NetworkGenome genome, int[] position) : void

MapElement
Class
Fields
    fitness : double
    genome : NetworkGenome
    genomeId : double
    position : int[]
    realFitness : double
Methods
    MapElement(NetworkGenome genome, int[] position)
    updateMapElementProperties() : void
```

Figure 10: MAP-Elites Class Diagram. The MAP-Elites implementation is made up of three classes. MAPElites is the main class and contains the instantiation of the map itself, which is described by the EliteMap class. The EliteMap class then makes use of the MapElements class which makes up each discrete cell in the map. Methods available in the three classes allow for genomes in the map to be compared, updated and replaced as evolution takes place.

3.2.1 MAP-Elites Implementation

One of the contributions of this research was to add MAP-Elites as an optional feature to the evolution framework. The MAP-Elites implementation is made up of three classes. The architecture of this class-set is provided in Figure 10.

3.2.2 ME-HyperNEAT Algorithm

The combination of HyperNEAT and MAP-Elites gives rise to what we have called ME-HyperNEAT. In order to implement ME-HyperNEAT and integrate MAP-Elites into the evolutionary process, it was necessary to modify the existing HyperNEAT code in two ways. First, to add an alternative select method that implements the MAP-Elites select functionality wherein elites (those agents that occupy unique positions in the Elite Map) are included in the selection for further evolution.

The second change was to implement an updateMAP method that acts at each generation of evolution wherein unique agents are stored in the Elite Map.

3.2.3 Performance and behavioural metrics

In this research we make use of selection of domain standard performance metrics so as to aid comparison with peer research. For evolutionary performance we focused analysis on search fitness, and task performance, and for MAP-Elites consideration we focused on global reliability, and coverage along with maximum task performance and average performance. The key performance metrics used are described in Table 3.

3.2.4 Analysis infrastructure

As part of the work completed, we developed a MATLAB based analysis infrastructure that would enable efficient analysis of the resultant simulation and evolution research data. This infrastructure is intended to be usable by future researchers.

Class name	Type	Description
EliteMaps.cs	Class	The N-dimensional map of elites, made up of MapElements.
MapElement.cs	Class	MapElements represent each discreet cell or location in the N-dimensional map.
MapElites.cs	Class	The parent class that implements the MAP-Elites algorithm.

Table 2: MAP-Elites Classes. The MAP-Elites implementation is made up of three classes. MapElites is the parent class tha implements the MAP-Elites algorithm. This class makes use of the EliteMap and MapElement classes to record and store genomes in the discretised N-dimensional map of elites.

Category	Performance metric	Description	Range
HyperNEAT	Search fitness (OS, NS, ONS, ME-OS, ME-NS, ME-ONS)	Search fitness describes how well the genome is able to meet the chosen evolutionary directive. Each variant has a different evaluation function, each is described in section 2.10	24.5 < OS < 107.9 15.9 < NS < 414.8 13.1 < ONS < 308.2 13.1 < ME-OS < 18.9 7.1 < ME-NS < 14.1 8.3 < ME-ONS < 37
RoboCup Keep-away Soccer	Task performance (TP)	Task performance is the measure of Keep-Away agents average hold time, measured in seconds.	9.6 < TP < 22.8
MAP-Elites	Global reliability (GR)	The average performance of the filled cells or niches in the feature-performance map (0 where cells are empty) across all runs for the same task.	0 < GR < 100%
	Coverage (C)	How many cells or niches of the feature-performance map a particular run of MAP-Elites can fill of the total possible.	0% < C < 100%

Table 3: Performance metrics without normalisation. The table describes our selected performance metrics and categorises them according to the particular area of study they below. Throughout evolutionary search, HyperNEAT makes use of Search fitness to direct evolution. The key metric in RoboCup Keep-away soccer, and our primary metric of interest is Task performance, which is a measure of how long keeper agents are able to keep the ball in their possession. We measure MAP-Elites through Global reliability and Coverage, which represent the reliability of solutions in the map, and the number of elites that have been discovered, respectively.

3.2.5 Experiments

The principal aim of this study is to investigate the efficacy of integrating an illumination algorithm into objective, non-objective, and hybrid search strategies within the context of multi-agent systems. Our focus is on assessing how this integration influences task performance and exploration of the search space in Keep-away soccer scenarios when employing MAP-Elites in conjunction with Novelty search, objective search, and hybrid novelty-objective search methods.

Our primary methodology involves evaluating the performance metrics of simulated Keep-away soccer agents that have undergone evolution using a version of HyperNEAT combined with MAP-Elites, which we have called ME-HyperNEAT. This investigation is conducted through experimentation, with the comparison of outcomes against a null hypothesis formulated as follows:

Search method	**Hybrid combination ratio**	**Select percentage**	**Task Performance**
OS	N/A	10%	91.5
OS	***N/A***	***30%***	***107.9***
OS	N/A	50%	107.9
OS	N/A	70%	28.4
OS	N/A	90%	24.5
NS	***N/A***	***10%***	***197.4***
NS	N/A	30%	14.5
NS	N/A	50%	117
NS	N/A	70%	70.8
NS	N/A	90%	24.4
ONS	30%	10%	18.2
ONS	30%	30%	113.7
ONS	***30%***	***50%***	***147.6***
ONS	30%	70%	28.8
ONS	30%	90%	12.6

Table 4: Hyperparameter tuning for select percentage for OS, NS, and ONS variants without normalisation. The table outlines the parameter sweep strategy we employed to discover a suitable Select percentage for our experimentation. The bolded and italicised rows indicate the highest performing Select percentages for OS, NS, and ONS. Similar experiments conducted for ME-OS, ME-NS and ME-ONS

Agents evolved for 2 vs. 3 Keep-away soccer with MAP-Elites augmented HyperNEAT does not yield a higher average task performance than either Objective search, Novelty Search or Hybrid novelty-objective directed implementations of HyperNEAT when not augmented by MAP-Elites.

In order to address our research questions, we needed to gather data for 6 experimental permutations (variants) derived from the unique combination of the three search directives: Novelty, Objective-search, Hybrid search, and the inclusion or exclusion of the MAP-Elites algorithm. The six permutations are listed in Table 6. We captured agent performance data at each generation of evolution for each of the 6 variants, as well as evolution data including speciation and genome age. A full description of the important performance metrics and the related data captured is given in section 3.2.3.

3.2.6 Hyper-Parameters

Previous work by Gomes, et al. [47] demonstrated that a medium to high hybrid combination ratio of 50%-80% yielded the best results, while Didi [8] found that a hybrid combination ratio of 40% yielded the best results in a similar experimental setup to ours. In this research, we found that weight of 30% yielded the best results.

To determine the best select percentage for OS, NS and ONS, experiments were configured to allow us to determine the right hyperparameter tuning in order to get the best task performance out of each evolution method. We performed a sweep through a range of select-percentages and once complete, we selected the best parameters for each experiment permutation. The bolded and italicised rows indicate the highest performing repetitions. Similar experiments conducted for ME-OS, ME-NS and ME-ONS indicated that a

select percentage of 60% was best in these cases. The rest of the parameters for experimentation were maintained from previous research work by Didi [8] These parameters are given in Table 5.

NE Parameters	**Setting**	**HyperNEAT CPPN**	**Functions**
Population Size	150	Identity	x
Generations	270	Gaussian	$e^{-2.5x^2}$
Maximum number of species	10	Bipolar Sigmoid	$\frac{2}{1+e^{-4.9x}}-1$
Maximum species population	30	Absolute value	$\lvert x \rvert$
Mutation Type	Gaussian	Sine	$sine(x)$
Weight value range	[-5.0, 5.0]	**Simulation Parameters**	**Setting**
Mutation rate	0.05	Number of Runs	10
Survival threshold	0.2	Iterations per task trial	4500
NS Parameters	**Setting**	Trials per generation	10
NS nearest neighbou**r** k	15	Agent positions	Random
Maximum archive size	1000	Environment size	20x20 grid
Compatibility threshold	3	Agent speed (per iteration)	1 grid cell
Behavioural threshold	0.03	Ball speed (per iterations)	2 grid cells
MAP-Elites Parameters	**Setting**	**Hybrid Search Parameters**	**Setting**
Map resolution	0.05	Hybrid combination ratio	30%

Table 5: Hyperparameters for experiments. Left: Neuro-Evolution (NE), Novelty Search (NS) parameters. MAP-Elites parameters. Right: HyperNEAT CPPN activation functions, Simulation Parameters and Hybrid Search parameters.

Experiment permutation	**Variant**	**Search Method**	**Incorporates MAP-Elites**	**Select Percentage**
1	OS	Objective search	No	30%
2	NS	Novelty search	No	10%
3	ONS	Hybrid search	No	50%
4	ME-OS	Objective search	Yes	60%
5	ME-NS	Novelty search	Yes	60%
6	ME-ONS	Hybrid search	Yes	60%

Table 6: Method variants, covering 6 experimental permutations derived from the unique combination of the three search directives: Novelty (NS), Objective-search (OS), Hybrid search (ONS), and the inclusion or exclusion of the MAP-Elites algorithm (denoted by the prefix ME). The table describes each variant and the select percentage we used for each variant (see 3.2.6).

3.2.7 Research Experimentation

The approach to experimentation was to set up sixty individual experimental runs, resulting from ten iterations of each of the six experiment permutations presented in Table 6 and record the output data. This would allow us to apply statistical significance testing on the data and obtain robust results.

3.2.8 Number of Evolution Generations

We wanted to know whether more generations would have a significant impact on task performance. However, with each generation of evolution taking approximately 33 minutes of computation on the University of Cape Town HPC infrastructure, we faced a considerable time challenge. We reasoned that 500 generations for each of the 60 experiment runs (10 independent runs for each of the 6 variants) resulting in approximately 2063 hours (86 days) of computation on average, when running 8 experiments in parallel at a time would be our absolute limit, but finding a smaller number of generations that would result in similarly satisfactory results would be ideal.

To address this challenge, we ran a sub-set of experiments up to 500 generations of evolution and observed what the minimum number of generations would be to obtain satisfactory results wherein there was no change in the task performance ranking of the 6 experimental permutations.

We observed that there was no task performance rank change after 270 generations. We therefore settled on 270 generations of evolution for all experiments, resulting in approximately 1114 hours (46 days) of computation on average, running 8 experiments in parallel at a time.

3.3 High Performance Computing

Each experiment took considerable time to complete and so it was necessary to make use of High-Performance Computing (HPC) infrastructure to obtain results in an achievable time.

In this research, we made use of the Centre for High Performance Computing (CHPC) infrastructure to complete experiments during the exploration and hyperparameter tuning, and then switched to the University of Cape Town HPC infrastructure for final experimentation.

A MATLAB based infrastructure was developed to facilitate the preparation of simulation files and job scripts in such a way as to avoid and ensure there was no human error in preparing experiment configurations for a large number of experiments.

3.4 Statistical tests

We repeated experiments 10 times for each variant and recorded the best performance achieved during each evolutionary run for statistical testing. We applied the Lilliefors test [51] to determine which datasets were non-parametric and which were from the normal family.

Task Performance and Search Fitness data across experimental runs were found to be non-parametric for the OS variant and from the normal family for all other variants. Coverage data across experimental runs were found to be non-parametric for the ME-ONS variant and from the normal family for all other variants. Global reliability data was found to be from the normal family for all variants.

These mixed results lead us to apply statistical testing using the two-sided Wilcoxon rank sum test, which is equivalent to the Mann-Whitney U-Test ($p < 0.05$)[52]. This test useful in that it is appropriate for both non-parametric and normal pair-wise comparisons. The statistical tests were applied in pair-wise comparisons between data-sets for the method variants, and complete overview of statistical test results are provided in Appendix A: Statistical Tests

3.5 Summary

This chapter outlines the systematic approach taken to examine the impact on performance through integration of MAP-Elites as an illumination algorithm into various HyperNEAT driven search strategies within a multi-agent system. This integration, termed ME-HyperNEAT, is significant as it enables us to explore capability related to the generation of high performing and diverse agent behaviours.

Our method centres on the training keeper agents using ME-HyperNEAT alongside the Novelty, Objective, and Hybrid novelty-objective search, and the experiments are conducted in a controlled simulation environment tailored for 2 vs. 3 Keep-away soccer scenarios.

This chapter presents key technical details regarding our methods and experiments. We describe modifications made to the evolution framework to accommodate MAP-Elites, resulting in ME-HyperNEAT. We discuss the development of a MATLAB-based analysis infrastructure for efficient data processing and the strategic use of High-Performance Computing (HPC) resources to optimize computational efficiency. Additionally, our selection of performance and behavioural metrics is provided and we describe our final experimental setup, encompassing experiment design, hyper-parameter selection, and the rationale behind determining the number of evolution generations utilised in the study.

4 Results and Discussion

In this chapter we will report on experiment results and compare variants to each other in terms of task performance, global performance, coverage, global reliability, and exploration of the behaviour search space.

We begin with a summary of the results obtained with each variant, and then continue on to systematically compare all 6 variants to each other. Our approach to this is first to compare the variants with MAP-Elites excluded from the algorithm (OS, NS, ONS) to each other, and then we compare each of these variants to their counterparts when MAP-Elites is included. We then conclude by making comparisons of all variants to each other.

This approach allows us to understand the difference in performance and behaviour for each of the 'standard' variants, and then understand how MAP-Elites influences each of them. The variants we will report on, and compare are described in Table 7 and the structure of this chapter is as follows:

- Overall results comparison (4.1)
- OS vs. NS vs. ONS (4.2)
- OS vs. ME-OS (4.3.1)
- NS vs. ME-NS (4.3.2)
- ONS vs. ME-ONS (4.3.3)

4.1 Overall Results Comparison

Results for the best performing genome of any run, at any generation (the best genome generated by the variant) are presented in Table 8. Search Fitness is the result of Equation (1 for OS and ME-OS; Equation(2 for NS and ME-NS; and(4 for ONS and ME-ONS (all presented in 2.10). Task performance is the Average Hold Time described in 2.1.2 and 3.2.3. Global Performance is the relative performance for each variant compared to the best task performance obtained in any run as described in 3.2.3. Coverage is the amount of the conceivably searchable behaviour space that was explored by the variant as described in 3.2.3. Global Reliability is the average performance of the filled cells or niches in the feature-performance map as described in 3.2.3.

