Biology is one of the main sources of inspiration in producing intelligence in artificial systems, and stimulated a broad area of research in nature-inspired computing. Inspired by the evolutionary process of biological systems, the research field known as Neuroevolution employs evolutionary computing to optimize ANNs. One of the key aspects in Neuroevolution is the approach used for encoding the ANNs. For instance, direct encoding aims to optimize ANNs by representing their parameters in the genotype of the individuals directly, whereas, indirect encoding aims to find optimum rules to develop or learning mechanisms to train ANNs during their lifetime.

A fundamental property of Biological Neural Networks is their plasticity, which allows them to modify their internal configurations during their lifetime. According to the current physiological understanding, these changes are performed on synapses based on the local interactions of the neurons. This form of learning may lead to a high level of adaptation due to its distributed and self-organized nature. However, the exact synaptic plasticity mechanism that leads to the emergence of a coherent learning behaviour is not well-understood.

In this work, we investigate direct and indirect aspects of Neuroevolution. In the case of direct encoding, we propose limited evaluation and cooperative co-evolution schemes to help scale evolutionary approaches to optimize ANNs with large number of parameters. In the case of indirect Neuroevolution, we propose an evolutionary approach to producing plasticity property in ANNs to facilitate learning during the lifetime of the networks. We employ genetic algorithms to evolve discrete learning rules that are capable of performing synaptic changes locally based on the pairwise binary activation states of neurons. Our evolved plasticity rules make it easy to understand how the synaptic changes are performed depending on the all possible pairwise binary activation states of the neurons. We study plasticity in three kinds of learning processes. First, the reinforcement signals are available after every action of the networks. Second, the reinforcement signals are received after a certain period of time and finally, in the case where the reinforcement signals are not available. We test evolved plasticity rules on foraging and maze tasks, and show that they are capable of training networks for these tasks.

https://anilyaman.com

9 789038 648910

Evolution of Biologically Inspired Learning in Artificial Neural Networks

Anil Yaman

# Evolution of Biologically Inspired Learning in Artificial Neural Networks

Anil Yaman

# Evolution of Biologically Inspired Learning in Artificial Neural Networks

Anil Yaman

# Evolution of Biologically Inspired Learning in Artificial Neural Networks

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op dinsdag 19 november 2019 om 16:00 uur

door

Anil Yaman

geboren te Tekirdağ, Turkije

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

| | |
|---|---|
| voorzitter: | prof. dr. Johan Lukkien |
| 1e promotor: | prof. dr. Mykola Pechenizkiy |
| 2e promotor: | dr. George Fletcher |
| co-promotor: | dr. Decebal C. Mocanu |
| leden: | prof. dr. Agoston E. Eiben (Vrije Universiteit Amsterdam) |
| | prof. dr. Nicolas Bredeche (Université Pierre et Marie Curie) |
| | prof. dr. Heinrich J. Wörtche |
| adviseur(s): | dr. Giovanni Iacca (University of Trento) |
| | dr. Matt Coler (Rijksuniversiteit Groningen) |

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

# Evolution of Biologically Inspired Learning in Artificial Neural Networks

Anil Yaman

Keywords: Biologically Inspired Learning, Plasticity, Plastic Artificial Neural Networks, Evolutionary Computing, Neuroevolution, Hebbian Learning

This research has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No: 665347.

SIKS Dissertation Series No. 2019-33
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.


Cover image: Neurogenesis I by Greg Dunn, 2018, www.gregadunn.com
Cover design by Anil Yaman, Simon van der Zon and Onur Çaylak
Printed by ipskampprinting.nl

A catalogue record is available from the Eindhoven University of Technology Library
ISBN: 978-90-386-4891-0

# Acknowledgments

I have been very fortunate to fill past four years with extremely valuable experiences that helped me to grow personally and professionally. During this time, I have interacted with and learned from many brilliant people. I would like to take this chance to express my gratitude.

I have had the unique opportunity to involve in the Phoenix project, and simultaneously pursue my PhD degree. This position has encouraged me to collaborate with a large number of researchers from various research fields. For making this job opportunity possible for me, I would like to thank Heinrich Wörtche, Matt Coler, Giovanni Iacca, and all the researchers involved in the writing and submission process of the Phoenix project, and the European Union's Horizon 2020 research and innovation programme for the funding.

I was also very fortunate to have multiple advisors who helped me to gain insights from multiple perspectives in my research and personal life. I am deeply grateful for their valuable advices, guidance and constant support during this process.

I would like to thank my promoters George and Mykola for supporting me in my research and allowing me to pursue my ideas which helped me greatly to gain confidence and develop my future research. I am also grateful to them for sharing their experiences and knowledge.

