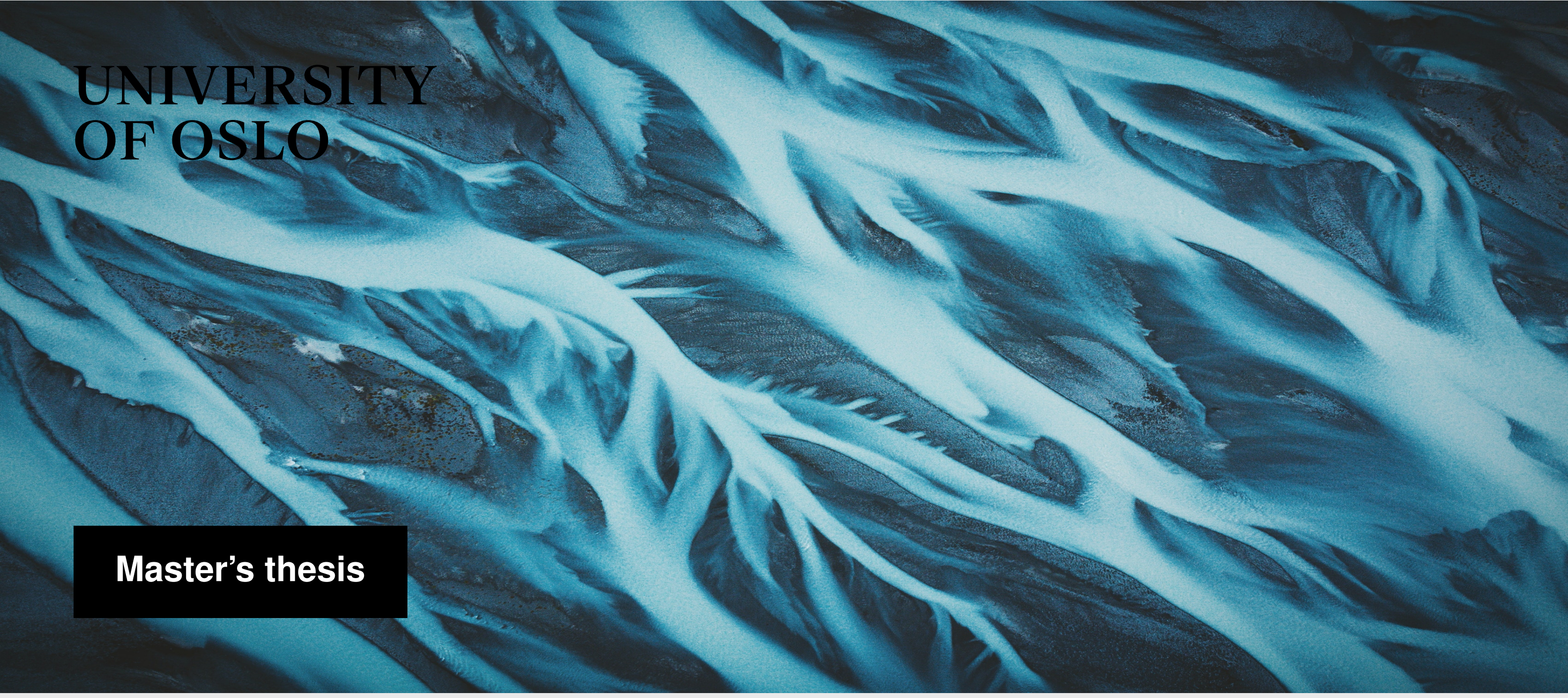


Cultivating Diversity: a Comparison of Diversity Objectives in Neuroevolution

Didrik Spanne Reilstad

Informatics: Robotics and Intelligent Systems
60 ECTS study points

Department of Informatics
Faculty of Mathematics and Natural Sciences

Spring 2023

Didrik Spanne Reilstad

Cultivating Diversity: a Comparison of Diversity Objectives in Neuroevolution

Supervisor:
Kai Olav Ellefsen

Abstract

Inspired by biological evolution's ability to produce the complexity that is human brains, *neuroevolution* utilizes evolutionary algorithms for optimizing the hyperparameters and structure of neural networks. However, evolutionary algorithms fail to produce the same type of diversity as biological evolution can with the abundant range of adaptable and complex organisms in nature. Encouraging *diversity* in neuroevolution has seen increased interest in recent years with methods such as Novelty Search and Quality-Diversity optimization. Another promising, but less explored approach, is to explicitly encourage diversity with an additional diversity objective. There is, however, a lack of knowledge regarding the relationship between the type of diversity encouraging objective and the characteristics of the targeted problem. For instance, should a diversity of brain structures, behaviors, or neural firing patterns be encouraged when optimizing a walking robot?

This thesis compares two different diversity objectives for each type of diversity, *behavioral*, *structural*, and *representational* diversity, on problems with unique characteristics. To make the comparison more informative, the following problem characteristics are studied: (1) Modularity, (2) Regularity, (3) Deceptiveness, and (4) Environment space representation. Results show a clear correlation between the performance of the type of diversity objective and the characteristics of the problem. *Ad hoc* behavioral diversity performed best on deceptive problems, structural diversity was most efficient on modular problems, and representational diversity outperformed other objectives on problems with a high degree of regularity. Furthermore, significant performance differences between diversity objectives of the same type were found.

A second contribution of the thesis is introducing a new type of diversity called representational diversity for encouraging a diversity of learned neural network representations. The first steps of how representational diversity can be adapted to neuroevolution are outlined. Early promising results show that representational diversity merits further study. There are indications that representational diversity is suitable for problems where neuron activations of neural networks potentially play an important role in solving the task.

Acknowledgements

I would like to thank my supervisor, Kai Olav Ellefsen, for invaluable guidance and support throughout the thesis work, and to fellow students for the interesting and inspiring discussions.

Lastly, I want to thank my family for their unyielding love and support.

Contents

List of Figures

List of Tables

List of Algorithms

1 | Introduction

1.1 Motivation

Historically, artificial intelligence (AI) research has successfully been able to solve specific problems that most humans are not very good at or automate laborious and repetitive tasks. For example playing chess, traffic control, vehicle steering, image recognition, speech recognition, diagnosis of medical diseases, and recommendation systems for social media and e-commerce [1, 2]. However, the direct approach of optimizing a single solution for a single problem prevalent in AI research neglects aspects of biological intelligence, such as autonomy, self-healing, social interaction, evolution, and learning, that make biological organisms highly adaptable and successful in unknown and changing environments [2]. Finding alternative learning methods to encompass these aspects is an area of much research in recent years, especially in fields like reinforcement learning, evolutionary computing, swarm intelligence, and embodied cognitive science.

Traditionally, artificial neural networks are trained using gradient-based learning through a combination of backpropagation and stochastic gradient descent [1]. An alternative to gradient-based learning, partly inspired by the ability of biological evolution to produce the complexity that is natural brains in animals and humans, is *neuroevolution* (NE). Neuroevolution harnesses the capabilities of an evolutionary algorithm to optimize the hyperparameters, but also the topology and activation function of neural networks, which are capabilities typically unavailable to gradient-based approaches [3]. The structure of a neural network can greatly impact its performance [4]. Additionally, instead of optimizing a single neural network, neuroevolution employs and maintains a population of neural networks, enabling extreme exploration and parallelization of solutions.

A remarkable feat of biological evolution is the diversity of complex organisms it produces, all highly adaptable and high-performing in their niche. An evolutionary algorithm attempts to emulate this same evolutionary process in order to facilitate the exploration of solutions but often fails to produce the same level of diversity even with evolutionary operators and traditional diversity-maintenance techniques. The lack of diversity often results in sub-optimal solutions as the population has converged to a local optimum.

Explicitly encouraging *diversity* has emerged as a way to prevent early convergence to local optima and make the evolutionary search more efficient and improve the overall performance [3]. One successful method in diversity-driven neuroevolution is Novelty Search [5]. Instead of searching for solutions with higher performance, as is usually the case, the idea is to search for solutions with *novel* behaviors and thus explore a more diverse set of solutions. Novelty search demonstrated impressive results by outperforming performance-based search, especially on deceptive problems.

Expanding on the work of novelty search, recent work on diversity-driven neuroevolution uses a multi-objective evolutionary algorithm to simultaneously optimize solutions according to a performance objective and a diversity objective. Using an additional diversity objective has demonstrated the ability to discover novel solutions and ultimately lead to better performance more efficiently [3]. A common type of diversity objective is *behavioral diversity*, which encourages different behaviors [6, 7]. *Genetic diversity* objectives can be used to encourage genetically different solutions [8, 9], whereas *structural diversity* objectives encourages structurally different solutions [4, 9, 10].

Encouraging diversity in neuroevolution with diversity objectives has demonstrated impressive results, but there is a lack of knowledge regarding the relationship between the type of diversity objective and the characteristics of the problem. For instance, if there is a type of diversity objective that is more effective for problems with certain characteristics. Little is known about how the characteristics of the targeted problem may affect the effectiveness of certain diversity objectives. Being able to determine the type of diversity objective to use based on only the characteristics of the problem would be very useful instead of having to compare multiple objectives every time. For example, being able to determine if a diversity of behaviors, network structure, or neuron firing patterns is most suitable for a mobile robot task.

Moreover, there are different advantages and limitations for each type of diversity objective. Behavioral diversity objectives can be quite effective but are limited by usually being domain-dependent and having to be adapted to each problem. In contrast, structural diversity objectives are domain-independent but can be expensive to compute. A research gap in diversity-driven neuroevolution is combining the advantages of both behavioral and structural diversity into a new type of diversity. A new type of diversity that is able to exploit both the behavior and structure of a neural network to encourage a diversity of *learned neural network representations.*

1.2 Research Goals

With the above-stated challenges of diversity-driven neuroevolution in mind, the primary research goal of this thesis is to:

- Compare the effectiveness of various diversity objectives in neuroevolution on a number of targeted problems with unique combinations of problem characteristics, and if possible, determine if specific diversity objectives are more effective on certain problems.

A secondary research goal of this work is to introduce a new type of diversity called *representational diversity* for encouraging a diversity of learned neural network representations. In short, the goal is to:

- Investigate if similarity metrics for measuring the similarity between learned neural network representations can be adapted to neuroevolution with neural networks of different topologies, and if so, determine if representational diversity has the potential to become a new type of diversity able to encompass both the structure and behavior of neural networks.

1.3 Contributions

The main contribution of this thesis is a comparison of different types of diversity objectives, and their effectiveness on different problems. Both behavioral and structural diversity objectives are compared, in addition to a new type of diversity introduced in this work called *representational diversity*. To make the comparison of the diversity objectives more informative, a set of interesting problem characteristics is proposed and the targeted problems are characterized according to these characteristics.

The contributions of this work can be summarized as follows:

- Identified relevant diversity objectives, two for each type of diversity: *behavioral*, *structural*, and *representational* diversity.
- Identified relevant problems with unique combinations of characteristics often targeted in neuroevolution. The targeted problems are: (1) The Retina problem, (2) The Tartarus problem, (3) A maze navigation problem, and (4) A robot locomotion problem.
- Proposed the following set of interesting problem characteristics: (1) Modularity, (2) Regularity, (3) Deceptiveness, and (4) Environment space representation. A formal characterization of the targeted problems was performed.

- Introduced a new type of diversity called *representational diversity*, and showed how this type of diversity can be adapted to neuroevolution. Representational diversity objectives were found to be especially effective on the robot locomotion problem, where novel neuron activations are important for discovering an effective gait.

Building on the above contributions, the *main* contribution of this thesis is as follows:

- Compared diversity objectives and found a correlation between the type of diversity objective and the problem characteristics. Behavioral diversity objectives were found to be especially effective on deceptive problems, whereas structural diversity objectives outperformed other diversity types on modular problems. Finally, representational diversity objectives were most effective on problems with a high degree of regularity.

1.4 Outline

The thesis is divided into the following seven chapters: (1) Introduction, (2) Background, (3) Methodology, (4) Experimental Setup, (5) Results and Analysis, (6) Discussion, and (7) Conclusion.

Chapter 2 introduces the theoretical background for neuroevolution and previous work on the use of diversity in neuroevolution. Chapter 3 presents the diversity objectives and the neuroevolution algorithm employed to compare the diversity objectives, including the motivation for introducing a new type of diversity. Finally, interesting problem characteristics are defined to make the comparison more informative.

Chapter 4 presents the targeted problems and a characterization of them using the characteristics previously defined in Chapter 3. Chapter 5 presents the results for all experiments performed and analysis describing the results. Chapter 6 discusses the results more in-depth with regard to the research goals. Finally, Chapter 7 summarizes contributions and findings, including future work suggestions.

2 | Background

Chapter 2 introduces the theoretical background for *neuroevolution* and the role of *diversity* in neuroevolution, including previous work in this area. In particular, how guiding evolution with diversity objectives can improve the efficiency and performance of neuroevolution algorithms. The chapter will provide the reader with a general understanding of *evolutionary algorithms* as an optimization algorithm, using *multi-objective optimization* for optimizing multiple objectives, and the structure and application of *artificial neural networks.*

2.1 Evolutionary Algorithms

2.1.1 Fundamentals of Evolutionary Algorithms

Evolutionary algorithms (EAs) are optimization algorithms inspired by biological evolution [11, 12]. EAs employ a *population* of candidate solutions called *individuals.* Using mechanisms such as *reproduction*, *mutation*, *crossover* and *selection* from biological evolution, the individuals are evaluated according to a *fitness function* and further optimized over multiple generations. A solution (or individual) is represented by a genetic representation called the *genotype*, which encodes the observable traits of the *phenotype* as *genes.* For example with neural networks, the genotype could be a list of neurons and a list of connections between neurons, and the phenotype could be the complete network as a graph.

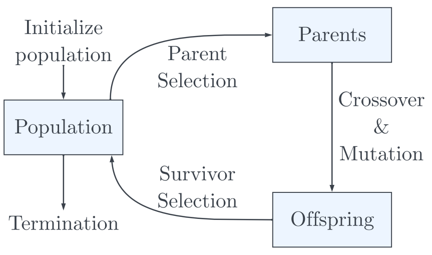


Figure 2.1: Illustration of a general evolutionary algorithm.

Algorithm 1 Evolutionary Algorithm

```
INITIALIZE population with random candidate solutions
EVALUATE each candidate
while not TERMINATION CONDITION do
    SELECT parents
    CROSSOVER pairs of parents
    MUTATE the resulting offspring
    EVALUATE new candidates
    SELECT individuals for next generation
```

The general evolutionary cycle is illustrated in figure 2.1, accompanied by pseudocode (see algorithm 1). For each generation, the individuals in the population are evaluated by a fitness function. The fitness of an individual characterizes how well that particular candidate solution solves the problem. Individuals from the current generation are then selected as parents. Using evolutionary operators such as crossover and mutation on the parents, offspring is created to partially or completely replace the old population for the next generation. Survivor selection dictates which offspring are to be included in the next generation. The process ends when a termination condition is met, either by finding a sufficiently optimal solution or reaching the generation limit.

In the early phase of the evolutionary process, few or none of the generated individuals will be good solutions for solving the problem. Over multiple generations of selecting fit individuals and propagating their genes to new offspring using evolutionary operators, the *fitness landscape* is incrementally explored and gradually steered towards optimal solutions. In evolutionary optimization, the fitness landscape is defined by the fitness function f mapping every possible solution (or genotype) to a fitness. For real-world problems, going through all possible solutions is impossible. Evolutionary algorithms, by employing a population of different solutions, facilitate exploring the fitness landscape and finding an acceptable solution.

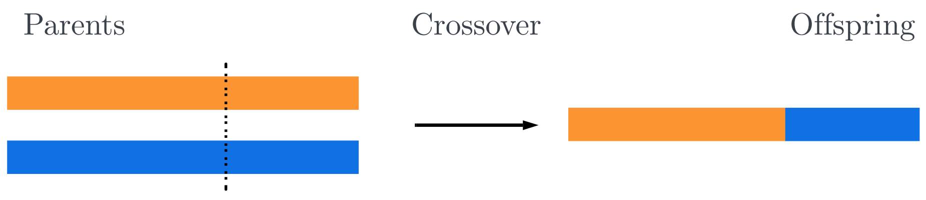
2.1.2 Balancing exploitation and exploration

There are two main competing forces driving the evolutionary process: *exploitation* of solutions and *exploration* of solutions [11–13]. When employing evolutionary algorithms, it is crucial to balance exploration and exploitation. If too much emphasis is placed on exploitation, the algorithm will quickly converge to a local optimum with a suboptimal solution. On the other hand, with too much emphasis on exploration, the evolutionary search does not find good solutions within a reasonable amount of time and may never converge [11–13].

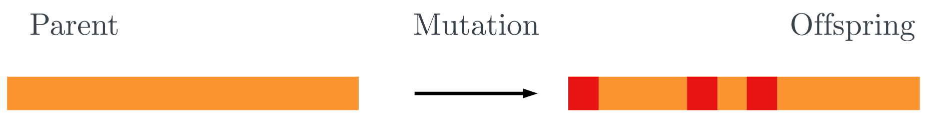
The type of parent/offspring selection used can greatly affect the selection pressure, i.e. exploitation of solutions. The level of selection pressure regulates the degree to which individuals with higher fitness are favored over individuals with lower fitness. Selection aims to increase or at least preserve the average quality of the candidate solutions in the population from one generation to the next. Common types of selection are tournament selection, roulette wheel selection, rank selection, and random selection. For example, the tournament selection method holds multiple "tournaments" among k random individuals from the population, and the winner of the tournament is selected as a parent for the next generation. The selection pressure is easily controlled by adjusting the tournament size k, resulting in high selection pressure for large values of k whereas a 1-way tournament with $k = 1$ is equivalent to random selection.

Elitism is commonly used to make sure that the evolutionary algorithm has a non-degrading performance. The best individual or the n best individuals from the previous generation is transferred to the next generation without changes to the genotype. Elitism ensures that the progress from previous generations is not lost when exploring other parts of the fitness landscape.

Evolutionary operators such as crossover and mutation create diversity and variation in the population facilitating exploration of solutions. Crossover, also called recombination, generates a new offspring by combining the genetic information of two parents. The offspring's genotype will consist of genes from both parents (see example in figure 2.2a). The resulting offspring will lie somewhere between or near its parents in the fitness landscape. The crossover operator exploits the genetic information evolved from previous generations that are known to be good but creates a new solution slightly different from its parents. Crossover as an evolutionary operator is analogous to sexual reproduction in biology.



(a) Crossover between the genes of two parents results in an offspring containing parts of genes from both parents. The crossover point is shown as a dotted line.



(b) Mutation of a single cloned parent results in an offspring containing the genes of the parent with one or more genes modified. Modified genes are shown in red.

Figure 2.2: Illustration showing how crossover and mutation behave, and what the effect is on the genes of the resulting offspring. The color bar is a general representation of the genotype. The change in color signifies a change in the respective genes.

Mutation introduces variation and diversity into the population by randomly modifying one or more genes in an individual (see example in figure 2.2b), akin to biological mutation. The mutation operator is applied to either a single cloned parent or the resulting offspring from crossover between two parents. How different the resulting offspring is from its parent will depend on the number of genes mutated and the type of mutation performed. Mutation is particularly important in EAs to explore different solutions in the fitness landscape. Whereas the crossover operator is at times not used due to its somewhat exploitative effect compared to mutation, and also using crossover in a *meaningful* way can be difficult depending on the complexity of the mapping between genotype and phenotype (e.g. list of nodes and edges to a complete neural network).

2.1.3 Diversity-preserving mechanisms

A common problem with population-based evolutionary algorithms without diversity maintenance mechanisms is that the best individual found may take over the whole population before the fitness landscape is properly explored [14]. The result is early convergence at a local optimum leading to a rapid reduction in the diversity of the population, and therefore only a limited part of the search landscape is repeatedly explored unnecessarily (see figure 2.3a). Introducing diversity-preserving mechanisms has been shown to significantly increase the likelihood of finding multiple optima more efficiently [14], and avoid premature convergence and enable effective crossover [15] compared to EAs with no diversity maintenance.

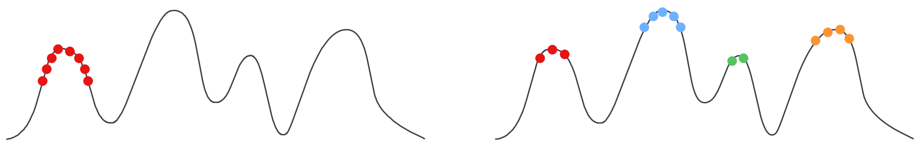

(a) Without niching **(b)** With niching

Figure 2.3: Illustration showing the effect of niching on a simple 2D fitness landscape example. To the left in **(a)**, with no niching, the candidate solutions cluster at a single peak in the fitness landscape. To the right in **(b)**, the candidate solutions cluster in groups, i.e. niches, at various peaks in the fitness landscape when niching is used.

Several mechanisms have been proposed in order to maintain a diverse population. A direct approach is *niching* [16], where individuals in the population are divided into *niches* and only reproduction with individuals in the same niche is allowed. The result is groups of similar solutions gathering at different local optima in the fitness landscape (see example in figure 2.3). A more indirect approach is *fitness sharing* [17], where the population is divided into niches, and individuals in each niche are forced to share their fitness. The idea is to control the number of individuals in each niche by forcing them to share fitness, thereby discouraging convergence to a single area of the fitness landscape. *Crowding* [18] is another indirect mechanism. The offspring have to compete with a similar already existing individual, which is found in a set of randomly selected individuals from the entire population. The idea is that selection pressure is only applied to similar candidate solutions, ensuring that offspring can only replace similar candidate solutions.

The aforementioned mechanisms help maintain diversity and avoid early convergence, but the performance of EAs is ultimately dependent on the fitness function. The fitness function is responsible for quantifying how well candidate solutions perform. As EAs are utilized to solve more complex and non-trivial tasks (e.g. robot controllers), the fitness function quickly becomes the limiting factor [19]. Designing a comprehensive fitness function for finding novel solutions to complex tasks is a difficult challenge, and often time-consuming because they are usually domain-dependent requiring varying degrees of *a priori* task knowledge from the designer. A too "simple" fitness function, even with diversity-preserving mechanisms, has been shown to be vulnerable to deceptive tasks and even become misleading in the search for a solution [7, 20, 21]. For this reason, *Multi–Objective Evolutionary Algorithms* (see section 2.2.2) and other *diversity-driven* methods such as Novelty Search (see section 2.6), have seen increased interest as a viable option to produce a more diverse set of high performing solutions.

2.2 Multi–Objective Optimization

Multi-objective optimization [22, 23] is concerned with the optimization of multiple objective functions simultaneously. Multi-objective optimization has been successfully applied to a wide variety of fields, such as economics, engineering and logistics, where decisions taken will often be a trade-off between two or more conflicting objectives. For example, maximizing the top speed of a car and maximizing the energy efficiency of a car are two conflicting objectives. Consequently, for non-trivial multi-objective optimization problems, no single solution will exist that fully optimizes both or all objectives.

Formally, a multi-objective optimization problem can be defined as

$$\begin{aligned} \text{Set of objectives } \mathbf{F}(\mathbf{x}) &= (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})), \\ \text{Minimize/Maximize } f_i(\mathbf{x}), &\quad \text{for } i = 1, 2, \ldots, k \\ \text{subject to } \mathbf{x} &\in S. \end{aligned} \tag{2.1}$$

where $k \geq 2$ is the number of objective functions and solution $\mathbf{x}$ is a decision vector of n decision variables, $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$. The solutions constitute a *feasible decision variable space* $S \subset \mathbb{R}^n$. S is the feasible set of decision vectors, meaning it is the set of solutions that satisfies all constraints and variable bounds applied. The set of objective functions $\mathbf{F}$ constitute the *objective space* $Z \subset \mathbb{R}^k$, and for each solution $\mathbf{x}$ there exists a point $\mathbf{F}(\mathbf{x}) = \mathbf{z}$

in the objective space. It is often useful to visualize the solutions **x** in the objective space with the corresponding points **z** (see figure 2.4a).

2.2.1 Pareto–Optimality

The challenge with multi-objective optimization is deciding which solutions are the most optimal. As described previously, no single solution exists that fully optimizes all objectives because of the trade-off between conflicting objectives. In order to find the set of optimal solutions, an ordering of all solutions is needed. For this purpose, *domination* is used to describe the relationship between two solutions. Domination between two solutions [22, 23] is defined as:

Definition 1 *A solution* $\mathbf{x}^{(1)}$ *is said to dominate the other solution* $\mathbf{x}^{(2)}$*, if both the following conditions are true:*

1. *The solution* $\mathbf{x}^{(1)}$ *is no worse than* $\mathbf{x}^{(2)}$ *in all objectives. Thus, the solutions are compared based on their objective function values (or location of the corresponding points,* $\mathbf{z}^{(1)}$ *and* $\mathbf{z}^{(2)}$*, on the objective space).*
2. *The solution* $\mathbf{x}^{(1)}$ *is strictly better than* $\mathbf{x}^{(2)}$ *in at least one objective.*

Consequently, a pair-wise comparison for a given set of solutions can be made to establish which solutions are dominated and which solutions are *non-dominated*. The set of non-dominated solutions contains the solutions that are not dominated by any other solutions in the same set. When visualized in the objective space, the non-dominated solutions make up a *non-dominated front* (see figure 2.4b).

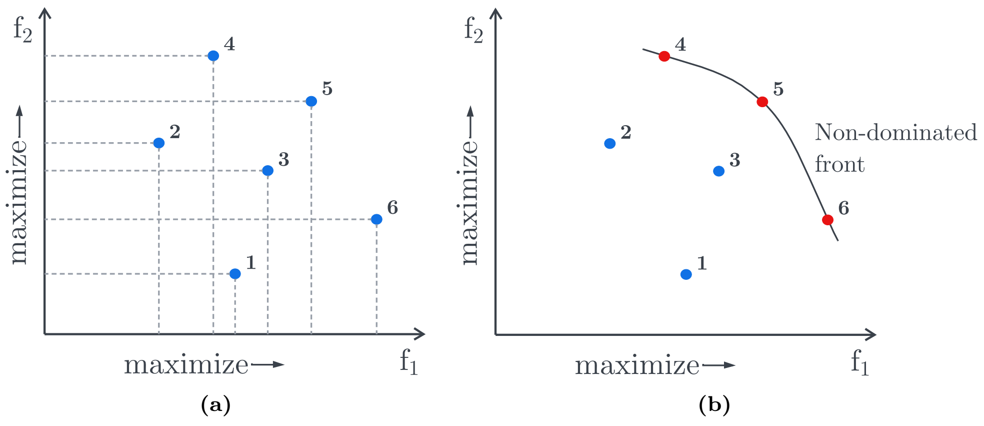


Figure 2.4: **(a)** A set of solutions visualized as points in the objective space. **(b)** The non-dominated front is visualized for the same set of solutions. The non-dominated solutions in this example are 4, 5, and 6. Figures adapted from [22].

A non-dominated solution is also called a Pareto-optimal solution. In terms of multi-objective optimization, a solution is a Pareto-optimal solution if no other existing solution dominates it. In other words, a solution is Pareto-optimal if none of the objectives can be further optimized without causing degradation of one of the other objectives. Thus, the set of non-dominated solutions is the set of Pareto-optimal solutions and these solutions make up the Pareto-front (non-dominated front). Choosing a single optimal solution among the Pareto-set will require the subjective preferences of a human decision-maker. To that end, it is advantageous to find an even distribution of Pareto-optimal solutions along the Pareto-front resulting in a more representative set of solutions to choose from.

2.2.2 Multi–Objective Evolutionary Algorithms

A lot of multi-objective optimization problems are often too complex to be solved using exact methods. *Multi-Objective Evolutionary Algorithms* (MOEAs) have been successfully applied to many of these problems, including real-world problems [24, 25]. Due to the inherent parallelism and use of a population of solutions, MOEAs are especially suited to multi-objective optimization problems as they can approximate the Pareto-set in a single optimization run. It is important to note that MOEAs can only approximate the Pareto-set and cannot guarantee that the Pareto-optimal solutions are found.

Despite the successful applications of MOEAs for solving multi-objective optimization problems, several challenges remain regarding the use of MOEAs. One of the challenges is *scalability*. As the number of objectives increases, the ability of MOEAs to search the objective space deteriorates and the number of solutions required to achieve an even distribution along the Pareto-front increases exponentially [26, 27].

Another challenge is *computationally expensive* objective functions. Evaluating candidate solutions is necessary in order to guide the search, and every candidate solution is usually evaluated for each generation. Using multiple expensive objective functions will therefore drastically increase the time needed to fully evaluate each candidate solution per generation. Techniques such as *fitness inheritance* and *problem approximation* can somewhat help reduce the computational cost, but can potentially come with a performance trade-off [28]. When applying fitness inheritance, an *inheritance proportion* decides if individuals will be assigned a fitness using the objective function as usual or the average fitness of the parents to reduce the total number of objective function evaluations. Problem approximation is a technique for replacing the original problem statement with another one that is approximately the same but easier to solve.

Lastly, as with single-objective evolutionary algorithms described in section 2.1, an essential challenge with MOEAs is the selection of individuals for reproduction and survival while also maintaining diversity.

A number of different MOEAs have been proposed to attempt to solve the aforementioned challenges encountered in multi-objective optimization. Parallel computing can drastically help mitigate the problem of scalability and expensive objective functions but not with selection and diversity maintenance. For this reason, the main difference between the various algorithms is the selection method. Most multi-objective evolutionary algorithms use a Pareto-based ranking scheme for selection.

Among them, the *Strength Pareto Evolutionary Algorithm 2* (SPEA2) [29] and *Non-dominated Sorting Genetic Algorithm-II* (NSGA–II) [30] were shown to perform significantly better across multiple test problems compared to other MOEAs [25, 29], and have become the standard. An important factor in the performance gap between SPEA2 and NSGA–II and the other algorithms is the use of elitism. Both SPEA2 and NSGA–II use elitism during selection, and algorithms with no elitist strategy showed an increase in performance when using SPEA2's elitist strategy. SPEA2 mainly differs from NSGA–II in how individuals are ranked. SPEA2 performs better in high-dimensional objective spaces but is computationally slower than NSGA–II [29]. NSGA–II is described in more detail in the following section 2.2.3.

2.2.3 NSGA–II

NSGA–II [30], short for *Non-dominated Sorting Genetic Algorithm-II*, is an efficient and well-established algorithm for multi-objective optimization problems. The algorithm implements three main features that make it successful while also being relatively computationally efficient:

- Elitism
- Explicit diversity-preserving mechanism
- Emphasis on non-dominated solutions

The step-by-step procedure of NSGA–II is illustrated and explained in figure 2.5. Elitism is preserved by keeping the previous parent population P_t for each generation and merging with the offspring population Q_t. Consequently, when the merged population is sorted into non-dominated fronts, the best solutions in the parent population will always be selected for the next parent population unless, in the unlikely event, all solutions in the offspring population are better.

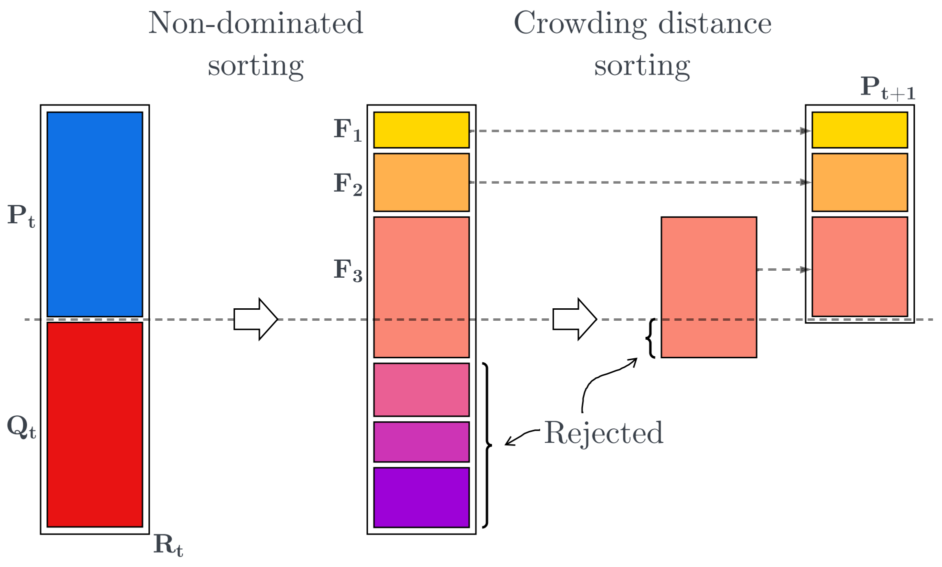
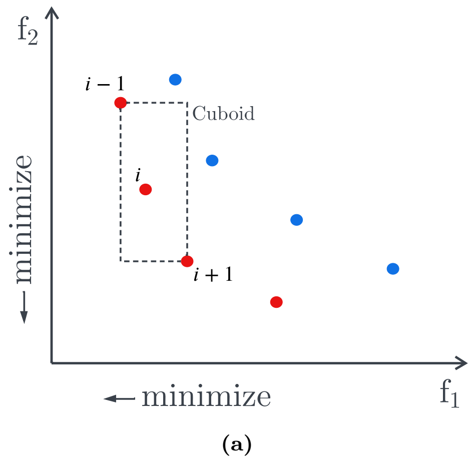


Figure 2.5: Illustration showing the NSGA–II step-by-step procedure. First, the parent population P_t and the offspring population Q_t are combined to form R_t. The population R_t is sorted into non-dominated fronts $\mathbf{F_i}$ (see fronts in figure 2.6b). Iteratively, in the order of their ranking, the solutions in each front are included in the new parent population P_{t+1} until the population limit is reached. The front for which the population limit would be reached if included is then sorted according to crowding-distance in descending order (see figure 2.6a). Finally, the new parent population P_{t+1} is then filled from this sorted front until the population limit is reached. The remaining solutions from this front and the remaining fronts are rejected. Crossover and mutation can now be applied to P_{t+1} to create a new offspring population Q_{t+1}. Figure adapted from [30].