As described in 3.4 we conducted 10 independent evolutionary runs for each variant. We recorded the best task performance obtained (at any generation) for each run and used these figures to perform significance testing. We present these results in Figure 11, which shows boxplots indicating the distribution of best achieved task performance across the 10 independent runs. The figure shows that there is a wide distribution of best performance for the variants. The least variation (most consistent results) was obtained from ONS (interquartile range of ~0.13) and ME-NS (interquartile range of ~0.09). The widest range of best task performance was observed in OS (interquartile range of ~0.26) and ME-ONS (interquartile range of ~0.29). The minimums and maximums at the end of the whiskers indicate the highest and lowest performance achieved for each variant. We observed that the best individual performance was achieved by ME-ONS, followed closely by ME-OS, and the worst individual performance was achieved by ONS, and followed by ME-ONS, perhaps indicating that hybrid search has the highest propensity to explore the search space irrespective of performance and thus find the widest range of performance. ME-OS had the highest median performance achieved (indicated by the red line in each box) indicating that ME-OS had the highest probability to develop strong solutions (solutions with task performance above 0.70). When comparing each distribution for statistical significance we observed that only ME-NS was significantly different to OS, ONS, ME-OS. No other comparisons yielded a significant difference for maximum task performance. Assiciated p-values are provided in Table 9-12 in Appendix A: Statistical Tests

Figure 12 shows the distribution of task performance for each variant's population at the generation in which the best task performance was discovered. This view allows us to understand whether each variant creates a population that is significantly better than another. The key insight we take from this is that although ME-ONS and ME-OS produced best performances that were not significantly different (Table 9, Appendix A), the population produced by ME-ONS was different (better) than ME-OS with statistical significance (Table 9, Appendix A).

Variant Name	Variant Description
OS	Objective-based Search
NS	Novelty Search
ONS	Hybrid Novelty-Objective based Search
ME-OS	MAP-Elites - Objective-based Search
ME-NS	MAP-Elites - Novelty Search
ME-ONS	MAP-Elites - Hybrid Novelty-Objective based Search

Table 7: Evolution method variants covering 6 experimental permutations derived from the unique combination of the three search directives: Novelty (NS), Objective-search (OS), Hybrid search (ONS), and the inclusion or exclusion of the MAP-Elites algorithm (denoted by the prefix ME).

Search method	Search Fitness	Task Performance	Map Coverage	Global Reliability	Generation of maximum performance
OS	0.70	0.94	99.5%	2.164%	34
NS	0.58	0.73	99.2%	2.131%	119
ONS	0.57	0.77	98.6%	2.118%	30
ME-OS	0.74	1.00	99.1%	2.168%	254
ME-NS	0.41	0.68	98.9%	2.144%	111
ME-ONS	1.00	1.00	98.8%	2.110%	265

Table 8: Normalised maximum task performance for each of the six variants along with normalised search fitness, global performance, coverage, global reliability, and the generation at which maximum performance was achieved for each variant in any run. The maximum number of generations for any given run was 265.

In terms of the MAP-Elites performance metrics, we note that OS had the highest coverage, followed by NS, ME-OS, ME-NS, ME-ONS, and ONS however none of these results were statistically significant (Table 11, Appendix A). The highest global reliability was achieved by ME-OS, followed by OS, ME-NS, NS, ONS, and ME-ONS. In this case, OS was significantly different to NS, ONS, and ME-NS, and ONS was significantly different to ME-OS (Table 12, Appendix A). The close results show that all variants performed fairly evenly with respect to global reliability in this task, with only OS seemingly to have the best global reliability with statistical significance (Table 12, Appendix A).

4.2 Non MAP-Elites Variants (OS, NS, ONS)

In this section we compare OS, NS and ONS against each other in terms of task performance, and capability in maximising useful usage of the behaviour search space, that is, the ability to both explore, and exploit.

4.2.1 OS vs. NS vs. ONS

Figure 13 shows the maximum task performance for OS, NS and ONS. From this figure we see that OS achieves the best performance of the three variants. Figure 15 shows how all three variants perform similarly at the start of the evolutionary process, but by generation 5, OS takes a leap forward and then retains its lead throughout the rest of the evolutionary process. NS and ONS remain quite close in performance all throughout the process. This observation is made clearer in Figure 14 and Figure 16 which show the average task performance for OS, NS and ONS. From this figure we see that OS shows the best average task performance across the entire population at each generation of evolution. Here we can see that OS outperforms NS and ONS on average, and by generation 7, the average task performance for OS is greater than that of NS and ONS, and this status is maintained throughout the rest of the evolutionary run. We also see that by generation 6, the average task performance is better for ONS than for NS and this status is retained throughout the rest of the evolutionary run. Figure 13 is annotated to show the generation where the maximum performance is achieved for each variant. The best task performance was achieved by OS, followed by ONS, and then NS (with statistical significance, Table 9, Appendix A). OS and ONS achieved maximum performance as early as generation 34 and 30, respectively, while NS took significantly longer, achieving maximum performance at generation 119. None of these variants achieve higher performance in the second half of the evolutionary run, showing that either there is no more potential for better performance, or a significant leap in evolution would be needed to overcome this performance plateau.

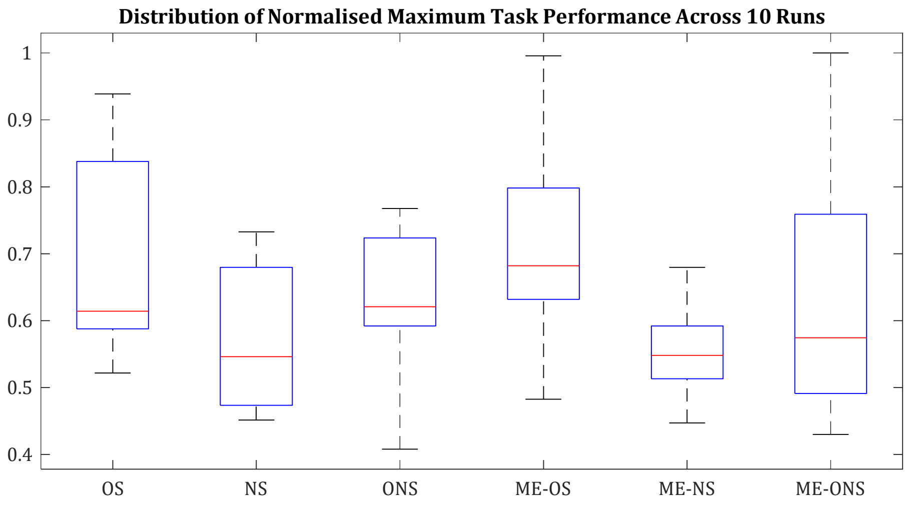


Figure 11: Distribution of normalised maximum task performance (average hold time, 2.1.2 and 3.2.3) across 10 runs for each variant (OS, NS, ONS, ME-OS, ME-NS, ME-ONS). ME-NS is significantly different to OS, ONS, ME-OS. No other comparisons yielded a significant difference for maximum task performance (two-sided Wilcoxon rank sum test, which is equivalent to the Mann-Whitney U-Test)

We now describe the way each variant is able to explore the search space in three chosen directions which have been chosen as the phenotypic descriptors for genomes. These are Number of passes (NP), Distance of the Ball from the Field Centre (DBFC), and Team Dispersion (TD). In addition, we analyse each variant's ability to explore and exploit the search space by assessing Global fitness, Coverage, and Global reliability.

We introduced the concept of an elite in 2.9 when reviewing MAP-Elites as the best performing genome reserving a unit of space in the behavioural search space. In order to understand and showcase each variant's ability to explore and exploit the search space, we made use of 3-dimensional plots where each elite is represented by a point on the plot with X, Y, and Z dimensions representing the three behavioural descriptors for genomes (NP, DBFC, TD, respectively). Additionally, the colour and size of each dot represents its task performance. The genome with the best task performance in each plot is also highlighted with an annotation to the plot.

Figure 17, Figure 18 and Figure 19 show these 3-D plots for the OS, NS and ONS variants respectively. What's interesting is that OS (Figure 17) seems to explore the search space in a symmetrical fashion, creating a symmetrical cone-like shape centred around its Y axis (NP), with a wide base of relatively low performing genomes at around 20 (0.15 after normalising) passes, and peaking towards the other end of the scale at 134 passes (1 after normalising). This is different to NS (Figure 18) which appears to have explored in a symmetrical way, especially noticeably exploring a low performing region towards the higher end of the Z (BDFC) and X (TD) axis, this is annotated as B in Figure 18. The best performing NS genome still appears towards the end of the Y axis scale (NP), indicating that this is a critical behaviour trait for task performance, however we do notice that there are more high performing genomes close to this region as compared to OS. As one would expect, the ONS plot (Figure 19) shows what appears to be a middle ground between OS and NS, where the cone structure is maintained but there is some NS-like exploration outside of the symmetrical shape seen in OS.

Table 8 provides the quantitative results of our Global fitness, Coverage, and Global reliability calculations. These findings indicate that, although NS and ONS include greater evolutionary incentives for exploring the search space compared to OS, OS still appears to outperform these other variants in this task. However, it is important to note that only the Global reliability measure shows statistical significance (Table 12, Appendix A).

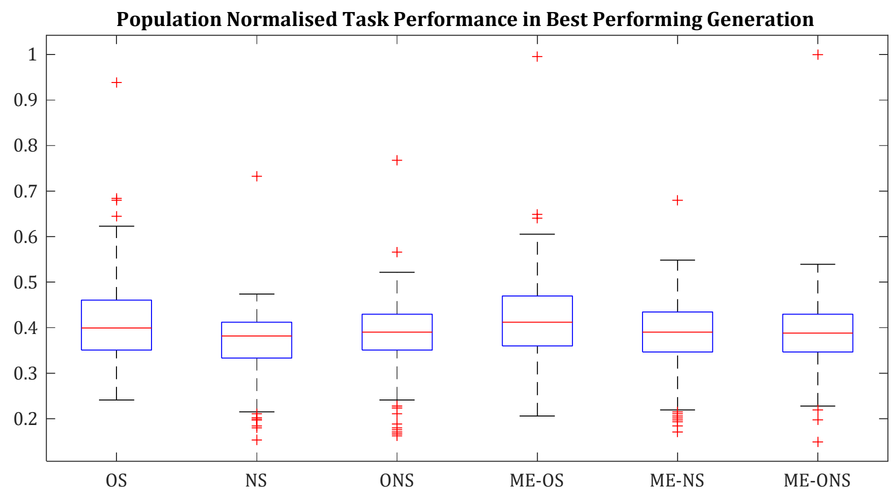


Figure 12: Best generation population comparison. Distribution of normalised task performance (average hold time, see 2.1.2 and 3.2.3) across 10 runs for the population in which the best task performance was achieved in each variant (OS, NS, ONS, ME-OS, ME-NS, ME-ONS). ME-OS is not significantly different to OS. ME-NS is not significantly different to NS or ONS. ME-ONS is not significantly different to ONS or ME-ONS. All other comparisons did yield a significant difference for maximum task performance (two-sided Wilcoxon rank sum test, which is equivalent to the Mann-Whitney U-Test, Table 9, Appendix A)

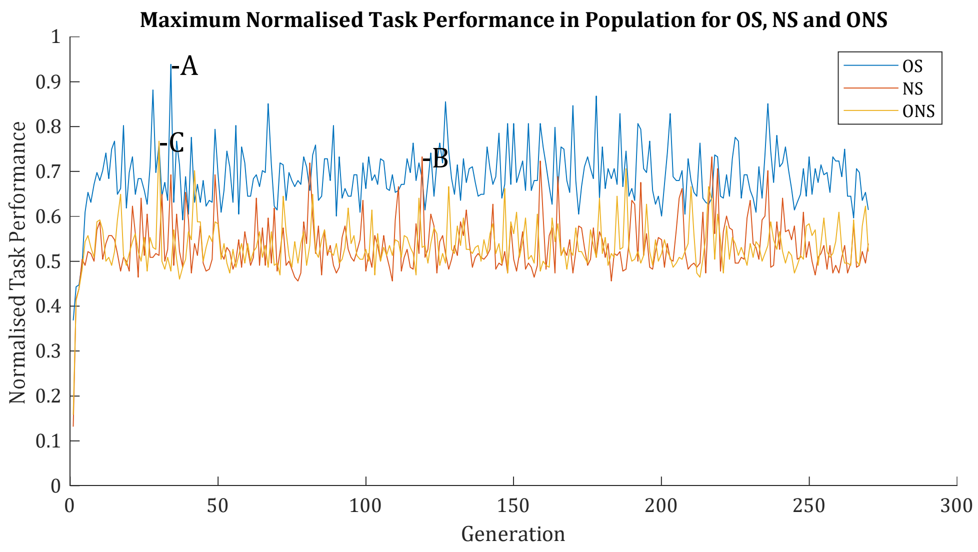


Figure 13: Maximum normalised task performance (average hold time, see 2.1.2 and 3.2.3) across 10 runs in population for OS, NS and ONS at each generation of evolution. The annotations A, B and C show the generation where the maximum performance is achieved for each variant, OS, NS, and ONS, respectively. OS reaches its peak performance early at 34 generations and on average performs better than NS and ONS for the remainder of evolution. NS and ONS reach their peak performance at generation 119 and 30 respectively, but on average perform similarly.

We have also created three surface plots (Figure 20 I - IX) per variant where the colour scale and Z-axis represent task performance, and the X-axis and Y-axis represent each of the 3 behavioural descriptors against each other. From these figures (subplot I, III, IV, VI, VII and IX) one can see the strong positive correlation between number of passes and task performance. Comparatively, there does not seem to be a strong correlation between BDFC or TD to task performance, but there are clear local minima and maxima.

For additional analysis we created "top-down" heat maps in Figure 21 that map out task performance for each behaviour descriptor combination. For example, we can see in subplot IIX that there is a particularly strong area of performance between 0.7 and 0.85 for BDFC, and between 0.75 and 0.83 for TD. The corollary is also true, that there is particularly weak performance around the edges of the explored behaviour space (with regards to these two behaviour descriptors).