I am grateful to Decebal for his constant encouragement and guidance in my research and in all aspect of the process of writing this thesis, graduation and planing the next step of my academic career. Many thanks also for creating opportunities for giving lectures and talks in several occasions which helped me improve a lot.

I am grateful to Giovanni for his close involvement in my research. I appreciate for always having the time for a discussion and sharing his deep knowledge and insights. I have learnt a lot during several organizational activities in con-

ferences and workshops. I would like to express my gratitude for generating these opportunities.

I am grateful to Matt for his supervision in the Phoenix project. I have learnt a lot from his feedback regarding my research, and especially about communicating and presenting my ideas and results. I also appreciate numerous discussions we engage regarding the topics in linguistics and philosophy.

I would like to thank my committee members Nicolas Bredeche, Gusz Eiben and Heinrich Wörtche for reviewing the thesis and providing their valuable feedback.

I have learnt enormously during the activities of the Phoenix project. I would like to thank Matt and Giovanni for their close collaboration, guidance and encouragement. I would like to thank Peter Baltus and Jan Haagh for their support, organization and management of the project. I would like to thank Eloisa Kompier for her contribution in designing a visual interface. Furthermore, I would like to thank all researchers, Erik Duisterwinkel, Gonenc Berkol, Ahmed Hallawa, Haoming Xin, Jaro De Roose, Karine Miras and Stephan Schlupkothen, Sina Sadeghpour, Martin Andraud, Dennis Grob, Dirkjan Krijnders, Eugenio Cantatore, Elena Talnishnikh, Gerd Ascheid, Gusz Eiben, Heinrich Wortche, Marian Verhelst, Pieter Harpe, Bob Puers that I had the privilege of collaborating during the Phoenix project. I would like to thank Léon Verhoeven for his collaboration in this project.

I would like to thank fellow PhD researchers and my first office mates Hedde Bosman and Erik Duisterwinkel when I first arrived in the Netherlands and started working at INCAS$^3$. I would also like to thank all research and support staff who I worked with at INCAS$^3$ for their warm welcome and help in settling in the Netherlands.

I would like to thank all my colleagues at the Eindhoven University of Technology for improving the quality of life during my stay in the Netherlands by offering their companionship in work and social life. I will most certainly miss the times when we worked, had discussions at the coffee corner, gathered during the lunch time and walked down from the stairs, organized social events, participated in sportive activities and had fun.

I would like to thank to PhD students, Alejandro Montes García, Jianpeng Zhang, Yulong Pei, Rafael Mantovani, Julia Efremova, Julia Kiseleva, Alexandr Maslov for their warm welcome when I have first joined the group. Many thanks to Jianpeng and Yulong for the discussions during our combined meetings. Thanks to Jianpeng, Rafael, Alejandro, and other PhD students, Yuhao Wang, Ricky Fajri, Masoud Mansoury, Pieter Gijsbers, Wilco van Leeuwen, Kaijie Zhu, and Ghada Sokar for sharing the office during these four years.

I would like to thank all PhD students, Tim van der Lee, Shiwei Liu, Ghada, Zahra Atashgahi, and guest researcher Davide Ferraro for interesting discussions during our AI meetings with Decebal.

Special thanks to Simon van der Zon for teaching me about interpretable machine learning and offering his suggestions for the content of my public talk. Many thanks also for his active role in social interactions, for having always the time to have a drink and a discussion, organizing seminars and discussion platforms, his involvement in many social and sportive activities.

I am grateful to Onur Çaylak for his constant support and encouragement, lecturing me in physics and mathematics, and always coming up with interesting problems that generate great opportunities for collaboration. Many thanks also to Björn Baumeier for his support and guidance in these collaborative works.

I would like to thank Vlado Menkovski for numerous discussions about research, science and life in general. I have appreciated his opinions and advices during these discussions. I would also like to thank him for introducing me to bouldering which was a great activity to clear my mind. I would like to thank Tim, Simon, Bilge Çelik, Pieter, Xin Du for participating in our bouldering sessions.

I would like to thank Wouter Duivesteijn for the lunch gatherings and social organizations where we engaged in many interesting discussions.

I would like to thank my colleagues Nikolay Yakovets, Natasha Stash, Joaquin Vanschoren, Odysseas Papapetrou, Ekaterini Ioannou, Samaneh Khoshrou, Xuming Meng, Loek Tonnaer, Sibylle Hess, Iftitahu Ni'Mah, Marijn van Knippenberg, Hamid Shahrivari Joghan, Hao Li, Alvaro Correia, Larissa Capobianco, Cassio de Campos, Negar Ahmadi, Lu Yin, Tianjin Huang, Sahithya Ravi, Prabhant Singh, Afrizal Doewes, Munehiro Kitaura, Thomas Mulder, Siddharth Ravi, and Oren Zeev Ben Mordehay for numerous discussions on variety of topics. I would like to thank Riet van Buul, José Jong, Ine van der Ligt and Margot van den Heuvel for their support and assistance. Many thanks to José for proofreading an earlier version of this thesis and providing her feedback. It was a privilege to work within the same department.