Preserving a diverse set of solutions is ensured via the crowding distance sorting step of the algorithm. During this step, the *crowding-distance* (see figure 2.6a) is used to estimate the density of solutions surrounding a particular solution in the same non-dominated front. Once all solutions in the front have been assigned a crowding-distance, they are sorted in descending order and are in order included in the next parent population until the population limit is reached. The remaining solutions are discarded. Essentially, the algorithm favors solutions with higher density, i.e. in a less crowded region of solutions, in order to preserve the diversity of the solutions in the front.

Finally, the non-dominated sorting step emphasizes non-dominated solutions (see figure 2.6b). Solutions are sorted into non-dominated fronts with increasing rank. The order by which the fronts are included in the next parent population P_{t+1} ensures that the approximated Pareto-set is always included. The computational complexity of the sorting is $O(MN^2)$ and governs the overall computational complexity of NSGA–II. One of the largest improvements between NSGA–II and its predecessor NSGA is the improvement in computational complexity of the non-dominated sorting step.



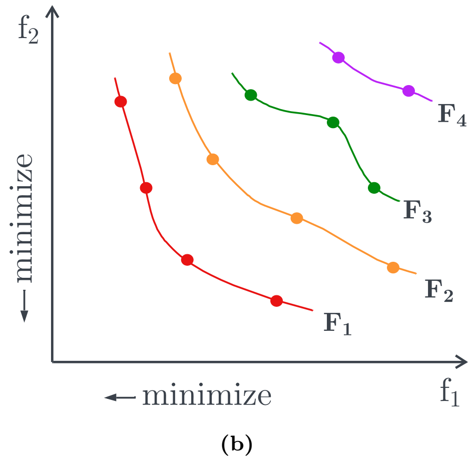


Figure 2.6: **(a)** Crowding-distance calculation of solutions (red) in the same non-dominated front. The crowding-distance for solution i is the average side length of the cuboid. The cuboid is formed using the nearest neighbors in the same front as vertices, i.e. $i-1$ and $i+1$. **(b)** Sorting of non-dominated fronts $\mathbf{F_i}$. The non-dominated sorting is performed iteratively by finding the current non-dominated front from the current set of solutions, assigning a rank and removing the front from the set. Repeat until the set of solutions is empty.

2.2.4 Multiobjectivization of single–objective problems

Mulitobjectivization [6, 31, 32] refers to the method of casting a single-objective optimization problem as a multi-objective one. The transformation to a multi-objective optimization problem is achieved by either adding additional objectives or decomposing the original objective function into multiple. In the case of decomposition, the evolutionary process can optimize the sub-tasks of a more complex problem simultaneously and thereby improve performance. Multiobjectivization with additional objectives has mostly been investigated by adding a diversity objective, with the purpose being to more explicitly balance exploitation and exploration [6]. The additional objectives are not the primary objective but are used to improve the search. Adding diversity objectives has demonstrated a substantial improvement in the efficiency and performance of evolutionary algorithms (see section 2.7).

2.3 Neural Networks

An *artificial neural network* (ANN), often simply called a neural network, is a machine learning method for learning representations and important features of data or a task domain. With the introduction of backpropagation, a gradient-based learning method (see section 2.3.2), even simple neural networks demonstrated promising results [33]. As computational resources and speeds increased, the neural architecture could be scaled up with millions of neurons and connections, i.e. deep learning and variants of deep learning such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Deep learning has revolutionized fields like computer vision, natural language processing, and speech recognition with impressive results in recent years [1].

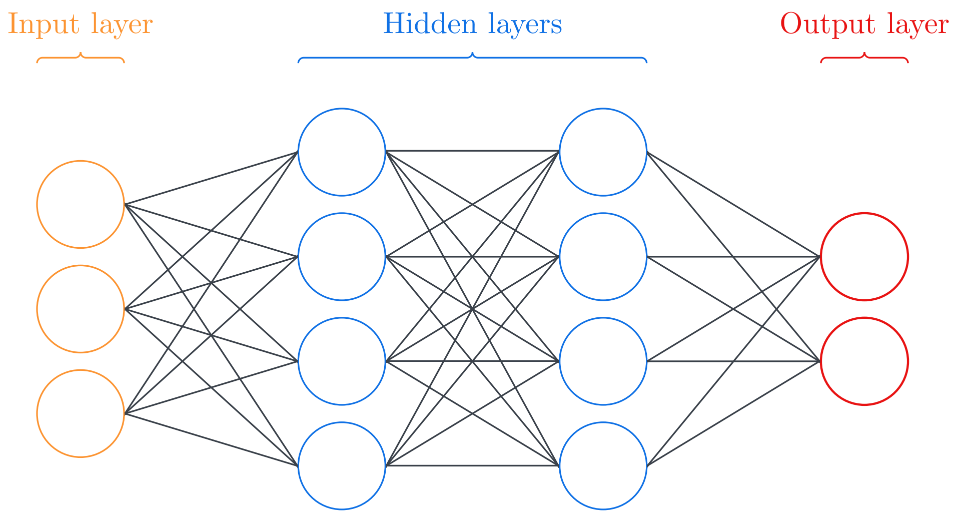


Figure 2.7: Example illustrating the architecture of a fully-connected neural network.

2.3.1 The architecture of a neural network

Inspired by biological neural networks in human and animal brains, neural networks are composed of *neurons* ordered in one or more *layers* (see figure 2.7). Each layer of neurons is connected with the previous layer and the next layer, and each *connection* is assigned a *weight*. The inputs x_{i-1} received by a neuron are the output from neurons in the previous layer, with the exception of the input layer which uses the actual data as input. The output a of a neuron is computed by multiplying the inputs with the weights w_i, adding bias b_i, and applying a non-linear activation function f on the resulting sum (see eq. 2.2). Commonly used activation functions are sigmoid, tanh, and ReLU.

$$a = f\left(b_i + \sum_{i=1}^{n} x_{i-1} w_i\right) \tag{2.2}$$

A *feed-forward* pass is performed by passing the data into the input layer and propagating it through the hidden layers until the output layer. The output of the network depends on the task being solved, e.g. outputs being class probabilities for a classification task or prediction for a regression task. Usually, an objective function (or loss function) is used to indicate how well the output matches the expected output given the input data. A neural network architecture of layered neurons can effectively approximate a function that maps the input data to the desired output. The *Universal Approximation Theorem* states that a neural network can approximate any arbitrarily complex function provided there are a sufficient number of hidden neurons [34].

2.3.2 Learning with backpropagation

Backpropagation [35] is an efficient algorithm for computing the gradients in a neural network, usually combined with an optimization algorithm such as *stochastic gradient descent* for optimizing the weights and biases of the network using gradients. This type of gradient-based learning technique is widely utilized for training neural networks, and is to date, the most efficient method for optimizing most neural networks [36]. After a feed-forward pass through the network, the backpropagation algorithm iterates backward through the layers starting with the output layer and computing the gradient of the loss (or objective) function with respect to the *weights* via the *chain rule*.

The backward pass through the layers is necessary because the error from the next layer is needed to compute the error for the neurons in the current layer. The magnitude and sign of the error determine how much the weights and biases for the neuron are changed in order to reduce the error of the output. In other words, backpropagation is an algorithm for propagating the error back through the network and using the computed gradients to optimize the weights and biases of the neurons in the network to reduce this error.

2.3.3 Limitations of gradient-based learning

One of the limitations of *gradient-based learning* is that the objective function is required to be differentiable. For a lot of real-world problems, a differentiable objective function does not exist. Propagating the error of the objective function backward through the network is not possible with a non-differentiable objective function. There are ways to avoid this limitation by for example differentiating subgradients using automatic differentiation [37]. Nevertheless, there is an interest in alternative optimization methods that can optimize non-differentiable objective functions and in general solve problems where gradient-based learning fails, such as reinforcement learning [38] and neuroevolution [3] (see section 2.4).

Moreover, gradient-based learning continually optimizes a single solution with a fixed network structure and can not easily explore multiple solutions with different structures. The structure of the neural network can greatly impact performance. Manually designing a neural network architecture is challenging and usually requires a domain-dependent architecture. A wider and denser network may be better suited for a particular task than a narrow and sparse network and vice versa. In some cases, the difference in performance may be insignificant but there is no way of knowing this beforehand as there is no systematic way to determine a suitable architecture. Finding a suitable neural network architecture will be time-consuming as it requires training each architecture to compare them [3].

2.4 Neuroevolution

Neuroevolution (NE) is a machine learning method that utilizes evolutionary algorithms for evolving neural networks. Neuroevolution enables the capability of optimizing the hyperparameters, activation function of neurons, and architecture of a neural network. The last two are typically unavailable to gradient-based approaches. Consequently, the use of population-based evolutionary algorithms for optimizing the architecture has removed the difficult challenge of manually designing the neural network architecture and the human bias that would be introduced with manual design. Although, there are cases where such human knowledge is beneficial (e.g. ConvNets for computer vision [1]). Early successful applications of NE were in the field of *evolutionary robotics*, which is concerned with evolving neural network controllers for robots. With the increasing availability of computing resources, more recent successful applications of NE include language processing, image generation, soft robotics, multi-task learning, and playing games [3, 39, 40].

The general procedure of neuroevolution is illustrated in figure 2.8. For each generation, every genotype in the population is transformed into a neural network (i.e. phenotype). All or a number of observations (e.g. sensor data) from the environment are used as input for the neural network. The output of the neural network affects the environment and causes the observations to change. How the outputs affect the environment is domain-dependent. After an attempt at the task or problem, the neural network is evaluated based on the performance and/or other metrics. Once all individuals in the population have been assigned a fitness evaluation, a selection method selects which individuals will be used to generate offspring for the next generation using evolutionary operators. The cycle is repeated until a termination condition is met.

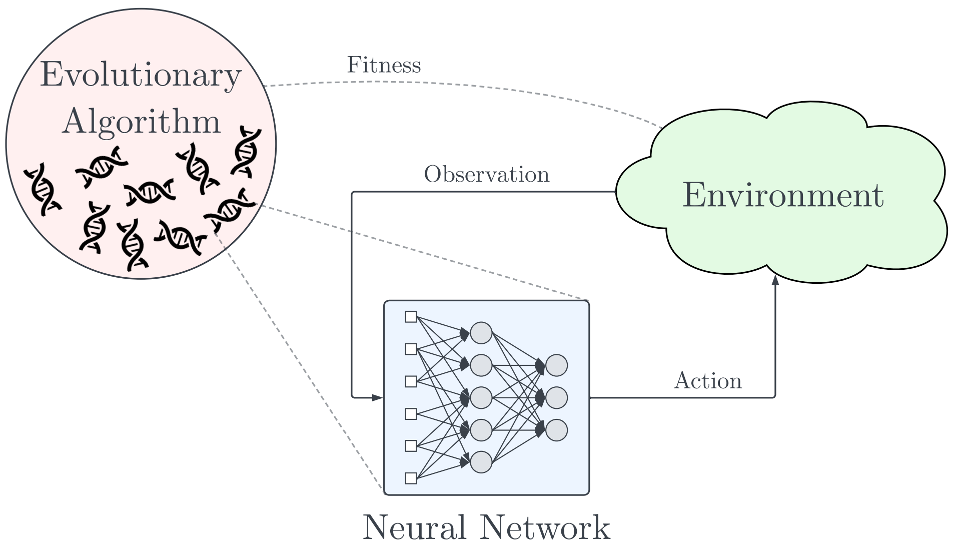


Figure 2.8: Illustration of neuroevolution. Each genotype in the population is transformed into a neural network phenotype. The neural network receives input from the environment and propagates it through the network to compute an output. The output causes a change in the environment which will update the observations. Fitness is assigned to the neural network according to its performance at the end of the evaluation. Selected individuals are used to generate the next population. Repeat until a termination condition is met. Figure adapted from [41].

2.4.1 NEAT

A successful and popular implementation of neuroevolution is the *NeuroEvolution of Augmenting Topologies* (NEAT) algorithm [42]. Unlike previous NE methods where only the weights and other hyperparameters are optimized, the NEAT algorithm also optimizes the neural network topology using crossover and mutation. The goal of NEAT was to present a solution to the following challenges:

- Finding a genetic representation that allows for meaningful crossover of different network structures without requiring complex structure analysis.
- Prevent individuals with structural innovation from being prematurely removed from the population.
- Minimizing the neural network structure throughout the evolutionary process.

Crossover between two neural networks in a meaningful way, or in other words a way that preserves the complex structure, is not a trivial task. The issue of crossover between two different neural networks was solved using a historical marking called an *innovation number*. The genotype is represented by a list of node genes and connection genes. Each connection gene contains information about the id of the out-node and in-node, connection weight, disabled/enabled bit, and an innovation number. Whenever a new gene appears through structural mutation, the gene is assigned a unique innovation number. The innovation number is used to match up the genes between two individuals to allow crossover.

To protect structural innovation, NEAT implements speciation (or niching) through the use of fitness sharing. With fitness sharing, individuals are divided into niches and share the fitness of their niche, preventing one niche from dominating the whole population. Individuals are assigned a niche using a compatibility distance δ. The pair-wise compatibility distance is measured as a linear combination of the number of disjoint genes (D), the number of excess genes (E), and the average weight difference of matching genes ($\overline{W}$).

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W} \tag{2.3}$$

The coefficients c_1, c_2 and c_3 can be tuned to adjust the importance of the three factors. N is the number of genes in the larger genome of the two. Important to note is that structural innovation through speciation does not guarantee different behavior considering there exist infinitely many different neural network structures that can be functionally identical according to the Universal Approximation Theorem [34].

Finally, the NEAT algorithm minimizes the neural network structure by initializing a uniform population of networks with no hidden nodes as the starting population. The networks will grow incrementally with new structures as mutation is applied throughout the evolutionary process. The structural innovation is protected by speciation, and the structures found to be useful will survive the selection phase. By starting with the most minimal structure and incrementally growing it, the number of hyperparameters required to optimize is reduced.

2.5 Neural Network Representation

One of the main reasons for the success of large neural networks, i.e. deep learning, is the ability to learn a good representation of data or a domain. Yet, understanding and characterizing the representations learned by neural networks remains an important challenge. Recent work on understanding neural network representations includes understanding if different neural networks learn the same representations [43, 44], whether or not the learned representation varies with the width and depth of the neural network [45], and how to measure the similarity between two neural network representations [46].

Of particular interest, with neuroevolution in mind, is measuring the similarity between learned representations. How to measure the similarity between representations is unclear, and most importantly, if it is even possible to extract any useful information from the intermediate layers of a neural network. One approach is to use the activations from the neurons in the intermediate layers of two networks and apply a similarity metric such as the *centered kernel alignment* (CKA) or *canonical correlation analysis* (CCA) [46]. These metrics are described in more detail below in sections 2.5.1 and 2.5.2 respectively.

2.5.1 Similarity between representations

Centered kernel alignment (CKA) [46] is a similarity index for measuring the similarity between the learned representation within and across neural networks. The representation of neural networks is expressed by representational similarity matrices. The similarity between two neural networks is calculated as follows with the problem statement:

Let $X \in \mathbb{R}^{n \times p_1}$ be the matrix of activations of p_1 neurons for n examples, and $Y \in \mathbb{R}^{n \times p_2}$ the matrix of activations of p_2 neurons for the same n examples. Without loss of generality, X and Y are assumed to be centered and that $p_1 \leq p_2$.

Dot Product-Based Similarity. The following equation relates the dot product between examples to the dot product between features (i.e. neurons):

$$\langle \mathrm{vec}(X, X^T), \mathrm{vec}(Y, Y^T) \rangle = \mathrm{tr}(XX^TYY^T) = \|Y^TX\|_{\mathrm{F}}^2 \tag{2.4}$$

The left-hand side of eq. 2.4 measures the similarity across the inter-example structure elements of the dot products XX^T and YY^T. The right-hand side calculates the exact same similarity by summing the squared dot products between every pair of *features* of X and Y, The advantage of this equality is that if n is larger than p_i, the similarity can be calculated faster across the features instead of the examples. Assuming X and Y are centered, it is implied that:

$$\frac{1}{(n-1)^2}\mathrm{tr}(XX^TYY^T) = \|\mathrm{cov}(X^T, Y^T)\|_{\mathrm{F}}^2 \tag{2.5}$$

Hilbert-Schmidt Independence Criterion. The Hilbert-Schmidt Independence Criterion (HSIC) generalizes eq. 2.4 and 2.5 to inner products from reproducing kernel Hilbert spaces. See [46, 47] for more details. HSIC is formulated as follows:

$$\mathrm{HSIC}(K, L) = \frac{1}{(n-1)^2}\mathrm{tr}(KHLH) \tag{2.6}$$

where $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ and $L_{ij} = l(\mathbf{y}_i, \mathbf{y}_j)$. k and l are kernels. With linear kernels no projection occurs, resulting in the inner product $k(\mathbf{x}, \mathbf{y}) = l(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$. Thus, HSIC yields eq. 2.5. H is the centering matrix $H_n = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^T$.

The original purpose of HSIC was to determine the statistical independence of two sets of variables but has since been used for various machine learning problems such as feature selection, clustering, dimensionality reduction, and kernel optimization [47].

Centered Kernel Alignment. Normalizing HSIC results in the centered kernel alignment which produces a similarity index from 0 to 1 as follows:

$$\mathrm{CKA}(K, L) = \frac{\mathrm{HSIC}(K, L)}{\sqrt{\mathrm{HSIC}(K, K)\mathrm{HSIC}(L, L)}} \tag{2.7}$$

2.5.2 Correlation between representations

Canonical Correlation Analysis (CCA) can be used to find bases (i.e. linear combinations) of the matrices X and Y, such that the correlation between them is maximized when the original matrices are projected onto these bases. CKA with linear kernels is closely related to CCA [46], and can likewise be used to measure the similarity between neural network representations.

Using the same problem statement as in section 2.5.1 with matrices of activations from neurons, the ith canonical correlation coefficient ρ_i for $1 \leq i \leq p_1$ is given by:

$$\begin{aligned} \rho_i = \max_{\mathbf{w}_X^i, \mathbf{w}_Y^i} & \ \mathrm{corr}(X\mathbf{w}_X^i, Y\mathbf{w}_Y^i) \\ \text{subject to } & \ \forall_{j<i}\ X\mathbf{w}_X^i \perp X\mathbf{w}_X^i \\ & \ \forall_{j<i}\ Y\mathbf{w}_Y^i \perp Y\mathbf{w}_Y^i \end{aligned} \tag{2.8}$$

where the vectors $\mathbf{w}_X^i \in \mathbb{R}^{p_1}$ and $\mathbf{w}_Y^i \in \mathbb{R}^{p_2}$ are the canonical weights that maximize ρ_i. The constraints in eq. 2.8 enforce orthogonality of the canonical variables. Finally, the mean squared CCA correlation R^2_{CCA} is used to calculate the goodness of fit of CCA:

$$R^2_{\mathrm{CCA}} = \frac{\sum_{i=1}^{p_1} \rho_i^2}{p_1} = \frac{\|Q_Y^T Q_X\|_{\mathrm{F}}^2}{p_1} \tag{2.9}$$

where $Q_X = X(X^TX)^{-1/2}$ and $Q_Y = Y(Y^TY)^{-1/2}$ represent the orthonormal bases for the columns of X and Y.

2.6 Encouraging diversity in Neuroevolution

2.6.1 Why encourage diversity?

One of the most remarkable feats of biological evolution is the ability to produce a diversity of complex organisms that are all high-performing in their niche. The biological diversity seen in nature today is the result of an evolutionary process that is over millions of years in the making. Diversity is a topic of much contention in the field of evolutionary biology, including how to measure diversity and whether to view diversity as a cause or as an effect of evolution. One group of researchers argues that diversity adds redundancy and robustness to the evolutionary process to create more stable and productive ecosystems. Contrary to the view of diversity as an informative signal, i.e. an effect, that describes the evolutionary forces that caused it [48, 49].

Regardless of the role diversity plays in evolution, it is an important aspect of evolution that requires further study in evolutionary computing. From the traditional point of view, evolution in evolutionary computing is primarily viewed as an optimizer. The problem with viewing evolution as an optimizer is how biological evolution does not strive toward any particular organism. The ability of evolution to produce diversity and complexity raises the question of whether a shift in perspective is required in the approach to evolutionary computing.

In recent years, research on neuroevolution (and evolutionary computing in general) has focused more and more on *diversity* [3]. Population-based evolutionary algorithms should, in theory, create diversity by themselves through evolutionary operators. In practice, they often converge too early and lack the diversity needed to avoid local optima. Explicitly encouraging diversity will drive exploration and avoid early convergence to local optima and can also be utilized to create a collection of diverse yet high-performing solutions rather than a single solution. Successful diversity-driven methods include multiobjectivization with diversity objectives, quality-diversity optimization, and open-endedness. Despite these successful methods, a considerable gap remains between the type of complexity nature evolves and the type of complexity evolutionary algorithms discovers.

2.6.2 Effect of diversity in Spaces

The effect of encouraging diversity will depend on the space in which one tries to encourage diversity. For instance, encouraging diversity in the *genotype space* with *genetic diversity* objectives [8, 9], will encourage genetically different individuals. In the *phenotype space*, *structural diversity* objectives [4, 9, 10] can be used to encourage structurally different individuals. By contrast, a more domain-dependent approach is to encourage different behaviors in the *behavior space* with *behavioral diversity* objectives [6, 7, 50]. As illustrated in figure 2.9, diversity in the aforementioned spaces does not guarantee different behaviors because of the non-injective mapping between them. Two genetically different individuals may result in an identical neural network and structurally different neural networks may produce the same behavior.

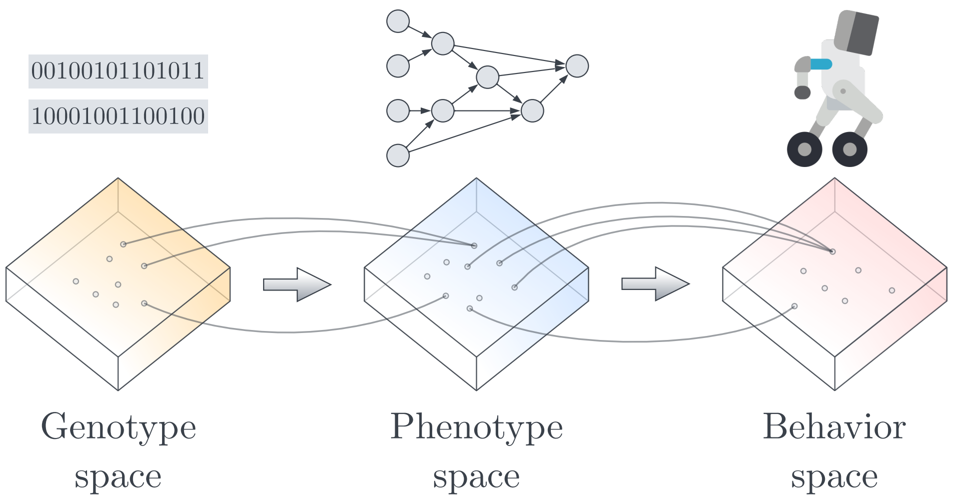
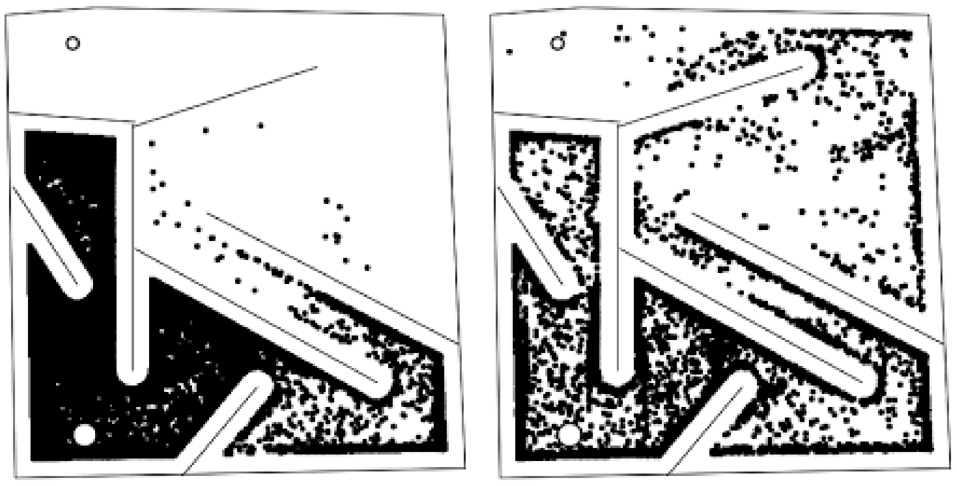


Figure 2.9: Illustration of the mapping between genotype-, phenotype-, and behavior space. The genotype space contains the genetic representations of candidate solutions. Evolutionary operators such as crossover and mutation are used to manipulate the genotypes. Genotypes are transformed into phenotypes (e.g. neural networks) in the phenotype space. The genotype-to-phenotype mapping is non-injective, meaning multiple genotypes can correspond to the same phenotype. The phenotypes are simulated during the task and assigned a fitness based on their behavior in the behavior space. The phenotype-to-behavior mapping is also non-injective considering two different neural networks can produce identical behavior. Figure adapted from [7].

2.6.3 Novelty Search

Novelty search [5] successfully illustrates the effect diversity has on the evolutionary search. Novelty search differs from the traditional approach in that the performance on the task is completely disregarded. Instead, the idea is to search for behavioral novelty alone. Compared to the traditional approach where novel individuals are discovered indirectly, novelty search is used to explicitly search for novel individuals. Counterintuitively, novelty search has been shown to outperform performance-based search on multiple tasks, especially *deceptive* tasks where there is an obvious local optimum [5, 6, 21]. Consequently, on deceptive tasks, the search for increased performance on a task may ultimately become an impediment. A comparison between performance-based search and novelty search is illustrated in figure 2.10.



(a) Performance-based search. **(b)** Novelty search.

Figure 2.10: The hard maze used in the maze experiment to introduce novelty search and show how objective functions can become misleading. The goal is for the robot in the lower left corner to reach the goal in the top left corner. The maze contains an attractive dead-end in the direction of the goal. Each black point represents the end location of a robot evaluated during the run. Novelty search finds a solution that is able to reach the goal, while performance-based search does not because of the deceptiveness of the task. Figures from [5].

The novelty metric used to measure the *uniqueness* of an individual's behavior should reward individuals diverging from behaviors seen before. Therefore, an archive is kept of past individuals with novel behavior at the time. The idea is to measure the distance from a new individual to the population and the archive in *behavior space*. For this purpose, the *sparseness* ρ at any point in the behavior space is used as a metric. New individuals at dense clusters in the behavior space are rewarded less because these clusters are areas with solutions already seen and therefore less novel. The sparseness ρ at point x is defined as

$$\rho(x) = \frac{1}{k}\sum_{i=0}^{k} dist(x, \mu_i) \tag{2.10}$$

where μ_i is the ith-nearest neighbor of x of the total k-nearest neighbors. The distance metric *dist* is a domain-dependent measure of the behavioral difference between two individuals, which in this case is the euclidean distance between their end locations.

2.6.4 Quality-Diversity Optimization

The success of novelty search has inspired other research focused on *Quality-Diversity* optimization algorithms. The purpose of Quality-Diversity algorithms is to generate a diverse collection of high-performing solutions. As with novelty search, Quality-Diversity is usually applied to the behavior space and attempts to generate solutions to fill the whole space even if they are not the absolute highest-performing solutions. Notable Quality-Diversity algorithms include *Novelty Search with Local Competition* (NSLC) [51] and *Multi-dimensional Archive of Phenotypic Elites* (MAP-Elites) [52].

2.7 Diversity Objectives in Neurevolution

This section gives an overview of the various diversity objectives used in previous work on Neuroevolution. The subsections are dedicated to the following diversity categories: (1) Behavioral diversity, (2) Structural diversity, and (3) Representational diversity. Unless specified otherwise, referring to the use or results of a diversity objective/distance implies that it was used in addition to performance with a multi-objective evolutionary algorithm, i.e. multiobjectivization (see section 2.2.4). Generally, the purpose of using a diversity objective is to search for individuals who are different from others. In more technical terms, this means searching for individuals that have a large distance from others according to a distance metric of the chosen objective.

2.7.1 Behavioral Diversity

A problem with measuring diversity in the genetic space or the phenotype space is the non-injective mapping between spaces illustrated in figure 2.9. As a result, the genotype and the phenotype will in many cases be a poor predictor of how the individual will perform and behave in the environment. The novelty search results demonstrate how effective measuring diversity in the behavior space can be, especially in deceptive environments. Encouraging behavioral diversity alone or in addition to performance has been successfully applied to a variety of tasks [5–7, 50].

Defining the behavior of an individual

For the purpose of generality, the behavior of an individual is often characterized by a *behavior vector* β. Furthermore, a behavioral distance measures the distance between individuals using this behavior vector. The contents of the behavior vector will depend on the environment domain and which behavioral distance is used. Behavioral distances are usually divided into two categories: (1) domain-dependent *ad hoc* distances, and (2) domain-independent generic distances. Behavior vectors are used to describe the behavior of individuals for both types of distances.

Ad hoc behavioral distance

An essential challenge of behavioral diversity is how to define the distance between two individuals in the behavior space. In general, the behavioral distance between two individuals is domain-dependent (*ad hoc*), meaning the distance is specific to the environment. In the maze experiment, a natural distance metric is the distance between the end location of two individuals or alternatively measuring the difference in the trajectory of the two individuals.

Formally, the *ad hoc* behavioral distance between two individuals, x and y, is defined as follows:

$$d_{\text{ad-hoc}}(x, y) = dist(\beta_x, \beta_y) \tag{2.11}$$

where β_x is the behavior vector of individual x and $dist$ is the metric used to compute the behavioral distance between two individuals. The behavior vector and distance metric are domain-dependent.

Defining the behavioral distance becomes increasingly harder when applied to more complex real-world problems. As with designing fitness functions, defining the behavioral distance usually requires human knowledge and can introduce human bias. While introducing human bias might be beneficial in some cases, normally the goal of machine learning is to limit the amount of human knowledge required.

Generic behavioral distance

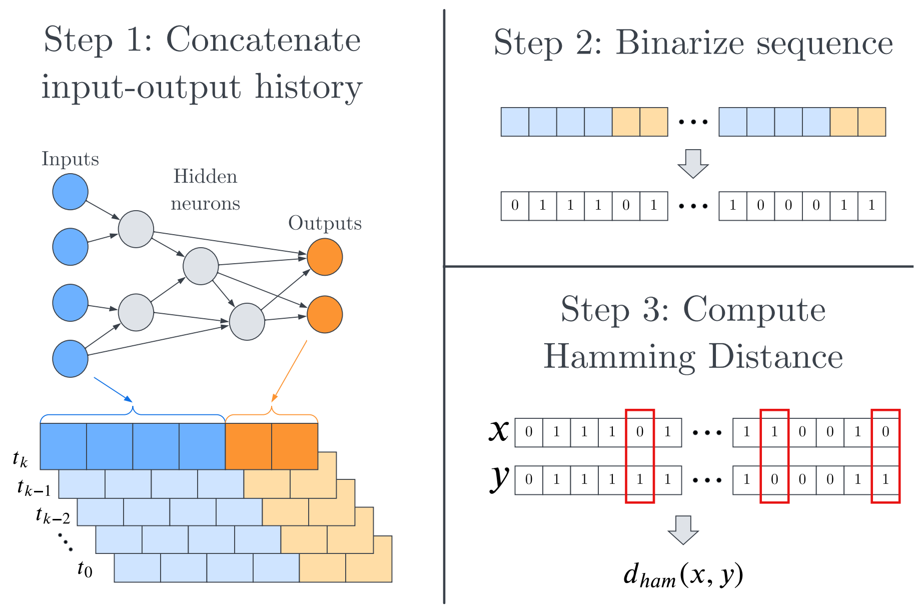
The alternative to *ad hoc* behavioral distance is a *generic* behavioral distance. With generic behavioral distances, the behavior vector and behavioral distance can be applied directly across domains with minimal adjustment regardless of how different the domains are. The generality of such behavioral distances removes the burden of designing a comprehensive behavioral distance specific to every domain. However, this generality is potentially less informative and can lead to a performance trade-off.

One study [53] compared the following generic behavioral distances on a grid-based optimization problem: (1) *Fitness difference* – the absolute difference in fitness between individuals x and y, (2) *Relative Entropy* – the expected number of extra bits required to encode β_x using the optimal code for β_y (Kullback-Leibler divergence), (3) *Normalized Compression Distance* (NCD) – the improvement in compressing β_y derived from using β_x as a compressed database, and (4) *Hamming distance* – the number of non-matching bits between binarized behavior vectors β_x and β_y. The last three distances use a behavior vector with the inputs and outputs of the neural network to define an individual's behavior (see step 1 in figure 2.11).

Of the distances analyzed, the three behavior-based distances (NCD, Hamming distance, Relative Entropy) outperformed the rest. NCD outperformed all of them because it is less sensitive to misaligned sequences and sequences of different lengths, but has a significantly higher computational cost. The Hamming distance was competitive with NCD but with a much lower computational cost. The *Hamming distance* is described in more detail below.

Hamming distance

The Hamming distance measures the behavioral distance between two individuals by counting the number of non-matching bits between their respective binary sequences. The sequence of an individual is the input and corresponding output history of the neural network for T steps concatenated together (see figure 2.11).



Step 1: Concatenate input-output history
Inputs
Hidden neurons
Outputs
t_k
t_{k-1}
t_{k-2}
t_0
Step 2: Binarize sequence
0 1 1 1 0 1 ··· 1 0 0 0 1 1
Step 3: Compute Hamming Distance
x 0 1 1 1 0 1 ··· 1 1 0 0 1 0
y 0 1 1 1 1 1 ··· 1 0 0 0 1 1
$d_{ham}(x, y)$

Figure 2.11: Illustration of the step-by-step procedure for computing the Hamming distance. Step 1 involves concatenating the input-output history of the neural network of an individual during the simulation. In step 2, the sequence of input and output values is binarized. Finally, step 3 computes the Hamming distance between the binarized sequence of two individuals x and y by counting the number of non-matching bits (see eq. 2.14). Figure for step 1 adapted from [54].