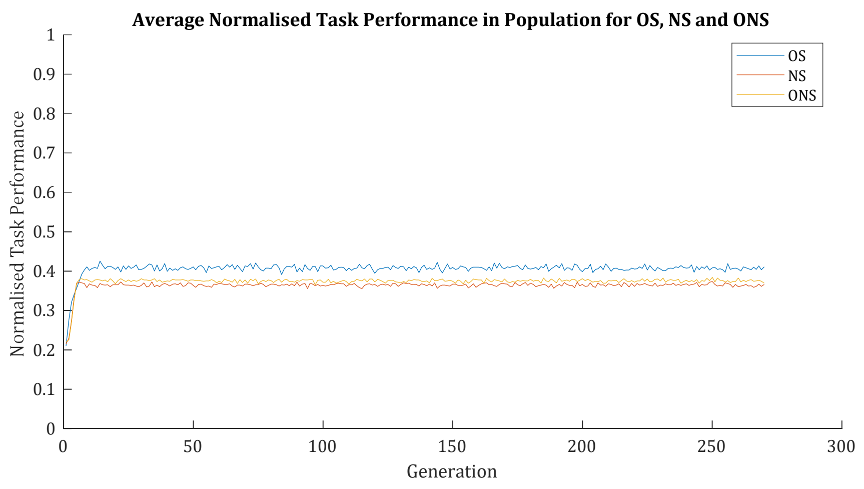


Figure 14: Average normalised task performance (average hold time, see 2.1.2 and 3.2.3) for the population across 10 runs for OS, NS, ONS. The figure demonstrates how OS is able to consistently produce a higher performing population of genomes throughout evolution.

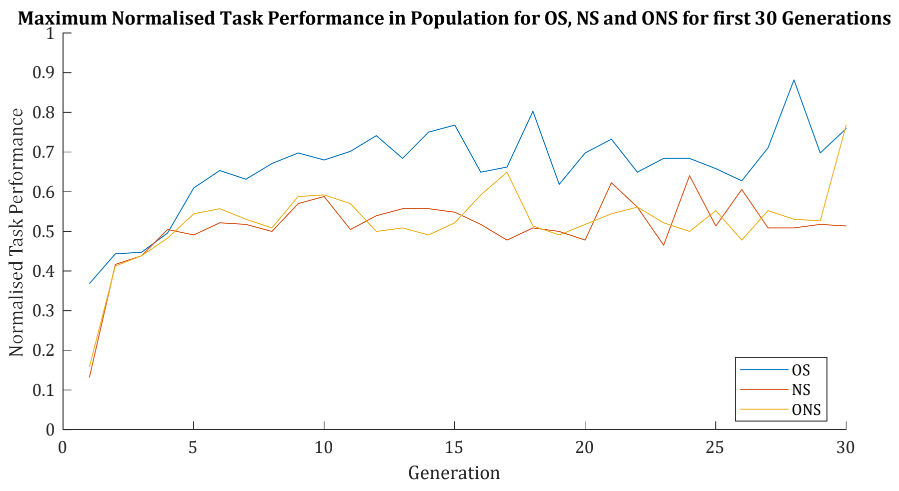


Figure 15: Maximum normalised task performance (average hold time, see 2.1.2 and 3.2.3) across 10 runs for each variant during the first 25 generations for OS, NS, ONS. The figure shows how OS was able to leap ahead of both NS and ONS in performance as early as generation 5.

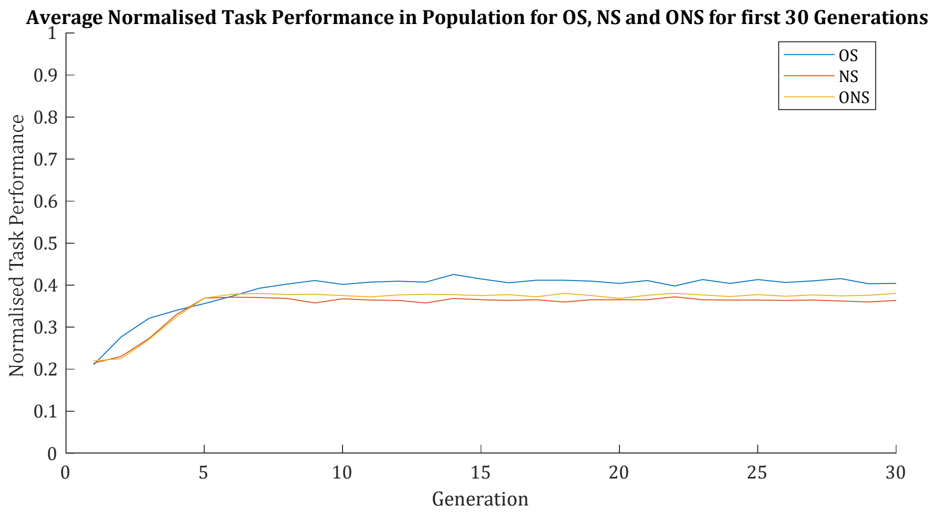


Figure 16: Average task performance (average hold time, see 2.1.2 and 3.2.3) in population across 10 runs over the first 25 generations for OS, NS, and ONS. The figure shows how OS was able to produce a population that had higher performanec on average from as early as generation 7.

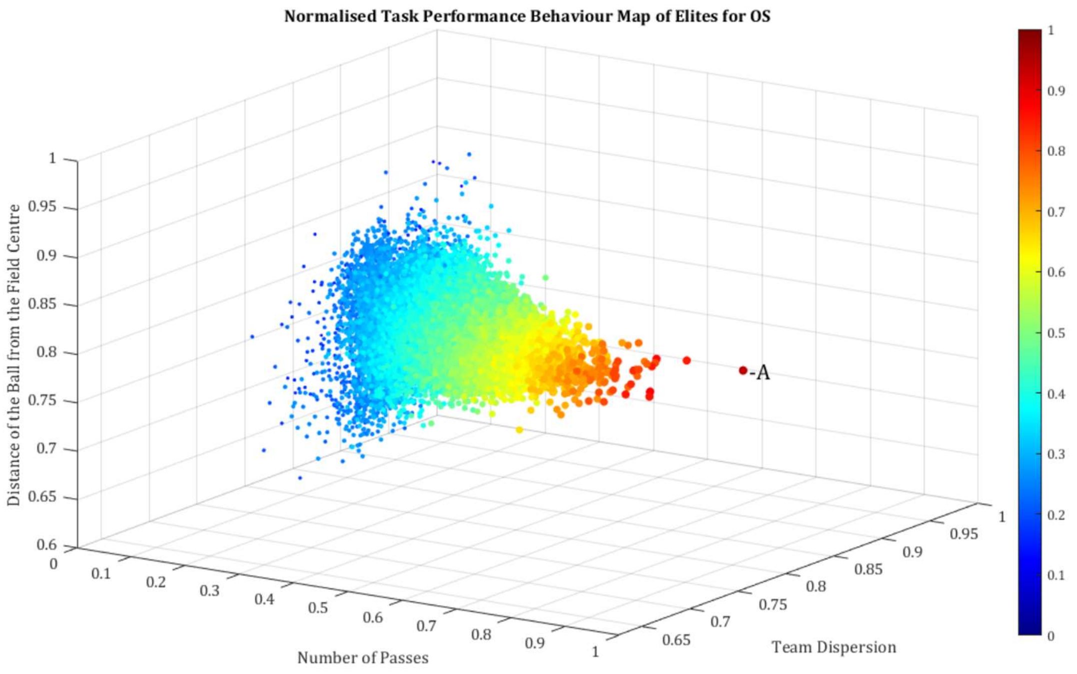


Figure 17: Task performance (average hold time, see 2.1.2 and 3.2.3) behaviour map of elites for OS. Annotation A indicates best performing elite in the map.

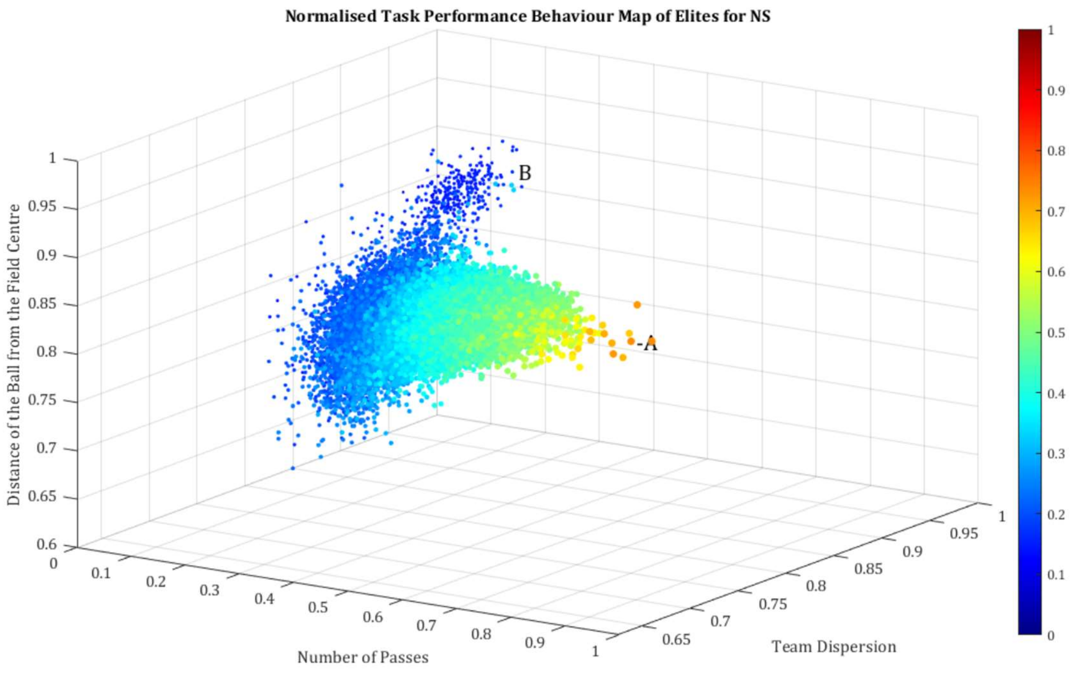


Figure 18: Task performance (average hold time, see 2.1.2 and 3.2.3) behaviour map of elites for NS. Annotation A indicates best performing elite in the map. Annotation B indicates an extended region of exploration.

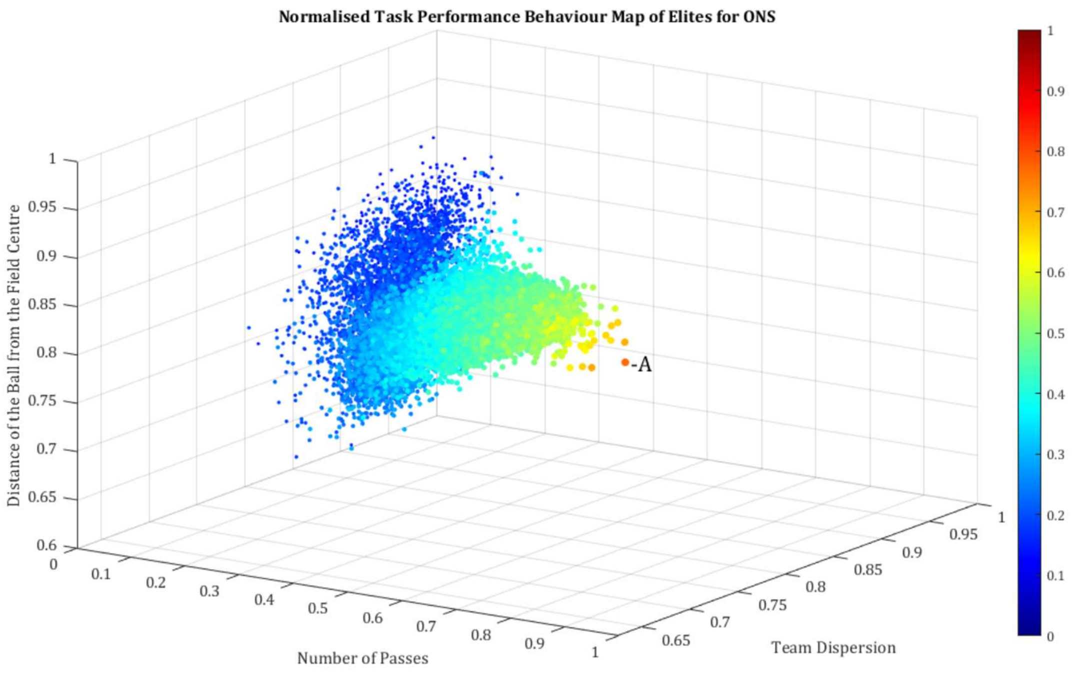


Figure 19: Task performance (average hold time, see 2.1.2 and 3.2.3) behaviour map of elites for ONS. Annotation A indicates best performing elite in the map.

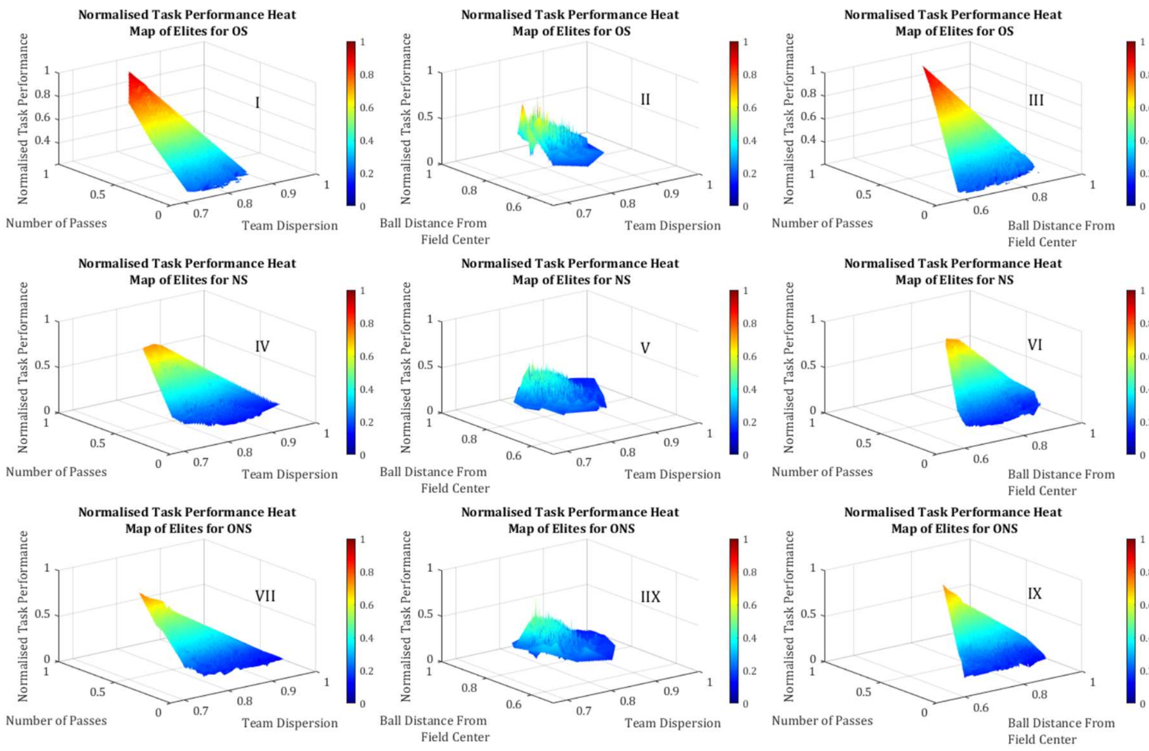


Figure 20: Interpolated 3-Dimensional analysis of task performance (average hold time, see 2.1.2 and 3.2.3) feature space for OS (top row I, II, III), NS (middle row, IV, V, VI), and ONS (bottom row, VII, IIX, IX). We have performed linear interpolations between elites in order to create smooth surfaces.