I would like to thank Fabio Caraffini for sharing his experience and knowledge in nature inspired search approaches, and for his contributions and support in several collaborative works.

I have learnt a lot by coaching the bachelor students in Honor's program organized by the Honor's Academy at the Eindhoven University of Technology. I would like to thank Marwan Hassani for the opportunity and his support during this program. I would also like to thank all students and coaches involved in

the program.

I have had an amazing experience while I was visiting Korea and Japan during my last year. I would like to thank Sang Wan Lee, JeeHang Lee, Takashi Ikegami and all the members in their groups for their hospitality. I would like to thank Mykola for his support for these research visits. I would like to thank Onur for accompanying me and sharing the experience during these visits.

I would like express my deep gratitude for my family for always being there and supporting me in all my decisions. I have always had the best moments when I go back home and stay or travel with you.

I would like to thank all my advisors, colleagues, family and friends for the discussions during the period of deciding my next step of my academic career. I would especially like to thank Mykola and Giovanni for their support during my job applications.

Special thanks to Greg A. Dunn for allowing the use of his wonderful artwork "Neurogenesis I" as the cover of this thesis. Many thanks to Simon and Onur for their help in the design of the cover. I would like to thank Joos Buijs for generating this thesis template and allowing for the public use.

Anil Yaman
Eindhoven, October 2019

# Summary

Biology is one of the main sources of inspiration in producing intelligence in artificial systems, and stimulated a broad area of research in nature-inspired computing. Among these approaches, artificial neural networks (ANNs) aim to model the information processing behavior of biological neural networks (BNNs), and evolutionary computing aims to design software and hardware by mimicking the working principles of biological evolution. Inspired by the evolutionary process of biological systems, the research field known as *Neuroevolution* employs evolutionary computing to optimize ANNs.

One of the key aspects in Neuroevolution is the approach used for encoding the ANNs. In the case of *direct encoding*, the topology and/or weights of the ANNs are directly represented within the genotype of the individuals. However, this approach may not be biologically plausible considering the amount of the information required to encode possible configurations of BNNs with a relatively small number of genes present in the genotype. Moreover, searching this large number of possible network configurations using direct encoding is likely to lead scalability issues. In addition to that, the ANNs optimized with this approach do not exhibit lifetime learning capabilities. A complementary approach, known as *indirect encoding* on the other hand, aims to find optimum rules to develop or learning mechanisms to train ANNs during their lifetime.

A fundamental property of BNNs is their plasticity, which allows them to modify their internal configurations during their lifetime. According to the current physiological understanding, these changes are performed on synapses (connections between two neurons) based on the local interactions of the neurons. This form of learning may lead to a high level of adaptation (as observed in BNNs) due to its distributed and self-organized nature. However, further research is needed to understand the emergent of a coherent learning behaviour from local learning rules.

Hebbian learning has been proposed to model the plasticity property in ANNs. According to the basic formalism of Hebbian learning, the synaptic efficiency between two neurons is increased/decreased depending on the correlations of their activations. However, this formalization is likely to suffer from instability as it introduces an indefinite increase/decrease of the synaptic efficiencies. Therefore, the plasticity rules may require further optimization to properly capture the dynamics needed for adjusting the network parameters.

A number of previous works suggested optimizing the parameters of some form of Hebbian plasticity rule using evolutionary approaches. Others suggested replacing Hebbian rules with an additional ANN to perform synaptic changes. However, these approaches may involve a high degree of complexity which may not be able to deliver insights into the dynamics of the learning process with plasticity property. Furthermore, some of the existing works evolve the initial synaptic weights and/or the connectivity of the networks in addition to the plasticity rule. However, evolving the initial synaptic weights of the networks increases the number of parameters to evolve and, in principle, can be decoupled from the evolution of plasticity rules (except for tasks where there is a need to adapt to changing conditions); on the other hand, evolving the connectivity of the networks may overfit the networks to a certain task, making it difficult to exhibit adaptive behavior.

In this thesis, we investigate direct and indirect aspects of neuroevolution. In the case of direct encoding, we propose limited evaluation and cooperative co-evolution schemes to help scale evolutionary approaches to optimize ANNs with large number of parameters. In the case of indirect neuroevolution, we propose an evolutionary approach to producing plasticity property in ANNs to facilitate learning during the lifetime of the networks. We employ genetic algorithms to evolve discrete learning rules that are capable of performing synaptic changes locally based on the pairwise binary activation states of neurons. Our evolved plasticity rules make it easy to understand how the synaptic changes are performed depending on the all possible pairwise binary activation states of the neurons.