The sequence vector of individual i can be defined as follows:

$$\beta_i = \Big[\{\mathbf{i}(t), \mathbf{o}(t)\}, t \in [0, T]\Big] \tag{2.12}$$

where $\mathbf{i}(t)$ is a vector of size n_i with the inputs of the neural network at time or step t, and $\mathbf{o}(t)$ is a vector of size n_o with the outputs of the neural network at time or step t. T is the total observation length. Without loss of generality, the values of β are assumed to be in the range $[0, 1]$. The sequence vector is binarized as follows:

$$\beta_{i,\text{bin}}(t) = \begin{cases} 1 & \text{if } \beta_i(t) > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

When the binary sequences of two individuals are of different lengths, only the bits for the smallest of the two sequences are counted. Finally, the Hamming distance between two individuals is computed by counting the number of non-matching bits as follows:

$$d_{\text{ham}}(x, y) = \sum_{k=1}^{\min(|\beta_x|,|\beta_y|)} 1 - \delta(\beta_{x,\text{bin}}(k), \beta_{y,\text{bin}}(k)) \tag{2.14}$$

where the Kronecker delta $\delta(i, j)$ is defined as:

$$\delta(i, j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

Comparing ad hoc and generic distances

Another study [55] extended the work of [53] by comparing generic behavioral distances to *ad hoc* behavioral distances on an evolutionary robotics task with a continuous environment space, in contrast to the discrete environment used in [53]. The task was for a mobile robot to collect multiple balls and return them to a basket. The following behavioral distances were analyzed:

- *Task-specific* (*ad hoc*): the euclidean distance between behavior vectors describing the number of times the robot was in a user-defined state;
- *Trajectory* (*ad hoc*): the difference between the trajectory of two individuals during simulation;
- *Fourier coefficients*: the euclidean distance between the first coefficients of a discrete Fourier transform of the behavior vector with the input-output history of the individuals;
- *Hamming distance.*

Out of all the behavioral distances studied, the Hamming distance led to the best performance with the *ad hoc* distances being the next best. However, it is emphasized in the study that most of the input values are already binarized, which might skew the results in favor of Hamming distance as opposed to the *ad hoc* distances. Nevertheless, the experiments in which no behavioral diversity was encouraged resulted in a significantly worse performance [55].

An empirical study [7] compared the traditional diversity technique *fitness sharing*, *NEAT*, and *multiobjectivization* with a behavioral diversity objective (*ad hoc* or Hamming) on three evolutionary robotics tasks. NEAT implements fitness sharing but divides individuals into niches by using the compatibility distance in the genotype space instead of performance. In addition, experiments with no diversity encouragement were used as control experiments.

The study concluded that multiobjectivization outperformed all other treatments, including single-objective fitness sharing, NEAT, and with no diversity. In particular, *ad hoc* and Hamming behavioral distance dominated the rest in all experiments. The generic Hamming distance demonstrated the ability to be as efficient as *ad hoc* behavioral distances [7].

Ad hoc vs generic

The preceding sections have highlighted the results of studies comparing various behavioral distances, both *ad hoc* behavioral distances and generic behavioral distances. In general, the studies showed that *ad hoc* led to the best performance, with Hamming distance as a close second. In some experiments, Hamming distance was as efficient as *ad hoc* distances. As expected, *ad hoc* distances usually outperform the rest due to being specifically designed for the domains in which they measure behavior. Surprisingly, Hamming distance is competitive with *ad hoc* distances in many of the studied experiments, including the mobile robot tasks with a continuous environment.

One could assume that *ad hoc* distances should be significantly more informative than Hamming distance, especially in continuous environments where binarizing continuous sensor values may abstract valuable information away. There are results to suggest this to some degree in the maze experiment studied in [7]. Hamming distance may not be able to differentiate between individuals in the dead-end from those outside the dead-end in the hard maze (see figure 2.10b). However, Hamming distance outperformed *ad hoc* distance on the ball-collecting task studied in [7, 55]. One hypothesis, as mentioned earlier, could be that Hamming distance benefits from input and outputs that are binarized or at least somewhat discrete instead of continuous values. Hamming distance also performed well on the more discrete grid-based optimization problem studied in [53], but was not compared to an *ad hoc* behavioral distance. Nevertheless, Hamming distance as a behavioral diversity objective merits further study to understand what type of problems it is most suited for.

2.7.2 Structural Diversity

As described earlier, there are limitations to using *ad hoc* behavioral diversity objectives. While the Hamming distance is a reasonable alternative, the generic characteristic of the metric has a trade-off by not being as informative on certain problems as *ad hoc* distances.

A domain-independent alternative is to encourage *structural diversity.* Guiding neuroevolution with structural diversity objectives will encourage structurally different individuals. Accordingly, a distance metric is required to measure the difference or similarity between two neural networks. From a more abstract perspective, a neural network is essentially a directed graph of nodes and edges. However, computing the distance between two graphs is an NP-hard problem, making a complete one-to-one connectivity distance metric not feasible due to the algorithmic complexity involved [50].

One solution is to use *approximate* structural distance metrics. Approximate structural distance metrics are less computationally expensive at the cost of accuracy. Such approximate distances include *edit distance*, *resistance distance*, *spectral distance* [56], and *graph probing* [57] to name a few. Yet, in a comparison with behavioral diversity objectives, using graph probing as a diversity objective did not improve performance [50].

Recent research suggests benefits to encouraging certain structural properties inspired by biology instead. Inspired by the adaptability, flexibility, and robustness of natural brains, it is believed that the structure of a neural network should express traits such as *modularity*, *regularity*, and *hierarchy* in order to exhibit the same behavioral complexity as humans and animals [58]. As a result, encouraging *modularity* [9, 10] in the neural network structure has been shown to improve the performance. In the same vein as modularity but at a higher level of abstraction, encouraging *modularity diversity* [4] has also demonstrated increased performance. Still, some questions remain regarding the effectiveness of structural diversity objectives. Considering the non-injective phenotype-to-behavior mapping (figure 2.9) and the universal approximation theorem (section 2.3.1), there is no guarantee that structurally different individuals will behave differently.

Modularity

Modularity [59] is a measure of the strength of division of a network into multiple communities (or modules, clusters, groups). A community is a subset of nodes of the network that is connected by a higher number of edges between them than is *statistically expected* on the basis of chance. In other words, modularity indicates how well a network can be divided into communities or modules. Generally, modularity can be defined as follows:

$$\begin{aligned} Q = &\text{ (fraction of edges within communities)} \\ &- \text{(expected fraction of such edges)} \end{aligned} \tag{2.16}$$

The expected number of edges between nodes can be approximated using the probability of an edge existing between nodes i and j of a large random network under certain assumptions (see [59] for more details). The modularity Q of a network that can be partitioned into c communities is defined as follows:

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j) \tag{2.17}$$

where A_{ij} is the adjacency matrix of the neural network, δ is the Kronecker delta, k_i is the degree of vertex i, c_i is the label of the community to which vertex i is assigned, and m is the total number of edges in the network. In general, optimizing for modularity is an NP-complete problem. In practice, a number of approximate optimization methods such as greedy algorithms and spectral methods are used to reduce computational complexity [59].

Modularity Diversity

The premise of modularity diversity [4] is that the interesting and significant differences between neural networks do not lie in the exact connectivity, but in their inherent higher-level modular structures. Modularity diversity calculates the distance between the modular decompositions of two neural networks. The calculation is limited to either the input or output neurons depending on where it is most relevant to measure the modular decomposition for the targeted problem see figure 2.12.

The calculation of modularity diversity involves two steps. First, the modular decomposition of the networks is estimated. Finally, how well the decompositions match is calculated as follows:

$$\Delta_{decomp} = 1 - \frac{\mathit{Uniformity}(M_{evo}, M_{comp}) + \mathit{Conflicts}(M_{evo}, M_{comp})}{2.0} \tag{2.18}$$

where M_{evo} and M_{comp} are the modular decomposition of the two individuals. *Uniformity* calculates the degree to which the modules in M_{comp} match a module of corresponding nodes in M_{evo}. *Conflicts* calculates the number of conflicts between M_{evo} and M_{comp} by counting the neurons in the modules of M_{evo} that belong to a different module in M_{comp}. Δ_{decomp} ranges from 0 to 1, where 0 indicates that the decompositions match perfectly and the worst possible match between the decompositions will result in a score of 1. See [4] for pseudocode on how to calculate the uniformity and conflicts between the two decompositions.

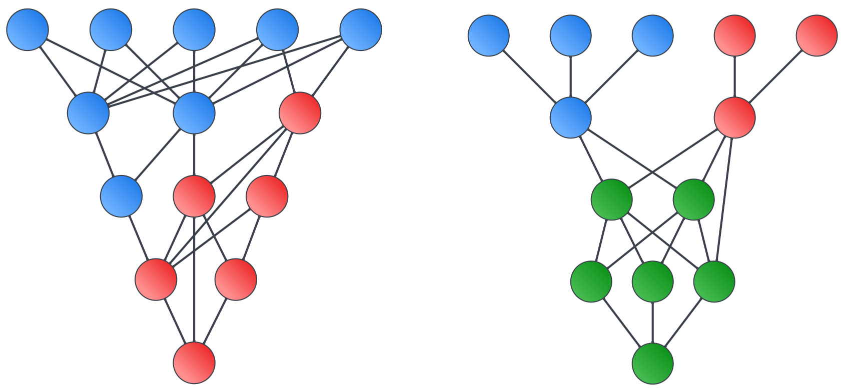


Figure 2.12: The modular decomposition of two neural networks used to calculate modularity diversity. The color of each node indicates which module it belongs to. In this example, calculating the modularity diversity for the input neurons is most informative as there is only one output node. The modular decomposition of **(a)** will have several conflicts with the modular decomposition of **(b)** because multiple corresponding input nodes belong to different modules. Figure adapted from [4].

2.7.3 Representational Diversity

As described in sections 2.7.1 and 2.7.2, there are advantages and limitations to using behavioral diversity and structural diversity respectively. With behavioral diversity, there is the challenge of defining an appropriate characterization of the behavior of individuals and how to measure the distance between behaviors. In addition, encouraging novel behavior during the evolutionary process can get stuck in so-called "novelty plateaus", wherein mutation and crossover of individuals do not immediately produce novel behaviors [54]. For example, a sequence of specific mutations may be required to produce novel behaviors. With structural diversity, there is the challenge of how to avoid structurally different individuals that produce identical behaviors. Furthermore, a lot of existing distance metrics for measuring the distance between neural networks are computationally expensive. There is a lack of research into attempts to combine the advantages of both types of diversity.

Combining the two in an attempt to measure how differently two neural networks "*think*" (used metaphorically) or how different their *learned representations* are, can perhaps be used to encompass both the structure and behavior of a neural network. Using both the generality of structural diversity and the ability to discover novel behaviors of behavioral diversity in a new type of diversity objective could be a new direction of research into diversity in neuroevolution. In this work, a diversity objective of this variety is dubbed

Representational diversity. Recalling figure 2.9 illustrating the various spaces in which one can encourage diversity, a representational diversity objective would thus encourage diversity in the *representation space*. This concept is illustrated in figure 2.13, where the learned representation of the neural network depends on the phenotype and its behavior.

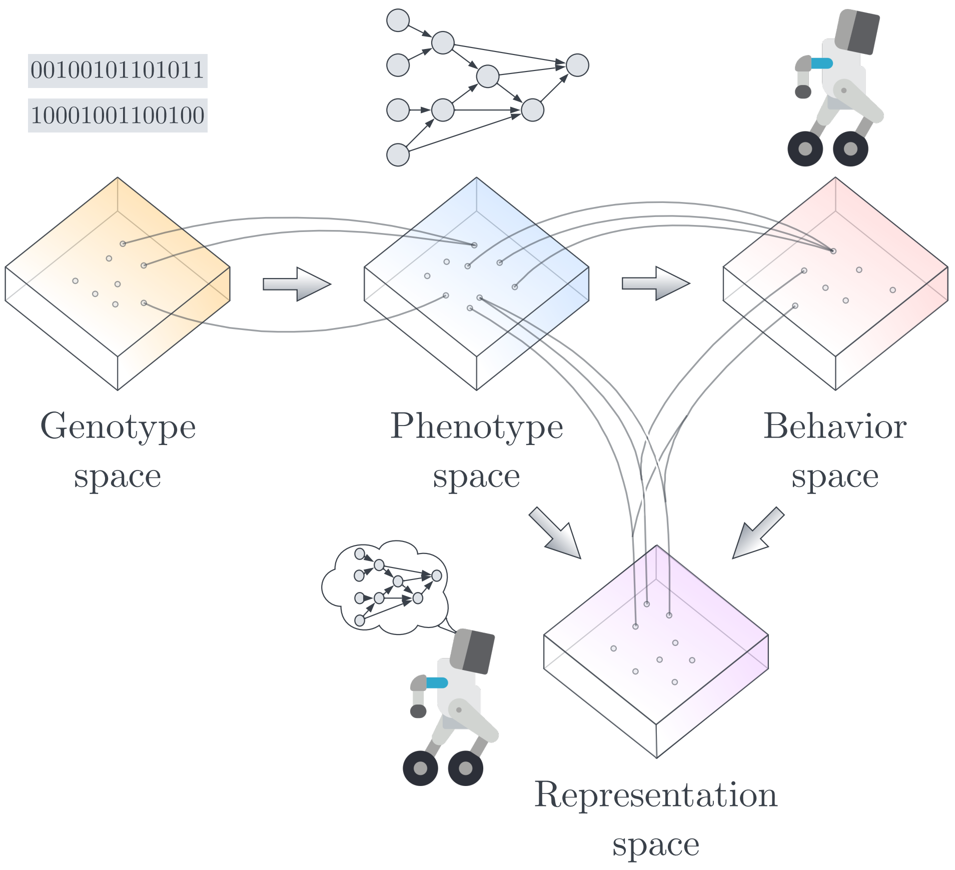


Figure 2.13: Illustration of representation space by extending figure 2.9. Learned representations in the representation space are mapped from a combination of the phenotype and its behavior during simulation. The phenotype-behavior-to-representation mapping is non-injective because two structurally different neural networks can produce identical behavior and thus learn the same representation.

The challenge with representational diversity then becomes how to define the representation of a neural network and how to calculate the distance between two representations. One study [54] introduced a new method called the *Creative Thinking Approach* (CTA), wherein evolved neural networks are encouraged to "think differently". In more technical terms, the study defines "creative thinking" as novel patterns of neural firing. Meaning neural networks with novel activation outputs are rewarded more.

CTA is an extension of how a behavior vector of the binarized input-output history of a neural network is used with the Hamming distance to calculate the behavioral distance between individuals. The difference is that CTA concatenates the activations from the neurons in the hidden layers for every step t in addition to the inputs and outputs (see figure 2.14). The study compared CTA to behavioral diversity with Hamming distance on the maze experiment and a ball-collecting problem. Using CTA was not more efficient than behavioral diversity on the maze experiment, but CTA outperformed behavioral diversity on variations of the ball-collecting problem.

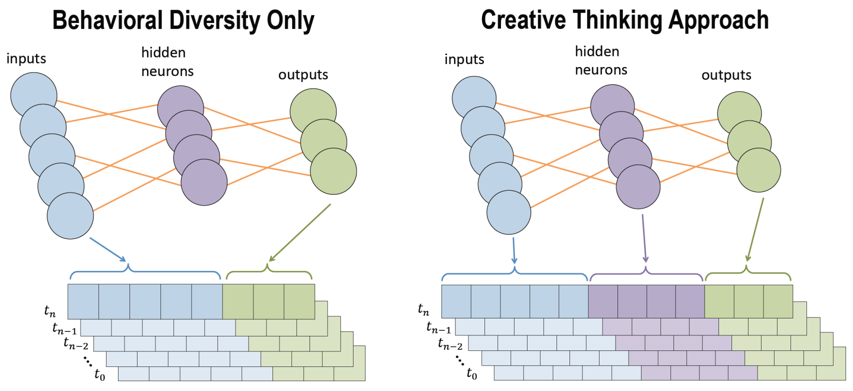


Figure 2.14: Illustration of the Creative Thinking Approach compared to behavioral diversity. Figure from [54].

The *Creative Thinking Approach* was shown to be somewhat successful in encouraging novel neuron firing patterns in the hidden layers to increase the efficiency of the search. However, one could argue whether binarizing the activations and using the Hamming distance is informative enough. For example, there is a significant difference between an activation of 0.51 and an activation of 1.0 of a neuron. Information like this is lost when binarized to either 0 or 1. As such, more research into how the representation of neural networks can be characterized to be as informative as possible and how to measure the distance between representations is required.

Using similarity metrics, such as CKA and CCA introduced in section 2.5, to measure the similarity between neural network representations in neuroevolution is a so far unexplored research direction that is investigated in this thesis. The motivation for employing a different characterization of the learned representation with these similarity metrics is described in more detail in Chapter 3 below.

3 | Cultivating Diversity

Chapter 3 presents the neuroevolution setup that enables the comparison of diversity objectives. More specifically, an overview of the types of mutations, the neural network encoding, and the multi-objective evolutionary algorithm implemented. Furthermore, the chapter describes the various diversity objectives that are compared in this thesis. Including the introduction of a new type of diversity called *representational diversity.* Finally, characteristics of interest are presented that can provide more information about the effectiveness of various diversity objectives on different problems, and thus, make the comparison performed in this work more informative.

3.1 Neuroevolution setup

In order to make a fair comparison between diversity objectives on different problems, a general neuroevolution setup that is applicable to all targeted problems is used. In certain domains, it may be advantageous to use a different neural network encoding or a different set of mutations. The decision to use the same setup across all problem domains is due to the following reasons:

- Ease of use across domains;
- Alternative neural network encodings are beyond the scope of this thesis;
- And most importantly, to perform a straightforward and fair comparison focused solely on diversity objectives, without other factors potentially affecting the results.

Only parameters such as archive threshold for *ad hoc* behavioral distances, mutation probabilities, parameter bounds, and activation functions are specific to each domain. Many of the aforementioned parameters are domain-dependent and specific parameters are required for the evolutionary search to be successful in the various targeted problems. The targeted problems are described in more detail in Chapter 4. Experimental parameters for the targeted problems are listed in appendix A.

3.1.1 Neural network encoding

A graph-based direct neural network encoding is used to represent and evolve neural networks. The encoding is a simplified version of the NEAT encoding, where neural networks are represented as a list of nodes and a list of edges with weight and bias values. Enable bit and innovation number are not used. This type of simple and lightweight encoding has been employed in many previous studies (e.g. [4, 6, 7, 50, 55]). The genotype is transformed into a directed graph, i.e. a neural network.

The NEAT algorithm could have been employed instead of the current setup and encoding. However, the implementation of niching and the use of crossover in NEAT will greatly influence the results. Consequently, the comparison of diversity objectives and any valuable insight into any differences or similarities between them will be affected. Multiobjectivization with a diversity objective also removes the need for niching to protect novel individuals that perform poorly in the beginning.

3.1.2 Evolving neural networks

The neural networks in the initial population are initialized as fully connected with no hidden neurons (i.e. every input neuron is connected to every output neuron), each with randomly sampled weights and biases. The best-performing individuals are evolved using evolutionary operators. Following previous studies (e.g. [4, 6, 7]), only mutations are employed to evolve networks with no crossover. The following two types of mutations are implemented: (1) Structural mutation, and (2) Parametric mutation.

- **Structural mutation**
  - Add neuron – a neuron is added by splitting an existing connection in two, wherein the original connection weight is kept for the two *new* connections.
  - Remove neuron – a randomly selected neuron is removed.
  - Add connection – a connection is added between two randomly selected neurons
  - Remove connection – a randomly selected connection is removed
- **Parametric mutation**
  - Weight mutation – a connection weight is mutated with *polynomial mutation*
  - Bias mutation – a neuron bias is mutated with *polynomial mutation*

Structural mutation. As the name suggests, structural mutation affects the structure of the neural network. All four of the structural mutations described above are each applied once *per* individual with a given mutation probability. Removing a neuron is not possible if there are no hidden neurons, i.e. the input and output neurons can not be removed. The mutation for removing a connection is restricted to only be possible if it does not leave a "dangling" neuron, meaning a neuron with either no input connection or no output connection.

Parametric mutation. This type of mutation affects the weights and biases of the connections and neurons in the neural network respectively. Parametric mutation is applied *per* parameter of an individual with a given mutation probability (i.e. per connection and neuron in the network). A few studies restrict weight and bias values to a set of discrete values in order to reduce the parameter search space. No restriction is used in this work except for a lower and upper bound.

Algorithm 2 Polynomial mutation scheme

```
1: function POLYNOMIAL_MUTATION
2:     $X_p \in [X_{lower}, X_{upper}] \leftarrow$ parameter value with bounded range
3:     $\eta_m \leftarrow$ distribution index
4:     $\alpha = \frac{1}{\eta_m+1} \leftarrow$ mutation power
5:     if random$(0,1) \leq$ mutation_probability then
6:         $\delta_1 = \frac{X_p - X_{lower}}{X_{upper} - X_{Lower}}$
7:         $\delta_2 = \frac{X_{upper} - X_p}{X_{upper} - X_{Lower}}$
8:         $r = $ random$(0,1)$
9:         if $r \leq 0.5$ then
10:            $\delta_q = \big(2r + (1-2r)(1-\delta_1)^{\eta_m+1}\big)^{\alpha} - 1.0$
11:        else
12:            $\delta_q = 1.0 - \big(2(1-r) + 2(r-0.5)(1-\delta_2)^{\eta_m+1}\big)^{\alpha}$
13:        $X_p := X_p + \delta_q \cdot (X_{upper} - X_{lower})$
14:        $X_p := \begin{cases} X_{lower} & \text{if } X_p < X_{lower} \\ X_{upper} & \text{if } X_p > X_{upper} \\ X_p & \text{otherwise} \end{cases}$
```

As in previous neuroevolution studies [4, 7], the *polynomial mutation* [60] scheme is used in this work to mutate parameters. See Algorithm 2 for pseudocode on how polynomial mutation is applied. With polynomial mutation, a distribution index η_m determines how different the mutated offspring will be from its parent. A small distribution index results in a more different mutated offspring as the mutation power α will be larger, and vice versa for a large distribution index. Due to the inherent elitism of NSGA–II, a small distribution index and thus more variation is used in this work.

3.1.3 Choice of multi-objective evolutionary algorithm

NSGA–II (see section 2.2.3) is used as the evolutionary algorithm for optimizing multiple objectives. The NSGA–II algorithm is well-established and has previously been used in various neuroevolution studies, such as [4, 6, 7, 21, 55] to name a few. Furthermore, NSGA-II was chosen over SPEA2 because the performance advantage with SPEA2 in high-dimensional objective spaces will be negligible when only two objectives are optimized. Additionally, NSGA–II is one of the most efficient multi-objective evolutionary algorithms compared to other multi-objective evolutionary algorithms.

For the comparison of diversity objectives, experiments with only performance and no diversity objective are also performed. Performance alone is used as a control experiment for the targeted problems. In the case of a single objective, NSGA–II is equivalent to an elitist evolutionary algorithm with tournament-based selection [30]. More efficient evolutionary algorithms for single-objective optimization exist (e.g. NEAT). However, as in previous studies (e.g. [4, 6, 7]), NSGA–II is used for optimizing performance alone to ensure that only the objectives affect the results and not any other factors related to the implementation.

3.1.4 General overview of the neuroevolution algorithm

The general overview and step-by-step procedure of the neuroevolution algorithm employed in this work is illustrated in figure 3.1. Algorithm 3 shows the pseudocode equivalent to figure 3.1. The parent population at generation t is denoted by P_t. Likewise, the offspring population is labeled as Q_t. The step-by-step procedure for running a neuroevolution experiment for a given problem environment is as follows:

1. Initially, at generation $t = 0$, the parent population P_t is empty. The offspring population Q_t is populated with minimal fully-connected neural networks, each with randomly sampled weights and biases. A minimal fully-connected neural network contains only input and output neurons, where every input neuron is connected to all output neurons. Additionally, as the parent population is empty at $t = 0$, the entirety of the offspring population becomes the parent population P_{t+1} for the next generation and no individuals are rejected in NSGA–II. In subsequent generations, NSGA–II merge and select non-dominated solutions as usual.

2. For the first step of the evolutionary cycle, the individuals in the offspring population Q_t are simulated in the problem environment (or domain). See algorithm 4.

3. After the simulation of individuals, individuals are evaluated according to the objectives, both the performance on the task and the diversity of the individual. How the diversity of an individual is defined depends on the objective function, and if they are compared to an archive or to the rest of the current population.

4. Once all individuals are evaluated according to the objectives, the offspring population Q_t is merged with the parent population P_t and the NSGA-II algorithm selects the non-dominated individuals for the next parent population P_{t+1} and discards the rest of the individuals. NSGA-II was previously described in section 2.2.3.

5. Next, any statistics that may be required for visualizing results later such as the best performance per generation, best individual, Pareto front, and other statistics of population P_{t+1} are reported and saved. See REPORT_STATISTICS (line 9) in algorithm 3.

6. Finally, the offspring population Q_{t+1} for the next generation is generated by mutating the individuals of P_{t+1} as was described in section 3.1.2 above. Next generation starts from step 2 again with the mutated offspring population Q_{t+1} until the success criterion for the simulated environment is reached or a set maximum number of generations is reached.

Algorithm 3 Neuroevolution algorithm (see figure 3.1)

```
1: function RUN($\mathcal{E}$ – environment, $\mathcal{O}$ – objectives, $N$ – max_generations)
2:     $t \leftarrow 0$, current generation
3:     $P_t \leftarrow$ empty parent population
4:     $Q_t \leftarrow$ initialized offspring population
5:     while not success_criterion and $t < N$ do
6:         $Q_t \leftarrow$ SIMULATE($Q_t, \mathcal{E}$)
7:         $Q_t \leftarrow$ EVALUATE($Q_t, \mathcal{O}$)
8:         $P_{t+1} \leftarrow$ NSGA–II($P_t, Q_t$)
9:         REPORT_STATISTICS($P_{t+1}, \mathcal{O}$)
10:        $Q_{t+1} \leftarrow$ MUTATE($P_{t+1}$)
11:        $t \leftarrow t + 1$
```

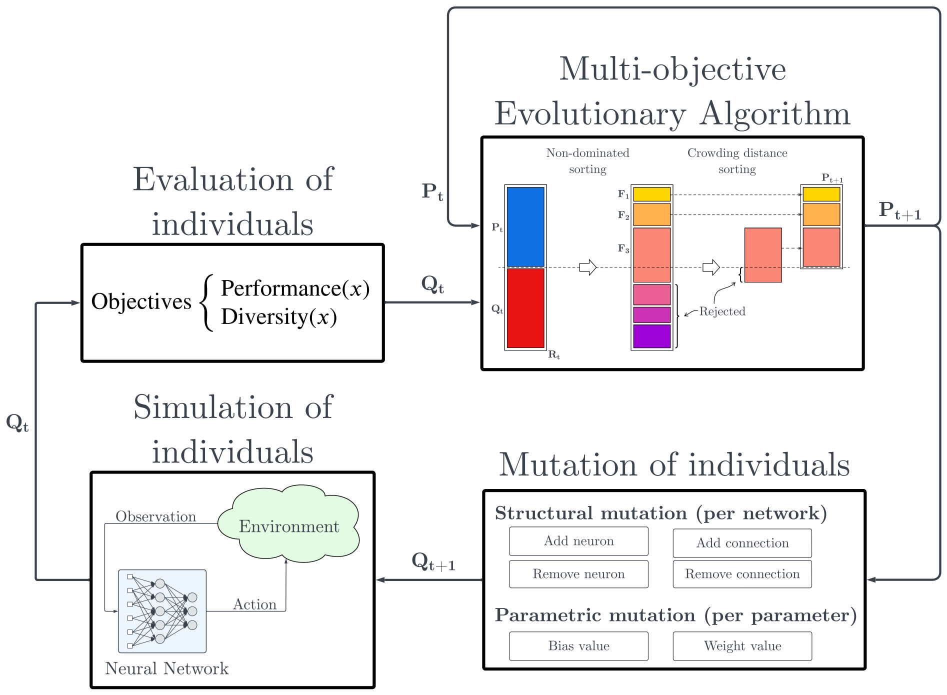


Figure 3.1: General overview of the neuroevolution algorithm. For every generation t, individuals in the offspring population Q_t are simulated in the environment of the targeted problem. Next, individuals are evaluated according to the objective functions employed. NSGA–II merges the offspring population Q_t and parent population P_t and selects non-dominated solutions for population P_{t+1} of the next generation. Finally, the offspring population Q_{t+1} for the next generation is generated by mutating the individuals of P_{t+1}. Repeat until a success criterion or the max number of generations is reached.

Algorithm 4 Simulation of individuals

```
 1: function SIMULATE($P$ – population, $\mathcal{E}$ – environment)
 2:     $K \leftarrow$ maximum number of simulation steps
 3:     for individual in $P$ do
 4:         neural_network $\leftarrow$ individual.GENOTYPE_TO_PHENOTYPE()
 5:         observation $\leftarrow$ $\mathcal{E}$.GET_INITIAL_OBSERVATION()
 6:         $i \leftarrow 0$
 7:         while $i < K$ do
 8:             action $\leftarrow$ neural_network.ACTIVATE(observation)
 9:             observation $\leftarrow$ $\mathcal{E}$.STEP(action)
10:             $i \leftarrow i + 1$
11:         individual.SAVE_SIMULATION_DATA()
```

Important to note that the pseudocode for simulating individuals is defined in more general terms in algorithm 4, and will be different depending on the targeted problem.

The function call SAVE_SIMULATION_DATA on line 11 of algorithm 4 is a general function for saving all data that is necessary to evaluate the performance and diversity of an individual later. Computing the diversity of an individual is usually done by comparing the individual to the rest of the population after all individuals have finished simulating. For example, with the Hamming distance, saving the input-output history for each individual during simulation is required in order to compare to the rest of the population after all individuals are simulated. The diversity objectives used in this thesis are presented in section 3.2 below.

3.2 Comparison of Diversity Objectives

This section presents the motivation for introducing representational diversity as a new type of diversity, and in more technical terms, how the learned representation is characterized and used. Afterward, all diversity objectives (i.e. experimental treatments) compared in this thesis are outlined.

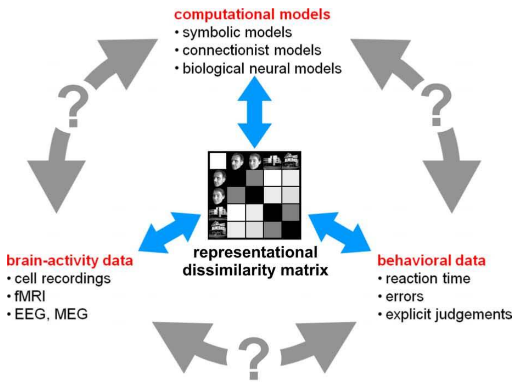
3.2.1 Introducing a new type of diversity

A fundamental challenge in neuroscience is to model the representations in human brains by quantitatively relating brain-activity measurements, behavioral measurements, and computational modeling. One study [61] suggested abstracting the measured brain activity and computing *representational dissimilarity matrices* (see figure 3.2) in order to characterize the information of a given representation. The correlation between activity patterns (in a brain or model) elicited by four stimulus images (i.e. input) was computed to successfully reveal the representational structures for different parts of the brain.

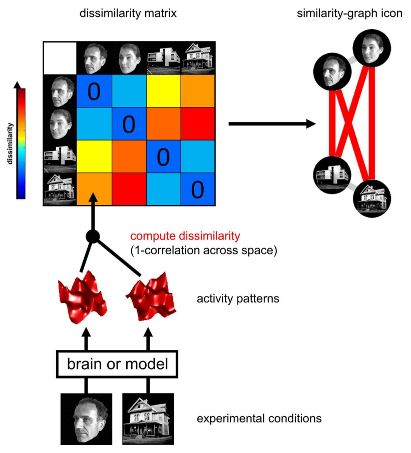
Inspired by this study and other studies on brain representations in neuroscience, researchers proposed similarity metrics (CKA, CCA) for measuring the similarities between learned representations of deep neural networks using neuron activations that were introduced in section 2.5. The purpose was to study the similarity between the learned representations of two neural networks that were initialized differently, and the proposed similarity index was found to reliably identify correspondences between learned representations [46].

3.2.2 Adapting representational diversity to neuroevolution

Bridging the gap to neuroevolution, no previous work has attempted to encourage a diversity of neural network representations other than in the *Creative Thinking Approach* (CTA) with a somewhat similar use of neuron



(a)



(b)

Figure 3.2: Characterizing representations in different parts of the brain with a *representational dissimilarity matrix.* **(a)** Illustrates the representational dissimilarity matrix as a hub that relates the three major branches of research in systems neuroscience: behavioral experimentation, brain-activity, and computational modeling. The matrix acts as a quantitative abstract representation that can be compared across the regions of the brain or a model. **(b)** Computation of the representational dissimilarity matrix by spatial correlation of activity patterns from a brain or model. Figures from [61].

activations. The motivation for proposing *representational diversity* as a new type of diversity objective was briefly outlined in section 2.7.3 of Chapter 2 and further inspired by research in neuroscience presented above. In short, the motivation for representational diversity as a new type of diversity in neuroevolution is as follows:

> Encouraging representational diversity is to encourage a diversity of *learned neural network representations*. That is, a diversity of learned representations dependent on the *structure* and *behavior* of the neural networks, with the additional advantage of being domain-independent.

In more technical terms, the learned representation of a neural network is characterized by a matrix of non-binarized neuron activations. Only activations from hidden neurons and output neurons are used, in contrast to CTA which binarizes the input-output history and hidden neuron activations collected in sequence (see figure 3.3). This way of representing the learned representation is arguably more informative, but this remains to be seen.

The similarity metrics *centered kernel alignment* (CKA) and *canonical correlation analysis* (CCA) can be applied to measure the similarity between learned representations of two individuals using the *representation matrix* in figure 3.3. Originally, these metrics were used on neural networks with identical topologies. Using neuroevolution in this work, the metrics are applied to evolved neural networks with different topologies. How this affects the ability to reliably find similarities remains to be seen.