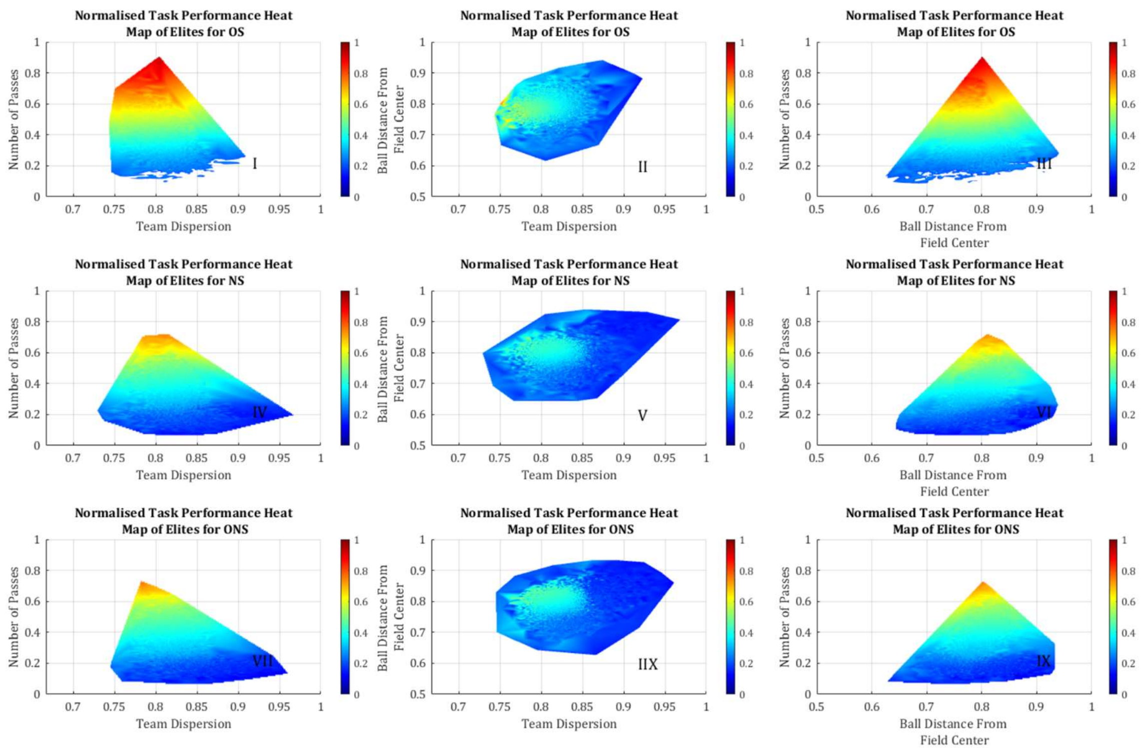


Figure 21: Interpolated task performance (average hold time, see 2.1.2 and 3.2.3) "top-down" heat map for OS (top row I, II, III), NS (middle row, IV, V, VI), and ONS (bottom row, VII, IIX, IX). for each behaviour descriptor combination. We have performed linear interpolations between elites in order to create smooth surfaces.

4.3 MAP-Elites Variants (ME-OS, ME-NS, ME-ONS)

In this section we report on and compare the results obtained by each of the previously discussed variants (OS, NS, ONS) against their MAP-Elite variant counterpart. We report on the impact for each of them when including MAP-Elites in the selection process.

Table 8 shows how incorporating MAP-Elites improves performance for OS and ONS, but interestingly reduced performance of NS. Figure 22 shows the task performance trend for ME-OS vs. ME-NS vs. ME-ONS. In this figure we see how the population of ME-OS outperforms the other variants populations for most of the evolutionary run and with statistical significance (Table 9, Appendix A) when comparing each variant's best performing generation.

Figure 24 shows the first 25 generations of evolution and shows us that OS performed better from generation 2, and maintained its superiority until evolution concluded, although only with a statistical significance of 65%. Figure 26 makes this trend clear by showing the average population performance for each generation.

Interestingly, and what is different to the OS, NS and ONS variants, is that the best performance was reached by the ME-OS and ME-ONS variants later in evolution. ME-OS achieved its best performance at generation 254, and ME-ONS achieved its highest performance at generation 265 (compared to generation 34, and 30 for OS and ONS respectively). Interestingly, ME-NS achieves its best performance at generation 111, which is at a similar stage to NS which achieved its best performance at generation 119.

In terms of performance comparisons, ME-ONS resulted in the best performing genome of these three, and in fact all variants with a task performance of 1. Although this result was not statistically significant, on average, the population of ME-ONS outperformed all other variants with statistical significance (Table 9, Appendix A).

ME-OS resulted in the second highest task performance at 0.99 and although this result was only statistically significant compared to ME-NS, the outperformance of the population was significantly different to the other variants (Table 9, Appendix A).

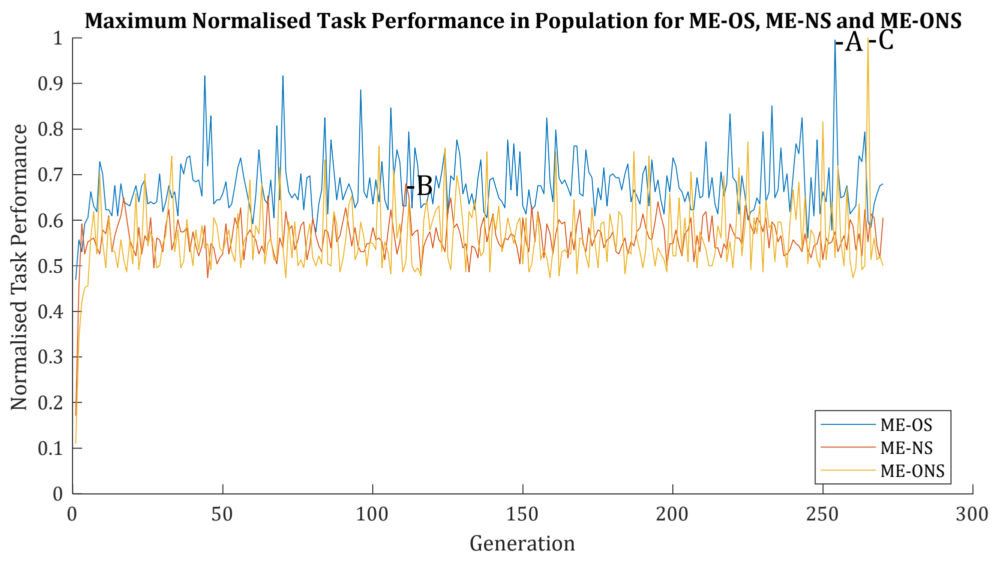


Figure 22: Maximum task performance (average hold time, see 2.1.2 and 3.2.3) over 10 runs for ME-OS, ME-NS and ME-ONS. Annotations A, B and C indicate the generation when each variant produced its best task performance.

ME-NS interestingly achieved the lowest performance of all variants (including non-MAP-Elites variants), and this result is statistically significant compared to OS, ONS, and ME-OS but not statistically significant (Table 9, Appendix A) when compared to any other variant.

It was also interesting to note that ONS was outperformed by OS until generation 265 when it apparently evolved a high performing genome that outperformed OS.

Figure 23 shows surface plots for ME-OS, ME-NS and ME-ONS. From these figures one can see similar trends to those found in OS, NS and ONS (see Figure 20). The major difference being that ME-OS and ME-ONS achieved higher task performances. These views also highlight the importance of the Number of Passes as the plots form triangular surfaces, seemingly searching a narrower and narrower band of TD and BDFC and yet achieving higher performance as long as NP increases. We will describe this search space exploration in more detail in the following sections which make comparisons between OS and ME-OS, NS and ME-NS, and between ONS and ME-ONS.

4.3.1 OS vs. ME-OS

Figure 27 shows the task performance trend for OS vs. ME-OS. Performance is close, but in later generations at generation 254, ME-OS discovers something important, resulting in a high performing solution. ME-OS resulted in a higher performing solution than OS, but not with statistical significance over the course of 10 runs (Table 9, Appendix A). Although the best task performance obtained was not statistically significant, on average, the population of ME-OS outperformed OS in task performance with statistical significance (Table 9, Appendix A), this is shown in Figure 28.

Figure 33 shows heat maps for task performance, comparing each behaviour descriptor. Here we can make similar observations to those we made in Figure 21 for the OS, NS and ONS variants. Namely, we see that there is a strong positive correlation between Task performance and Number of Passes. We also see that there is a strong region of performance between 0.75 and 0.83 for TD and between 0.73 and 0.85 for BDFC. The difference here between OS and ME-OS is that ME-OS was able to better exploit the importance of NP, thus resulting in the triangular shape peaking in the high-performance zone at the top of subplot IV and VI in Figure 33.

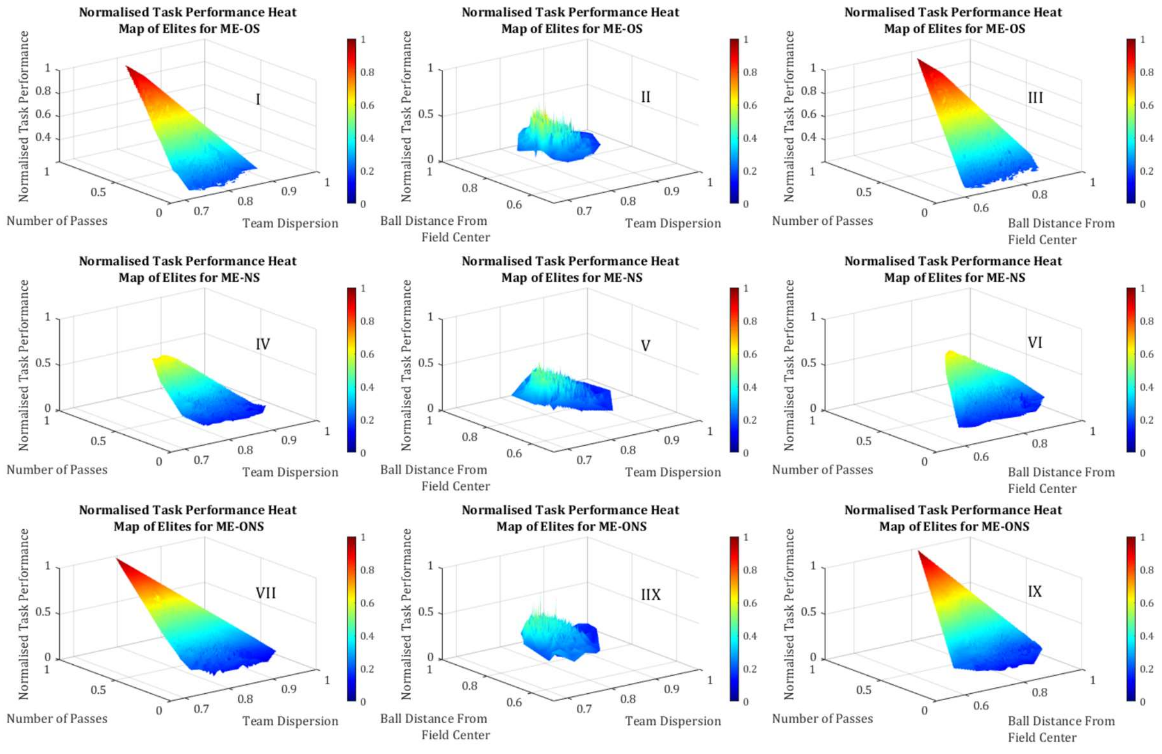


Figure 23: Interpolated 3-Dimensional analysis of task performance (average hold time, see 2.1.2 and 3.2.3) feature space for ME-OS (top row I, II, III), ME-NS (middle row, IV, V, VI), and ME-ONS (bottom row, VII, IIX, IX). We have performed linear interpolations between elites in order to create smooth surfaces.

What is unexpected is that OS seems to have explored the low performance space at the lower range of NP more than ME-OS. This is perhaps due to MAP-Elites resulting in more directionality in the evolutionary process as the elites in the map drive evolution towards their centre of mass, compared to OS which tends to explore more equitably in directions of higher performance.

4.3.2 NS vs. ME-NS

Figure 29 shows the trend of task performance at each generation for NS and ME-NS. Interestingly NS outperformed ME-NS, unlike is the case with the other variants which performed better with ME than without, reaching its best performance of 0.7325 at generation 119. ME-NS reached its best performance of 0.6798 at generation 111. This difference in performance was not statistically significant (Table 9, Appendix A), however the population was significantly different in performance (Table 14, Appendix A), which is also seen in Figure 30 which shows average population task performance at each generation.