We study plasticity in three kinds of learning processes. First, the reinforcement signals are available after every action of the networks. Second, the reinforcement signals are received after a certain period of time and finally, in the case where the reinforcement signals are not available. We test evolved plasticity rules on foraging and maze tasks, and show that they are capable of training networks for these tasks.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Biological systems and organisms are main sources of inspiration in approaches to producing/mimicking intelligence in artificial systems [19]. Among these approaches, artificial neural networks (ANNs) are computational models inspired by the information processing behavior of biological neural networks (BNNs).

Conventionally, ANNs consist of a number of artificial neurons organized within a certain connectivity pattern that processes information through neuron activations and map the inputs to desired outputs [34]. The goal is to find optimum parameter settings of an ANN (topology, connectivity and their weights) that can map maximum number of inputs to desired outputs correctly without overfitting [34]. To achieve that, mathematical optimization approaches (i.e. stochastic gradient descent) are used to adjust the connection weights during the training phase. Consecutively, the optimized model is used without making any changes in its configuration (unless some form of change detection algorithm is used, and the model is retrained). Although, conventional training and optimization approaches are successfully applied in many domains, they are often regarded as biologically implausible [51].

## 1.1 Evolution of Artificial Neural Networks

We can discuss three levels of organization observed in nature. These levels are referred as the evolutionary (phylogenetic), developmental (ontogenetic) and lifetime (epigenetic) [90] as depicted in Figure 1.1. The evolutionary level is concerned with the evolutionary process of species which operates over gen-

erations and allows organisms to adapt their environment through variation and selection. The behaviors produced through the developmental and lifetime learning processes contribute to the fitness of the agent and provides feedback to the selection mechanism. The developmental level refers to the developmental process of a multi-cellular organism starting from a single cell. The environmental factors can have an influence in the developmental process of the individuals, however, the process is mainly instructed by the genetic code. Finally, the lifetime level refers to the changes that occur during an individuals lifetime, usually in response to the behaviors of the individuals, and allows them to adapt/learn.

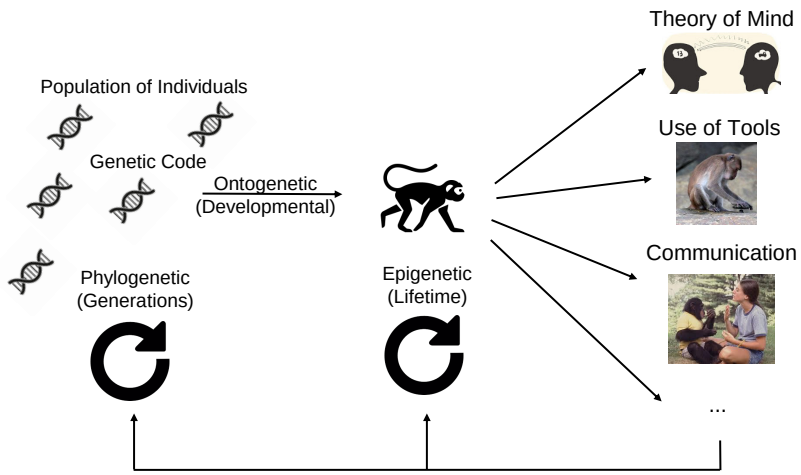



Figure 1.1: Levels of organization in nature.

Inspired by the evolutionary process of BNNs, the field of research known as Neuroevolution employs Evolutionary Computing (EC) to optimize ANNs [26, 96, 119]. The approaches used in neuroevolution can roughly be grouped into two groups that are referred as: *direct* and *indirect encoding* [26]. In direct neuroevolution, the parameters of ANNs (topology, connectivity and their weights) are directly represented into the genotype of the individuals [117]. In indirect encoding on the other hand, some form of rules and/or learning algorithms are encoded into the genotype to develop/optimize ANNs during their lifetime [48, 114, 116]. It is not a straightforward task to optimize the parameters

of the networks directly by using direct encoding, because the number of possible configuration of networks increases exponentially depending on the number of neurons making the optimization task difficult to scale. This drawback may be overcome by indirect encoding approaches, since the number of parameters optimized is usually considerably less then the actual parameters of the ANN models.

Biological evidence indicates that the configuration of BNNs are not directly specified by the genotype of the individuals [48]. Rather, due to their property referred as "plasticity property", they are learnt during their lifetime by changing their configuration in response to their interactions within the environment. According to the current physiological understanding, the plasticity property arises due to the synaptic modifications performed locally in individual synapses (connections) [25, 102]. Moreover, the reinforcement signals received based on these interactions appear to modulate the learning process. These modifications cause increase/decrease in the efficiency of neurons activating one another to converge on a global network behavior. Hebb [35] proposed a learning mechanism known as Hebbian learning/rule to model the plasticity property in ANNs. According to this rule, a synaptic efficiency between two neurons is strengthen based on the correlations of their activations.