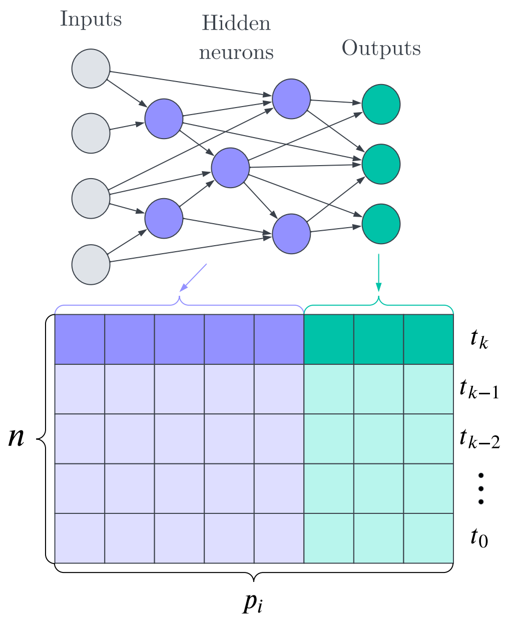


Figure 3.3: Characterization of a neural network representation. The learned representation is represented by a matrix of activations of p_i neurons in the neural network of individual i for n examples (or samples). The activations are appended to the matrix during simulation. Contrary to CTA in figure 2.14, activations are not binarized and collected in a matrix instead of a sequence. Additionally, only activations from hidden and output neurons are used (i.e. no input values).

Computing the similarity between the representation matrix of two individuals, x and y, involves matrix multiplication of matrices of size $(n_x \times p_x)$ and $(n_y \times p_y)$ respectively. Matrix multiplication requires the first dimension of each representation matrix to be equal (i.e. $n_x = n_y$) but the number of neurons can be different (i.e. $p_x \neq p_y$). The former is not guaranteed to be the case for problems often targeted in neuroevolution such as evolutionary robotics tasks (e.g. a simulated robot failed earlier than another robot). In the case where the number of samples, n_x and n_y, of two representation matrices are not equal, only corresponding samples for the smallest of the two are considered (see figure 3.4).

In short, neural networks can be evolved with a different number of neurons but the number of samples considered has the be the same when computing the similarity between representations. A more mathematical definition of CKA and CCA was previously defined in section 2.5. See section 3.2.3 below on how these similarity metrics are used to define the representational diversity of an individual.

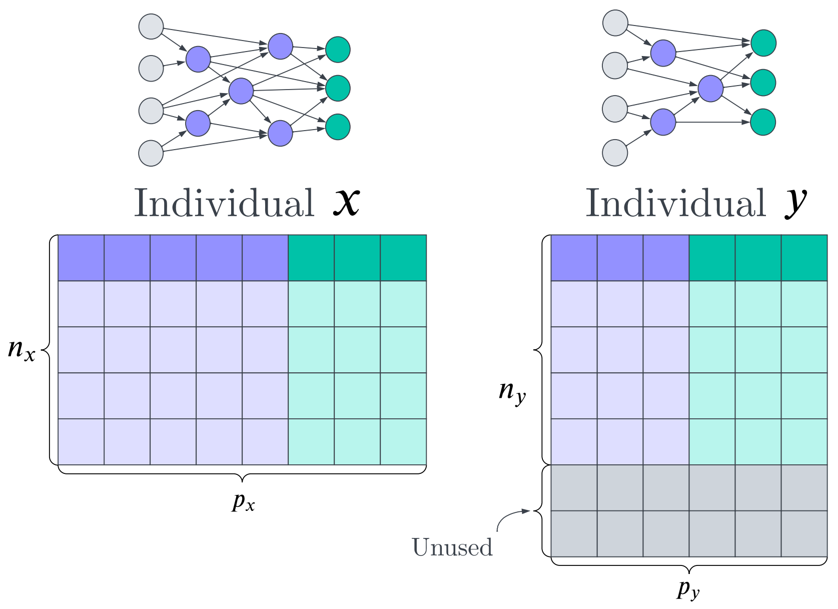


Figure 3.4: Computing similarity between representation matrices of two individuals x and y. Using the similarity metrics CKA or CCA requires an equal number of samples $n_x = n_y$. The number of neurons p_x and p_y in individual x and y respectively can be different. If the number of samples is different, only the corresponding samples of the representation matrix with the smallest number of samples are used while the remaining samples of the largest matrix are unused (grey).

3.2.3 Experimental treatments

A comparison of diversity objectives is performed on four targeted problems. In total, seven experimental treatments are applied to every problem. An experimental treatment is defined as the combination of a performance objective and a diversity objective as follows:

$$\text{Maximize} \begin{cases} F(x) \\ Diversity(x) \end{cases} \tag{3.1}$$

where $F(x)$ is the performance objective. The task is to optimize both objectives, but performance is the *primary* objective while the diversity objective is the *secondary* objective used to help the search. A baseline treatment with only the performance objective is also employed as a control experiment. The purpose of the control experiment is to determine if a traditional evolutionary approach without any diversity objective can solve the problem as efficiently as with diversity objectives. Of the remaining six, two treatments for each type of diversity described in Chapter 2 are chosen (i.e. behavioral, structural, representational). The following experiments are performed in this thesis:

(1) *Performance Alone*

$$\text{Maximize } F(x) \tag{3.2}$$

See Chapter 4 for how the performance F is evaluated in the targeted problems.

(2) *Performance + Novelty*

$$\text{Maximize} \begin{cases} F(x) \\ D(x) = \frac{1}{k}\sum_{i=1}^{k} d_{\text{ad-hoc}}(x, \mu_i) \end{cases} \tag{3.3}$$

See Chapter 4 for $d_{\text{ad-hoc}}$ behavioral distances of every targeted problem.

(3) *Performance + Hamming distance*

$$\text{Maximize} \begin{cases} F(x) \\ D(x) = \frac{1}{|P|}\sum_{j \in P} d_{\text{ham}}(x, j) \end{cases} \tag{3.4}$$

See equation 2.14 for d_{ham} distance.

(4) *Performance + Modularity*

$$\text{Maximize} \begin{cases} F(x) \\ D(x) = Q(x) \end{cases} \tag{3.5}$$

See equation 2.17 for Q score calculation.

(5) *Performance + Modularity Diversity*

$$\text{Maximize} \begin{cases} F(x) \\ D(x) = \frac{1}{|P|}\sum_{j \in P} \Delta_{decomp}(x, j) \end{cases} \tag{3.6}$$

See equation 2.18 for Δ_{decomp} distance.

(6) *Performance + Centered Kernel Alignment*

$$\text{Maximize} \begin{cases} F(x) \\ D(x) = \frac{1}{|P|}\sum_{j \in P} 1 - \text{CKA}(x, j) \end{cases} \tag{3.7}$$

See equation 2.7 for CKA similarity index.

(7) *Performance + Canonical Correlation Analysis*

$$\text{Maximize} \begin{cases} F(x) \\ D(x) = \frac{1}{|P|}\sum_{j \in P} 1 - R^2_{\text{CCA}}(x, j) \end{cases} \tag{3.8}$$

See equation 2.9 for R^2_{CCA} similarity index.

For ease of reference when discussing the results or the use of specific experimental treatments, the abbreviations in table 3.1 below are used.

Table 3.1: Summary of all experimental treatments and their abbreviations. The abbreviations are used for ease of reference of the various diversity objectives in the subsequent discussion of results.

Treatment	**Objectives**
PA	Performance Alone
NOVELTY	Performance + Novelty
HAMMING	Performance + Hamming distance
MOD	Performance + Modularity (Q-score)
MODDIV	Performance + Modularity Diversity
CKA	Performance + Centered Kernel Alignment
CCA	Performance + Canonical Correlation Analysis

To summarize, a total of seven experimental treatments are employed with a control experiment (PA) and two treatments with a different diversity metric for each diversity type (i.e. behavioral, structural, and representational). All treatments with a diversity objective, except NOVELTY and MOD, define the diversity of an individual as the average distance to the rest of the individuals in the population P. A few other implementation details to note for a few of the treatments:

- NOVELTY – The novelty calculation is the behavioral distance from individual x to its k nearest neighbors of the archive (i.e. not to all in the current population P). An individual is added to the archive if the computed novelty is above a threshold. The threshold for each targeted problem is listed in appendix A.
- MOD – The modularity treatment is slightly different from the rest as no comparison to the rest of the current population or an archive is required to calculate the modularity of an individual.
- CKA & CCA – The representational diversity treatments use a similarity index. Encouraging diversity thus means minimizing similarity. For the sake of keeping all experiments as purely maximization problems, minimizing similarity is converted into maximizing diversity by subtracting the similarity index from 1 (see eq. 3.7 and 3.8). A small similarity index will result in a large diversity score, and vice versa.
- CKA – As was briefly mentioned in section 2.5.1 on CKA, the similarity calculation is performed across the features (i.e. neurons) instead of the samples. The calculation is faster as the targeted problems in this work involve a significantly larger number of simulation steps (i.e. samples) than the number of neurons the networks will ultimately evolve.

3.2.4 Parameter tuning, experiments, and statistical testing

The experimental parameters are specific to each targeted problem, with different mutation probabilities, activation functions, etc. (see appendix A). To ensure differences between treatments are only due to different diversity objectives, the same parameters are used for treatments applied to the same problem. Setting the parameters is done by using similar values used in previous studies and further adjusting them based on initial test experiments if necessary. However, comprehensive parameter tuning has not been performed due to computational constraints and time. Limited computational resources and a large number of parameters and possible combinations of them, make parameter tuning time-consuming.

Likewise, a population size of 100 is employed due to the number of experiments required. All experiments are repeated 50 times with random seeds (i.e. different stochastic events). With 7 targeted problems (including variations of the same problem), 7 treatments, and 50 runs per treatment, the total number of experiments performed is: $7 \cdot 7 \cdot 50 = 2450$. All statistical significance testing of results presented in Chapter 5 apply the Mann-Whitney U test.

3.3 Problem characteristics of interest

In order to identify which diversity objectives are most suitable for which type of problem, a number of interesting characteristics are studied in this work. The characteristics of a problem are the qualitative aspects of the problem and its structure that are prominent and important. Some studies focus specifically on problems that express one of these aspects, such as *modularity* in [4, 10] and *deceptiveness* in [5, 21]. However, a comprehensive set of important problem characteristics has not previously been established. To fill this gap, four characteristics of interest are proposed in this thesis:

- Modularity
- Regularity
- Deceptiveness
- Environment space

The above characteristics are studied because they are important aspects expressed by many of the commonly targeted problems in neuroevolution. Naturally, this list is by no means exhaustive but covers both previously studied characteristics and also characteristics that merit further study. The targeted problems presented in Chapter 4 are, in part, selected in order to compare diversity objectives on problems that have *unique* combinations of these characteristics. The targeted problems are characterized according to the characteristics. An informal definition of each characteristic and motivation for studying them is given below.

3.3.1 Modularity

Modularity is understood to be the degree to which the problem structure can be decomposed into independent sub-tasks and the potential for a modular neural network structure to evolve. In other words, the problem lends itself nicely to a modular structure, both for solving the task and for the neural network structure. It should be noted that a modular structure may not be required to solve the problem.

Modularity is an important aspect that has previously been studied in neuroevolution [4, 10, 62]. In general, modularity has been studied as an important *organizing principle* in biological neural networks of human and animal brains [63] and a key driver of *evolvability* – the ability to rapidly adapt to novel environments [10]. In addition, modularity has been shown to facilitate the evolution of novel *network activity* [64]. The role of modularity in evolution remains an important area of research. Therefore, modularity as a problem characteristic is studied in this work.

3.3.2 Regularity

Regularity is expressed if there is a repeating or oscillatory nature to the problem and its structure. Examples of problems exhibiting regularity are problems where solving a repeating sub-task is required, a sequence of different sub-tasks is required to be solved, or discovering an oscillatory pattern is useful for solving the task. In other words, there is a pattern to the problem, and discovering such a pattern is useful for solving the problem.

Patterns appear frequently in evolution (and nature in general) through symmetry, repetition of identical structures, repetition with varying structures, and regular properties [65]. It is generally accepted that regularity and oscillatory patterns in the human brain are widespread and serve an important role. Studies on such patterns in biological neural networks include the *co-emergence* of regularity and complexity in neural activity [66] and how *neural oscillatory activity* in the brain responds to rhythmic stimulus (e.g. speech, music, and sensorimotor input) from the environment and are also theorized to be *predictive* processes [67].

In neuroevolution, such patterns and regularities were partly the inspiration for developing a new type of neural network encoding called *compositional pattern producing network* (CPPN) [65]. CPPNs compose the network as a graph of functions and can be evolved to discover and produce complex patterns. Discovering such oscillatory patterns is an important aspect of robot locomotion tasks that are often targeted in neuroevolution (e.g. [4, 5, 21]). The relationship between environmental input patterns and neural network activity patterns, both biological and artificial, is important and is why regularity is studied in this work.

3.3.3 Deceptiveness

Deceptiveness as a problem characteristic has been the subject of several studies on evolutionary algorithms (EAs), especially with a focus on diversity maintenance techniques [5, 20, 21]. Including showcasing the efficiency of novelty search in deceptive domains (see section 2.6.3). In [5], Lehman and Stanley refer to deception as a definition of problem hardness as follows:

> A deceptive problem is one in which a reasonable EA will not reach the desired objective in a reasonable amount of time. That is, by exploiting objective fitness in a deceptive problem, a population's trajectory is unlikely to uncover a path through the search space that ultimately leads to the objective.

The authors emphasize that this definition is an intuitive definition and does not account for the impact of parameters, problem representation, or search operators on the performance of EAs. This previously established intuitive definition of deceptiveness in [5] is also used in this work.

Generally, the deceptiveness of a problem is often linked to the objective function and is the degree to which the problem contains obvious sub-optimal solutions (i.e. local optima) that are easily converged to and normally hard to avoid. Deceptive problems are usually harder to solve with a traditional evolutionary algorithm using a reasonable objective function for the problem domain.

3.3.4 Environment space

Environment space as a characteristic is understood to be how *discrete* or *continuous* the problem structure is. More specifically, problems where an agent or robot controlled by a neural network interacts with either a continuous environment with infinite states or a discretized environment with a finite number of states. Also included is the representation and use of input and output values. Depending on the environment, input values can be discrete integers, continuous real-valued numbers, or a combination of the two. Problems also differ in how the output values of the neural networks are interpreted and subsequently affect the environment. For example, the output values can be applied as they are or discretized in some way to represent an action or prediction of some sort.

Defining an appropriate representation of a problem is an important aspect of evolutionary computation (and optimization in general) that can greatly affect performance. Many studies use vastly different representations for the same problem, which changes the problem difficulty and with it the reported performance of evolutionary algorithms [68]. There is, in particular, a general lack of research into how the representation of the environment space affects the effectiveness of diversity objectives. Whether certain diversity objectives are more effective in discrete or continuous environments, or if the representation of the environment space has any effect at all on the evolutionary search with diversity objectives. For this reason, the environment space representation is chosen as one of the characteristics that are studied in this work.

4 | Experimental Setup

Chapter 4 presents the experimental setup for the thesis. The following four problems are targeted: (1) the Retina problem, (2) the Tartarus problem, (3) Maze navigation, and (4) Robot locomotion. The main reasons for choosing the targeted problems are:

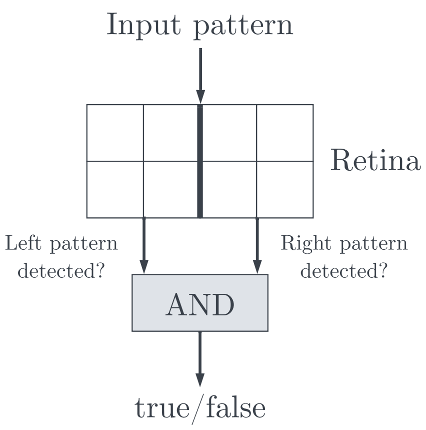
- They have previously been used in related research to study diversity-driven neuroevolution;
- Some of them are hard to solve using traditional methods, despite their relative simplicity;
- They are easy to set up and reproduce from scratch or with existing simulator frameworks available for a few of the problems;
- They have distinct characteristics, thus giving more valuable insight into the role of various diversity objectives described in Chapter 3.

The following sections are dedicated to the targeted problems and variations of them. The sections describe the problem, implementation setup, and neural network details. In the final section, the problems are characterized according to the characteristics defined in Chapter 3.

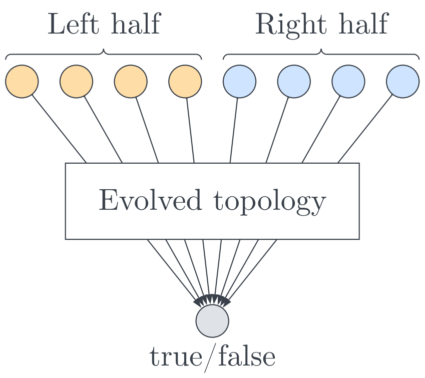
4.1 The Retina problem

The `Retina` problem is a *pattern-recognition* problem where the objective is to correctly classify input patterns (see figure 4.1). The problem has been used in several previous studies focused on the evolution of modular structures in neural networks. In particular, to study if modular structures appear in neural networks when applied to modular problems [10, 62] and to study structural diversity objectives in neuroevolution [4, 9]. The 8-bit input pattern contains two independent parts, left and right. The input pattern is classified as 1 (*true*) if both the left and right parts contain a target pattern, otherwise, it is classified as 0 (*false*). The simulated environment is from [69] with associated code repository[1].

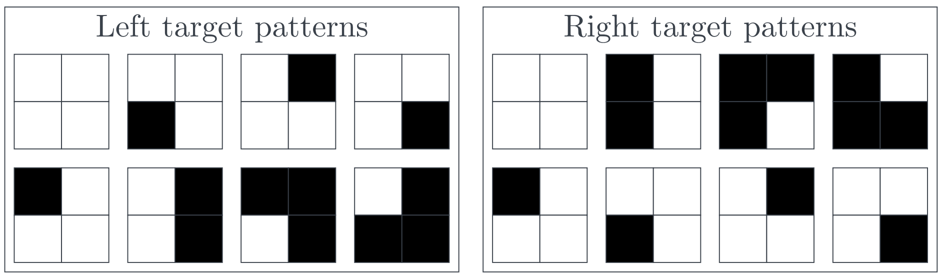
[1]https://github.com/PacktPublishing/Hands-on-Neuroevolution-with-Python



(a) The retina for detecting input patterns.



(b) Neural network setup for the retina problem.



(c) Target patterns for the left and right side of the retina.

Figure 4.1: The retina takes combinations of input patterns and classifies them as *true* if both halves of the retina contain a target pattern and *false* if not. A black square in a pattern will be passed to the neural network as a value of 1.0, and a white square as a value of 0.0. The left and right target patterns are mirrored depending on which half of the retina it is a target of, including a few shared target patterns. Figures adapted from [10].

The motivation is to see independent parts of a visual scene by abstracting the left and right sides of a retina. The `Retina` problem is highly modular because the left and right parts are completely independent of each other, which lends itself nicely to neural networks with two distinct modules. Despite the modular structure of the `Retina` problem, evolving neural networks can also tend to produce non-modular structures [62]. The problem is fairly straightforward with no obvious local optimum the neural network can converge to.

The problem setup follows the same setup used in previous studies [4, 10], but with a slightly different neural network setup. In previous studies, the neural networks are limited to a fixed number of layers, a maximum number of neurons per layer, and a set of discrete values for the weights and biases. A neural network setup with no structural limits and continuous weights and biases is used in this thesis to enable a similar setup across all problem domains.

Neural network details. The neural networks are initialized as a fully-connected network with 8 input neurons and 1 output neuron (see figure 4.1b). The evolved neural network should output a positive value (*true*) whenever the left half of the retina contains a left target pattern *and* the right half of the retina contains a right target pattern. Otherwise, the neural network outputs a negative value (*false*). To ensure the neurons output a value in the range $[-1, 1]$, the *tanh* function is used as the activation function. A parameter λ determines the slope of the activation function between its limits. As in [4, 10], λ is set to 20 to resemble a step function with a very steep slope.

Performance. The performance F of a neural network is evaluated by presenting all 256 possible combinations of input patterns and counting the number of errors (i.e. wrongly classified patterns):

$$F(x) = 1 - \frac{errorCount}{256} \tag{4.1}$$

where *errorCount* is the number of wrongly classified patterns out of the 256 possible combinations. Thus, the performance score ranges between 0 and 1. With a score of 1, all inputs were correctly classified. All inputs were wrongly classified with a score of 0.

Ad hoc distance. No *ad hoc* behavioral distance has previously been defined for the `Retina` problem. The Hamming distance was used in [4] to represent behavioral diversity. As a result, this thesis defines the *ad hoc* behavioral distance as the euclidean distance between the outputs of two neural networks for a set of test patterns as follows:

$$d_{\text{ad-hoc}}(x, y) = \|\beta_x - \beta_y\| \tag{4.2}$$

where the behavior vector β_x of individual x consists of the set of *raw* output o of the neural network for k test patterns.

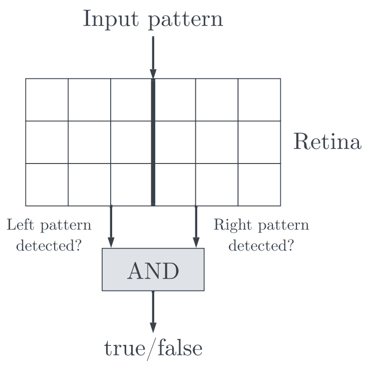
$$\beta_x = \{o^{(1)}, \ldots, o^{(k)}\} \tag{4.3}$$

In this work, 4 pairs of test patterns are used to produce the behavior vector of an individual. The chosen test patterns are different from the target patterns in figure 4.1c. The reason is to differentiate between individuals that behave differently on *unseen* patterns and disregard the performance on the task. Consequently, the behavior definition is domain-dependent as domain knowledge is required to design the test patterns.

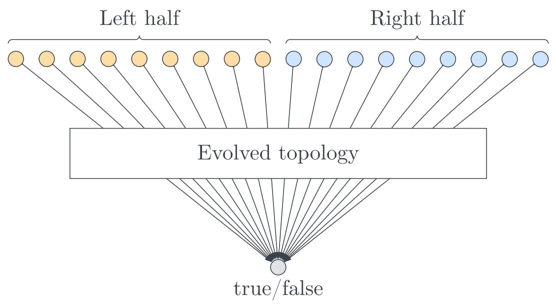
Success criterion. An individual is successful if all 256 combinations of target patterns are classified correctly. The individual has achieved the maximum performance score, and thus, the evolutionary search is stopped.

Number of inputs. 8: 4 inputs for the left half of the retina and 4 inputs for the right half of the retina.

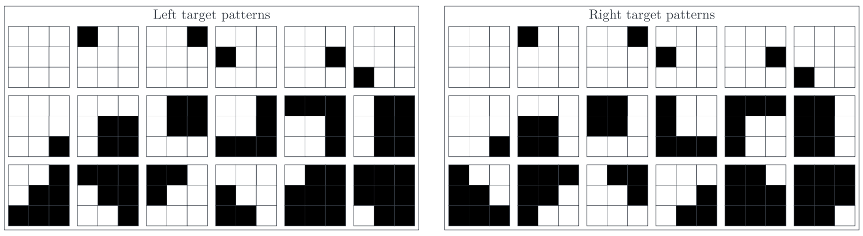
Number of outputs. 1: the classification of the input pattern, either *true* or *false*.



(a) The retina for detecting input patterns for the 3x3 version.



(b) Neural network setup for the 3x3 version now with a total of 18 input neurons.



(c) Left and right target patterns for the extended 3x3 version.

Figure 4.2: The 3x3 retina problem with more and larger target patterns. The task of pattern-recognition remains the same. As with the 2x2 patterns in the original version, the left and right 3x3 target patterns are mirrored including a few shared target patterns.

4.1.1 A harder Retina problem

Early testing of the `Retina` problem indicated very similar results for many of the diversity objectives, which is not very informative for a comparison of them. This prompted the need for a more difficult version of the problem. The original `Retina` problem is not tunable in terms of difficulty, and a more difficult version of the problem does not currently exist. Therefore, a new and harder version of the `Retina` problem with larger and more target patterns is implemented in this thesis (see figure 4.2).

The task is the same as in the original version, but the problem is harder now with a pattern size of 3x3 and an increased number of patterns to be classified correctly. As a result of the increased size of the target patterns, the neural network will now have 18 input neurons (see figure 4.2b). In this extended version of the `Retina` problem, there are 18 target patterns for each half of the retina. The neural networks are be presented with a total of 1296 combinations of target patterns. To differentiate between the original version and the new extended version, they are from this point on referred to as the `2x2-Retina` problem and the `3x3-Retina` problem respectively.

4.2 The Tartarus problem

The `Tartarus` problem [53, 70, 71] is a grid-based optimization problem. The *Minigrid*[2] library is customized to simulate the environment. The problem environment is a 6×6 grid world containing movable blocks and an agent representing a bulldozer with the ability to push the blocks (see figure 4.3). The objective is for the agent to locate and move each of the blocks to the edge of the environment. A limit of 80 moves is afforded the agent, during which the agent can perform three possible actions to navigate the environment: *turn left*, *turn right*, or *go forward*. If there is a block ahead of the agent while going forward, the block is only pushed if the block is not against another block or a wall. The agent can achieve a maximum score of 10, with 1 point received for each block at the edge of the environment and 2 points for any block in the corners.

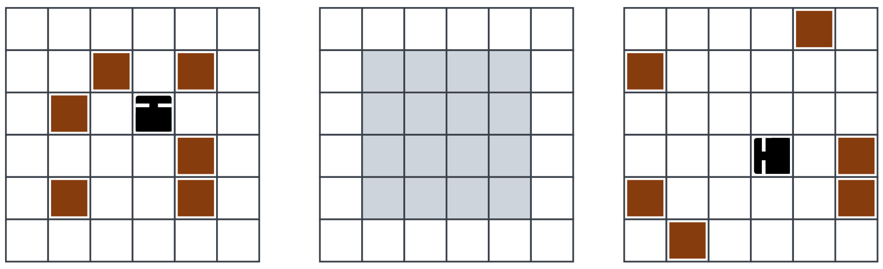

(a) Example of an initial board configuration. **(b)** Valid start positions for the blocks and the agent. **(c)** Example of a final board configuration.

Figure 4.3: The Tartarus problem. Movable blocks in brown, and the agent in black with the ability to push blocks. The goal is for the agent to push all blocks to the edge of the environment within 80 moves. The agent receives a point for each block at the edge, and two points for blocks in the corners. To navigate the environment, the agent can perform three possible actions: turn left, turn right, or go forward. Figures adapted from [71].

[2]https://minigrid.farama.org

Despite its simplicity and small grid world, the `Tartarus` problem is challenging. The agent does not have any prior knowledge about its starting position or orientation in the environment. The only information available to the agent is the eight surrounding grid cells and the previous action performed by the agent. Two distinct situations may appear identical to the agent but can require entirely different actions (i.e. perceptual aliasing).

While the task is challenging in itself, it is also somewhat deceptive as there is a possibility of an agent learning a simple *mechanical* behavior that is better than random behavior but does not exhibit the complex behavior required to achieve a maximum score [70]. Due to the small grid, the *mechanical* behavior of simply pushing a block forward whenever encountered can earn a mediocre score between 4 and 6 points. The agent must learn a more complex behavior in order to achieve a maximum score within the move limit. It requires learning the behavior of first locating and pushing a block to the edge, moving around the block, and finally pushing the block into the corner. Accordingly, the agent (i.e. neural network) requires a type of memory element to remember the previous observation or action taken in order to learn a more complex behavior. A *recurrent neural network* (RNN) with a fixed topology is used in [53, 70] for this purpose.

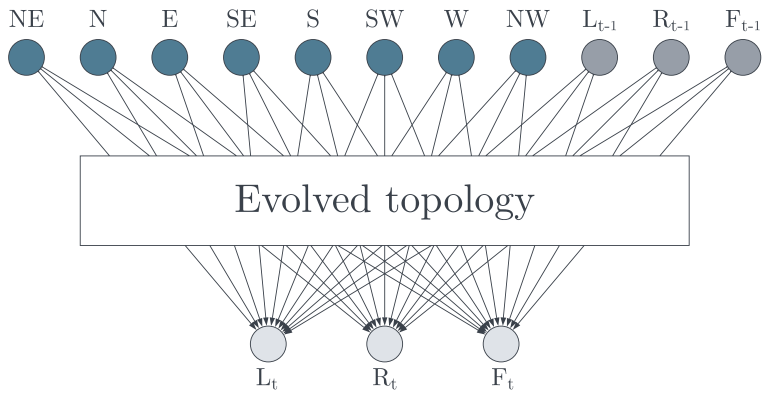


Figure 4.4: Neural network setup for the Tartarus problem. Outputs of the neural network are turn left (L_t), turn right (R_t), and go forward (F_t). The inputs (NE, . . . , NW) are the surrounding the eight squares, and the inputs ($L_{t-1}, R_{t-1}, F_{t-1}$) are the one-hot encoded values of the previous neural network output.

Neural network details. Alternatively, as is employed in this work, the action of the previous step taken by the agent is used as input to inform the action of the current step. The output values of the network for the previous step are one-hot encoded and used as input for the current step. The neural networks are initialized as a fully-connected network with 11 input neurons and 3 output neurons. The evolved neural network outputs a value for each of the three possible actions at step t (see figure 4.4). The action with the largest output value is chosen as the action taken by the agent. Neurons use the *sigmoid* activation function to ensure output values in the range $[0, 1]$.

Performance. The performance F of a neural network is evaluated by the average score achieved on a set of k board configurations of blocks:

$$F(x) = \frac{1}{k} \sum_{c \in C} stateEvaluation(x, c) \tag{4.4}$$

where c is a board configuration in the set C of k board configurations. The number of points achieved by individual x on configuration c is returned by *stateEvaluation*. In this work, a set with 30 board configurations is used.

All possible starting configurations of blocks are generated beforehand. Of these, the *partially solvable* configurations are removed and only the *fully solvable* ones are kept (see figure 4.5). For every experiment (i.e. run), the set C is created by randomly sampling k *fully solvable* configurations once. All neural networks are evaluated on the configurations in C.

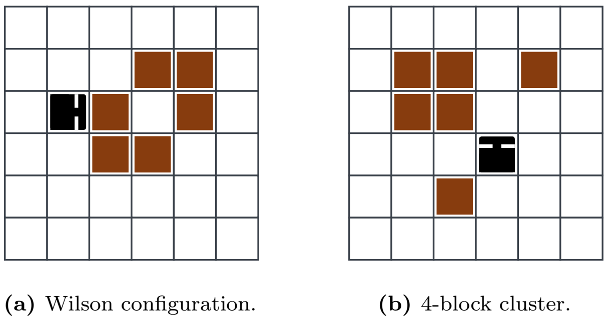

(a) Wilson configuration. **(b)** 4-block cluster.

Figure 4.5: Two types of partially solvable Tartarus configurations. Pushing a block that is up against another block is not possible, thus configurations with 4-block clusters as is shown in **(b)** are only partially solvable. Likewise, the Wilson configuration in **(a)**, will result in a 4-block cluster regardless of which block is pushed. Figures adapted from [71].

Ad hoc distance. The *ad hoc* behavioral distance between two individuals, x and y, is defined as the Manhattan distance between the corresponding blocks in the two sets of final board configurations (i.e. after 80 moves):

$$d_{\text{ad-hoc}}(x, y) = \text{Manhattan}(\beta_x(i), \beta_y(i)) \tag{4.5}$$

where the behavior vector β_x of individual x contains the final positions of corresponding blocks after 80 moves for all k sampled configurations:

$$p^{(c)} = \{(x_1, y_1), \ldots, (x_6, y_6)\} \tag{4.6}$$

$$\beta_x = \left\{p^{(c)}, c \in [1, k]\right\} \tag{4.7}$$

where $p^{(c)}$ is a vector with the final position of the 6 blocks in board configuration c. Important to note that the Manhattan distance calculation is between *corresponding* blocks. Thus, individuals are rewarded for solving board configurations differently from other individuals (e.g. blocks in the corners). The *ad hoc* behavioral distance was first defined in [70].

Success criterion. An individual is successful if an average score of 10.0 is achieved on all sampled board configurations. In other words, getting the maximum score for all board configurations. The evolutionary search is then stopped.

Number of inputs. 11: 8 for the grid cells surrounding the agent and 3 for the one-hot encoded outputs of the previous step.

Number of outputs. 3: one for each of the three possible actions, *turn left*, *turn right*, and *go forward*.

4.2.1 A deceptive version of the Tartarus problem

In addition to the above-described problem setup used in [53], there exists an explicitly *deceptive* version of the Tartarus problem used in [70]. In the `Deceptive-Tartarus` problem, blocks in the corners are still awarded 2 points but blocks at the edges are awarded -1 points. In this version of the problem, the optimal strategy is to push a block to each of the four corners and leave the remaining two blocks in the middle of the grid giving 0 points and thus resulting in a maximum score of 8. The new scoring scheme is deceptive because it penalizes neural networks that converge to the local optima of mechanically pushing blocks until it hits a wall and is awarded points.

4.3 Maze Navigation problem

A Maze Navigation problem was used to demonstrate the efficiency of Novelty Search [5] and has since been used in various other studies [6, 7, 21, 72]. More specifically, the maze domain is usually designed to be deceptive to a reasonable fitness function. In many of the maze domains previously used, there is an attractive dead-end (i.e. local optima) that individuals will almost always converge to. Due to the deceptiveness of the maze, searching for novelty or diversity has been shown to lead to better performance rather than pure fitness.

The task is for a mobile robot to navigate a maze domain from a starting point to a goal within a fixed time limit. The maze domains contain cul-de-sacs that prevent a direct route to the goal and require the robot to properly navigate the environment. In this work, two different maze domains

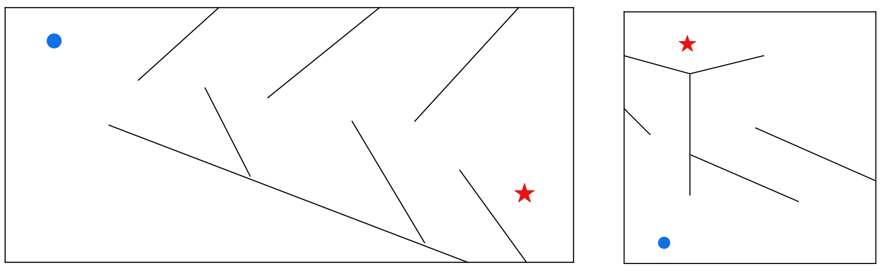

(a) Medium maze. **(b)** Hard maze.