The performance trend seems to indicate less of an oscillatory pattern of performance with ME-NS than with NS, where low and high ranges of task performance are achieved successively throughout evolution. It appears that ME when coupled with NS might actually be limiting exploration of the search space. We can see this more clearly in Figure 18 and Figure 37 which show the map of elites in 3 dimensions. NS explores the space more freely than ME-NS. In these figures observe that the NS variant stretches out further into the search space, covering a greater range in the X (NP) direction, and takes a greater detour into the region denoted B in the figure. We also see in these figures that ME-NS appears to search more low performing areas at the task performance level of around 0.44 (teal colour in the figures), this exploration is perhaps at the expense of performance since we now know that NP is the most critical behavioural feature for task performance.

Figure 34 shows subtle differences between the search space exploration for NS vs. ME-NS. The shape of each surface is similar, but not identical. We see in subfigure I and IV that NS explores the relationship between NP and TD more thoroughly than ME-NS. Subfigure II and V show that the two variants explored different finding different low performance regions but discovered the same hot spot of performance around BDFC = 0.8 and TD = 0.79. Subfigures III and VI show again that NS was able to exploit NP more effectively than ME-NS, perhaps because ME-NS spent more time exploring a greater range of BDFC at the lower range of task performance < 0.22.

4.3.3 ONS vs. ME-ONS

Figure 31 shows the trend of best fitness achieved in each generation for ONS and ME-ONS. ME-ONS outperforms ONS with a task performance of 1 at generation 265 compared to 0.77 at generation 30 for ONS. However, this result was not statistically significant (Table 9, Appendix A), and neither was performance for the full population of ME-ONS compared to ONS. This result can be understood when we consider that ME-ONS performed similarly to ONS for most of the evolutionary run, until something important was discovered at generation 265, resulting in a high performing genome for ME-ONS. This observation tells us that there is likely more task performance to be achieved if more generations of evolution are performed.

In ad hoc experimentation, we performed evolution for up to 500 generations, and in those runs did not find that results improved significantly enough to warrant the additional computational time. However, we do concede the recommendation that it may be worth attempting to run these experiments for significantly extended periods of around 600 to 1000 generations in the future.

Looking at Figure 19 and Figure 38 we can see how far of a leap was taken by ME-ONS to evolve its best performing elite (annotated in the figures as A). Looking at how the behaviour dimensions relate to task performance in Figure 35 shows how ME-ONS was able to develop better solutions overall by exploiting NP. We observe that these two variants explored the low performing areas of the TD vs. BDFC dimensions differently, however similarly to all other variants, they were able to find hot spots of performance around BDFC = 0.8 and TD =0.79.

4.4 MAP-Elites Performance Metrics

We selected Coverage and Global Reliability as MAP-Elites performance metrics. These results are provided in Table 8.

Coverage results showed that OS was able to discover the most elites in the search space, followed by NS, ME-OS, ME-NS, ME-ONS, and finally ONS. However, these results were not statistically significant (Table 11, Appendix A) enough to make this claim certain. The similarity in results does however show that in this collective behaviour task, all variants explored a similar proportion of the behaviour space.

Global reliability shows that ME-OS was able to produce the best average performance of all elites in the discovered behaviour space, followed by OS, ME-NS, NS, ONS, and ME-ONS. Statistical significance was found for OS vs. NS, ONS, ME-NS and for ONS vs. ME-OS (Table 12, Appendix A). This finding shows us that OS and ME-OS may have potential to produce better average elite performance throughout the behaviour space, but the lack of statistical significance across all comparisons indicates that further research would be needed to make this claim with confidence.

4.5 Comparisons with previous research

Previous research by Didi [8] on which this research is based corroborate our research in that they found the best task performance was achieved by ONS, followed by OS, and then NS, the same hierarchy as our research when excluding the MAP-Elites variants. When including the MAP-Elites variants, we found that the best task performance was found by ME-ONS, followed by ME-OS, then OS, ONS, NS, and finally ME-NS.

Didi [8] also described that the best performing agents demonstrated high number of passes with little team dispersion and distance to the field centre. They also found that with OS the best performing agents had low team dispersion and average distance from the centre of the field. Comparing this to our results, we found that number of passes was the critical behaviour characteristic and was highly correlated with task performance for all variants. We found that team dispersion was important as well, but all variants were able to find what appeared to be a local (or perhaps global) optimal area of performance between 0.775 and 0.825. We found that ball distance from field centre was uncorrelated with task performance, which also appears to confirm what was found by Didi [8]. In future research we recommend that more behaviour descriptors are investigated to determine whether there are other behaviour traits that could aid our understanding of the behaviour space beyond the three behaviour traits investigated so far.

Didi [8] found that NS was able to explore more diverse solutions but was not able to evolve high performing agents in comparison to OS and ONS. We can confirm this result, with the additional insight that combining MAP-Elites increases this phenomenon even further, demonstrated by our results that showed ME-NS resulted in a lower task performance than NS, as well as a lower global reliability and coverage (Table 8).

4.6 Summary

In our experiments we discovered that the best task performance was obtained by ME-ONS, a combination of MAP-Elites, and hybrid objective, non-objective search. Although this result was not statistically significant (Table 9, Appendix A), we believe it provides good weight of evidence to the hypothesis that MAP-Elites does indeed improve the performance of evolution in the Keep-Away task, which is a type of collective behaviour task. Additionally, our findings were that ME-OS outperformed OS, and ME-ONS outperformed ONS. This too lends weight of evidence to our finding.

On the other hand, we found that MAP-Elites seemed to decrease performance of the NS variant in that ME-NS performed worse than NS (although not with statistical significance, Table 9, Appendix A). Thus, future researchers may find value in avoiding this combination.

Our second objective was to investigate the impact of MAP-Elites on the ability of OS, NS, and ONS to explore the behaviour search space. We found that MAP-Elites did not significantly improve Coverage, or Global Reliability. However, MAP-Elites did significantly improve our ability to analyse how behaviour related to task performance. Visualisations of the map enabled us to clearly determine the importance of Number of Passes in achieving high performing agents and gave us clear indications of high performance and low performance regions. We posit that MAP-Elites is highly effective as an illumination algorithm and provides great insight into how to perform follow up research.

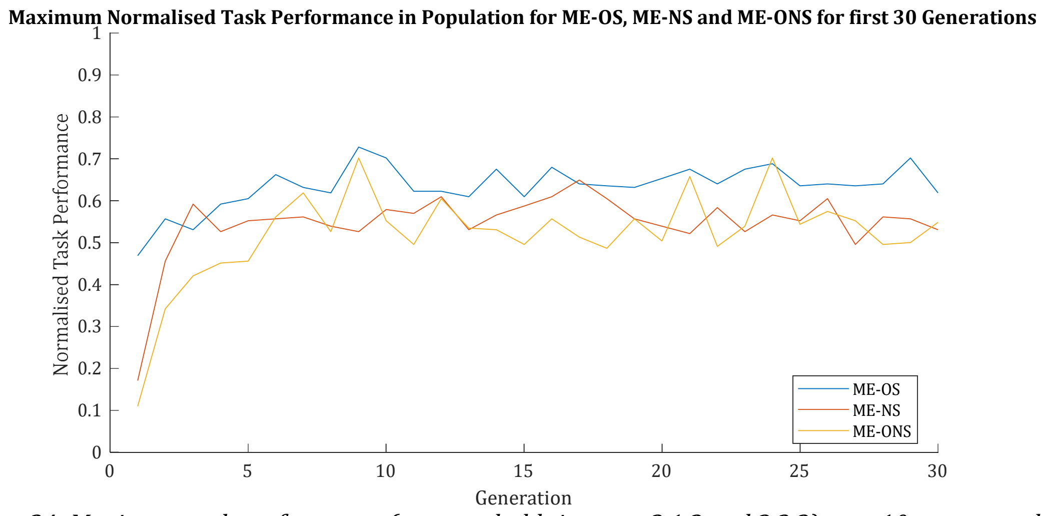


Figure 24: Maximum task performance (average hold time, see 2.1.2 and 3.2.3) over 10 runs over the first 25 generations for ME-OS, ME-NS, and ME-ONS. The figure shows how ME-OS was able to leap ahead of both ME-NS and ME-ONS in performance as early as generation 4.

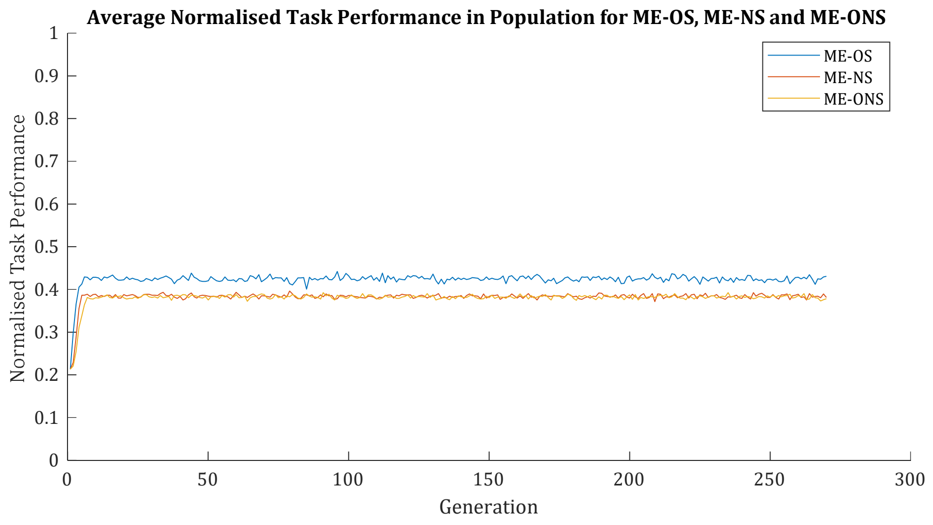


Figure 25: Average task performance (performance (average hold time, see 2.1.2 and 3.2.3) over 10 runs for ME-OS, ME-NS, and ME-ONS). The figure demonstrates how ME-OS is able to consistently produce a higher performing population of genomes throughout evolution.

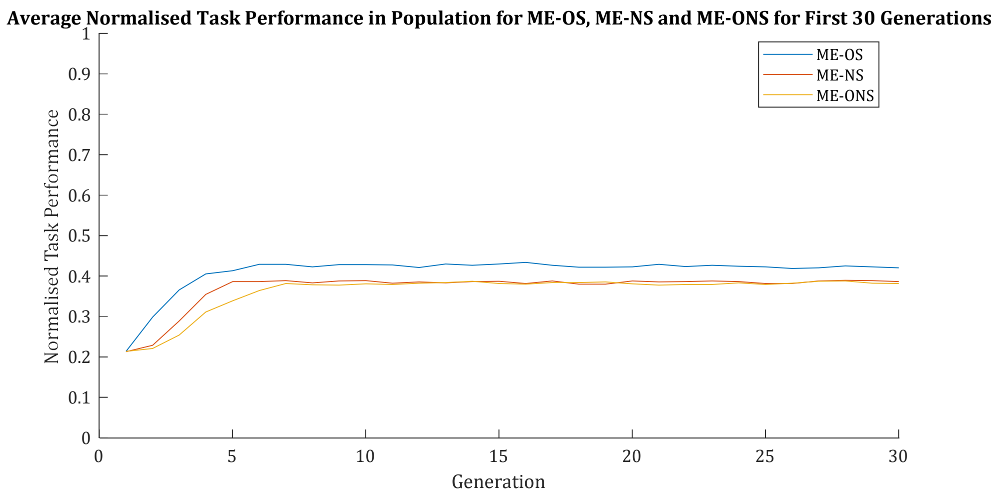


Figure 26: Average task performance (average hold time, see 2.1.2 and 3.2.3) over 10 runs for the first 25 generations: for ME-OS, ME-NS, and ME-ONS. The figure shows how ME-OS was able to produce a population that had higher performance on average from as early as generation 1.

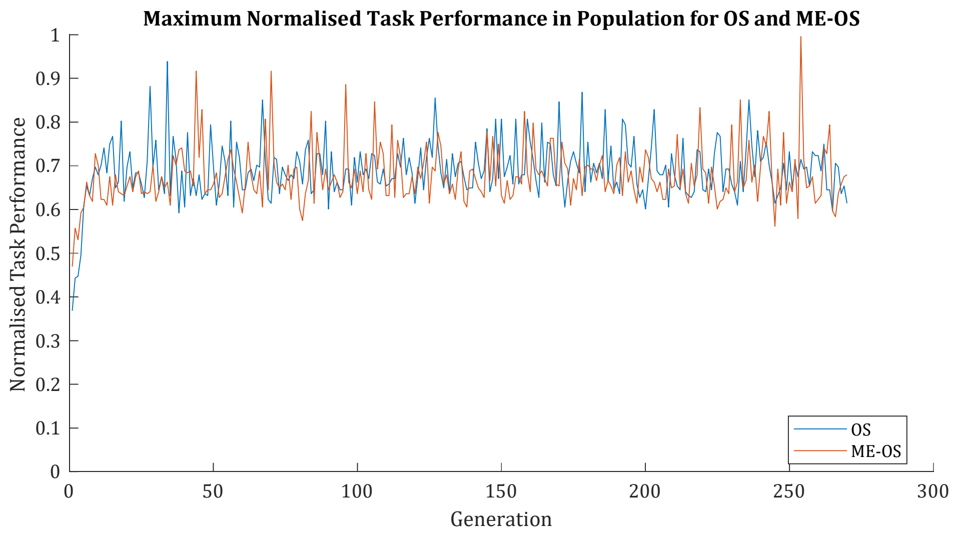


Figure 27: Maximum task performance (average hold time, see 2.1.2 and 3.2.3) over 10 runs for OS and ME-OS. The figure shows similar oscillatory performance until late in evolution when a leap of performance was discovered by ME-OS.