## 1.2 Problem Formulation

Evolutionary computing plays an important role in optimizing ANNs in the cases of direct and indirect encoding. The behavior of the EC algorithms is controlled by its parameters (i.e. types of evolutionary operators and their parameters) to balance the trade-off between exploration and exploitation during an evolutionary search process. Therefore, it is crucial to find the parameter settings that can perform an evolutionary search process as efficient as possible.

There are two bottlenecks that make it difficult to scale evolutionary approaches for optimizing ANNs [117] using direct neuroevolution. These bottlenecks are: the dimensionality of the problem because classic evolutionary algorithms tend not to scale well for searching large parameter spaces; and the evaluation phase which is time-consuming because it takes long time to evaluate large number of samples to assign the fitness values. Therefore, it is required to develop approaches for these two bottlenecks to scale direct neuroevolution to optimize ANNs with large number of parameters.

The plasticity property allows networks to learn during their lifetime. When plasticity is performed based on local interactions of the neurons, it is likely

to lead to a distributed and self-organized form of learning. Hebbian learning is a promising approach to produce plasticity property in ANNs. However, it is likely to suffer instability since it's basic formalism can introduces indefinite synaptic increase [103]. Therefore, it is crucial to find/optimize plasticity rules and/or higher order regularization mechanisms to provide stable and autonomous learning in ANNs [116].

Reinforcement signals play a crucial role in a Hebbian learning process because they provide feedbacks to make it more/less likely to produce behaviors that lead to rewards/punishments. However, if the reinforcement signals are available only after a certain period of time (episode), based on a sequence of action steps, it becomes challenging to associate the neuron activations to the reinforcement signals. This problem is known as "distal reward problem" which makes it impossible to perform changes in the configuration of ANNs straightforwardly [43, 94, 114]. Therefore, to perform synaptic changes based on delayed reinforcement signals after a certain period of time, it is required to associate the activations of the neurons during this period with the delayed reinforcement signals.

In some learning tasks, the reinforcement function can be extremely sparse, providing only a binary reward/punishment signal when a correct/incorrect behavior is performed. In this case, there would be no way of distinguishing between two unsuccessful behaviors to guide the search process due to the binary reinforcement signal that distinguishes only a successful/unsuccessful behavior. This makes it impossible to search the possible behavior space. We refer to this problem as the "needle in a haystack" problem where *needle* refers to the behavior that can solve the task, *haystack* is the search space and there is no fitness value available unless the behavior that solves the task is found [36].

## 1.3 Research Questions and Research Approach

We address the challenges outlined in Section 1.2 with the research questions and approaches provided below. The research and the developments of the methods proposed for these research questions contribute in the understanding of learning, memory and problem solving capabilities of BNN inspired information processing approaches, which is crucial in mimicking their behavior in artificial systems. This goal may allow the development of the control/decision making systems that exhibit autonomous and continual learning capabilities.

Q0: How to find (near-)optimum parameter settings of the EC algorithms to

perform an efficient evolutionary search?

In order to gain knowledge in the search behavior of evolutionary computing algorithms, we conduct research in two main approaches to parameter setting problem in EC to contribute to Q0 [33, 110–112]. First, we aim to make use of the existing knowledge in parameter setting problem in the literature. There is a great deal of theoretically and empirically established knowledge in the literature that suggests optimum parameter settings for certain types of problems. We propose to make use of this knowledge by collecting it from the literature and formalizing it to be used in algorithms for solving problems [110]. Second, if there is no prior knowledge for setting the parameters of the algorithm, it is required to set the parameters either arbitrarily, or tuning them before or during an evolutionary process. Since different problems may require different optimum parameter settings, which may also change during an evolutionary process, we propose a self-adaptive parameter control strategy to adapt the algorithm parameters during an evolutionary run [112]. We propose a co-evolutionary approach inspired by artificial immune system algorithms to co-evolve the solutions and algorithm parameters in two populations. We discuss a summary of these works in Chapter 2 and provide the results of the self-adaptive parameter control approach in Appendix A.

Q1: How to address two bottlenecks in direct neuroevolution to help scale evolutionary approaches to optimize ANNs with large number of parameters?

To address the two bottlenecks in direct neuroevolution, we propose using the limited evaluation (LE) and cooperative co-evolution (CC) schemes. The LE scheme aims to perform evaluations on smaller number of samples to reduce the computing time during the evaluation process, and the CC scheme aims to split the parameters of a large-scale optimization and evolve them separately in subpopulations. These two approaches are used to evolve the connection weights of large feedforward ANNs using direct encoding [117]. We present the results of our approach in Chapter 3.

Q2: How to define local learning rules to introduce plasticity property in ANNs, and find/optimize the rules that can achieve global autonomous learning behavior?