Figure 4.6: The medium and hard maze domains. The blue circle represents the starting position of the robot, and the red star is the goal. Both maze domains are deceptive with attractive local optima. The maze domains in **(a)** and **(b)** were also used in [5].

are used. The first maze, the `Medium-Maze`, is designed to have multiple deceptive dead-ends on the path to the goal (see figure 4.6a). The robot must learn to weave between the dead-ends on the way to the goal. The second maze, the `Hard-Maze`, is designed to be even more deceptive. The robot must explore areas away from the goal that initially leads to lower fitness before the goal can be reached. In this maze domain, there is an attractive local optimum very close to the goal (see figure 4.6b). Simulated maze domains are from [69] with associated code repository[3].

Important to note is that the task is not a pathfinding problem but to find a neural network that can navigate the maze by itself using only the sensors it is equipped with [5]. In a pathfinding problem, the optimal path is unique to every maze layout while for this maze problem, an evolved neural network would be able to navigate any maze, successfully or unsuccessfully, regardless of the layout. The neural network does not necessarily find the optimal path.

The robot is equipped with 3 range sensors and a 4 pie-slice goal radar (see figure 4.7a). Range sensors measure the distance to the nearest obstacles, and the goal radar indicates which direction the goal is in relation to the perspective of the robot. The robot moves by setting the angular and linear velocity of the robot that determines how fast the robot is turning left or right and the forward or backward speed of the robot respectively.

[3]https://github.com/PacktPublishing/Hands-on-Neuroevolution-with-Python

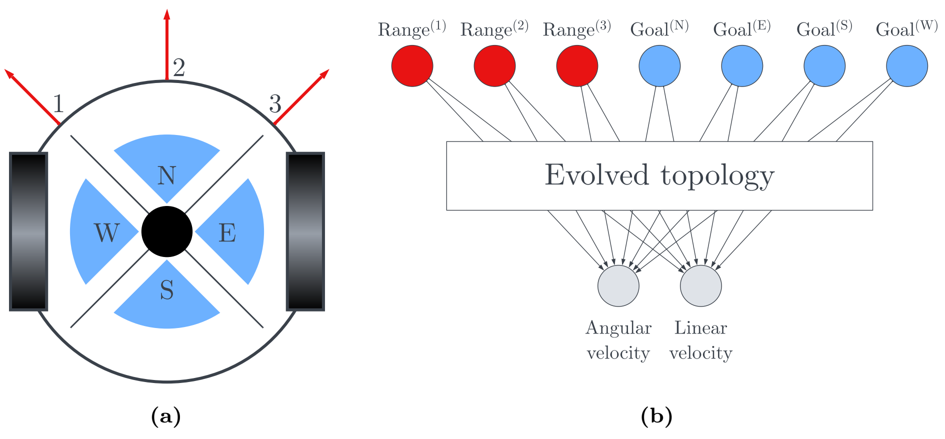


Figure 4.7: **(a)** Simulated mobile robot. The numbered red arrows are the range sensors (e.g. laser range finder) of the robot that return the normalized distances to the nearest obstacles in these directions. The four pie-slice goal sensor in blue indicate in which direction the goal is from the perspective of the robot (i.e. a compass). **(b)** The neural network setup for the mobile robot. Outputs of the neural network set the angular and linear velocity of the robot. The inputs are three range sensor distances (red) and the four pie-slice goal sensor (blue). Figure **(a)** adapted from [7].

Neural network details. The neural networks are initialized as fully-connected with 7 input neurons and 2 output neurons (see figure 4.7b). The input values for the range sensors are normalized to between $[0, 1]$. For the goal sensor, only one of the inputs will be active (i.e. input of 1.0) and the rest inactive (i.e. input of 0.0) at all times. The evolved neural network outputs a value for the angular and linear velocity of the robot as control signals. The velocities are used to update the position and heading of the robot at each time step t for a total of T time steps. Control signals should be in the range $[-0.5, 0.5]$. Therefore, neurons use the *sigmoid* activation function with range $[0, 1]$, and outputs from the output neurons are subtracted by 0.5.

Performance. The performance F of a neural network is evaluated by the euclidean distance from the robot to the goal at the end of the simulation. The performance is normalized to be in the range $[0, 1]$. A value of 1 means that the robot reached the goal:

$$F(x) = \frac{d_0 - d_T}{d_0} \tag{4.8}$$

where d_0 is the initial distance between the robot and the goal at its starting position, and d_T is the distance to the goal at the end of the simulation after time T.

Ad hoc distance. The *ad hoc* behavioral diversity of an individual is defined as the average distance from its end location to the end location of its

k-nearest neighbors (i.e. sparseness). The same *novelty* metric was described in section 2.6.3 about Novelty Search [5] where an archive of previously novel individuals is utilized. The distance between two individuals, x and y, is defined as follows:

$$d_{\text{ad-hoc}}(x, y) = \|p_x - p_y\| \tag{4.9}$$

where p_x and p_y is the end location of individual x and y in the maze at the end evaluation. The distance metric is the euclidean distance between their end locations.

Success criterion. An individual is successful if the goal is reached by the robot by the end of the simulation. The evolutionary search is stopped because the maximum score is achieved.

Number of inputs. 7: 3 range sensors and 4 inputs for the goal sensor.

Number of outputs. 2: 1 for the angular velocity of the robot and 1 for the linear velocity of the agent.

4.4 Robot Locomotion problem

The Robot Locomotion problem is, as the name implies, a *locomotion* problem where a robot has to transport itself from one location to another. Wheeled robots are usually very efficient and easy to control, but another form of locomotion may be more appropriate if the robot has to traverse rough terrain or interact with and in a human environment. This type of problem has previously been targeted in neuroevolution studies [4, 5, 9, 21] with various robot configurations (e.g. bipedal robot or hexapod).

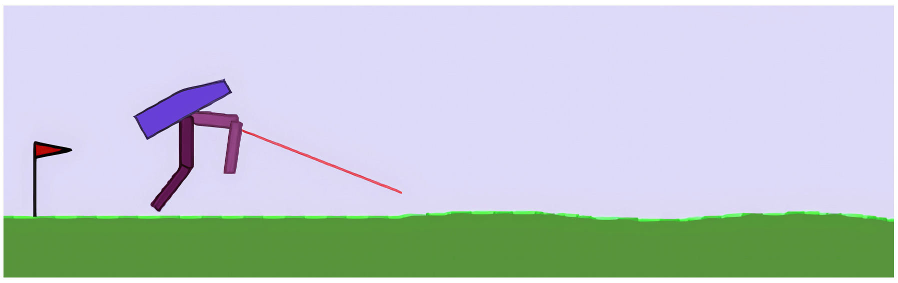
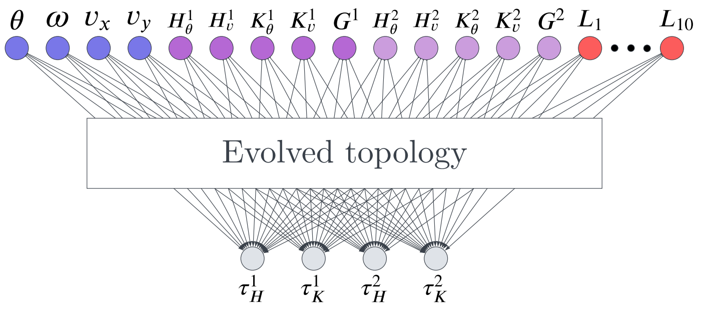


Figure 4.8: Simulated bipedal walker for the Robot locomotion problem. The 2D environment is generally flat and the goal is for the bipedal walker to reach the end of the environment (not pictured) within a time limit.

In this work, a `Bipedal-Walker` is simulated (see figure 4.8). The *Gymnasium*[4] library, originally developed by OpenAI, is used as the simulation framework for this problem. More specifically, the *BipedalWalker-v3*[5] environment is used. The bipedal walker has a hull (or head/body) with a constant size and weight, and two legs with two segments each of constant length. A hip joint connects each leg to the hull and a knee joint connects the two segments of each leg. The bipedal walker utilizes lidar rangefinder measurements to observe its environment and also has various information about the current angles and velocities of its hull and two legs. The objective is for the bipedal walker to reach the end of the environment within a set time limit without falling.

The task is difficult because the neural network must learn a gait that is efficient and fast enough to reach the end within the time limit. Learning such a gait requires coordination of the two legs, balancing the hull, and discovering an *oscillatory* pattern [5]. Similar to how human babies first learn to crawl before learning to walk, the initial population of neural networks tends to perform badly. However, encouraging diversity to discover useful "stepping stones" (e.g. crawling) can be beneficial for learning a *natural* gait. The problem is slightly deceptive as there is a local optimum of lunging forward at the start. Although, this local optimum will eventually be surpassed by neural networks that learn slow and inefficient gaits able to move beyond this distance in the environment.



θ ω v_x v_y H_θ^1 H_v^1 K_θ^1 K_v^1 G^1 H_θ^2 H_v^2 K_θ^2 K_v^2 G^2 L_1 $\cdots$ L_{10}
Evolved topology
τ_H^1 τ_K^1 τ_H^2 τ_K^2

(a)

θ	Hull angle
ω	Hull angular velocity
v_x	Hull velocity in x-direction
v_y	Hull velocity in y-direction
H_θ^i	Hip joint angle (leg i)
H_v^i	Hip joint velocity (leg i)
K_θ^i	Knee joint angle (leg i)
K_v^i	Knee joint velocity (leg i)
G^i	Ground contact flag (leg i)
L_j	Lidar rangefinder j
τ_H^i	Torque on hip (leg i)
τ_K^i	Torque on knee (leg i)

(b)

Figure 4.9: (a) The neural network setup for the simulated bipedal walker. The inputs are the various angles and velocities for the hull and the hips and knees of the legs. In addition to these, inputs also include ground contact flags to indicate when a leg touches the ground and the laser rangefinder measurements. The outputs are the torque (or motor speed) values applied to the hips and knees of each leg. **(b)** Table of symbols representing the various inputs and outputs in **(a)** with a descriptive explanation.

[4]https://gymnasium.farama.org
[5]https://gymnasium.farama.org/environments/box2d/bipedal_walker/

Neural network details. The neural networks are initialized as fully connected with 24 input neurons and 4 output neurons (see figure 4.9a). The evolved neural network outputs torque (or motor speed) values that are applied to the hip and knee joints of each leg. The torque values should be in the range $[-1, 1]$. Neurons use the *tanh* function as the activation function to ensure values in this range. Some studies [4, 9] use a *compositional pattern producing network* (CPPN) encoding. However, this thesis uses a direct neural network encoding as is used in [5, 21] to enable a similar setup across all problem domains.

Performance. The performance of a neural network is evaluated by the number of points accumulated. Points are accumulated by the bipedal robot during simulation based on the distance walked. A robot reaching the end of the environment will have accumulated 300+ points. The robot is penalized with -100 points if it falls. Applying torque to the motors of the joints costs a small amount of points. Thus, a more efficient neural network controller will achieve a better score than an inefficient one.

Ad hoc distance. The behavior of an individual is defined as the offset of the bipedal robot's center of mass sampled during the simulation. The offset of the center of mass is calculated as follows:

$$x'_k = \mathrm{sign}(x_k - x_0) \cdot (x_k - x_0)^2 \tag{4.10}$$

$$y'_k = \mathrm{sign}(y_k - y_0) \cdot (y_k - y_0)^2 \tag{4.11}$$

where (x_0, y_0) is the initial center of mass of the bipedal walker and (x_k, y_k) is the center of gravity of the biped walker at the kth sample during the simulation. The behavior vector β_i of individual i is defined by concatenating all pairs of (x'_k, y'_k) for m samples:

$$\beta_i = \{(x'_1, y'_1, \ldots, x'_m, y'_m)\} \tag{4.12}$$

The *ad hoc* behavioral distance between two individuals, x and y, is calculated in the same way as in the Maze Navigation problem in section 4.3, but using the euclidean distance between behavior vectors for the k nearest neighbors:

$$d_{\text{ad-hoc}}(x, y) = \|\beta_x - \beta_y\| \tag{4.13}$$

This way of measuring novel behaviors rewards individuals with unique gaits compared to others, and most importantly, is ignorant of the performance on the task. A slow and stable crawling gait is rewarded just as much as a fast and unstable jumping gait. Defining the behavior of an individual for the robot locomotion problem varies between studies depending on the robot configuration and other implementation details. The behavior definition and behavioral distance used in this work are the same as in [5, 21].

Success criterion. An individual is successful if a score of 300+ points is reached. The points are accumulated over the course of the simulation, and reaching 300+ means the robot managed to reach the end without falling. The evolutionary search is stopped.

Number of inputs. 24: 2 for the hull angle and hull angular velocity, 2 for the horizontal and vertical velocity, 4 for the angles and velocities of both hip joints, 4 for the angles and velocities of both knees, 2 for the ground contact flags of each leg, and 10 lidar rangefinder measurements.

Number of outputs. 4: motor speed for the 4 joints at both hips and knees.

4.5 Characterizing the targeted problems

A formal characterization of commonly targeted problems is lacking in previous research. As was stated in Chapter 3, this thesis attempts to characterize the targeted problems with the purpose of potentially gaining more valuable insight into the various diversity objectives. In particular, if some diversity objectives are more suitable for problems with certain characteristics than others.

The problems are classified according to which degree it expresses the characteristics: *modularity*, *regularity*, and *deceptiveness.* In addition, problems are characterized by their environment space: *discrete* or *continuous.* A summary is given in table 4.1 with every targeted problem. Subsequent sections outline the justification for this characterization. In situations where multiple versions of the same problem are characterized differently, the versions are separated and vice versa if characterized the same (e.g. `Tartarus` and `Retina` respectively). It should be noted that the characterization performed in this work is subjective.

Table 4.1: Summary of the characteristics of the targeted problems. The problems are characterized according to which degree it expresses the traits: modularity, regularity, and deceptiveness. *None* means that the problem does not exhibit the trait at all. *Some* indicates that the problem somewhat exhibits the trait. Problems with a trait characterized as *A lot*, exhibit the trait to a high degree. They are also categorized by whether the environment space for the problem is *discrete* or *continuous.*

Targeted problem	**Modularity**			**Regularity**			**Deceptiveness**			**Environment**	
	None	Some	A lot	None	Some	A lot	None	Some	A lot	Discrete	Continuous
Retina (2x2 and 3x3)			✓	✓			✓			✓	
Tartarus	✓				✓			✓		✓	
Deceptive-Tartarus	✓				✓				✓	✓	
Medium-Maze	✓			✓				✓			✓
Hard-Maze	✓			✓					✓		✓
Bipedal-Walker		✓				✓		✓			✓

4.5.1 Characteristics of the Retina problem

Modularity

As mentioned in section 4.1, the `Retina` problem is highly modular. The problem structure is modular because of the two independent parts of the retina and their respective target patterns. Such a modular structure can potentially be advantageous to networks that evolve into two modules, each learning their respective target patterns. However, a modular network is not required to perform well.

Regularity

The `Retina` problem does not exhibit any regularity. There is no obvious regular (or repeating) structure to the task that neural networks are required to learn. Examples of regularity include solving a sequence of sub-tasks or learning an oscillatory pattern.

Deceptiveness

The `Retina` problem is not deceptive. There are no obvious local optima that an individual can easily converge to. Guiding the evolutionary search using the performance is not misleading.

Environment space

The environment space of the `Retina` problem is discrete. The inputs are discrete with values of either 0 or 1. The outputs are discretized to indicate a classification of true or false. Furthermore, the activations of the neural networks will be more discrete due to using the tanh activation function with a steep slope parameter (see section 4.1).

4.5.2 Characteristics of the Tartarus problem

Modularity

The `Tartarus` problem is not modular. A modular structure does not appear obvious as the outputs are probabilities for the three actions and depend equally on all inputs. Perhaps one could argue that the two parts of the inputs, i.e. the surrounding cells and the one-hot encoded previous outputs, can potentially encourage evolving two modules. However, both the previous outputs and the surrounding cells are dependent on each other and will affect all the outputs equally.

Regularity

The `Tartarus` problem is somewhat regular in that the agent has to solve the same sub-task multiple times, i.e. push every block to the edge. The repeating task of pushing blocks indicate regularity to some degree, but the inputs and outputs do not exhibit any obvious regular pattern like is obvious in `Bipedal-Walker`.

Deceptiveness

The standard version of the `Tartarus` problem is somewhat deceptive as there is an obvious local optimum where the agent only learns to randomly push blocks to the edges and thus achieves a mediocre score of $4-6$ points instead of the maximum 10 points.

The change in how points are awarded, with negative points at edges but not in the corners, makes the `Deceptive-Tartarus` problem highly deceptive. A significant local optimum for the agent is to avoid pushing any block in order to avoid negative points and leave all blocks in the middle with a total score of 0 points. The local optimum is only overcome when the agent learns the behavior of pushing blocks to the corners, which is not trivial.

Environment space

The environment space of the `Tartarus` problem is discrete. The grid world is composed of cells where the blocks and agent positions are represented as discrete coordinates. The surrounding cells as inputs to the agent are discrete, including the one-hot encoded outputs of the previous step. The largest of the three output values determines which of the three possible actions the agent will take for the current step.

4.5.3 Characteristics of the Maze Navigation problem

Modularity

The maze navigation problem is not modular. There is no clear division of the problem structure that could benefit from a modular structure. An argument could be made for separate modules for each type of sensor, i.e. range sensors and goal radar. However, both will affect the output velocities equally and there is no obvious benefit from separating the two.

Regularity

The maze navigation problem does not exhibit any regularity. No repeating tasks are required to be solved or oscillatory patterns to be learned by the neural network. In the maze domain of `Medium-Maze`, it could potentially be beneficial to learn to weave between the local optima but this is not obvious nor required to solve the task.

Deceptiveness

The maze navigation problem was designed to be deceptive. The level of deceptiveness depends on the layout of the maze domain. In `Medium-Maze`, the local optima are placed on the path to the goal such that getting stuck in the next optimum will also be closer to the goal. Consequently, `Medium-Maze` is somewhat deceptive as it is possible for the robot to reach the goal by going from one local optimum to the next.

By contrast, the layout of the maze domain in `Hard-Maze` requires the robot to move away from the direction of the goal in order to reach it. In addition, there is a very attractive local optimum in the direction of the goal that is highly deceptive. This makes `Hard-Maze` highly deceptive.

Environment space

The environment space of the maze navigation problem is continuous. The mobile robot receives continuous distance values from the range sensors and outputs continuous velocity values that affect its position and heading in the environment. The environment in which the robot navigates is continuous, i.e. continuous position values.

4.5.4 Characteristics of the Robot Locomotion problem

Modularity

The robot locomotion problem is characterized as slightly modular. While a successful gait requires coordination between the legs, a module for each leg in the hidden layers could potentially evolve naturally and be beneficial. The inputs involving the hull and rangefinders should affect both legs, but an obvious modular decomposition would be for the inputs with angles and velocity of the joints of each leg. Still, some coordination is required and independently operated legs would likely not lead to a successful gait. Thus, `Bipedal-Walker` is only slightly modular.

Regularity

The robot locomotion problem exhibits a high degree of regularity. The most important aspect of a successful gait is the repeating or oscillatory pattern. The evolved `Bipedal-Walker` needs to learn a regularity of this nature in order to traverse the entire length of the environment within the time limit.

Deceptiveness

The robot locomotion problem is somewhat deceptive. An obvious local optimum at the start of the simulation is for the bipedal robot to lunge as far as possible. However, this local optimum is relatively quickly overcome even by bipedal robots with slow and inefficient gaits.

Environment space

The environment space of `Bipedal-Walker` is continuous. All input values will be in a continuous range, and outputs set continuous torque values on the joints of each leg. The position (or center of mass) of the bipedal robot in the environment is continuous.

5 | Results and Analysis

Chapter 5 presents the results and the analysis of the following experiments: (1) The main experiments for each of the targeted problems, (2) Summary figures illustrating the overall performance and success rate of each treatment across all problems, and (3) The computational run-time per generation for each treatment.

Analysis of results for individual experiments is given in this chapter and not the overall implications of the results. The subsequent Chapter 6 discusses the results as a whole and their significance with regard to the research goals.

5.1 Main experiments

The results for the main experiments were obtained by simulating the targeted problems on *Fox* – a high-performance computing cluster provided by the University of Oslo. The `Retina` problems were simulated on a single core due to not being very computationally expensive. The rest of the problems were simulated with 10 cores per run, with several runs running in parallel.

5.1.1 How to interpret the results

Figures show the median best performance with bootstrapped 95% confidence interval of 50 runs. The significance bars at the bottom with markers indicate if there is a statistical significance in performance between the selected treatment on the left and the other treatments on the right per generation. No marker indicates no statistical difference at that generation. For every targeted problem, the figure presents the results with all treatments. In this figure, the best-performing treatment is selected as the treatment all statistical significance testing is compared against.

A significance bar is shown in figure 5.1 with an explanation of how they can be interpreted. The use of significance bars is not common in all fields of research. However, due to the stochastic nature of evolutionary algorithms, significance bars can inform how significant the results are and can help determine if the results have any statistical weight to them. Significance bars were also used in [4, 9, 73]. It is important to note that relying too much on statistical significance testing can be problematic. Statistical testing with p-values is not an objective measure and should be interpreted with caution, as there are situations where statistical significance can be discovered "randomly" but is in reality not the case.

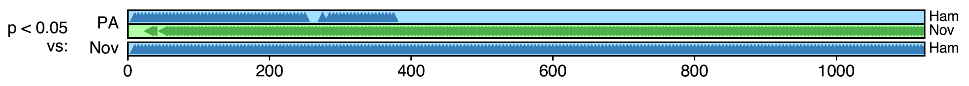


Figure 5.1: Illustration of significance bars. The left and right treatments are tested against each other using the Mann-Whitney U test, where a marker indicates a significant difference ($p < 0.05$) for that generation and no marker indicates no significance ($p \geq 0.05$). In this example, PA is compared against both diversity treatments (HAMMING and NOVELTY), and NOVELTY is compared against HAMMING. There is a significant difference for all generations between PA and NOVELTY, and between NOVELTY and HAMMING. As for PA vs HAMMING, the difference is only significant for the first 400 generations or so.

In addition to the figure with all treatments, figures showing only the treatments with objectives of the same type of diversity (behavioral, structural, and representational) are included in appendix B. The purpose is to more clearly see the difference between them. These figures also include the control experiment (PA) with statistical testing of diversity treatments compared against PA, and an additional significance bar for statistical testing between the diversity treatments (see figure 5.1).

5.1.2 2x2–Retina

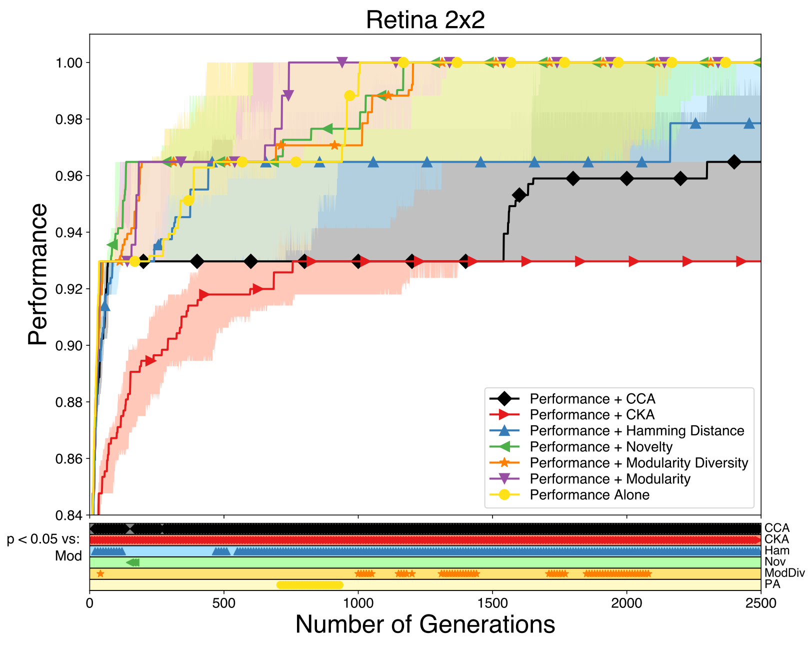


Figure 5.2: The performance of all experimental treatments on the 2x2-Retina problem. The best performance score is reached fastest by the Mod, ModDiv, PA, and Novelty treatments.

The results for the `2x2-Retina` problem are shown in figure 5.2. From studying the results, it becomes apparent that the treatments Mod, ModDiv, PA, and Novelty leads to the best performance. Between them, there is no significant difference in performance except perhaps Mod being slightly more efficient but the difference is not definitive. Of the behavioral diversity treatments, Novelty performed better than Hamming (see figure B.1). Whereas for the structural diversity treatments, no significant difference between Mod and ModDiv was evident (see figure B.2). As for the representational diversity treatments, CCA performed significantly better than CKA (see figure B.3).

The results in figure 5.2 are in line with the results in previous work [4], although with some differences. As in [4], the ModDiv and Mod treatments were both able to achieve the maximum performance score. Interestingly, in [4], the Mod treatment was significantly slower to converge, and PA was not able to fully reach a maximum performance score. One hypothesis for the difference in the performance of these treatments is due to the slightly different setups as was briefly mentioned in 4.1. With discrete weights, the problem is more difficult to solve for PA, and the structure and connectivity of the networks become more important. Whereas Mod potentially benefits from no restrictions on the number of layers and neurons per layer.

5.1.3 3x3–Retina

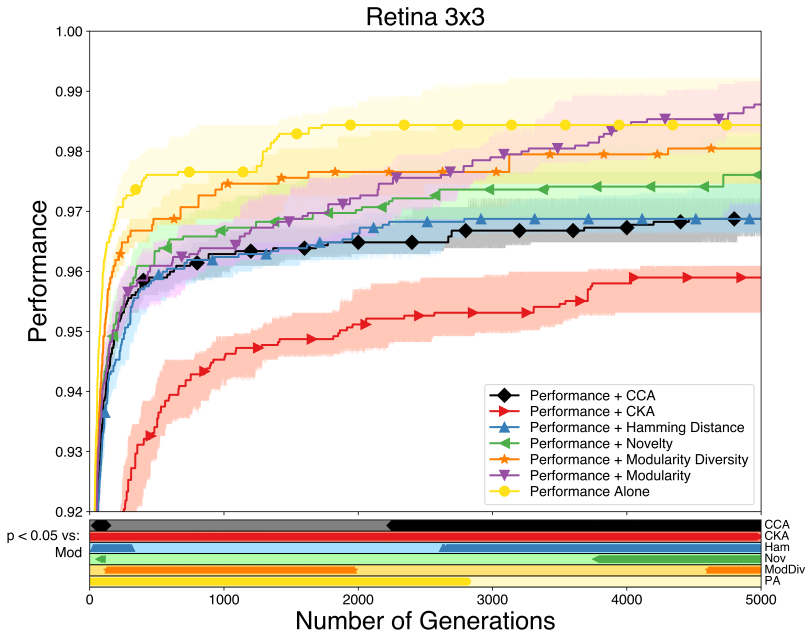


Figure 5.3: The performance of all experimental treatments on 3x3-Retina. The Mod, ModDiv, and PA treatments lead to the best performance with Mod initially slower but overtaking both towards the end.

The results for the harder `3x3-Retina` problem are shown in figure 5.3. Studying the results, the same trends as for the `2x2-Retina` problem can be seen but the problem has become harder to solve as was intended. The treatments Mod, ModDiv, and PA lead to the best performance. Interestingly, PA and ModDiv are initially the most efficient, whereas Mod is slower but overtakes the other two towards the end. The Mod treatment may have potentially reached the maximum performance score of 1.0 if it had been allowed to run for more generations.

The Novelty treatment is slightly worse in this harder version of the problem as there was a significant performance difference between PA and the behavioral diversity treatments (see figure B.4). As for the structural diversity treatments, results indicate that Mod is slower than both ModDiv and PA for the first 2500 generations or so, but is able to overtake both towards the end (see figure B.5). The results for the representational diversity treatments show CKA and CCA as significantly worse than PA. Moreover, CCA consistently leads to better performance more efficiently compared to CKA (see figure B.6).

Overall, the results for the treatments on both the `2x2-Retina` problem and `3x3-Retina` problem match closely with very few differences. None of the treatments consistently achieved the maximum score in the `3x3-Retina` problem.

5.1.4 Tartarus

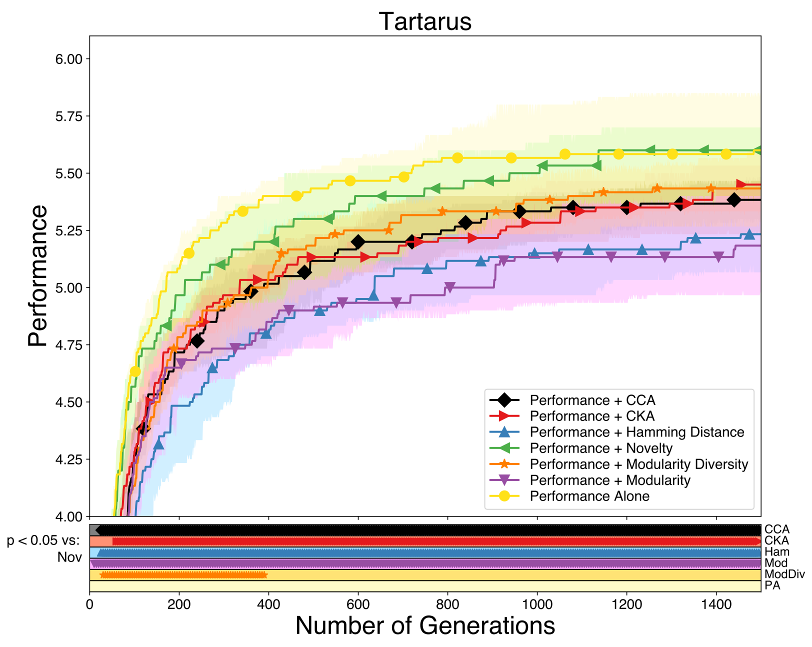


Figure 5.4: The performance of all experimental treatments on Tartarus. Novelty and PA treatments lead to the best performance. ModDiv and CKA look like close seconds, but PA significantly outperforms CKA but not ModDiv (see figures B.8 and B.9).

The results in figure 5.4 indicate that the Novelty and PA treatments lead to the best performance on the `Tartarus` problem, with ModDiv being the third best treatment. Although, none of the treatments were able to reach the maximum performance score of 10 and avoid the local optimum of between $4 - 6$ points. The worst-performing treatments were Hamming and Mod, whereas the remaining treatments performed slightly better but with no significant difference between them. Of the structural diversity treatments, ModDiv significantly outperformed Mod (see figure B.8). There was no significant difference between the representational diversity treatments (see figure B.9).

The results are somewhat in line with previous work [53], where treatments also tended to reach a performance of between 4.0 and 6.0 with one treatment reaching 7.0 (see NCD in [53]). However, a direct comparison between results is not possible as a recurrent neural network with a fixed topology was used in previous work. Nevertheless, the HAMMING treatment in [53] was able to reach a performance of 6.0 in one of the experiments, which may suggest that a recurrent neural network is a more suitable neural network architecture. Using the previous outputs as input, as is employed in this work, is potentially not a sophisticated enough type of memory element for the agent to learn the necessary sequence of actions to push blocks into the corners.

5.1.5 Deceptive–Tartarus

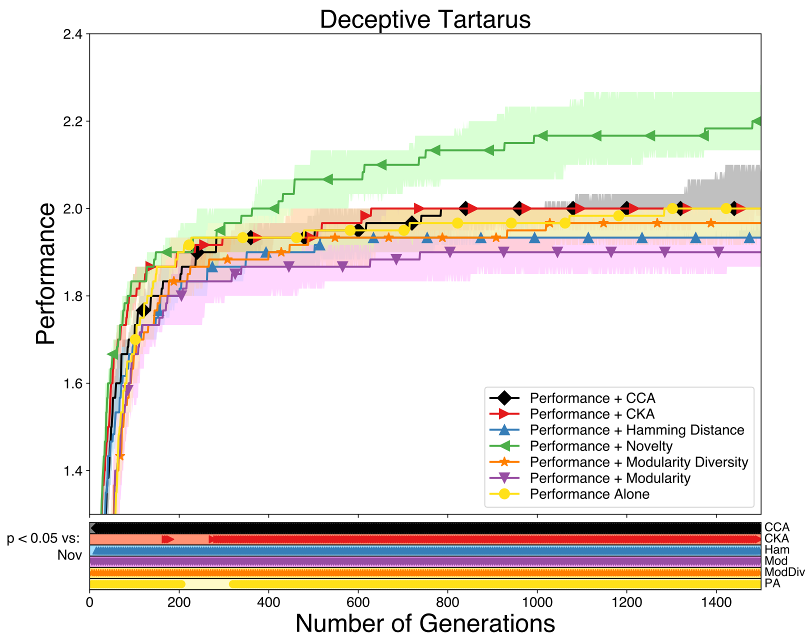


Figure 5.5: The performance of all experimental treatments on Deceptive-Tartarus. The NOVELTY treatment significantly outperforms all other treatments. All other treatments converge to a performance of around 2.0. None was able to reach maximum performance.

Looking at the results for the `Deceptive-Tartarus` problem in figure 5.5, it is apparent that the NOVELTY treatment performs significantly better than the rest of the treatments. Achieving a score of over 2.0 means that the agent is able to move more than a single box to the corners on average for all 30 board configurations (or alternatively more than one box in the corners but negative scores from boxes at edges). The remaining treatments converge to the local optimum with a performance of around 2.0. The structural and representational diversity treatments displayed no significant difference them respectively (see figures B.11 and B.12). The Novelty treatment significantly outperformed Hamming (see figure B.10).