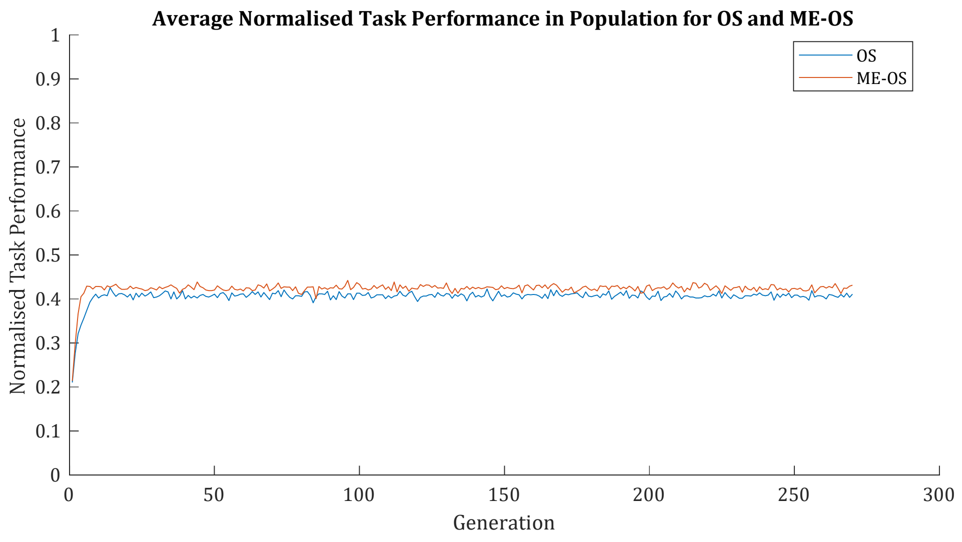


Figure 28: Average task performance (average hold time, see 2.1.2 and 3.2.3) over 10 runs for OS and ME-OS. The figure shows how on average, the population produced by ME-OS was able to outperform OS.

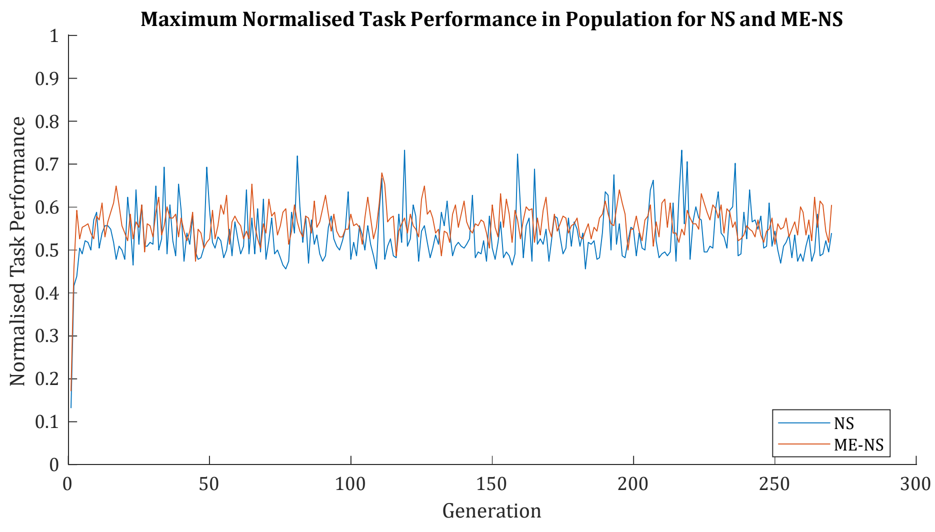


Figure 29: Maximum task performance (average hold time, see 2.1.2 and 3.2.3) over 10 runs for NS and ME-NS. The figure shows how on average, the population produced by ME-NS was able to outperform NS.

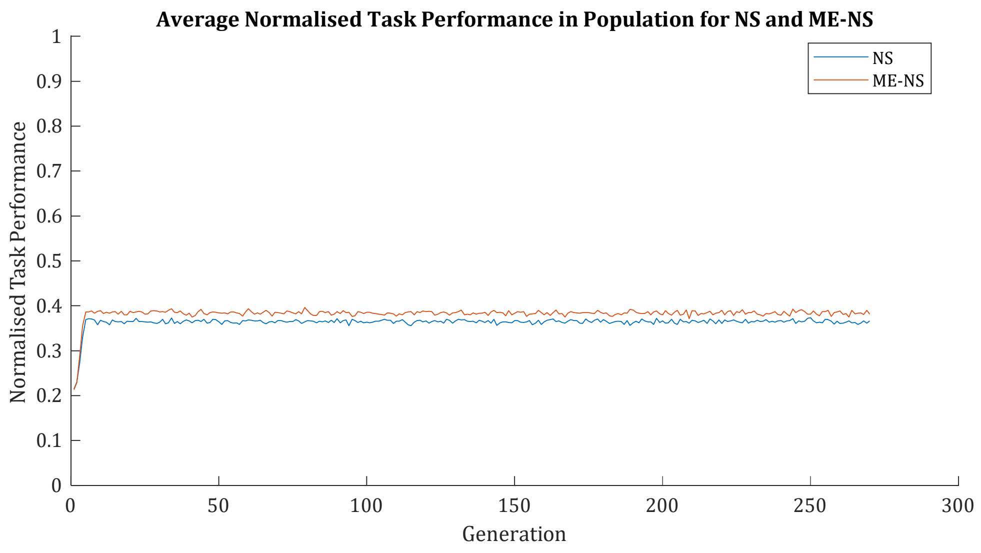


Figure 30: Average task performance (average hold time, see 2.1.2 and 3.2.3) over 10 runs *for NS and ME-NS. The figure shows how on average, the population produced by ME-NS was able to outperform NS.*

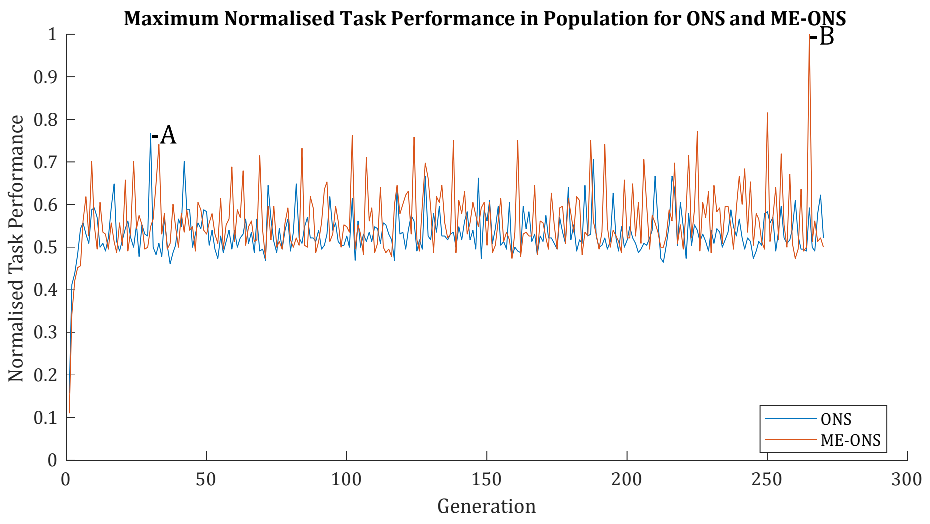


Figure 31: Maximum task performance (average hold time, see 2.1.2 and 3.2.3) over 10 runs for ONS and ME-ONS. Annotation A and B denote the generations at which peak performance was discovered for ONS, and ME-ONS respectively.

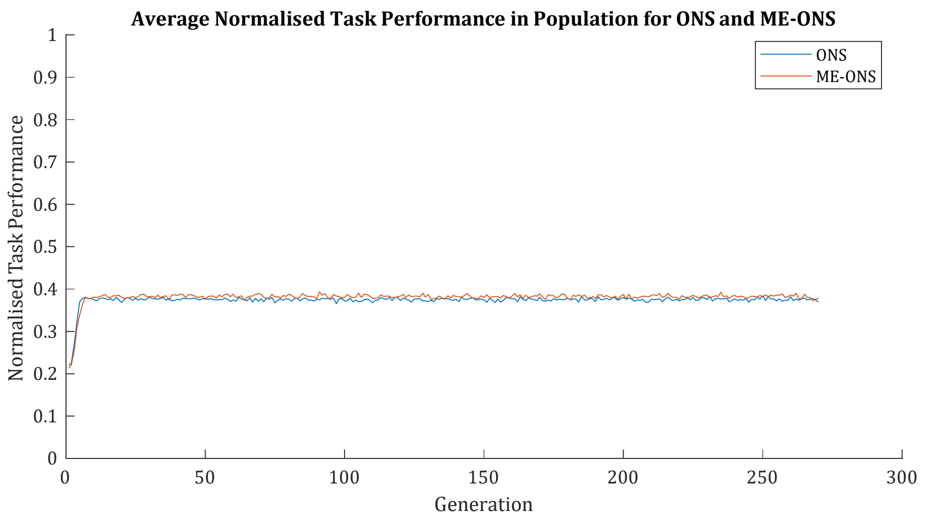


Figure 32: Average task performance (average hold time, see 2.1.2 and 3.2.3) over 10 runs for ONS and ME-ONS. The figure shows how similar the performance was for the variant's populations.

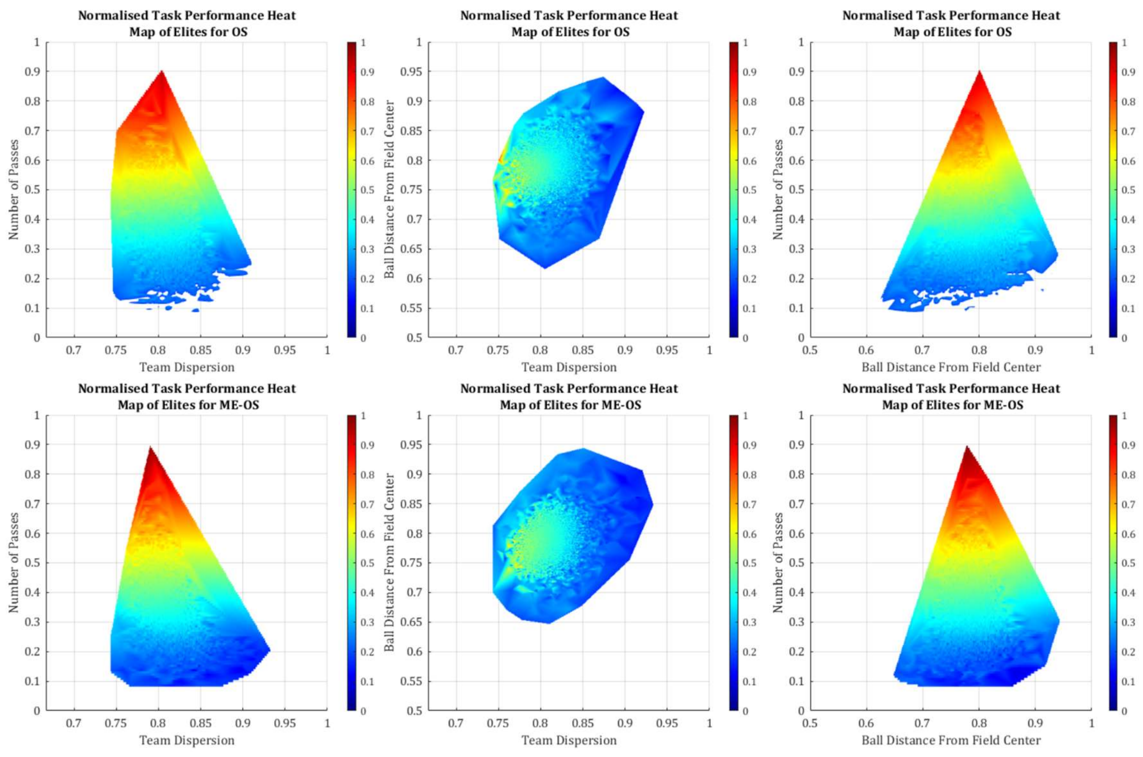


Figure 33: Interpolated task performance (average hold time, see 2.1.2 and 3.2.3) heat map for OS (top row) and ME-OS (bottom row) for each behaviour descriptor combination.

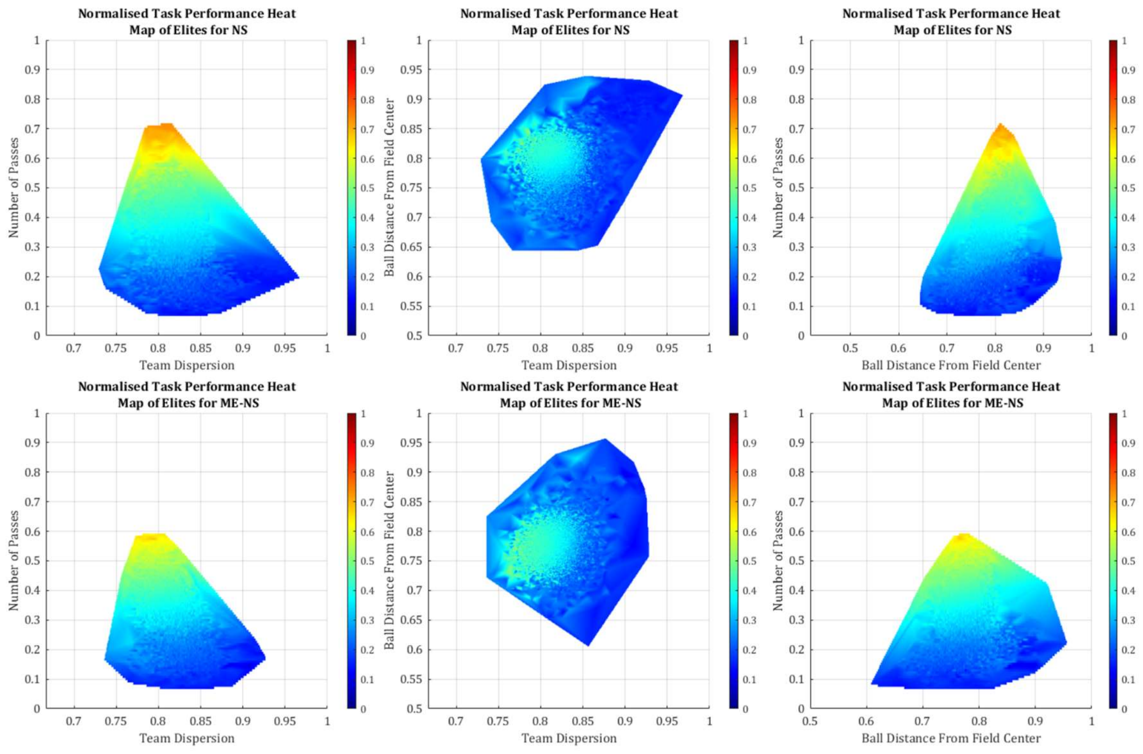


Figure 34: Interpolated task performance (average hold time, see 2.1.2 and 3.2.3) heat map for NS (top row) and ME-NS (bottom row) for each behaviour descriptor combination.