We propose performing synaptic changes in each synapse based only on the local activation of pre- and post-synaptic neurons (antecedent and subsequent neurons in respect to a synapse) in an ANN and a reinforcement signal that is

received after every action [116]. In order to limit the possible combination of neuron activations, we use a binary activation function to allow the neuron activations to be either one or zero. Using this approach, we are able to represent the activation states of two neurons using two bits which can provide one of four possible activation states. Similarly, we binarize the reinforcement signals to limit the possible number of external feedback deceived from the environment and use $-1/+1$ to indicate whether an ANN performed desired/undesired action. One additional bit is included into the representation to indicate the state of the reinforcement signal. Thus, we designed the synaptic plasticity rules to provide a synaptic update decision for each one of these eight possible states. We use three possible synaptic update decision as: $-1/0/+1$ to indicate decrease/stable/increase in the synaptic weights for each of these eight possible states. Consequently, we arrive at $3^8$ possible distinct synaptic update rules in total. In addition, we include a continuous value to the synaptic rules to indicate the learning rate which adjusts the magnitude of the synaptic change.

We employ genetic algorithms to find the synaptic plasticity rules that can perform learning in feedforward ANNs on a foraging task. We encode the synaptic update rules using a nine dimensional strings (one for the learning rate and eight for all the possible states of the binary activations of pre- and post-synaptic neurons and a binary reinforcement signal) that is used as the genotype of the individuals. We introduce seasonal changes in the foraging task to assess the adaptation capabilities of the ANNs trained by the evolved synaptic plasticity rules. We provide the results of our approach in Chapter 4.

Q3: How to associate the neuron activations in an ANN with the delayed reinforcements to allow learning in the distal reward cases?

We propose keeping track of the pairwise activations of all pre- and post-syanptic neurons within an ANN to associate a delayed signal with the neuron activations [114]. We introduce a data structure we refer as the neuron activation traces (NATs) in each synaptic connection to record the frequency of four possible activation states of neurons. At the end of a process (when a delayed reinforcement is received), we binarize the NATs using a threshold to reduce their number of possible states. We replace the synaptic update rule representation with the NATs to perform synaptic updates based on this data structure. Thus, the delay synaptic changes are performed based on the binary states of the NAT states and a binary reinforcement signal adding up to $2^5 = 32$ possible synaptic state that can result one of three possible synaptic updates as decrease/stable/increase. Consequently, there are a total of $3^{32}$ possible synaptic update rules.

We extend the genotype representation of the genetic algorithm to encode the delayed synaptic plasticity rules, and use genetic algorithms to find the rule to allow learning in maze navigation task. We include four continuous parameters into the genotype for the learning rate, threshold for binarizing the NATs, and two parameters for the networks. We use recurrent neural networks (RNNs) to control the agents. The results of our approach are given in Chapter 5.

Q4: How to introduce learning in the cases of needle in a haystack problem where there is no reinforcement signals to guide the learning process?

Since there is no reinforcement signal in the case of the needle in a haystack problem, we propose novelty producing synaptic plasticity (NPSP) where the plasticity rules are optimized to generate as many novel behavior as possible for a certain number of episodes to find the behavior that can solve the task [113]. The NATs are used to provide information about the neuron activations to make the synaptic changes that can lead to a novel behavior. We use genetic algorithms to find NPSP rules with a representation similar to the representation used for finding delayed synaptic plasticity rules, except the binary reinforcement signal was not included into the genotype. Thus, in the NPSP, there are in total of $2^4 = 16$ possible states of the NATs and each can take one of three synaptic update decisions as $-1/0/+1$ (decrease/stable/increase). Consequently, there are $3^{16}$ possible NPSP rules. We test the learning capabilities of the RNNs with NPSP on a complex deceptive maze tasks where the agent is required to learn a complex sequence of action to achieve the goal state.

## 1.4 Main Contributions and Thesis Outline

Several works that contribute to parameter setting problem in evolutionary computing are summarized in Chapter 2. First, there are a large collection of works that study optimal parameter settings for types of problems. This knowledge is rarely put into use in algorithms. We propose to collect this knowledge and make it available for reuse in similar problems [110]. Second, we propose a self-adaptive approach to tune algorithm parameters during an evolutionary process [112]. Most of the self-adaptive parameter control approaches in the literature aim to adapt algorithm parameters by including them into the genotype of the individuals. In our approach, we propose a co-evolutionary approach where an ensemble of search strategies co-evolve with the candidate solutions.