The results for Novelty and PA are similar to the results in previous work [70], where a recurrent neural network with a fixed topology was used like in the `Tartarus` problem, and a linear combination of the performance and novelty of an individual was used to rank individuals instead of using separate objectives. Nevertheless, the same trend is apparent with Novelty significantly outperforming PA. The treatments in [70] were simulated for a larger number of generations and Novelty reached a performance of 3.0. While not conclusive, there are indications that Novelty could approach the same type of performance in figure 5.5 if allowed to run longer.

5.1.6 Medium–Maze

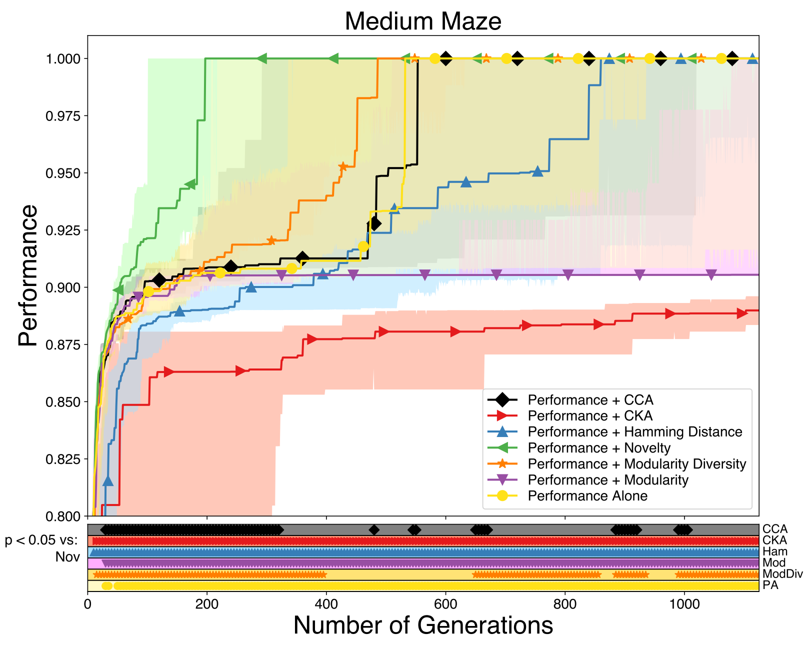


Figure 5.6: The performance of all experimental treatments on Medium-Maze. Novelty significantly outperforms all other treatments, with some uncertainty about the significance compared to CCA. The treatments ModDiv, PA, and Hamming also reach the goal within 1500 generations, albeit slower than Novelty.

As shown in the results for the `Medium-Maze` problem in figure 5.6, Novelty outperforms all other treatments. Although, there is some uncertainty about the significance compared to CCA. Overall, the Novelty treatment leads to the best performance most efficiently out of all treatments. The treatments ModDiv, CCA, Hamming, and PA are all able to consistently reach the goal of the maze and thus the maximum performance score within 1500 generations. Mod is only able to reach the goal a few times, whereas CKA never reaches the goal.

The results for the `Medium-Maze` problem with Novelty and PA are consistent with the results in a previous study [5]. It should be noted that the confidence interval is large for a few of the treatments, especially PA, where the many local optima can be seen in the sudden jumps in the performance of treatments. Interestingly, there is a significant difference between treatments of the same type of diversity. This is apparent for behavioral diversity with Novelty outperforming Hamming (see figure B.13). Likewise for structural diversity with ModDiv outperforming Mod and representational diversity with CCA outperforming CKA (see figures B.14 and B.15 respectively).

In addition to studying the performance, the areas of the maze explored by treatments were also investigated. However, no clear difference was seen between treatments except CKA exploring significantly less of the maze. A more clear difference can be seen for the `Hard-Maze` problem below. The figures for the `Medium-Maze` problem are included in appendix B.

5.1.7 Hard–Maze

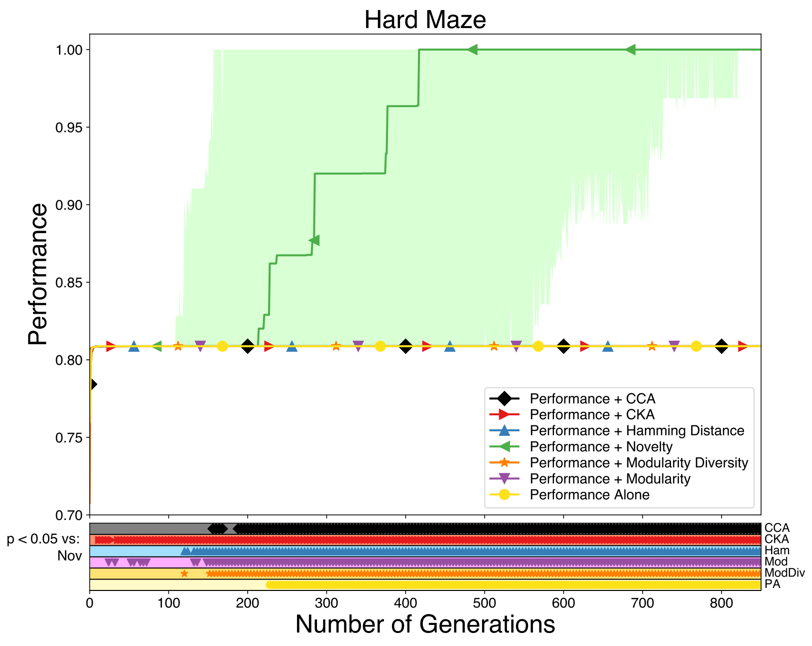


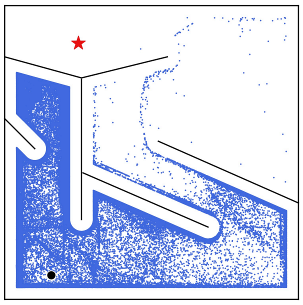
Figure 5.7: The performance of all experimental treatments on Hard-Maze. Novelty is the only treatment able to consistently reach the goal. All other treatments converge to the attractive local optimum.

Figure 5.7 shows the results for the treatments on the `Hard-Maze` problem. All treatments were simulated for 1500 generations but only 850 generations are shown as there was no change for the remaining generations. Only Novelty was able to consistently reach the goal, whereas the rest of the treatments were not able to reach the goal and all converged to the local optimum with a performance score of 0.81. Like in previous studies [5, 6], Novelty is consistently able to reach the goal while PA always converges to the local optimum when comparing Novelty and PA on the `Hard-Maze` problem.

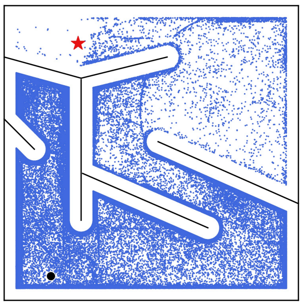
The `Hard-Maze` problem is a special case where there is a single attractive local optimum and treatments immediately converge to it because of the high degree of deceptiveness. For this reason, separate figures with each type of diversity are not shown. Instead, figures of the maze showing which areas each diversity objective explored are presented below.

Figure 5.8 shows the areas explored by the robot for each treatment over all 50 runs. Each dot represents the end position of a simulated individual. The end positions of individuals in the population are sampled every 10th generation.

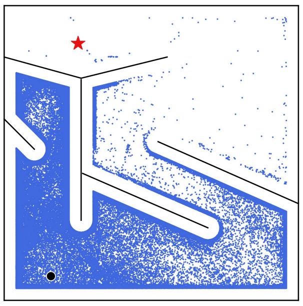
Most notable is the Novelty treatment compared to the rest. In particular, Novelty explores much more of the open area in the top right corner and also near the goal (i.e. red star). The results for Novelty are not surprising as individuals exploring new areas are rewarded with higher novelty. The Hamming treatment was able to reach the area near the goal but not at the same rate as Novelty (see figure 5.8c and 5.8b). Comparing the areas explored by the structural diversity objectives, there was no apparent advantage to either ModDiv or Mod (see figures 5.8e and 5.8d respectively). As for the representational diversity objectives, the CCA treatment explored significantly more of the area closer to the goal than the CKA treatment (see figures 5.8g and 5.8f respectively).



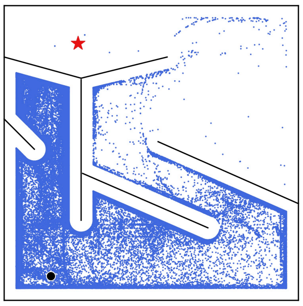
(a) PA



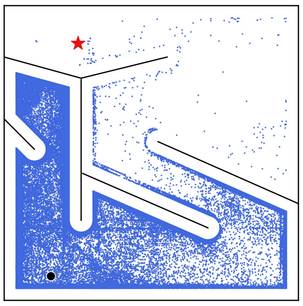
(b) Novelty



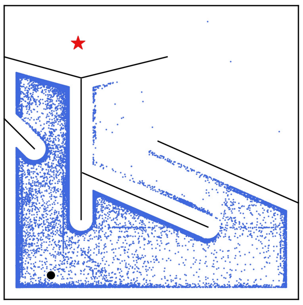
(c) Hamming



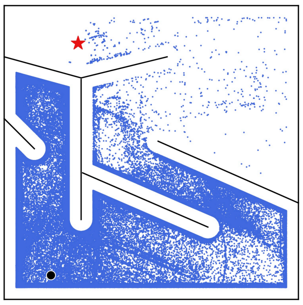
(d) Mod



(e) ModDiv



(f) CKA



(g) CCA

Figure 5.8: Illustration of how differently each treatment explored the maze. Each dot is the end position of a simulated robot. Most notable is Novelty with the most coverage.

5.1.8 Bipedal–Walker

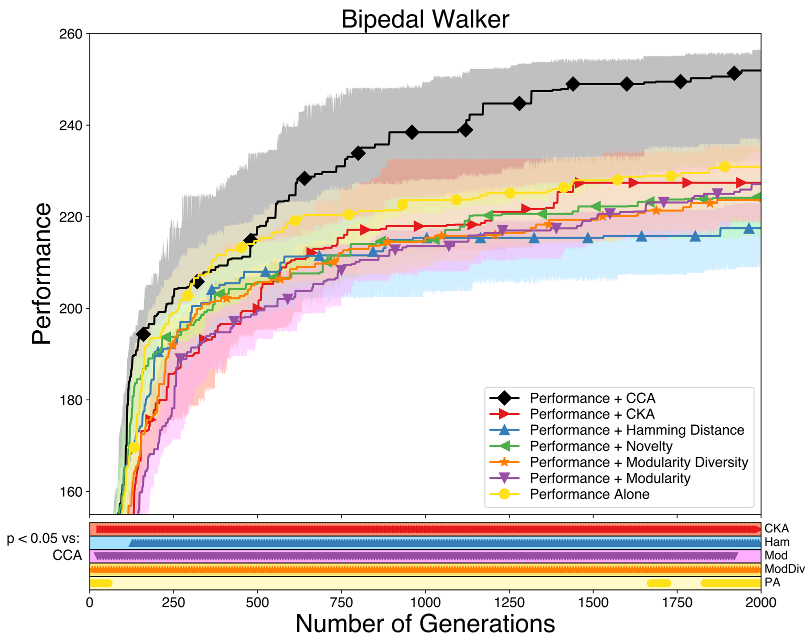


Figure 5.9: The performance of all experimental treatments on Bipedal-Walker. The CCA treatment significantly outperformed all other treatments except PA until around generation 1800. While the median performance for CCA is larger than PA, the 95% confidence interval is large and thus more uncertainty about the significance between them. None of the treatments were able to reach the maximum performance score.

Analyzing the results in figure 5.9 for the `Bipedal-Walker` problem, the CCA treatment significantly outperforms the rest of the treatments. There is more uncertainty about the performance difference between CCA and PA, where CCA leads to a much higher median performance score but with a large 95% confidence interval. At around generation 1800, the difference in performance becomes significant and is likely to continue if allowed to run for more generations. CKA performs on par with PA but significantly worse than CCA (see figure B.18).

There was no significant difference between the structural diversity treatments MOD and MODDIV (see figure B.17). PA is significantly more efficient than MOD and MODDIV for the first 1000 generations, but for subsequent generations, the difference in performance is not significant. Of the behavioral diversity treatments, NOVELTY performed only slightly better than HAMMING but not significantly. PA outperformed both behavioral diversity treatments but outperformed only HAMMING by a significant margin (see figure B.16).

Investigating if there is a correlation between the diversity objectives and the discovered gaits would be interesting, but is out of the scope of this thesis. Nevertheless, to understand what types of behaviors are discovered during the evolutionary search, the gait of a few of the best genomes for the various treatments is shown. Figures 5.10, 5.11, 5.12, and 5.13 below show examples of the most common type of gaits discovered.

A fairly normal gait was discovered where both legs are moved in coordination to achieve locomotion with neither leg as the more dominant one (see figure 5.10). However, in the aforementioned gait, neither leg was moved in front of the other and vice versa in an alternating way which is common in the human gait. This trend of keeping one leg in front of the other leg for the entire duration was common for all the gaits that were investigated.

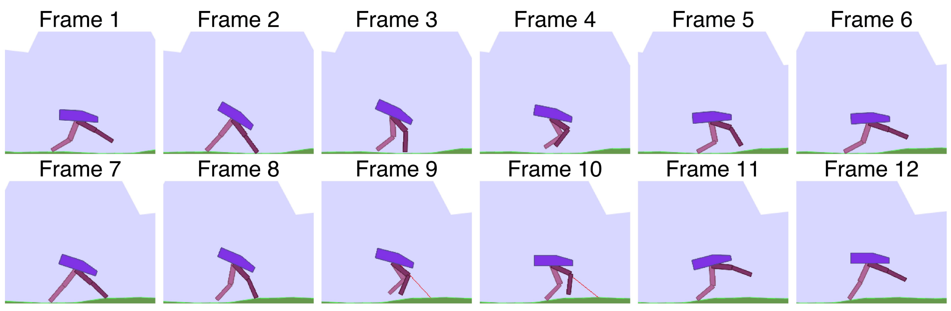


Figure 5.10: Normal gait.

Another type of gait, shown in figure 5.11, was discovered where the leg furthest back is used to *launch* itself forward in the air and the other leg furthest forward is used for support. By keeping the leg in front as straight as possible, the impact is dampened and more control is achieved to prevent falling forward when launching itself. A launching gait can be very fast but is more unstable.

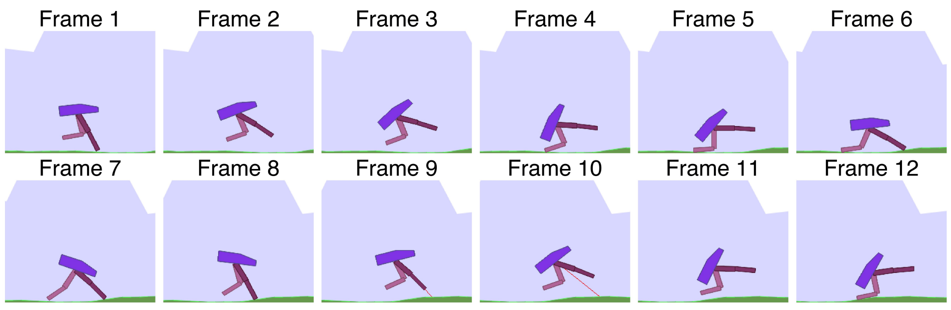


Figure 5.11: Launching gait.

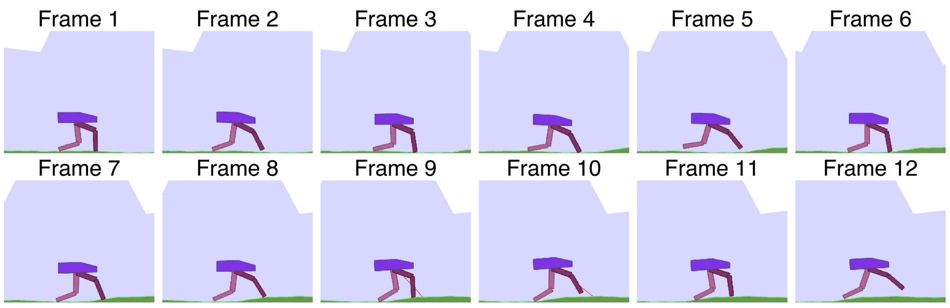


Figure 5.12: Crawling gait.

By contrast, the gait shown in figure 5.12 illustrates a *crawling* gait. With this type of gait, the bipedal walker is in a crouched configuration and achieves locomotion with limited leg movement. The bipedal walker alternates between applying a small force to the joints of each leg, but the change in the angle of each joint is minimal. This type of gait is very stable but significantly slower as there is less forward movement.

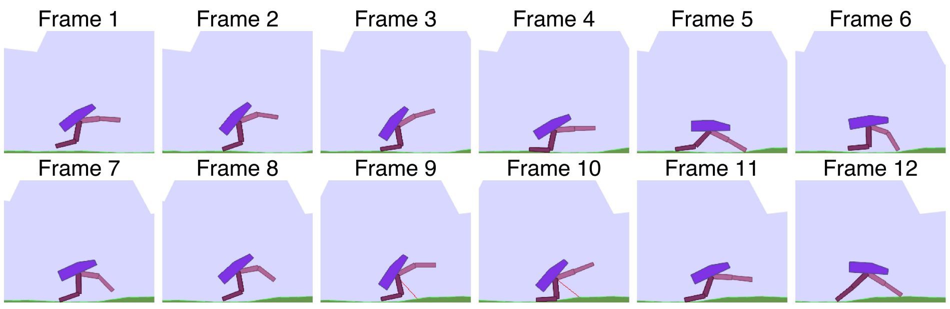


Figure 5.13: Dragging gait.

Lastly, a *dragging* gait was discovered as illustrated in figure 5.13. With a gait of this type, the leg furthest forward is used to drag the bipedal walker forward and the other leg is used for support and stability. Usually, the supporting leg has a sharp angle at the knee joint for more support. Like the crawling gait, the dragging gait is very stable but slow.

5.2 Summary of performance and success rate

Figure 5.14 and 5.15 below show the median best performance score and the success rate respectively for all treatments across all targeted problems. These figures are intended to be a visual illustration showing differences between treatments but they do not indicate any significance between treatments.

A notable observation for the median best performance in figure 5.14 is that NOVELTY generally performs well across all problems with the exception of the `Bipedal-Walker` problem. Furthermore, MOD performs well on the `2x2-Retina` and `3x3-Retina` problems but performs much worse on the other targeted problems. CCA resulted in much higher performance compared to other treatments on the `Bipedal-Walker` problem. CKA appears to perform poorly or at least resulted in mediocre performance on all problems. Likewise, the HAMMING treatment results in sub-par performance except on the `Medium-Maze` problem. Overall, the control treatment PA performs well and is never the worst treatment.

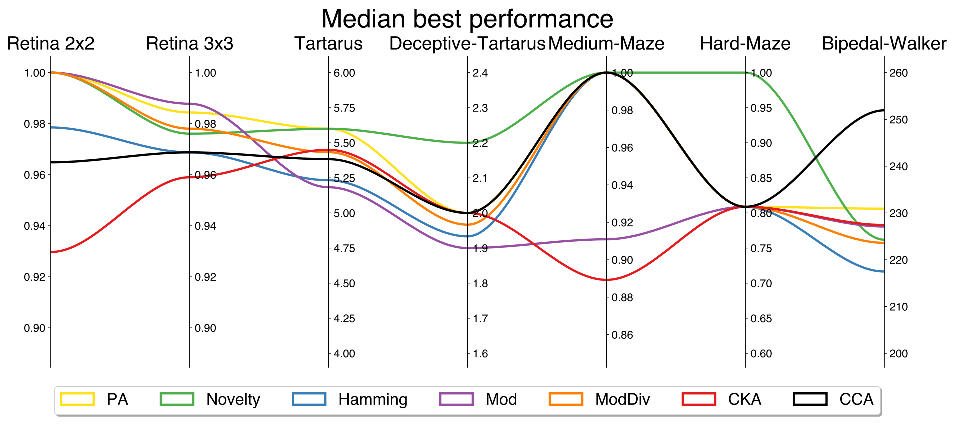


Figure 5.14: Median best performance for 50 runs of treatments across all targeted problems. The axis for each problem is scaled to show the differences between treatments if there is one, and does not always show the maximum possible performance score.

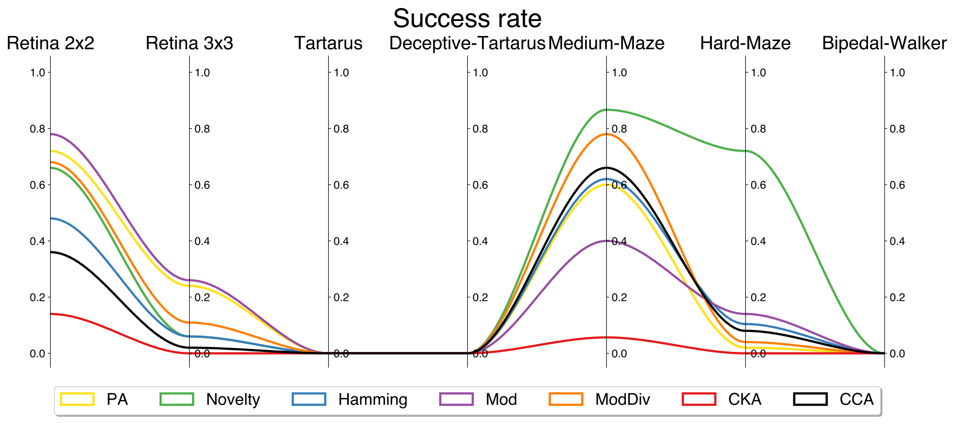


Figure 5.15: Success rate of treatments across all targeted problems. The success rate is the fraction of runs where the success criterion was reached, i.e. the maximum performance score. None of the treatments were able to reach the success criterion for the Bipedal-Walker, Tartarus, and Deceptive-Tartarus problems.

Looking at the success rates in figure 5.15, none of the treatments were able to reach the success criterion for the `Tartarus`, `Deceptive-Tartarus`, and `Bipedal-Walker` problems. PA and the structural diversity treatments ModDiv and Mod has a higher success rate on both retina problems, whereas Novelty has a much higher success rate on the maze navigation problems. The CKA treatment appears to consistently have the worst success rate across all targeted problems.

5.3 Computational run-time overhead experiment

Figures 5.16, 5.17, 5.18, and 5.19 below show the computational run-time per generation for each of the treatments for the targeted problems. Times are in seconds. Figures for multiple versions of the same problem (e.g. `2x2-Retina` and `3x3-Retina`) are not included here in the analysis as there was practically no difference in run-time ratio between treatments. The figures for the remaining versions, i.e. `2x2-Retina`, `Medium-Maze`, and `Deceptive-Tartarus`, are included in appendix B.

One note of caution is that the run-time per generation for treatments is not comparable across problems since they are simulated with a different number of cores. However, the diversity computation is not performed in parallel meaning the ratio between treatments on different problems is still useful to study.

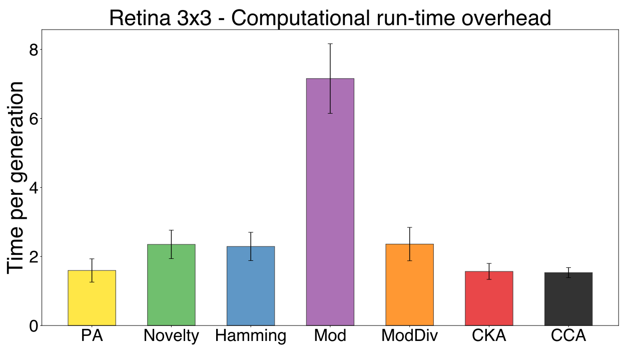


Figure 5.16: The mean run-time per generation on 3x3-Retina. Time units are in seconds with standard deviation also shown. Simulated using a single core. Mod required significantly more time per generation compared to other treatments, as it is more computationally intensive for larger neural networks.

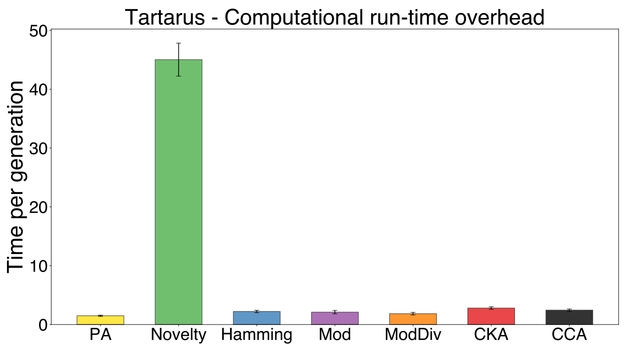


Figure 5.17: The mean run-time per generation on Tartarus. Time units are in seconds with standard deviation also shown. Simulated using 20 cores. Computing NOVELTY is significantly slower than other treatments. There was no significant difference in run-time between the other treatments.

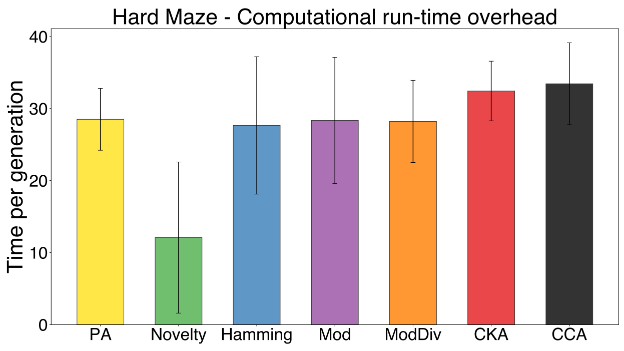


Figure 5.18: The mean run-time per generation on Hard-Maze. Time units are in seconds with standard deviation also shown. Simulated using 10 cores. NOVELTY is shown as significantly faster compared to the other treatments, but this is likely due to reaching the goal very quickly for many of the runs as is evidenced by the large standard deviation.

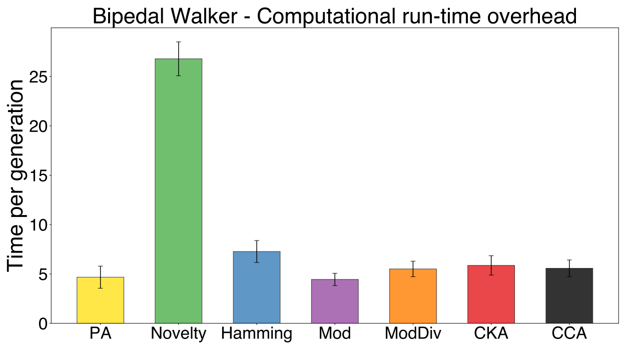


Figure 5.19: The mean run-time per generation on Bipedal-Walker. Time units are in seconds with standard deviation also shown. Simulated using 10 cores. NOVELTY is significantly slower than other treatments. No significant difference in run-time between the other treatments.

Looking at the run-time per generation for the `Retina` problem in figure 5.16, the Mod treatments were significantly slower than the rest of the treatments. A likely reason for this is that maximizing the performance and modularity for the `Retina` problem are not very conflicting objectives, and thus, neural networks can continually grow larger without any performance trade-off. A problem encountered during simulation was that neural networks would grow indefinitely when using Mod on both versions of the `Retina` problem and a limit of 50 hidden neurons had to be set for the networks. This problem was not encountered for the other targeted problems, further supporting the hypothesis that performance and modularity are not particularly conflicting objectives for the `Retina` problem.

The results for the `Tartarus` problem and the `Bipedal-Walker` problem in figure 5.17 and 5.19 respectively, show the Novelty treatment as significantly slower than other treatments for both problems. Novelty being much more computationally intensive on these problems is not unexpected as the novelty score calculation involves computing the distance between large behavior vectors sampled during simulation. In addition, the run-time per generation will exponentially increase as the archive grows. Therefore, a maximum archive size of 500 was used for the `Tartarus` problem and the `Bipedal-Walker` problem. The difference in time per generation between Novelty and the other treatments would likely be larger if an archive of unlimited size was employed.

The run-time per generation for the `Hard-Maze` problem in figure 5.18 shows that the representational diversity treatments (CKA and CCA) are slightly slower compared to the other treatments. Novelty appears to be significantly faster than other treatments, but this result should be interpreted with caution. A likely reason for Novelty being fast is because it reaches the goal very quickly for many of the runs, as is evidenced by the large standard deviation. Nevertheless, the behavior vector for the maze navigation problems is significantly smaller than for the `Tartarus` and `Bipedal-Walker` problems. Accordingly, the novelty score calculation is therefore much less computationally intensive.

6 | Discussion

Chapter 6 discusses the key implications of the results in this work with respect to the research goals of this thesis. Possible explanations and hypotheses for the findings are discussed, including references to similar or contradicting findings of previous studies. A short summary is given of the key insights discussed in each section.

6.1 Differences between objectives of the same type of diversity

A clear trend from the results in Chapter 5 is that there are significant differences between diversity objectives of the same type (i.e. behavioral diversity, structural diversity, and representational diversity).

6.1.1 Ad hoc behavioral diversity outperforms generic behavioral diversity

NOVELTY outperformed HAMMING across almost all targeted problems. These results broadly support the findings of previous studies where *ad hoc* behavioral diversity generally resulted in better performance compared to generic behavioral diversity [7]. It is not unexpected that a domain-dependent behavioral distance specifically defined for the domain is better than a generic behavioral distance. There is a clear performance trade-off for generic behavioral distances over *ad hoc* behavioral distances. As was briefly mentioned in section 2.7.3, it can be argued that valuable information is lost when binarizing the input-output history with HAMMING.

Despite the performance difference, the computational run-time overhead between the NOVELTY and HAMMING is something that has to be considered. The results in section 5.3 show that NOVELTY is computationally expensive for very large behavior vectors compared to HAMMING. This is evident for the `Tartarus` problems and the `Bipedal-Walker` problem, whereas the behavior vectors for the remaining problems are significantly smaller.

Reducing the size of the behavior vectors can somewhat mitigate this by using fewer samples m during simulation for the `Bipedal-Walker` problem and considering only a subset of the k board configurations in the `Tartarus` problems. However, reducing the amount of information contained in the behavior vectors may result in a slight performance trade-off. Alternatively, the distance to the nearest neighbors can be computed in parallel.

Somewhat surprising is the performance of HAMMING on the `Tartarus` problem compared to previous work [53], where HAMMING was able to achieve a performance of up to 7.0 instead of 5.5 in this work (see figure B.7). Additionally, in a more continuous environment in [55], it was suggested that HAMMING may perform better if the input and output values are already somewhat discrete as is the case for the `Tartarus` problem. A possible reason for the difference in the performance of HAMMING could be the use of a different memory element that is important to solve the problem (i.e. previous action as input instead of using a recurrent neural network).

6.1.2 Mixed performance for structural diversity with a slight edge to Modularity Diversity

As with the behavioral diversity treatments, the same trend can be seen for the structural diversity treatments of MODDIV and MOD for a few of the problems. MODDIV and MOD performed similarly on the `Bipedal-Walker` problem and both `Retina` problems, but MODDIV significantly outperformed MOD on the `Medium-Maze` problem and both `Tartarus` problems. One possible reason for the performance difference is that the modular decomposition of the input neurons is more important for solving the latter problems, while the modular decomposition of the network as a whole is as important for the former problems. The `Hard-Maze` problem is a special case where the high level of deceptiveness makes interpreting the differences between treatments difficult.

Additionally, MODDIV was found to be significantly more efficient for the first generations but MOD often catches up towards the end (see `Retina` and `Bipedal-Walker` problems). An explanation for this may be that a diversity of the modular decomposition of the input neurons makes it easier to discover high-performing individuals early in the search, but is limited in the amount of diversity that can be encouraged for only the input neurons. Whereas encouraging a diversity of the modular decomposition of the network as a whole prevents early convergence and can facilitate the continual discovery of new better-performing individuals. Although this is not always the case for MOD, as evidenced by the results for the `Medium-Maze` problem and both `Tartarus` problems showing no improvement for many generations.

6.1.3 Significant differences between the similarity metrics of representational diversity

Similarly, there are indications of a significant difference between the representational diversity treatments on a few of the targeted problems. The CCA treatment consistently outperforms CKA in terms of both performance and efficiency. The difference between CKA and CCA is especially noticeable on the less deceptive problems with few or no attractive local optima (i.e. the `Bipedal-Walker`, `Medium-Maze`, and both `Retina` problems). The representational diversity treatments and possible explanations for differences between them are discussed in more detail in section 6.3 below.

6.1.4 Summary

The main differences between diversity objectives of the same type of diversity can be summarized as follows:

- Not surprisingly, the *ad hoc* behavioral diversity objective outperformed the generic behavioral diversity objective on almost all targeted problems. However, the performance advantage of NOVELTY should be weighed against the high computational run-time overhead on certain problems (i.e. `Tartarus` and `Bipedal-Walker`).
- The structural diversity treatments MODDIV and MOD performed similarly, but with a slight performance edge to MODDIV on a few of the problems (i.e. `Medium-Maze` and `Bipedal-Walker`). MODDIV was found to be more efficient early in the search but converged earlier, whereas MOD was slower but did not converge as easily.
- CKA consistently outperformed CCA in terms of both performance and efficiency, especially on less deceptive problems with few or no local optima.

6.2 Relationship between the performance of diversity objectives and problem characteristics

The primary research goal of this thesis was to compare diversity objectives and determine if specific diversity objectives are more effective on certain problems. With regard to this research goal, there is a clear correlation between the performance of diversity objectives and problem characteristics in the results of Chapter 5.

6.2.1 Behavioral diversity and deceptiveness

The most striking result is the correlation between behavioral diversity and the deceptiveness of the problem. More specifically, the NOVELTY treatment performed significantly better on problems exhibiting a high degree of deceptiveness such as the `Taratrus`, `Deceptive-Tartarus`, `Medium-Maze`, and `Hard-Maze` problems. Recalling the characterization of targeted problems in table 4.1, the aforementioned problems were all characterized with a high degree of deceptiveness. The only exception is the `Bipedal-Walker` problem which is characterized by some deceptiveness. However, the deceptive local optimum in this problem is easily overcome when any gait is discovered, effective or not.