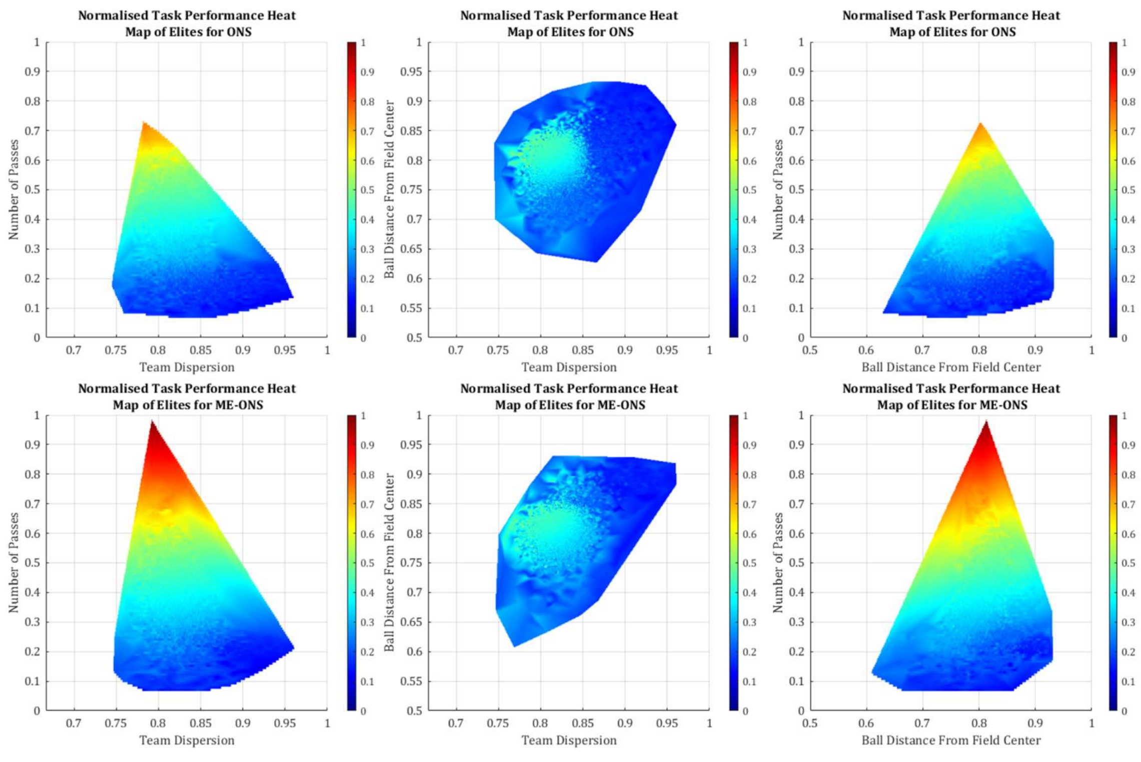


Figure 35: Interpolated task performance (average hold time, see 2.1.2 and 3.2.3) heat map for ONS (top row) and ME-ONS (bottom row) for each behaviour descriptor combination.

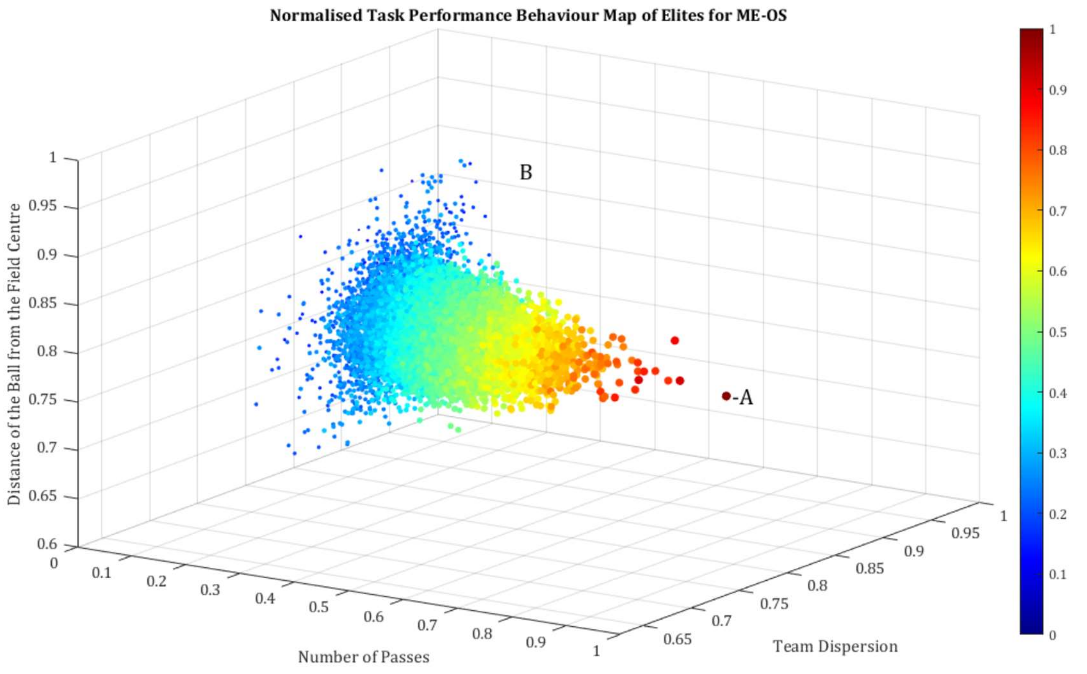


Figure 36: Normalised task performance (average hold time, see 2.1.2 and 3.2.3) behaviour space for ME-OS. Annotation A indicates best performing elite in the map. Annotation B indicates an extended region of exploration.

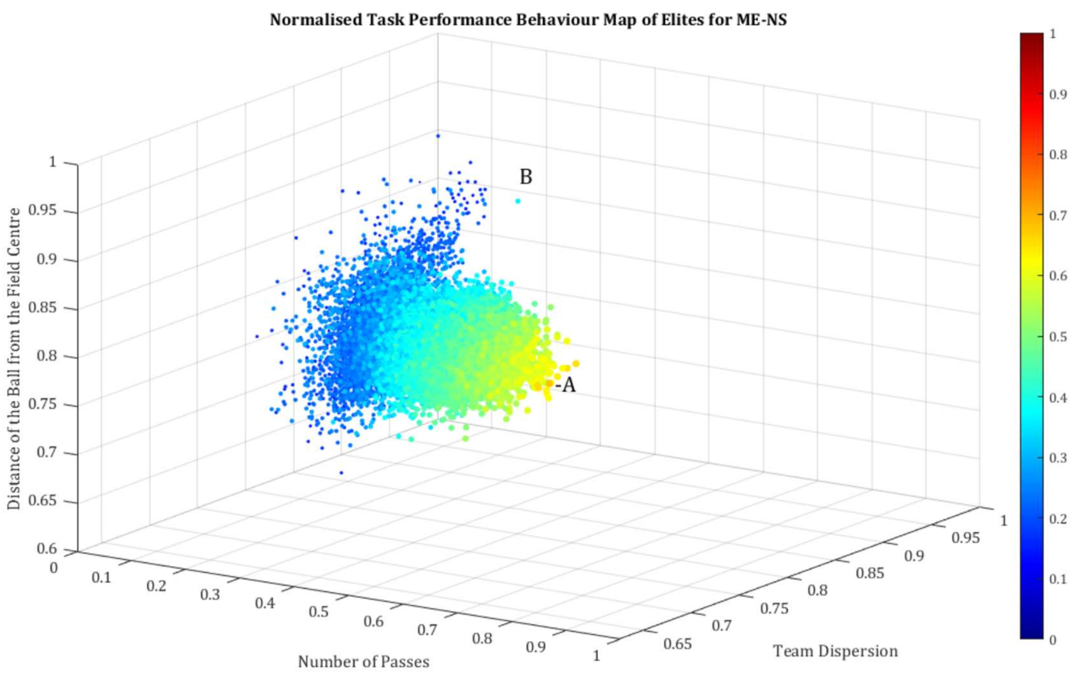


Figure 37: Normalised task performance (average hold time, see 2.1.2 and 3.2.3) behaviour space for ME-NS. Annotation A indicates best performing elite in the map. Annotation B indicates an extended region of exploration.

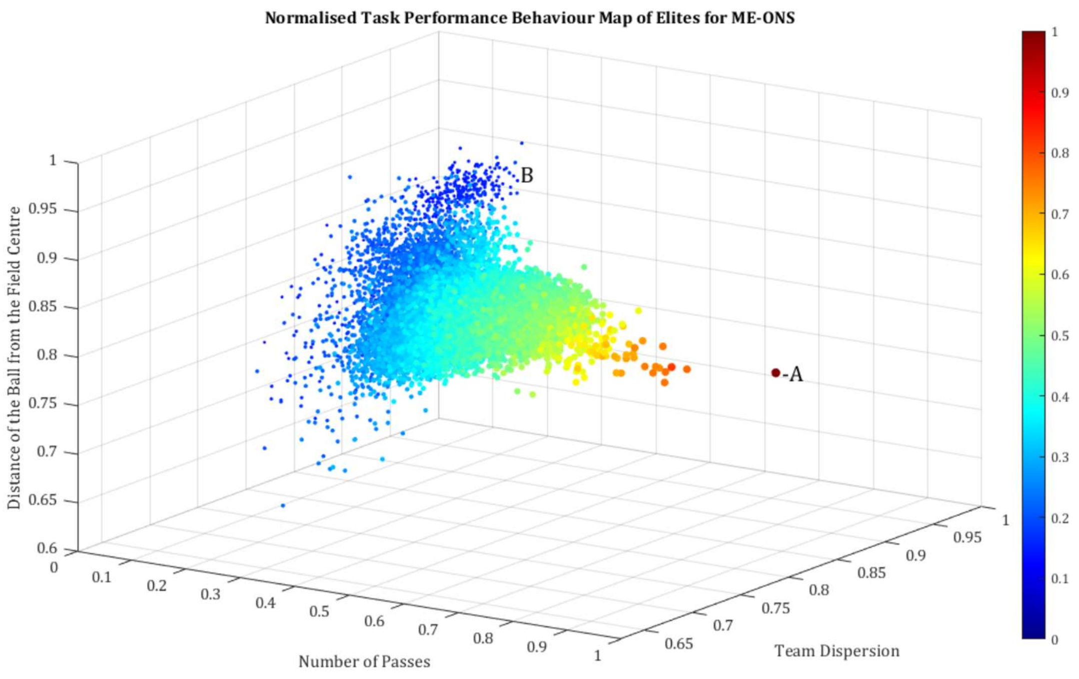


Figure 38: Normalised task performance (average hold time, see 2.1.2 and 3.2.3) behaviour space for ME-ONS. Annotation A indicates best performing elite in the map. Annotation B indicates an extended region of exploration.

5 Conclusion

In this thesis, our primary research objective was to assess the value of incorporating an illumination algorithm into objective, non-objective and hybrid search in the context of multi-agent systems. We made this assessment by way of experimentation in which we evaluated task performance and the ability of these methods to explore the behaviour search space.

Our experiments and analysis allowed us to explore the impact on exploration and exploitation of the search space, and develop recommendations for future researchers looking to evolve, and develop high performing agents in collective behaviour tasks. The subsequent sections provide a summary of the contributions made by this study as well as potential future avenues for research.

5.1 Contributions

We found that combining hybrid search with an illumination algorithm like MAP-Elites has a positive impact on the ability to evolve high performing cooperative agents in a complex system. We also found that combining MAP-Elites with Novelty search doesn't appear to have a positive benefit on either exploitation to develop high task performance, or exploration of the search space to discover more behaviour types. In general, this study also showed the explanative power of illumination to understand the behaviour space of the agents. The ability to visualise, analyse and explore the behaviour space thanks to the map of elites enabled us to discover and demonstrate that one key behaviour (Number of passes) had the greatest impact on task performance.

Based on our results it would seem that those looking to develop adaptive collective robotic controllers for complex tasks would do well to implement some combination of hybrid search and an illumination algorithm. In our Keep-away case study, it was clear that illumination can highlight behaviours of most importance to the task. This is an important contribution to consider for practical application. For example, our results indicate that developers of Keep-away robots should concentrate development on the ability of the robot to effectively pass the ball as many times as possible. Perhaps this would mean developing a fast-kicking action, and the ability to quickly target a fellow keeper, rather than spending time developing the robots' ability to traverse the field quickly or keep the ball in a certain area of the field. This recommendation can also be applied to other multi-agent, collective tasks such multi-robot search and rescue applications. In such a scenario, the knowledge of key behaviours would enable practitioners to direct design and development efforts towards the elements that most strongly benefit performance.

Although not a focus of this study, we were interested to understand the impact on evolutionary performance and behaviour space exploration that novelty search could have as compared to MAP-Elites as an illumination algorithm. We found that at least in the case of Keep-away, illumination was more effective than novelty search in exploring the high-performance behaviour space.

5.2 Known Limitations

In this research we conducted 10 independent iterations for each of the 6 variants (OS, NS, ONS, ME-OS, ME-NS, ME-ONS). This limited dataset meant that we were not always able to find statistical significance. Future researchers should perform 20 independent iterations for each variant in order to overcome this limitation.

Experimentation were computationally heavy and this limited our ability to conduct longer evolutionary runs. There was some evidence that more leaps in performance may be available if evolution could continue. Our recommendation is that future researchers conducting similar experiments run experiments for between 600 and 1000 generations.

5.3 Future

In future research we recommend that more behaviour descriptors are investigated to determine whether there are other behaviour traits that could aid our understanding of the behaviour space beyond the three behaviour traits investigated so far.

We also suggest considering additional Keep-away behaviour descriptors to ascertain if there exist other traits that could contribute to a more comprehensive understanding of the behaviour space, beyond the three traits included in this study. Future researchers should consider including distance travelled, length of pass (distance between keepers when passing the ball), angles between keepers and takers.

In previous research, Didi [8] explored solution complexity (topological complexity of evolved CPPNs) and we posit that it would be valuable to understand the impact of MAP-Elites on this point of interest. Didi [8] also made an important contribution by analysing the impact of increasing task complexity up to 6 vs. 5 Keepaway and future researchers are encouraged to replicate this scheme of increasing task complexity for experiments conducted in this research.