We contribute to the field of neuroevolution with direct encoding to scale evolutionary optimization approaches to optimize ANNs with large number

of parameters [117]. Some works used CC scheme in neuroevolution. For instance, the Symbiotic Adaptive Neuroevolution (SANE) aims to evolve two populations, one for neurons and another for the network "blueprints". The blueprints are used to specify which neurons to use to construct an ANN [64]. The Enforced SubPopulations (ESP) evolves all incoming and outgoing weights of each neuron in a subpopulation [31], and Cooperative Synapse Neuroevolution (CoSyNE) initiates a subpopulation for each connection [30]. With respect to these works, we considers the post-synaptic neurons as the building blocks of an ANN, thus, evolve all incoming (pre-synaptic) weights of each neuron by representing them in a separate subpopulation. Moreover, we employ a limited evaluation scheme to reduce the time during the fitness evaluation phase. Our results show that the LE scheme, while reducing the time required for the fitness evaluation, used with the CC achieves better accuracy results than a standard differential evolution algorithm for optimizing the parameters of ANNs. These results are presented in Chapter 3.

Some of the previous work suggested optimizing the parameters of a form of Hebbian based plasticity rule that is defined as an equation to specify synaptic updates based on the activations of pre- and post-synaptic neurons [27, 69, 91]. Others, suggested using separate ANNs to perform synaptic updates [72, 81]. In contrast, we propose to evolve simple discrete rules that perform synaptic updates based on the pairwise binary activation states of pre- and post-synaptic neurons. In addition, we introduce synaptic competition in pre-synaptic connections promote self-organization [116]. Moreover, some of the previous works evolve the initial synaptic weights and/or the connectivity of the networks with the the plasticity rule. In our case, we only evolve plasticity rules and assess their capability of training randomly initialized ANNs. Furthermore, we introduce environmental changes that the ANNs encounter during their lifetime and assess their adaptation capabilities. Our results, provided in Chapter 4, show that the evolved plasticity rules are capable at adapting ANNs effectively under changing environmental conditions.

In the context of time dependent plasticity, eligibility traces are proposed to keep track of neuron activations in the case of distal reward problem to associate delayed reinforcement signals [28, 43, 94]. In Chapter 5, we extend the discrete synaptic plasticity rules proposed in Chapter 4, and introduce the neuron activation traces to keep track of pairwise binary neuron activations over a certain period of time to associate delayed reinforcement signals in discrete time recurrent neural networks [114]. In addition, we introduce a novel way of providing the reinforcement signals by using positive or negative reinforcement signals that are provided based on the performance of the agent relative to the previ-

ous episode. We test our evolved synaptic plasticity rules on a maze navigation task and show that the results of the RNNs, trained using the evolved delayed synaptic plasticity, achieve better results in a smaller number of episodes then when they are trained using the Hill Climbing algorithm.

In Chapter 6, we introduce a novel approach we refer as the novelty producing plasticity to allow learning in tasks where there is no reinforcement signals to guide the learning process. We refer to this problem as the "needle in a haystack" problem. To best of our knowledge, the NPSP is the first attempt to perform synaptic updates to produce novel behaviors based on the neuron activations. Our results show that the number of novel behavior produced by the NPSP rules is greater than the number of novel behavior produced by the random search and random walk algorithms.

Finally in Chapter 7, we discuss conclusions, limitations and future work.

# Chapter 2

# Background

In this chapter[1], we provide a brief introduction to some of the topics that reappear throughout this thesis. These topics include evolutionary computing, artificial neural networks, and neuroevolution.

## 2.1 Evolutionary Computing

Evolutionary computing is an umbrella term for a collection of meta-heuristic search algorithms inspired by the evolutionary process in nature [13, 24, 29]. A pseudocode of an evolutionary algorithm (EA) is shown in Algorithm 1.

[1] This chapter is partially based on:
[112] Anil Yaman, Giovanni Iacca, Matt Coler, George Fletcher, and Mykola Pechenizkiy. Multi-strategy differential evolution. In Kevin Sim and Paul Kaufmann, editors, *Applications of Evolutionary Computation*, pages 617–633, Cham, 2018. Springer International Publishing
[110] Anil Yaman, Ahmed Hallawa, Matt Coler, and Giovanni Iacca. Presenting the eco: Evolutionary computation ontology. In Giovanni Squillero and Kevin Sim, editors, *Applications of Evolutionary Computation*, pages 603–619, Cham, 2017. Springer International Publishing
[111] Anil Yaman, Giovanni Iacca, and Fabio Caraffini. A comparison of three differential evolution strategies in terms of early convergence with different population sizes. In *LeGO 2018 - International Workshop on Global Optimization, 18-21 September, Leiden, The Netherlands*, 2018
[11] Onur Çaylak, Anil Yaman, and Björn Baumeier. Evolutionary approach to constructing a deep feedforward neural network for prediction of electronic coupling elements in molecular materials. *Journal of Chemical Theory and Computation*, 15(3):1777–1784, 2019. PMID: 30753071
[33] Ahmed Hallawa, Anil Yaman, Giovanni Iacca, and Gerd Ascheid. A framework for knowledge integrated evolutionary algorithms. In Giovanni Squillero and Kevin Sim, editors, *Applications of Evolutionary Computation*, pages 653–669, Cham, 2017. Springer International Publishing