This correlation between performance and problem characteristic suggest that behavioral diversity, particularly *ad hoc* behavioral diversity, is especially suitable for deceptive problems. For very deceptive problems, domain knowledge may be required to solve the task, in contrast to domain-independent diversity objectives of the same or a different type that do not encourage the necessary diversity required to solve the task. The results for Novelty are not surprising as it has been shown to perform well on deceptive tasks in previous studies [5–7, 20], but there was a lack of knowledge about how other types of diversity objectives performed on deceptive problems such as the ones targeted in this work.

6.2.2 Structural diversity and modularity

Moreover, there appears to be a correlation between modularity and structural diversity objectives. Both structural diversity treatments, ModDiv and Mod, perform well on problems exhibiting a high degree of modularity, namely the `2x2-Retina` and `3x3-Retina` problem. This is also consistent with the hypothesis, proposed earlier when analyzing the run-time per generation, that performance and modularity are not very conflicting objectives for these problems. This is further evidenced by how the structure of neural networks in the population was able to continually grow without any performance trade-off for the aforementioned problems but not for other problems. The results for structural diversity are consistent with the findings of [4], but this work has also produced results showing how other types of diversity do not perform as well as structural diversity.

The structural diversity objectives do not perform as well on the `Bipedal-Walker` problem characterized by some modularity. However, as was mentioned previously in section 4.5, a modular decomposition is arguably less important to be successful on this problem as coordination between the legs is required and the oscillatory pattern is more essential to produce an effective gait.

6.2.3 Representational diversity and regularity

There are some indications of a correlation between representational diversity and regularity as a problem characteristic. CCA in particular performed very well on the `Bipedal-Walker` problem characterized by a high degree of regularity. One explanation for this is that representational diversity objectives are, by encouraging novel neuron activations, able to produce novel oscillatory patterns and by extension better gaits. However, the CKA treatment performed significantly worse than CCA on this problem. Possible reasons for this are discussed in section 6.3.

Despite the success on the `Bipedal-Walker` problem, the representational diversity objectives did not perform as well on the other problems characterized by some regularity (i.e. `Tartarus` and `Deceptive-Tartarus`). This discrepancy could be attributed to how the activations of the hidden neurons are not as informative and important for solving the aforementioned problems, and thus, encouraging this type of diversity does not help with the evolutionary search. Another possible reason is that characterizing the `Tartarus` problem as exhibiting regularity is not accurate when compared to the `Bipedal-Walker` problem. A regularity of oscillatory patterns can perhaps not be equated to a regularity of sub-tasks. In the future, it could be necessary to distinguish between oscillatory regularity and sub-task regularity.

6.2.4 No clear relationship between the diversity and the environment space representation

As for the environment space characteristic, no clear correlation can be deduced as the different types of diversity objectives perform well on both discrete and continuous problems. This suggests that the other problem characteristics (i.e. modularity, regularity, and deceptiveness) have a larger impact on the performance and efficiency of diversity objectives than how discrete or continuous the problem environment is.

6.2.5 Summary

The following relationships between the performance of diversity objectives and the problem characteristics were found:

- A correlation between the performance of *ad hoc* behavioral diversity and the deceptiveness of the problem was found. Domain knowledge seems to be especially useful for solving deceptive problems, which is consistent with similar findings of previous studies.
- A correlation between the performance of structural diversity and the modularity of the problem was found. It appears that encouraging

both performance and modularity are not very conflicting objectives on modular problems.

- A correlation between the performance of representational diversity and the regularity of the problem was found. Representational performed especially well on problems where discovering an oscillatory pattern (i.e. novel neuron activations) is important.
- No clear relationship between the diversity objectives and the representation of the environment space (i.e. discrete or continuous). The other problem characteristics are suggested to be more important.

6.3 Representational diversity as a diversity type in neuroevolution

Representational diversity was introduced as a new type of diversity in this thesis. It could be argued that representational diversity is no different from behavioral diversity. Both types of diversity use neural network outputs, with behavioral diversity only using neuron activations from the output neurons. Furthermore, when the Creative Thinking Approach (CTA) was introduced in [54], it was mentioned that the proposed method extends the way the Hamming distance is used. However, the results in this thesis show that representational diversity merits consideration as its own diversity type separate from behavioral diversity. CCA significantly outperforming both behavioral and structural diversity on the `Bipedal-Walker` problem provides support for representational diversity as a diversity type different from the already established types in neuroevolution.

Interestingly, there was a significant difference in performance between the CKA and CCA treatments across all targeted problems. As mentioned in the analysis earlier, CCA outperformed CKA on almost all problems. These results are surprising considering how much better the CKA similarity metric was found to reliably identify corresponding representations in neural networks compared to CCA in [46].

One explanation is that CCA is superior to CKA when encouraging a diversity of learned representations in neural networks of different topologies. In other words, CCA may be better for computing the similarity between *misaligned* neuron activations. In [46], a fixed network structure was used with exact corresponding neurons. Finding corresponding (or at least somewhat corresponding) neurons of networks with different topologies in neuroevolution is not possible without complex structure analysis. Thus, CCA is perhaps more suitable to measure the similarity across layers and neurons of the network and does not need an exact match between activations of networks. The ability of CCA to compare misaligned activations (i.e. different network topologies) was argued in [74] to be because the similarity metric is invariant to invertible linear transformations. Whereas, the CKA similarity metric is *not* invariant to invertible linear transformations [46].

Another factor that may help explain the difference in performance between CCA and CKA is the measured range of values of the representational diversity of individuals. Figures 6.1 and 6.2 show the Pareto fronts for CKA and CCA for one of the 50 runs on the `Bipedal-Walker` problem. The Pareto fronts for every 20th generation are visualized with the color reflecting the generation.

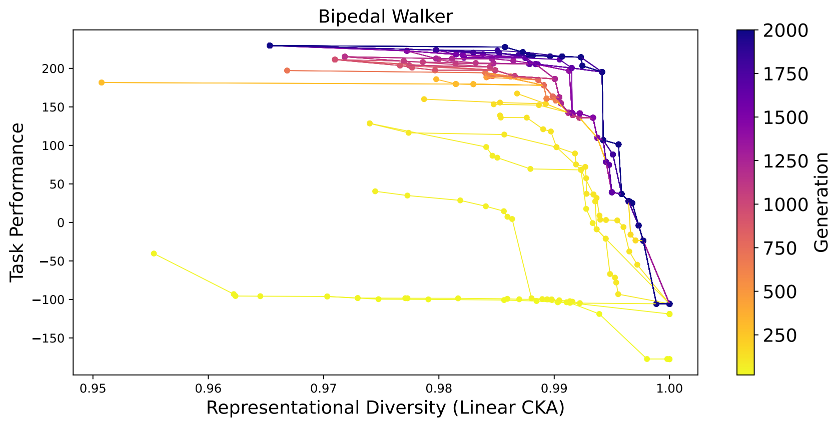


Figure 6.1: Pareto front for one of 50 runs with the CKA treatment on the Bipedal-Walker problem. Fronts are shown for every 20th generation. The color of the front indicates the generation. For this particular run, the final performance score was 229.

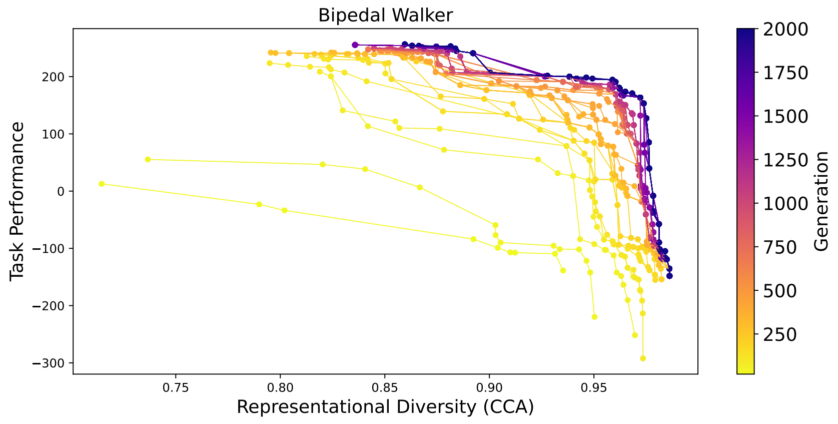


Figure 6.2: Pareto front for one of 50 runs with the CCA treatment on the Bipedal-Walker problem. Fronts are shown for every 20th generation. The color of the front indicates the generation. For this particular run, the final performance score was 256.

Studying the range of representational diversity values on the x-axis, it becomes apparent that the range is a lot smaller for CKA than with CCA. For CKA in figure 6.1, the representational diversity of individuals ranges from 0.95 to 1.0 indicating very high diversity (i.e. very low similarity between neural network representations). Whereas for CCA in figure 6.2, the range of values is from 0.70 to 1.0. The aforementioned ranges for both CKA and CCA are generally consistent for all 50 runs. This suggests that CCA is potentially more informative because it is able to differentiate better between representations of neural networks of different topologies. Being able to better differentiate between individuals could facilitate a more diverse population and more exploration of solutions, and as a result, better performance on the problem. Further investigation is needed to confirm if there is any merit to the claim that CCA is better able to differentiate between neural network representation in neuroevolution.

6.3.1 Summary

- Representational diversity was shown to merit consideration as a new type of diversity in neuroevolution. Performed especially well on problems where the neuron activations are important for solving the task. However, more investigation into representational diversity is needed.
- A significant performance difference between the similarity metrics CKA and CCA was found, with CCA significantly outperforming CKA. Possible reasons for this are that CKA is more informative when neuron activations are misaligned for neural networks of different topologies or that CKA is better at differentiating between neural networks of different topologies.

6.4 Population size and parameter setting

A population size of 100 was chosen for all targeted problems due to the computational resources and time required to run all experiments 50 times. Likewise, no comprehensive parameter tuning was performed except for small test runs. However, it can not be discounted that some treatments could perform better with different parameter settings. One way to overcome this would be to employ an automatic tuning procedure without the need to rely on subjective intuition, and the parameters are tuned for each experiment according to the same criteria [75]. Still, automatic tuning would require multiple runs per experiment because of the stochastic nature of evolutionary algorithms.

Nevertheless, the lack of parameter tuning and a relatively small population size does not render the present work meaningless. The results in this work are mostly consistent with many of the previous studies targeting the same problems but with larger population sizes and different experimental parameters. Additionally, results are consistent even for problems where the neural network architecture/encoding is different. This consistency of results across studies supports the assumption that diversity is an important factor.

7 | Conclusion

Encouraging diversity in neuroevolution with diversity objectives has seen increased interest in years, and demonstrated impressive results in terms of efficiency and performance of the evolutionary search. What is not yet clear is the relationship between the type of diversity objective and the characteristics of the problem, and if there is one, how to determine which types of diversity objectives are more suitable for which problems. Additionally, a type of diversity objective that is able to encompass both the structure and behavior of neural networks has not been extensively explored yet.

This thesis set out to compare diversity objectives on problems with unique problem characteristics often targeted in neuroevolution, and determine if there is a correlation between the performance of the type of diversity objective and the problem characteristics. Furthermore, representational diversity was proposed as its own diversity type, including the first steps on how to adapt representational diversity distances to neuroevolution. The main findings of this work are as follows:

- A clear correlation between the performance of diversity objectives and the problem characteristics. *Ad hoc* behavioral diversity was found to outperform all other types of diversity on deceptive problems, whereas structural diversity objectives performed best on problems expressing a high degree of modularity. Representational diversity showed promising results on a problem with the characteristic of regularity.
- A significant performance difference between diversity objectives of the same type was found for many of the targeted problems.
    - Of the behavioral diversity objectives, the *ad hoc* behavioral diversity objective outperformed the *generic* behavioral diversity objective on all targeted problems.
    - The difference between the structural diversity objectives was less striking. Encouraging a diversity of modular decomposition of the *input neurons* was more effective in the early phase of the evolutionary search than encouraging a diversity of modular decomposition of the *network* as a whole on a few of the problems.

    - The difference in the performance of representational diversity objectives with different similarity metrics was significant. Results suggest that similarity metrics with invariance to invertible linear transformations are more suitable for neuroevolution (i.e. neural networks with different topologies) than those without.

- Early promising results for representational diversity as a new type of diversity in neuroevolution. There are indications that encouraging representational diversity can be powerful on tasks where novel neuron activations can lead to better performance, such as robot locomotion problems. This thesis has made the first steps in introducing representational diversity and adapting it to neuroevolution, and the findings suggest that this new type of diversity merits further study.

7.1 Future work

This work has presented the initial indications of a correlation between the performance of the type of diversity objective and the characteristics of the problem. Future work should validate the findings of this work on other problems to see if this correlation still holds. More parameter tuning would also be beneficial for a potential performance improvement, especially experimenting with mutation rates. A different initialization scheme could also be interesting to explore, where the initial neural networks start with a number of hidden neurons and are not fully connected. This way of initialization may improve exploration as neural networks grow more different over generations than if they have the same starting topology.

Furthermore, exploring alternative multi-objective evolutionary algorithms (e.g. SPEA2) to study if they are more suited to neuroevolution than NSGA–II is something that has yet to be attempted. Many of the comparisons of multi-objective evolutionary algorithms usually use a set of test problems, but it is not known if this translates well to targeted problems in neuroevolution.

Performing the same experiments with a different encoding, architecture, or algorithm could be further explored. Although NEAT was originally intended to be single-objective, using a multi-objective evolutionary algorithm with NEAT is something that should be investigated. Additionally, investigate if the speciation of NEAT with a diversity objective would improve the efficiency and performance or not. Moreover, use a *recurrent neural network* architecture for the Tartarus problem or a *compositional pattern-producing network* (CPPN) encoding for a robot locomotion problem instead of using the same architecture or encoding across all problems. The purpose would be to study if certain diversity objectives potentially have a greater impact if a more suitable encoding/architecture is used for specific problems.

Finally, further study of representational diversity as a whole is needed, especially to validate that representational diversity is more suitable for problems with a high degree of regularity. In particular, combining the ability of CPPNs to produce complex patterns and representational diversity could potentially be effective on problems with a high degree of regularity. Although, as discussed earlier, distinguishing between problems with oscillatory regularity and sub-task regularity should be considered in future studies. Further investigation into the claim that invariance to invertible linear transformations has a great impact on the applicability of representational diversity objectives to neuroevolution should also be a priority.

Appendices

A | Experimental parameters

Table A.1: All experimental parameters that are common for all targeted problems.

Experimental parameter	**Value**
population size	100
neural network initialization	fully-connected without hidden neurons
parametric mutation	polynomial mutation
distribution index (η_m)	10
k-nearest neighbors	15

Table A.2: Experimental parameters specific to the 2x2-Retina problem.

Experimental parameter	**Value**
number of generations	2500
novelty threshold	0.85
max archive size	N/A
feed-forward	True
number of inputs	8
number of outputs	1
activation function	tanh (with $\lambda = 20$)
mutation rate – add neuron	0.1
mutation rate – remove neuron	0.05
mutation rate – add connection	0.2
mutation rate – remove connection	0.1
mutation rate – weight mutation	0.1
mutation rate – bias mutation	0.05
bias min value	-10.0
bias max value	10.0
weight min value	-5.0
weight max value	5.0

Table A.3: Experimental parameters specific to the 3x3-Retina problem.

Experimental parameter	**Value**
number of generations	5000
novelty threshold	0.85
max archive size	N/A
feed-forward	True
number of inputs	18
number of outputs	1
activation function	tanh (with $\lambda = 20$)
mutation rate – add neuron	0.1
mutation rate – remove neuron	0.05
mutation rate – add connection	0.2
mutation rate – remove connection	0.1
mutation rate – weight mutation	0.1
mutation rate – bias mutation	0.05
bias min value	-10.0
bias max value	10.0
weight min value	-5.0
weight max value	5.0

Table A.4: Experimental parameters specific to both the Tartarus problem and the Deceptive-Tartarus problem.

Experimental parameter	**Value**
number of generations	1500
novelty threshold	2.0
max archive size	500
feed-forward	True
number of inputs	11
number of outputs	3
activation function	sigmoid
mutation rate – add neuron	0.1
mutation rate – remove neuron	0.05
mutation rate – add connection	0.2
mutation rate – remove connection	0.1
mutation rate – weight mutation	0.2
mutation rate – bias mutation	0.05
bias min value	-10.0
bias max value	10.0
weight min value	-10.0
weight max value	10.0

Table A.5: Experimental parameters specific to both the Medium-Maze problem and the Hard-Maze problem.

Experimental parameter	**Value**
number of generations	2000
novelty threshold	10.0
max archive size	N/A
feed-forward	True
number of inputs	7
number of outputs	2
activation function	sigmoid
mutation rate – add neuron	0.1
mutation rate – remove neuron	0.05
mutation rate – add connection	0.2
mutation rate – remove connection	0.1
mutation rate – weight mutation	0.15
mutation rate – bias mutation	0.1
bias min value	−10.0
bias max value	10.0
weight min value	−5.0
weight max value	5.0

Table A.6: Experimental parameters specific to both the Bipedal-Walker problem.

Experimental parameter	**Value**
number of generations	2000
novelty threshold	1500.0
max archive size	500
feed-forward	True
number of inputs	24
number of outputs	4
activation function	tanh
mutation rate – add neuron	0.1
mutation rate – remove neuron	0.05
mutation rate – add connection	0.3
mutation rate – remove connection	0.15
mutation rate – weight mutation	0.2
mutation rate – bias mutation	0.1
bias min value	−10.0
bias max value	10.0
weight min value	−5.0
weight max value	5.0

B | Additional results

B.1 2x2–Retina

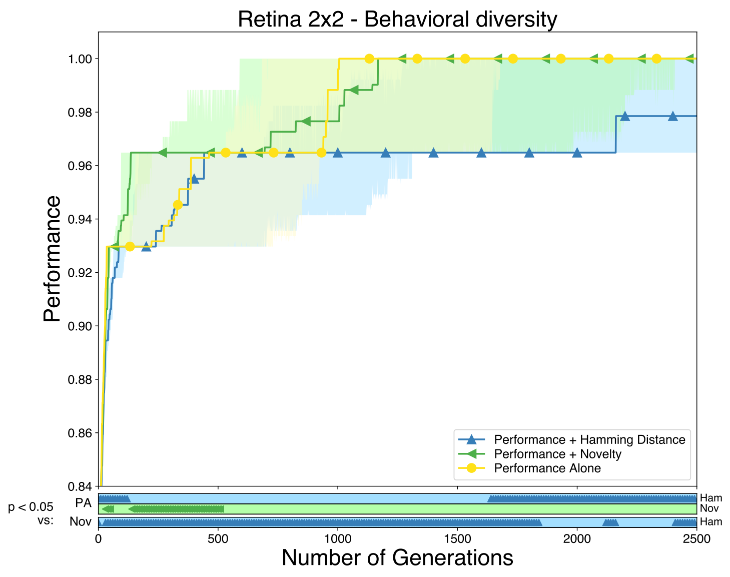


Figure B.1: The performance of behavioral diversity treatments on 2x2-Retina. Of the behavioral diversity treatments, NOVELTY leads to better performance and is slightly more efficient than PA for the first 500 generations. HAMMING sometimes achieves the maximum performance score towards the end but not consistently enough.

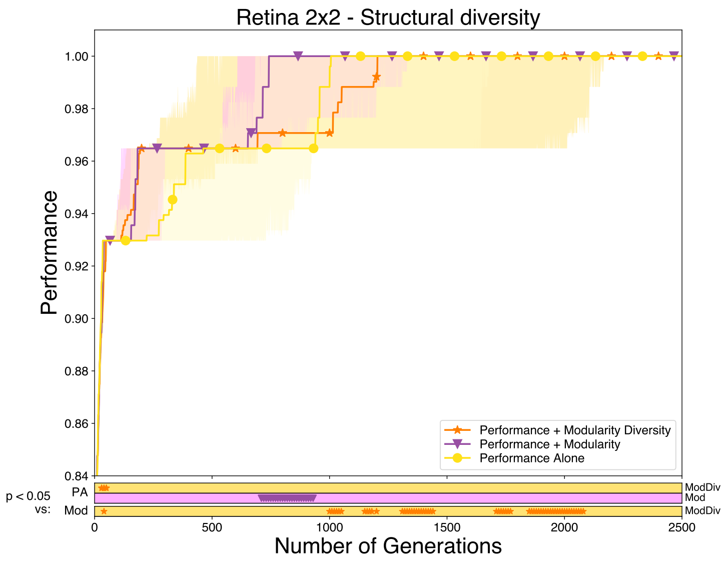


Figure B.2: The performance of structural diversity treatments on 2x2-Retina. The best performance score is reached by both Mod and ModDiv with neither being more efficient. PA performs on par with both structural diversity treatments.

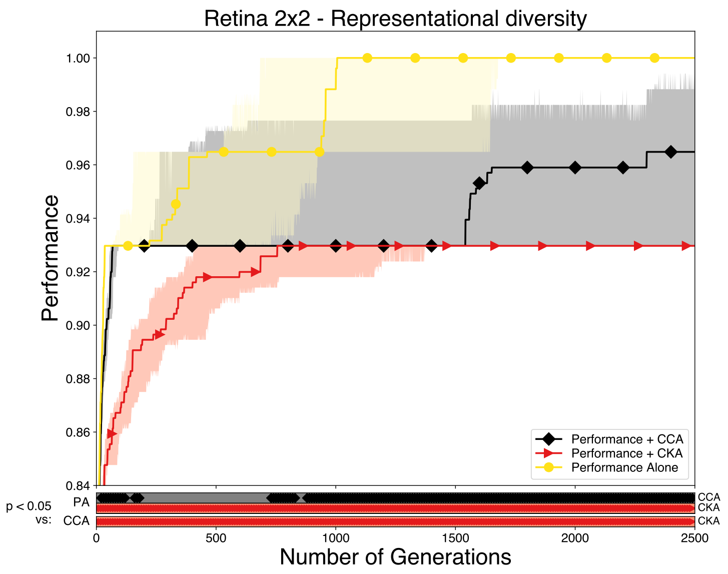


Figure B.3: The performance of representational diversity treatments on 2x2-Retina. Both representational diversity treatments CKA and CCA perform significantly worse than PA. CCA performs better than CKA but does not achieve the maximum performance.

B.2 3x3–Retina

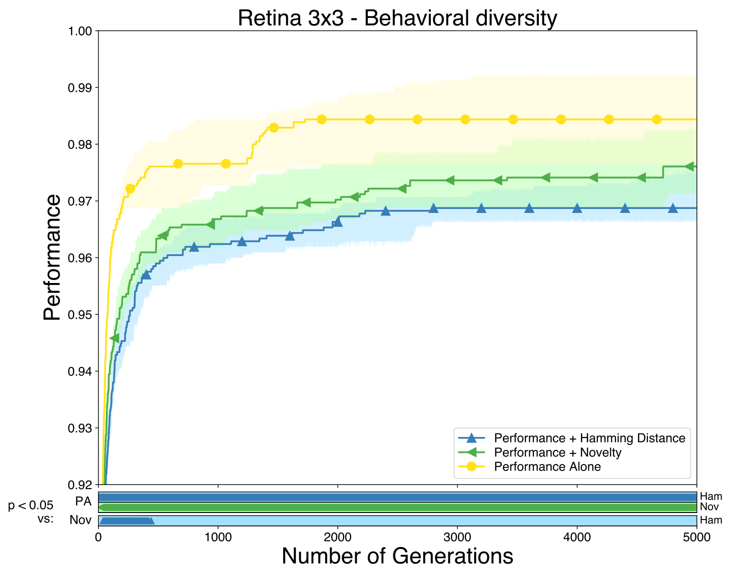


Figure B.4: The performance of behavioral diversity treatments on 3x3-Retina. Both NOVELTY and HAMMING perform significantly worse than PA. NOVELTY leads to slightly better performance compared to HAMMING.

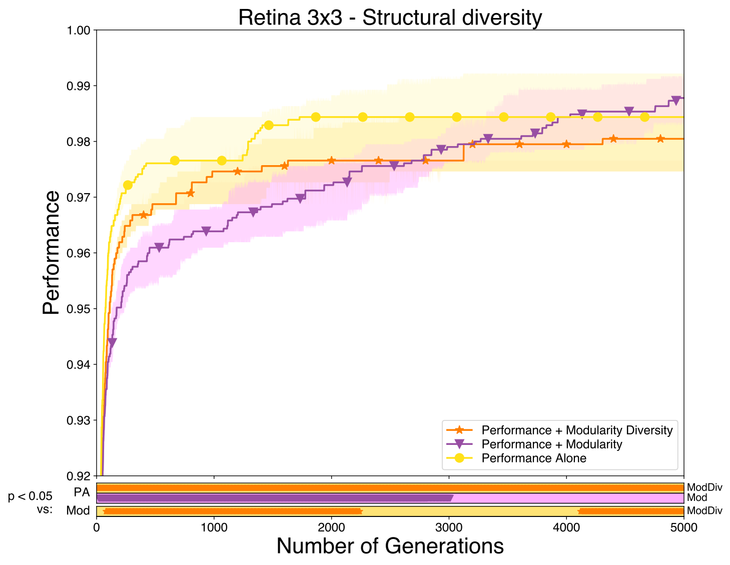


Figure B.5: The performance of structural diversity treatments on 3x3-Retina. MOD leads to the best performance including the control PA, while the MODDIV treatment is slightly worse. MOD is slower than both MODDIV and PA for the first 2500 generations but overtakes both of them as the other two converge.

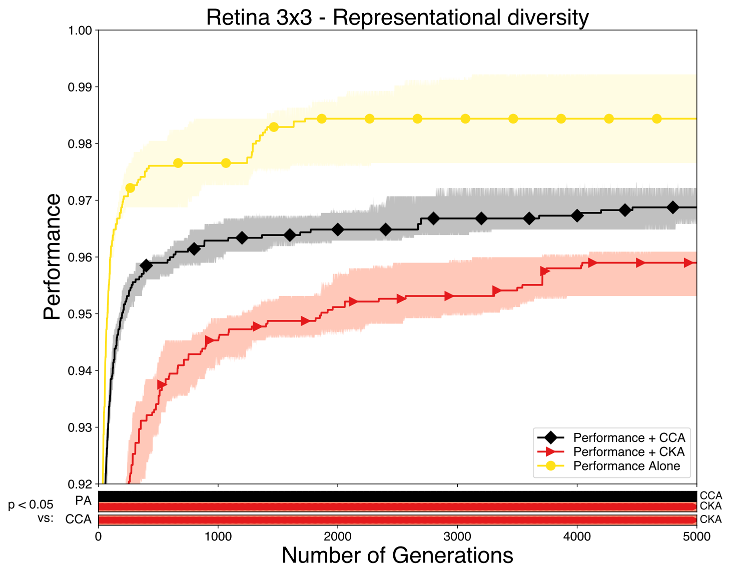


Figure B.6: The performance of representational diversity treatments on 3x3-Retina. Both representational diversity treatments perform significantly worse than PA. Between the two diversity treatments, CCA performs better and is more efficient than CKA.

B.3 Tartarus

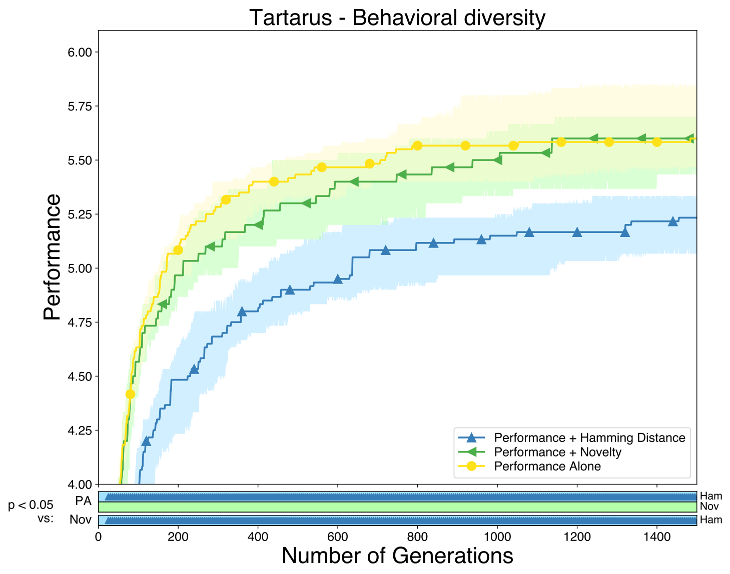


Figure B.7: The performance of behavioral diversity treatments on Tartarus. The Novelty treatment performs significantly better than Hamming. There was not any significant difference in performance between Novelty and PA.

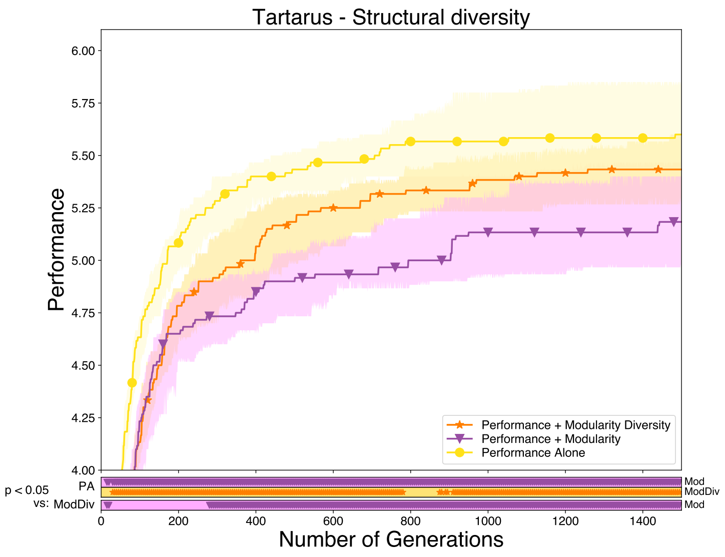


Figure B.8: The performance of structural diversity treatments on Tartarus. PA significantly outperforms both structural diversity treatments. Of the two structural diversity treatments, ModDiv significantly outperforms Mod.

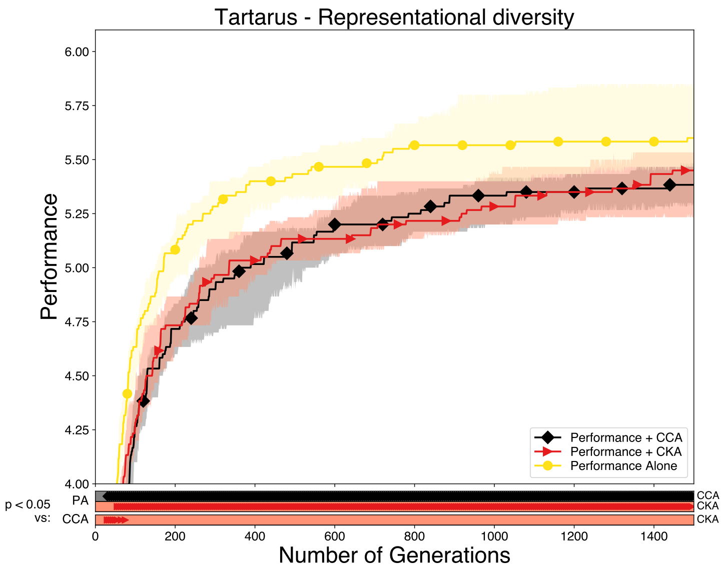


Figure B.9: The performance of representational diversity treatments on Tartarus. Both representational diversity treatments, CKA and CCA, perform similarly with no significant difference between them. PA outperforms both representational diversity treatments.

B.4 Deceptive–Tartarus

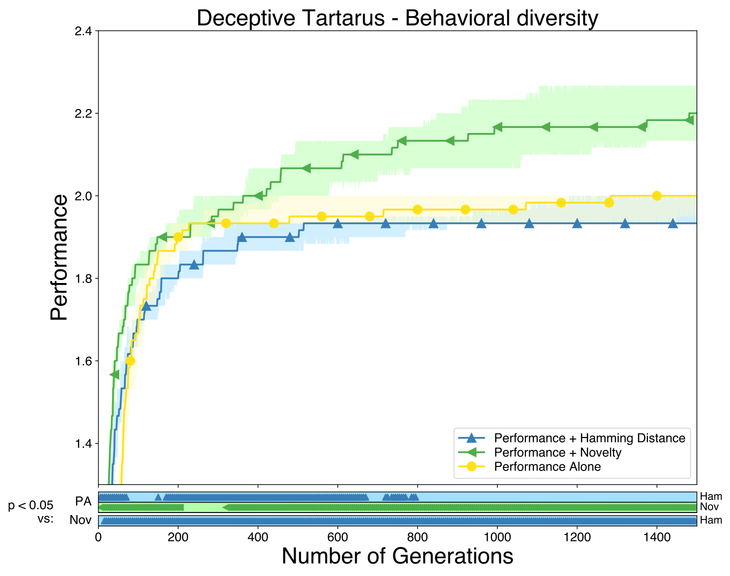


Figure B.10: The performance of behavioral diversity treatments on Deceptive-Tartarus. Of the behavioral diversity treatments, Novelty significantly outperformed Hamming. The difference between Hamming and PA was only significant for the first 400 generations.

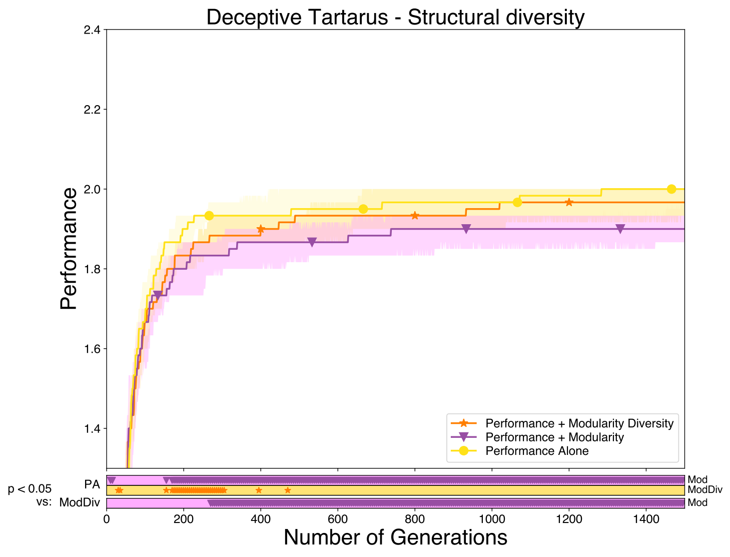


Figure B.11: The performance of structural diversity treatments on Deceptive-Tartarus. Both the ModDiv and PA treatments performed similarly. The Mod treatment performed slightly worse than ModDiv and PA.