Finally, we were constrained by the time it took for experiments to run and thus recommend that future researchers consider refactoring and modifying the simulation and evolution framework to run asynchronously in parallel. Computational evolution inherently lends itself to parallel processing owing to the fact that an entire population of agents could be simulated in parallel with no interdependence. If such a modification were to be made, we anticipate a speed improvement of up to 150 times (assuming a population of 150).

References

1. K. Stanley, “Compositional pattern producing networks: A novel abstraction of development,” *Genetic programming and evolvable machines*, vol. 8, no. 2, 2007, pp. 131-162.

2. N. Kohl and R. Miikkulainen, “Evolving neural networks for fractured domains,” *Proc. Proceedings of the 10th annual conference on Genetic and evolutionary computation*, 2008, pp. 1405-1412.

3. H. Kitano, et al., “Robocup: The robot world cup initiative,” *Proc. Proceedings of the first international conference on Autonomous agents*, ACM, 1997, pp. 340-347.

4. S. Russell and P. Norvig, “Artificial Intelligence A Modern Approach Third Edition,” *Book Artificial Intelligence A Modern Approach Third Edition*, Series Artificial Intelligence A Modern Approach Third Edition, ed., Editor ed.^eds., PEARSON, 2010, pp.

5. C. Darwin, “ORIGIN OF SPECIES,” *The Athenaeum*, no. 2174, 1869, pp. 861-861.

6. F. Gomez and R. Miikkulainen, “2-d pole balancing with recurrent evolutionary networks,” *ICANN 98*, Springer, 1998, pp. 425-430.

7. G. Nitschke and S. Didi, “Evolutionary Policy Transfer and search Methods for Boosting Behavior Quality: robocup Keep-away case study,” *Frontiers in Robotics and AI*, vol. 4, 2017, pp. 62.

8. S. Didi, “Neuro-evolution behavior transfer for collective behavior tasks,” 2018.

9. J. Lehman and K.O. Stanley, “Exploiting open-endedness to solve problems through the search for novelty,” *Proc. ALIFE*, 2008, pp. 329-336.

10. J.. Mouret and J. Clune, “Illuminating search spaces by mapping elites,” *arXiv preprint arXiv:1504.04909*, 2015.

11. P. Verbancsics and K.O. Stanley, “Evolving static representations for task transfer,” *Journal of Machine Learning Research*, vol. 11, no. May, 2010, pp. 1737-1769.

12. J. Lehman and K.O. Stanley, “Abandoning objectives: Evolution through the search for novelty alone,” *Evolutionary computation*, vol. 19, no. 2, 2011, pp. 189-223.

13. A. Cully, et al., “Robots that can adapt like animals,” *Nature*, vol. 521, no. 7553, 2015, pp. 503.

14. A. Mackworth, “On seeing robots,” *Computer Vision: Systems, Theory and Applications*, World Scientific, 1993, pp. 1-13.

15. I. RoboCup Federation, “A Brief History of RoboCup,” https://www.robocup.org//a_brief_history_of_robocup.

16. H. Kitano, et al., “The RoboCup synthetic agent challenge 97,” *Proc. Robot Soccer World Cup*, Springer, 1997, pp. 62-73.

17. P. Stone, et al., “Keepaway soccer: From machine learning testbed to benchmark,” *Proc. Robot Soccer World Cup*, Springer, 2005, pp. 93-105.

18. A. Hussein, et al., “Deep Imitation Learning with Memory for Robocup Soccer Simulation,” *Proc. International Conference on Engineering Applications of Neural Networks*, Springer, 2018, pp. 31-43.

19. H. Kitano, et al., “Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research,” *Proc. Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on*, IEEE, 1999, pp. 739-743.

20. G. Nitschke, "Emergence of cooperation: state of the art," *Artif Life*, vol. 11, no. 3, 2005, pp. 367-396; DOI 10.1162/1064546054407194.

21. A. Pietro, et al., "Learning in RoboCup keepaway using evolutionary algorithms," *Proc. Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, Morgan Kaufmann Publishers Inc., 2002, pp. 1065-1072.

22. P. Stone, et al., "Reinforcement learning for robocup soccer keepaway," *Adaptive Behavior*, vol. 13, no. 3, 2005, pp. 165-188.

23. U.o.T.a. Austin, "Learning to Play Keepaway," 2013; http://www.cs.utexas.edu/~AustinVilla/sim/keepaway/.

24. P. Stone and R. Sutton, "Keepaway soccer: A machine learning test bed," *Proc. Robot Soccer World Cup*, Springer, 2001, pp. 214-223.

25. S. Raza, et al., "Teaching coordinated strategies to soccer robots via imitation," *Proc. Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, IEEE, 2012, pp. 1434-1439.

26. M. Campbell, et al., "Deep blue," *Artificial intelligence*, vol. 134, no. 1-2, 2002, pp. 57-83.

27. P. Darwen, "Computationally intensive and noisy tasks: Co-evolutionary learning and temporal difference learning on backgammon," *Proc. Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, IEEE, 2000, pp. 872-879.

28. R. Lippmann, "An introduction to computing with neural nets," *IEEE Assp magazine*, vol. 4, no. 2, 1987, pp. 4-22.

29. J. Friedman, et al., *The elements of statistical learning*, Springer series in statistics New York, NY, USA:, 2001.

30. S. Grossberg, "Recurrent neural networks," *Scholarpedia*, vol. 8, no. 2, 2013, pp. 1888.

31. R. Sutton, et al., *Reinforcement learning: An introduction*, MIT press, 1998.

32. M. Taylor, et al., "Comparing evolutionary and temporal difference methods in a reinforcement learning domain," *Proc. Proceedings of the 8th annual conference on Genetic and evolutionary computation*, ACM, 2006, pp. 1321-1328.

33. V. Mnih, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, 2015, pp. 529.

34. M. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, 2009, pp. 1633-1685.

35. Z. Buk, et al., "NEAT in HyperNEAT substituted with genetic programming," *Proc. International conference on adaptive and natural computing algorithms*, Springer, 2009, pp. 243-252.

36. K. Stanley, et al., "A hypercube-based encoding for evolving large-scale neural networks," *Artificial life*, vol. 15, no. 2, 2009, pp. 185-212.

37. S. Doncieux, et al., "Evolutionary Robotics: What, Why, and Where to," *Frontiers in Robotics and AI*, vol. 2, 2015; DOI 10.3389/frobt.2015.00004.

38. K. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, 2002, pp. 99-127.

39. P. Angeline, et al., "An evolutionary algorithm that constructs recurrent neural networks," *IEEE transactions on Neural Networks*, vol. 5, no. 1, 1994, pp. 54-65.

40. F. Gruau, et al., “A comparison between cellular encoding and direct encoding for genetic neural networks,” *Proc. Proceedings of the 1st annual conference on genetic programming*, MIT Press, 1996, pp. 81-89.

41. X. Yao, “Evolving artificial neural networks,” *Proceedings of the IEEE*, vol. 87, no. 9, 1999, pp. 1423-1447.

42. J. Mouret and S. Doncieux, “Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity,” *Proc. Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, IEEE, 2009, pp. 1161-1168.

43. C. Colas, et al., “Scaling map-elites to deep neuroevolution,” *Proc. Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 67-75.

44. D. Tarapore, et al., “How do different encodings influence the performance of the MAP-Elites algorithm?,” *Proc. Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ACM, 2016, pp. 173-180.

45. C. Mailer, et al., “Evolving gaits for damage control in a hexapod robot,” *Proc. Proceedings of the Genetic and Evolutionary Computation Conference*, 2021, pp. 146-153.

46. A. Eiben and C.A. Schippers, “On evolutionary exploration and exploitation,” *Fundamenta Informaticae*, vol. 35, no. 1-4, 1998, pp. 35-50.

47. J. Gomes and A.L. Christensen, “Generic behaviour similarity measures for evolutionary swarm robotics,” *Proc. Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 2013, pp. 199-206.

48. J. Gomes, et al., “Evolution of swarm robotics systems with novelty search,” *Swarm Intelligence*, vol. 7, 2013, pp. 115-144.

49. J. Gomes, et al., “Devising effective novelty search algorithms: A comprehensive empirical study,” *Proc. Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 2015, pp. 943-950.

50. P. Verbancsics, “Effective Task Transfer through Indirect Encoding,” 2011.

51. H.W. Lilliefors, “On the Kolmogorov-Smirnov test for the exponential distribution with mean unknown,” *Journal of the American Statistical Association*, vol. 64, no. 325, 1969, pp. 387-389.

52. W. Press, et al., “ea Numerical Recipes,” *Book ea Numerical Recipes*, Series ea Numerical Recipes, ed., Editor ed.^eds., Cambridge University Press, 1986, pp.

Appendix A: Statistical Tests

The experiments were conducted iteratively ten times for each variant, and the optimal performance achieved during each evolutionary run was documented for subsequent statistical analysis. The Lilliefors test [51] was utilized to categorize datasets as either non-parametric or belonging to the normal distribution.

Upon analysis, it was observed that Task Performance and Search Fitness data exhibited non-parametric behavior for the OS variant, while conforming to a normal distribution for all other variants. Similarly, Coverage data displayed non-parametric tendencies for the ME-ONS variant, contrasting with the normal distribution observed in other variants. Global reliability data, however, exhibited a normal distribution across all variants.

Given these varied outcomes, we proceeded to perform statistical testing using the two-sided Wilcoxon rank sum test, which is equivalent to the Mann-Whitney U-Test ($p < 0.05$) [52]. This test was chosen for its suitability in comparing pairs of datasets, irrespective of whether they followed a parametric or non-parametric distribution. The statistical tests were conducted through pair-wise comparisons among datasets corresponding to different method variants and p values from statistical test results are provided in Table 9-14.

Task performance						
	OS	**NS**	**ONS**	**ME-OS**	**ME-NS**	**ME-ONS**
OS	NaN	0.070	0.880	0.623	0.028	0.345
NS	0.070	NaN	0.162	0.070	0.970	0.325
ONS	0.880	0.162	NaN	0.273	0.037	0.880
ME-OS	0.623	0.070	0.273	NaN	0.021	0.427
ME-NS	0.028	0.970	0.037	0.021	NaN	0.545
ME-ONS	0.345	0.325	0.880	0.427	0.545	NaN

Table 9: Task performance statistical tests for pair-wise comparison at 95% confidence interval (p < 0.05, Wilcoxon rank sum test) for each variant (OS, NS, ONS, ME-OS, ME-NS, ME-ONS).

Global fitness						
	OS	**NS**	**ONS**	**ME-OS**	**ME-NS**	**ME-ONS**
OS	NaN	0.070	0.880	0.623	0.028	0.345
NS	0.070	NaN	0.162	0.070	0.970	0.325
ONS	0.880	0.162	NaN	0.273	0.037	0.880
ME-OS	0.623	0.070	0.273	NaN	0.021	0.427
ME-NS	0.028	0.970	0.037	0.021	NaN	0.545
ME-ONS	0.345	0.325	0.880	0.427	0.545	NaN

Table 10: Global fitness statistical tests for pair-wise comparison at 95% confidence interval (p < 0.05, Wilcoxon rank sum test) for each variant (OS, NS, ONS, ME-OS, ME-NS, ME-ONS).

Coverage						
	OS	**NS**	**ONS**	**ME-OS**	**ME-NS**	**ME-ONS**
OS	NaN	0.385	0.623	0.521	0.910	0.791
NS	0.385	NaN	0.496	0.089	0.212	0.121
ONS	0.623	0.496	NaN	0.140	0.571	0.173
ME-OS	0.521	0.089	0.140	NaN	0.473	1.000
ME-NS	0.910	0.212	0.571	0.473	NaN	0.791
ME-ONS	0.791	0.121	0.173	1.000	0.791	NaN

Table 11: Coverage statistical tests for pair-wise comparison at 95% confidence interval (p < 0.05, Wilcoxon rank sum test) for each variant (OS, NS, ONS, ME-OS, ME-NS, ME-ONS).

Global reliability						
	OS	**NS**	**ONS**	**ME-OS**	**ME-NS**	**ME-ONS**
OS	NaN	0.045	0.038	0.910	0.021	0.162
NS	0.045	NaN	0.734	0.089	1.000	0.186
ONS	0.038	0.734	NaN	0.026	0.734	0.186

ME-OS	0.910	0.089	0.026	NaN	0.054	0.571
ME-NS	0.021	1.000	0.734	0.054	NaN	0.307
ME-ONS	0.162	0.186	0.186	0.571	0.307	NaN

Table 12: Global reliability statistical tests for pair-wise comparison at 95% confidence interval (p < 0.05, Wilcoxon rank sum test) for each variant (OS, NS, ONS, ME-OS, ME-NS, ME-ONS).

Search fitness						
	OS	**NS**	**ONS**	**ME-OS**	**ME-NS**	**ME-ONS**
OS	NaN	0.021	0.162	0.623	0.011	0.241
NS	0.021	NaN	0.140	0.014	0.473	0.307
ONS	0.162	0.140	NaN	0.186	0.241	0.734
ME-OS	0.623	0.014	0.186	NaN	0.011	0.345
ME-NS	0.011	0.473	0.241	0.011	NaN	0.910
ME-ONS	0.241	0.307	0.734	0.345	0.910	NaN

Table 13: Search fitness statistical tests for pair-wise comparison at 95% confidence interval (p < 0.05, Wilcoxon rank sum test) for each variant (OS, NS, ONS, ME-OS, ME-NS, ME-ONS).

Population task performance at best generation (respectively to each variant)						
	OS	**NS**	**ONS**	**ME-OS**	**ME-NS**	**ME-ONS**
OS	NaN	0.000	0.046	0.353	0.028	0.044
NS	0.000	NaN	0.031	0.000	0.051	0.033
ONS	0.046	0.031	NaN	0.001	0.881	0.880
ME-OS	0.353	0.000	0.001	NaN	0.001	0.001
ME-NS	0.028	0.051	0.881	0.001	NaN	0.935
ME-ONS	0.044	0.033	0.880	0.001	0.935	NaN

Table 14: Task performance statistical tests for the population of genomes at the generation that produced the best task performance for each variant respectively. Pair-wise comparison conducted at 95% confidence interval (p < 0.05, Wilcoxon rank sum test) for each variant (OS, NS, ONS, ME-OS, ME-NS, ME-ONS).