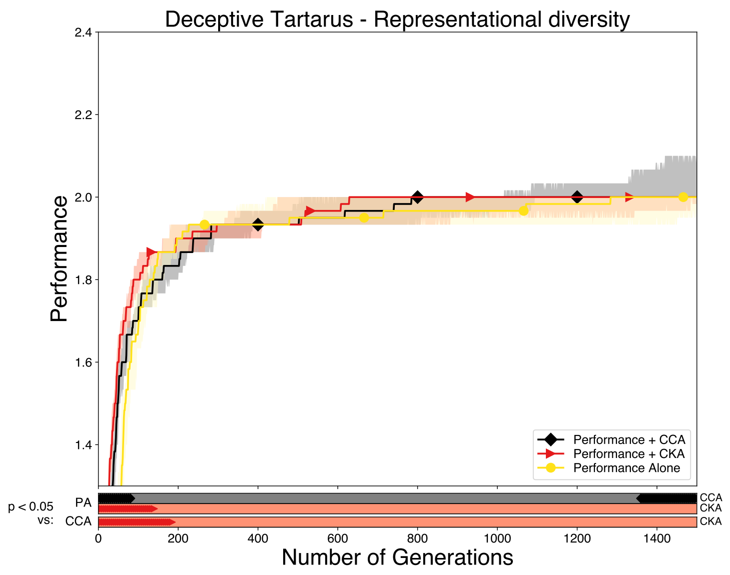


Figure B.12: The performance of representational diversity treatments on Deceptive-Tartarus. No significant performance difference can be seen between CKA, CCA, and PA. There are some indications that CCA could potentially reach a score over 2.0 if allowed to run for longer than 1500 generations.

B.5 Medium–Maze

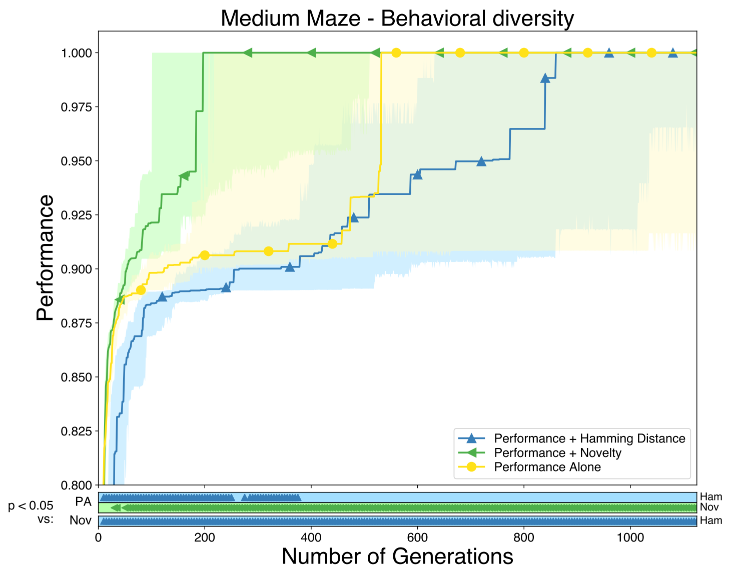


Figure B.13: The performance of behavioral diversity treatments on Medium-Maze. Novelty was significantly more efficient than both Hamming and PA. The Hamming treatment was slightly slower than PA, but the difference was not significant.

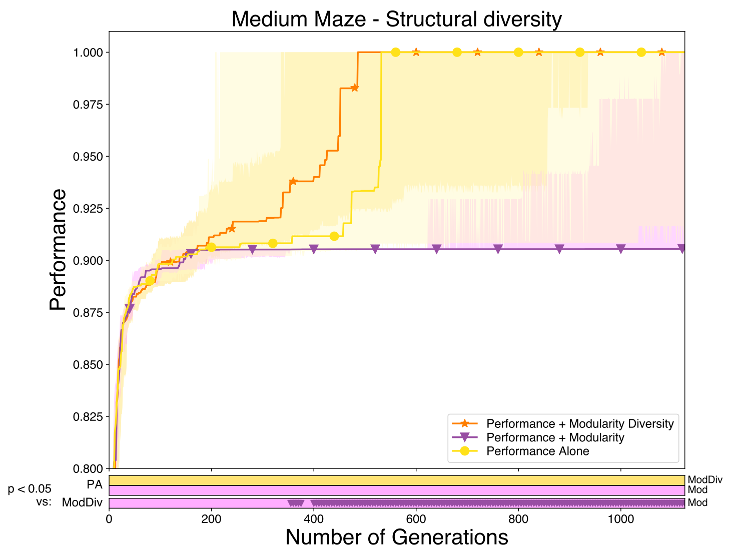


Figure B.14: The performance of structural diversity treatments on Medium-Maze. ModDiv performed similarly to PA, but significantly outperformed Mod. The Mod treatment was only able to reach the goal in a couple of instances and otherwise converged to one of the deceptive local optima. ModDiv significantly outperformed Mod.

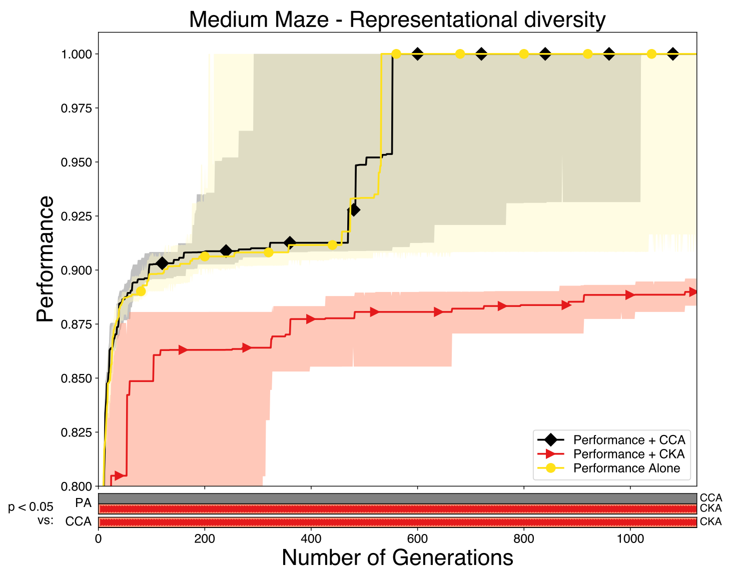


Figure B.15: The performance of representational diversity treatments on Medium-Maze. CCA performed similarly to PA. CKA performed significantly worse than both CCA and PA, and never reached the goal of the maze.

B.6 Bipedal

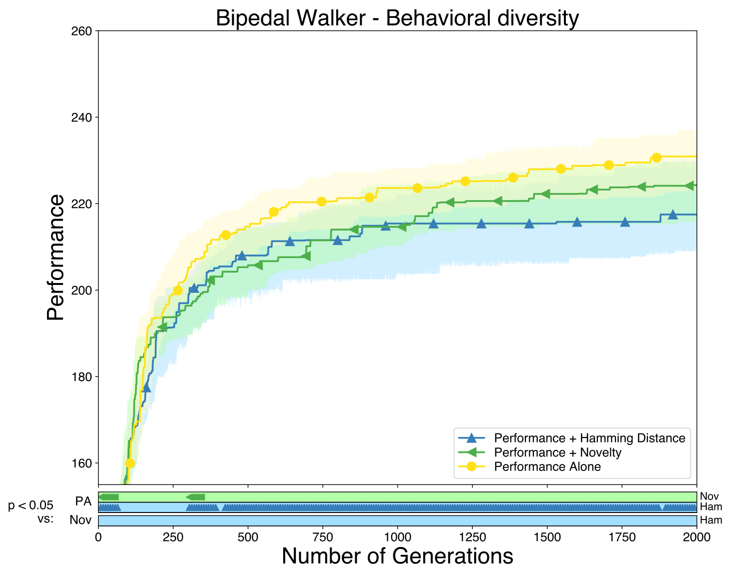


Figure B.16: The performance of behavioral diversity treatments on Bipedal-Walker. None of the behavioral diversity treatments outperform PA. NOVELTY is only slightly better than HAMMING but the difference is not significant.

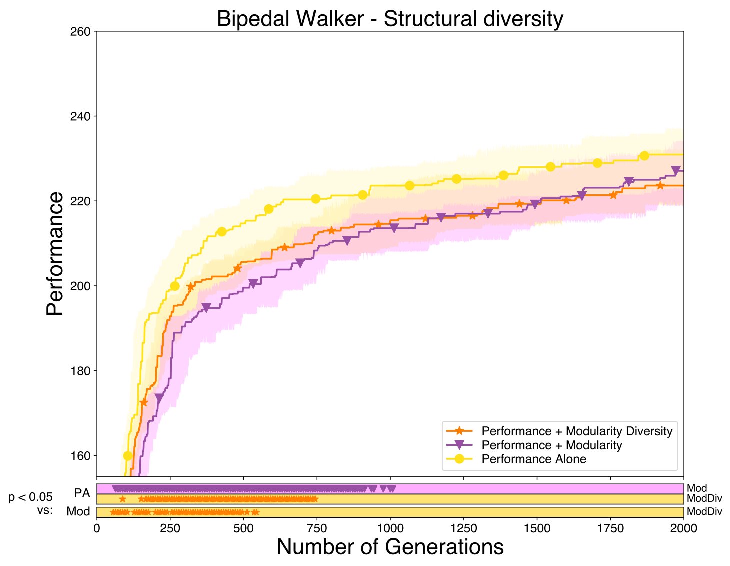


Figure B.17: The performance of structural diversity treatments on Bipedal-Walker. PA outperforms MOD and MODDIV for the first 1000 generations. For subsequent generations, the performance difference is not significant.

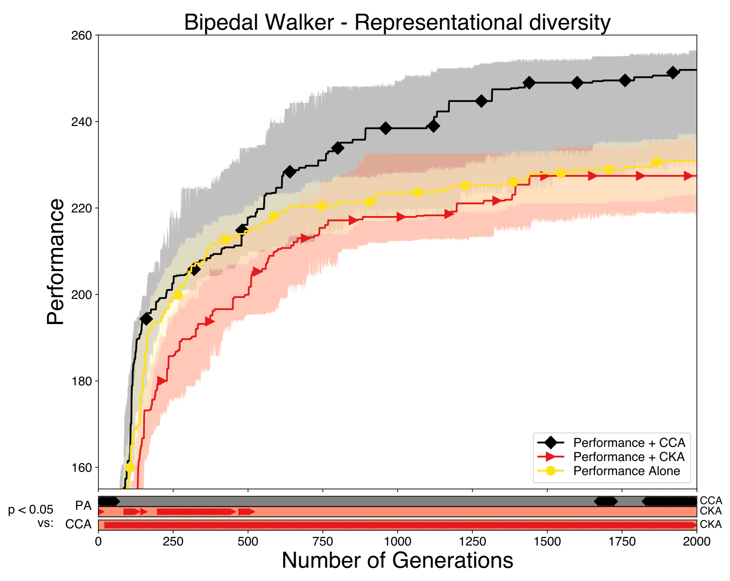


Figure B.18: The performance of representational diversity treatments on Bipedal-Walker. The CCA treatment outperforms both CKA and CCA. However, the performance difference between CCA and PA only becomes significant at around generation 1800 because of the large confidence interval.

B.7 Computational run-times

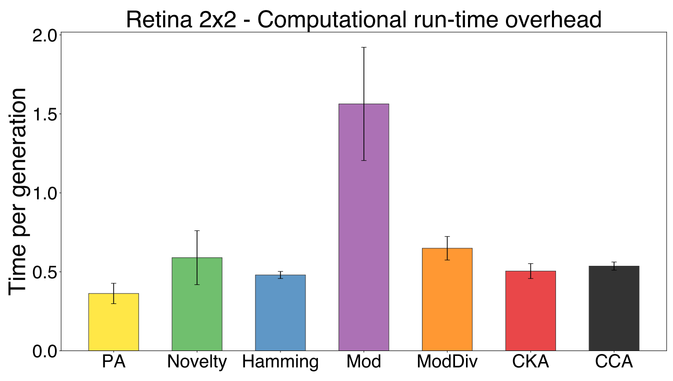


Figure B.19: The mean run-time per generation on 2x2-Retina. Time units are in seconds with standard deviation also shown. Simulated using a single core. MOD slower than the rest of the treatments.

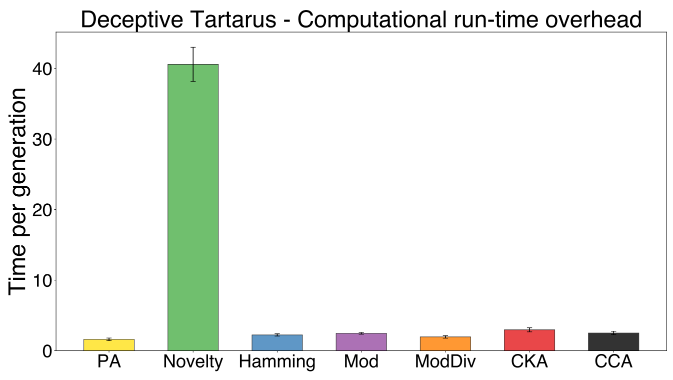


Figure B.20: The mean run-time per generation on Deceptive-Tartarus. Time units are in seconds with standard deviation also shown. Simulated using 20 cores. Computing NOVELTY is significantly more computationally intensive, and thus requires more time per generation compared to the other treatments.

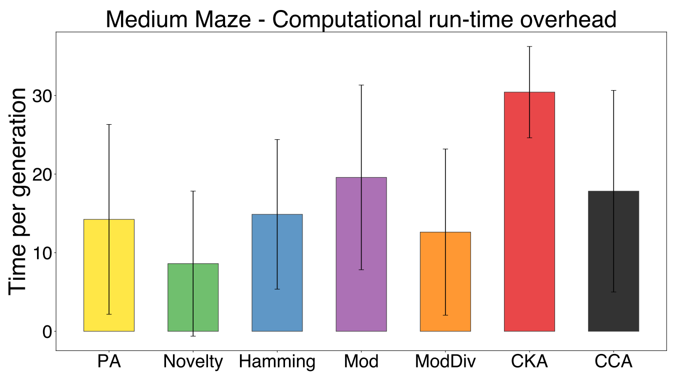
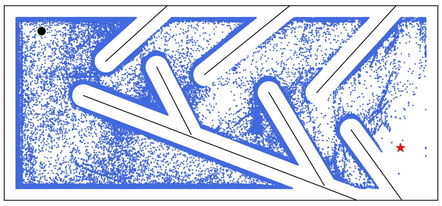
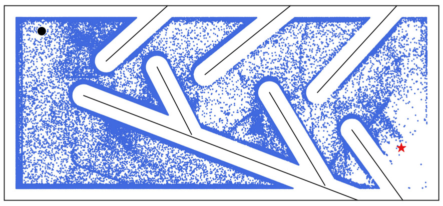
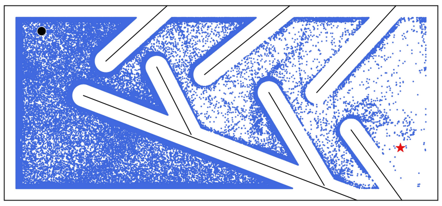


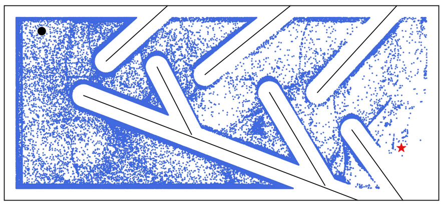
Figure B.21: The mean run-time per generation on Medium-Maze. Time units are in seconds with standard deviation also shown. Simulated using 10 cores. Caution should be taken when analyzing the times for this experiment. The reason for the large standard deviation is due to treatments sometimes reaching a maximum score very early and never on other runs. Thus, any run-time difference between treatments is mainly due to larger neural networks and not due to the diversity computation.

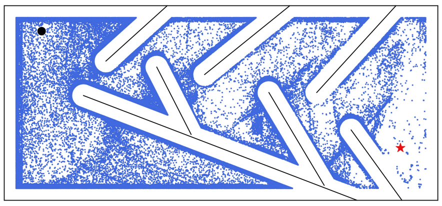
B.8 How treatments explored the Medium-Maze

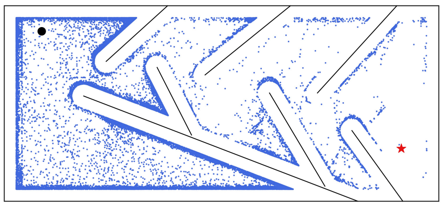

(a) PA


(b) Novelty


(c) Hamming


(d) Mod


(e) ModDiv


(f) CKA

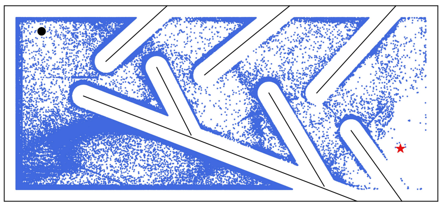

(g) CCA

Figure B.22: Illustration of how differently each treatment explored the Medium-Maze. Each dot is the end position of a simulated robot. There was no notable difference between treatments, except for CKA. CKA explored significantly less of area of the maze than other treatments.

Bibliography

[1] LeCun, Y., Bengio, Y., and Hinton, G. "Deep learning." *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[2] Floreano, D. and Mattiussi, C. *Bio-inspired artificial intelligence: theories, methods, and technologies.* MIT press, 2008.

[3] Stanley, K. et al. "Designing neural networks through neuroevolution." *Nature Machine Intelligence* 1 (Jan. 2019). DOI: 10.1038/s42256-018-0006-z.

[4] Ellefsen, K. O., Huizinga, J., and Torresen, J. "Guiding Neuroevolution with Structural Objectives." *Evolutionary Computation* 28.1 (Mar. 2020), pp. 115–140. ISSN: 1530-9304. DOI: 10.1162/evco_a_00250. URL: http://dx.doi.org/10.1162/evco_a_00250.

[5] Lehman, J. and Stanley, K. "Abandoning Objectives: Evolution Through the Search for Novelty Alone." *Evolutionary computation* 19 (June 2011), pp. 189–223. DOI: 10.1162/EVCO_a_00025.

[6] Mouret, J.-B. "Novelty-Based Multiobjectivization." Vol. 341. Feb. 2011, pp. 139–154. ISBN: 978-3-642-18271-6. DOI: 10.1007/978-3-642-18272-3_10.

[7] Mouret, J.-B. and Doncieux, S. "Encouraging Behavioral Diversity in Evolutionary Robotics: an Empirical Study." *Evolutionary Computation* 20.1 (2012), pp. 91–133. DOI: 10.1162/EVCO_a_00048. URL: https://hal.archives-ouvertes.fr/hal-00687609.

[8] Toffolo, A. and Benini, E. "Genetic Diversity as an Objective in Multi-Objective Evolutionary Algorithms." *Evol. Comput.* 11.2 (May 2003), pp. 151–167. ISSN: 1063-6560. DOI: 10.1162/106365603766646816. URL: https://doi-org.ezproxy.uio.no/10.1162/106365603766646816.

[9] Huizinga, J., Mouret, J.-B., and Clune, J. "Does Aligning Phenotypic and Genotypic Modularity Improve the Evolution of Neural Networks?" *Proceedings of the 25th Genetic and Evolutionary Computation Conference (GECCO).* ACM. Denver, France, 2016, pp. 125–132. DOI: 10.1145/2908812.2908836. URL: https://hal.inria.fr/hal-01402502.

[10] Clune, J., Mouret, J.-B., and Lipson, H. “The evolutionary origins of modularity.” *Proceedings. Biological sciences / The Royal Society* 280 (July 2013), p. 20122863. DOI: 10.1098/rspb.2012.2863.

[11] Eiben, A. and Smith, J. *Introduction To Evolutionary Computing.* Vol. 45. Jan. 2003. ISBN: 978-3-642-07285-7. DOI: 10.1007/978-3-662-05094-1.

[12] De Jong, K. *Evolutionary Computation – A Unified Approach.* MIT Press, Jan. 2006. ISBN: 9780262041942.

[13] Črepinšek, M., Liu, S.-H., and Mernik, M. “Exploration and Exploitation in Evolutionary Algorithms: A Survey.” *ACM Comput. Surv.* 45.3 (July 2013). ISSN: 0360-0300. DOI: 10.1145/2480741.2480752. URL: https://doi.org/10.1145/2480741.2480752.

[14] Friedrich, T. et al. “Analysis of Diversity-Preserving Mechanisms for Global Exploration.” *Evolutionary Computation* 17 (2009), pp. 455–476.

[15] Sudholt, D. “The Benefits of Population Diversity in Evolutionary Algorithms: A Survey of Rigorous Runtime Analyses.” Jan. 2020, pp. 359–404. ISBN: 978-3-030-29413-7. DOI: 10.1007/978-3-030-29414-4_8.

[16] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning.* 1st. USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN: 0201157675.

[17] Goldberg, D. E. and Richardson, J. “Genetic Algorithms with Sharing for Multimodal Function Optimization.” *Proceedings of the Second International Conference on Genetic Algorithms and Their Application.* Cambridge, Massachusetts, USA: L. Erlbaum Associates Inc., 1987, pp. 41–49. ISBN: 0805801588.

[18] Mengshoel, O. and Goldberg, D. “The Crowding Approach to Niching in Genetic Algorithms.” *Evolutionary computation* 16 (Feb. 2008), pp. 315–54. DOI: 10.1162/evco.2008.16.3.315.

[19] Nelson, A. L., Barlow, G. J., and Doitsidis, L. “Fitness functions in evolutionary robotics: A survey and analysis.” *Robotics and Autonomous Systems* 57.4 (2009), pp. 345–370. ISSN: 0921-8890. DOI: 10.1016/j.robot.2008.09.009. URL: https://www.sciencedirect.com/science/article/pii/S0921889008001450.

[20] Krčah, P. “Solving deceptive tasks in robot body-brain co-evolution by searching for behavioral novelty.” *2010 10th International Conference on Intelligent Systems Design and Applications.* 2010, pp. 284–289. DOI: 10.1109/ISDA.2010.5687250.

[21] Lehman, J., Stanley, K. O., and Miikkulainen, R. “Effective Diversity Maintenance in Deceptive Domains.” *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. GECCO ’13. Amsterdam, The Netherlands: Association for Computing Machinery, 2013, pp. 215–222. ISBN: 9781450319638. DOI: 10.1145/2463372.2463393. URL: https://doi.org/10.1145/2463372.2463393.

[22] Branke, J. et al., eds. *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Berlin, Heidelberg: Springer-Verlag, 2008. ISBN: 9783540889076.

[23] Miettinen, K. *Nonlinear multiobjective optimization*. Vol. 12. Springer Science & Business Media, 1999.

[24] Zitzler, E. and Thiele, L. “Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach.” *IEEE Transactions on Evolutionary Computation* 3.4 (1999), pp. 257–271. DOI: 10.1109/4235.797969.

[25] Zitzler, E., Deb, K., and Thiele, L. “Comparison of multiobjective evolutionary algorithms: Empirical results.” *Evolutionary computation* 8.2 (2000), pp. 173–195.

[26] Knowles, J. and Corne, D. “Quantifying the Effects of Objective Space Dimension in Evolutionary Multiobjective Optimization.” eng. *Evolutionary Multi-Criterion Optimization*. Vol. 4403. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 757–771. ISBN: 9783540709275.

[27] Purshouse, R. and Fleming, P. “On the Evolutionary Optimization of Many Conflicting Objectives.” eng. *IEEE transactions on evolutionary computation* 11.6 (2007), pp. 770–784. ISSN: 1089-778X.

[28] Santana-Quintero, L. V., Montaño, A. A., and Coello, C. A. C. “A Review of Techniques for Handling Expensive Functions in Evolutionary Multi-Objective Optimization.” *Computational Intelligence in Expensive Optimization Problems*. Ed. by Tenne, Y. and Goh, C.-K. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 29–59. ISBN: 978-3-642-10701-6. DOI: 10.1007/978-3-642-10701-6_2. URL: https://doi.org/10.1007/978-3-642-10701-6_2.

[29] Zitzler, E., Laumanns, M., and Thiele, L. “SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization.” Vol. 3242. Jan. 2001.

[30] Deb, K. et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II.” *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197. DOI: 10.1109/4235.996017.

[31] Knowles, J. D., Watson, R. A., and Corne, D. W. “Reducing local optima in single-objective problems by multi-objectivization.” *Evolutionary Multi-Criterion Optimization: First International Conference, EMO 2001 Zurich, Switzerland, March 7–9, 2001 Proceedings 1*. Springer. 2001, pp. 269–283.

[32] Handl, J., Lovell, S. C., and Knowles, J. “Multiobjectivization by Decomposition of Scalar Cost Functions.” *Parallel Problem Solving from Nature – PPSN X*. Ed. by Rudolph, G. et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 31–40. ISBN: 978-3-540-87700-4.

[33] LeCun, Y. et al. “Backpropagation Applied to Handwritten Zip Code Recognition.” *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[34] Hornik, K., Stinchcombe, M., and White, H. “Multilayer feedforward networks are universal approximators.” *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8.

[35] Rumelhart, D., Hinton, G., and Williams, R. “Learning representations by back-propagating errors.” *Nature* (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0.

[36] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[37] Baydin, A. G. et al. “Automatic Differentiation in Machine Learning: A Survey.” *J. Mach. Learn. Res.* 18.1 (Jan. 2017), pp. 5595–5637. ISSN: 1532-4435.

[38] Sutton, R. S. *Reinforcement learning : an introduction*. eng. Cambridge, Mass., 1998.

[39] Such, F. et al. “Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning” (Dec. 2017).

[40] Risi, S. and Togelius, J. “Neuroevolution in Games: State of the Art and Open Challenges.” *CoRR* abs/1410.7326 (2014). arXiv: 1410.7326. URL: http://arxiv.org/abs/1410.7326.

[41] Gomez, F., Schmidhuber, J., and Miikkulainen, R. “Accelerated Neural Evolution through Cooperatively Coevolved Synapses.” *Journal of Machine Learning Research* 9 (May 2008), pp. 937–965. DOI: 10.1145/1390681.1390712.

[42] Stanley, K. and Miikkulainen, R. “Efficient evolution of neural network topologies.” *Proceedings of the 2002 Congress on Evolutionary Computation. CEC’02 (Cat. No.02TH8600)*. Vol. 2. 2002, pp. 1757–1762. DOI: 10.1109/CEC.2002.1004508.

[43] Li, Y. et al. *Convergent Learning: Do different neural networks learn the same representations?* 2015. DOI: 10.48550/ARXIV.1511.07543. URL: https://arxiv.org/abs/1511.07543.

[44] Wang, L. et al. "Towards understanding learning representations: To what extent do different neural networks learn the same representation." *Advances in neural information processing systems* 31 (2018).

[45] Nguyen, T., Raghu, M., and Kornblith, S. *Do Wide and Deep Networks Learn the Same Things? Uncovering How Neural Network Representations Vary with Width and Depth.* 2020. DOI: 10.48550/ARXIV.2010.15327. URL: https://arxiv.org/abs/2010.15327.

[46] Kornblith, S. et al. "Similarity of Neural Network Representations Revisited." *CoRR* abs/1905.00414 (2019). arXiv: 1905.00414. URL: http://arxiv.org/abs/1905.00414.

[47] Wang, T., Dai, X., and Liu, Y. "Learning with Hilbert–Schmidt independence criterion: A review and new perspectives." *Knowledge-Based Systems* 234 (2021), p. 107567. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2021.107567. URL: https://www.sciencedirect.com/science/article/pii/S0950705121008297.

[48] Maclaurin, J. and Sterelny, K. *What is biodiversity?* University of Chicago Press, 2008.

[49] Brooks, D. R. and McLennan, D. A. *The nature of diversity: an evolutionary voyage of discovery.* University of Chicago Press, 2012.

[50] Mouret, J.-B. and Doncieux, S. "Using Behavioral Exploration Objectives to Solve Deceptive Problems in Neuro-evolution." *The 11th Annual conference on Genetic and evolutionary computation (GECCO'09).* Montréal, Canada: ACM, 2009, pp. 627–634. DOI: 10.1145/1569901.1569988. URL: https://hal.archives-ouvertes.fr/hal-00473132.

[51] Lehman, J. and Stanley, K. O. "Evolving a Diversity of Virtual Creatures through Novelty Search and Local Competition." *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation.* GECCO '11. Dublin, Ireland: Association for Computing Machinery, 2011, pp. 211–218. ISBN: 9781450305570. DOI: 10.1145/2001576.2001606. URL: https://doi.org/10.1145/2001576.2001606.

[52] Mouret, J. and Clune, J. "Illuminating search spaces by mapping elites." *CoRR* abs/1504.04909 (2015). arXiv: 1504.04909. URL: http://arxiv.org/abs/1504.04909.

[53] Gomez, F. "Sustaining diversity using behavioral information distance." Jan. 2009, pp. 113–120. DOI: 10.1145/1569901.1569918.

[54] Li, J., Storie, J., and Clune, J. "Encouraging Creative Thinking in Robots Improves Their Ability to Solve Challenging Problems." *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. GECCO '14. Vancouver, BC, Canada: Association for Computing Machinery, 2014, pp. 193–200. ISBN: 9781450326629. DOI: 10.1145/2576768.2598222. URL: https://doi.org/10.1145/2576768.2598222.

[55] Doncieux, S. and Mouret, J.-B. "Behavioral diversity measures for evolutionary robotics." *IEEE congress on evolutionary computation*. IEEE. 2010, pp. 1–8.

[56] Wills, P. and Meyer, F. G. "Metrics for graph comparison: A practitioner's guide." *PLOS ONE* 15.2 (Feb. 2020), pp. 1–54. DOI: 10.1371/journal.pone.0228728. URL: https://doi.org/10.1371/journal.pone.0228728.

[57] Lopresti, D. and Wilfong, G. "A fast technique for comparing graph representations with applications to performance evaluation." eng. *International journal on document analysis and recognition* 6.4 (2003), pp. 219–229. ISSN: 1433-2833.

[58] Striedter, G. F. *Principles of brain evolution.* Sinauer associates, 2005.

[59] Leicht, E. A. and Newman, M. E. J. "Community Structure in Directed Networks." *Physical Review Letters* 100.11 (Mar. 2008). DOI: 10.1103/physrevlett.100.118703. URL: https://doi.org/10.1103/PhysRevLett.100.118703.

[60] Deb, K. *Multi-objective optimization using evolutionary algorithms.* eng. Chichester, 2008.

[61] Kriegeskorte, N., Mur, M., and Bandettini, P. A. "Representational similarity analysis-connecting the branches of systems neuroscience." *Frontiers in systems neuroscience* (2008), p. 4.

[62] Kashtan, N. and Alon, U. "Spontaneous evolution of modularity and network motifs." *Proceedings of the National Academy of Sciences* 102.39 (2005), pp. 13773–13778. ISSN: 0027-8424. DOI: 10.1073/pnas.0503610102. eprint: https://www.pnas.org/content/102/39/13773.full.pdf. URL: https://www.pnas.org/content/102/39/13773.

[63] Mountcastle, V. B. "The columnar organization of the neocortex." *Brain: a journal of neurology* 120.4 (1997), pp. 701–722.

[64] Espinosa-Soto, C. and Wagner, A. "Specialization can drive the evolution of modularity." *PLoS computational biology* 6.3 (2010), e1000719.

[65] Stanley, K. "Compositional pattern producing networks: A novel abstraction of development." *Genetic Programming and Evolvable Machines* 8 (June 2007), pp. 131–162. DOI: 10.1007/s10710-007-9028-8.

[66] Fuchs, E. et al. "Coemergence of regularity and complexity during neural network development." *Developmental neurobiology* 67.13 (2007), pp. 1802–1814.

[67] Zoefel, B., Ten Oever, S., and Sack, A. T. "The involvement of endogenous neural oscillations in the processing of rhythmic input: More than a regular repetition of evoked neural responses." *Frontiers in neuroscience* 12 (2018), p. 95.

[68] Rothlauf, F. and Rothlauf, F. *Representations for genetic and evolutionary algorithms.* Springer, 2006.

[69] Omelianenko, I. *Hands-On Neuroevolution with Python: Build high-performing artificial neural network architectures using neuroevolution-based algorithms.* Packt Publishing Ltd, 2019.

[70] Cuccu, G. and Gomez, F. "When Novelty is Not Enough." *Proceedings of the 2011 International Conference on Applications of Evolutionary Computation - Volume Part I.* EvoApplications'11. Torino, Italy: Springer-Verlag, 2011, pp. 234–243. ISBN: 9783642205248.

[71] Griffiths, T. D. and Ekárt, A. "Improving the Tartarus problem as a benchmark in genetic programming." *Genetic programming.* Ed. by McDermott, J. et al. Lecture Notes in Computer Science. NLD: Springer, Mar. 2017, pp. 278–293. URL: https://publications.aston.ac.uk/id/eprint/30345/.

[72] Lehman, J. et al. "Safe Mutations for Deep and Recurrent Neural Networks through Output Gradients." *CoRR* abs/1712.06563 (2017). arXiv: 1712.06563. URL: http://arxiv.org/abs/1712.06563.

[73] Ellefsen, K. O., Mouret, J.-B., and Clune, J. "Neural Modularity Helps Organisms Evolve to Learn New Skills without Forgetting Old Skills." *PLOS Computational Biology* 11.4 (Apr. 2015), pp. 1–24. DOI: 10.1371/journal.pcbi.1004128. URL: https://doi.org/10.1371/journal.pcbi.1004128.

[74] Raghu, M. et al. "Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability." *Advances in neural information processing systems* 30 (2017).

[75] Smit, S. and Eiben, A. "Comparing parameter tuning methods for evolutionary algorithms." eng. *2009 IEEE Congress on Evolutionary Computation.* IEEE, 2009, pp. 399–406. ISBN: 1424429587.