**Algorithm 1** A pseudocode of an Evolutionary Algorithm

1: **procedure** EvolutionaryAlgorithm($n, p_c, p_m$)
2: $\boldsymbol{X} \leftarrow initialize(n)$ ▷ randomly initialize $n$ number of individuals each encoding a candidate solution to a problem
3: $F \leftarrow evaluate(\boldsymbol{X})$ ▷ $f_i \in F$ for each $x_i \in X$
4: **while** termination criterion is not satisfied **do**
5: $\boldsymbol{X}' \leftarrow select(\boldsymbol{X}, F)$
6: $\boldsymbol{X}'' \leftarrow crossover(\boldsymbol{X}', p_c)$
7: $\boldsymbol{X} \leftarrow mutate(\boldsymbol{X}'', p_m)$
8: $F \leftarrow evaluate(\boldsymbol{X})$
9: **end while**
10: **end procedure**

EAs operate on a population of individuals that encode solutions to a problem. Taking the terminology from biology, the representation scheme of the individuals is called as the *genotype*, an individual solution is referred as *genetic string* (a.k.a. *chromosome*), and each component (dimension) within a genetic string is called a *gene*.

In EAs, a population of individuals are initialized randomly. Each individual is evaluated to assign a *fitness value* to measure how well they solve the problem. The main part of the algorithm performs *selection* and *reproduction with variation* that aims iteratively to improve the fitness values of the solutions. The selection operator functions as a sort of a filter by selecting the individuals with better fitness values to construct next generation of individuals. One of the most commonly used selection operator, known as *roulette wheel selection*, selects individuals with a probability proportional to their fitness values. The stochasticity of the selection process may occasionally cause the best individuals to disappear from the population. Therefore, *elitist* selection scheme is introduced to transfer top $k$ best ranked individuals directly to the next generation during the selection procedure.

Biologically inspired evolutionary operators (i.e. *crossover* and *mutation*) are used to construct the next generation from existing solutions with variation. The algorithm is run for a certain number of iterations or until a satisfactory solution is found. It is also possible to define a stopping criterion to check whether there is a fitness improvement made for a certain of generations.

There are different types of evolutionary algorithms. Each type has its own representation scheme and evolutionary operators. Genetic Algorithms

(GA) [38] represent individuals as binary strings and use crossover and mutation operators. In Genetic Programming (GP) [49], candidate solutions are represented by tree-structured programs; and crossover and mutation operators are used. Evolution Strategies (ES) [80, 86] use real-valued representations and mainly the mutation operator. Differential Evolution (DE) [99] is a continuous optimization algorithm that uses real-valued representation and employs crossover and mutation operators. In Sections 2.1.1 and 2.1.2, we discuss genetic and differential evolution algorithms respectively.

**Parameter Setting in EAs**

As it is the case for all search algorithms, the selection of the parameters in EAs (i.e. population size, type of selection operator, type of crossover operator and its probability, mutation probability) play an important role in the search process [23, 45, 50]. Eiben *et al.* categorized the parameters setting problem into two main categories, parameter tuning and parameter control [23]:

1. *Parameter tuning* aims to find the appropriate parameter settings offline, before an evolutionary run. The parameter tuning process can be performed by trial and error, from studies in the literature [20, 110], or by using settings of similar problems [33].

2. *Parameter control* on the other hand, aims to adjust the parameter settings during an evolutionary process because the goodness of a parameter setting varies depending on the state of the search [15]. *Deterministic*, *adaptive* and *self-adaptive* methods have been proposed for the parameter control task [23].

Parameter setting in EAs is one of the main research topics that provides a fundamental knowledge in performing an efficient search. We contribute to this topic with a number of works outlined below:

1. In literature, there is a large collection of works that theoretically and empirically study optimum parameter settings for certain types of problems. We aim to collect and formalize this knowledge to establish a link between algorithm settings and problem types [110].

2. We propose a framework that aims to extract features of the problem type based on the distribution of the population during an evolutionary search to associate these features with empirically established optimum parameters settings [33].

3. We compare three different strategies and various population sizes in differential evolution algorithm to analyse their behavior [111].

4. We propose a self-adaptive parameter control approach where we initialize strategies and their parameters in a separate population and co-evolve with the solutions [112]. The details of this work can be found in Appendix A.

### 2.1.1 Genetic Algorithms

Genetic Algorithms are a popular branch of EAs usually applied to discrete optimization problems. In a standard genetic algorithm [29, 38], candidate solutions are represented as fixed length binary strings.

The crossover operator in GAs generates two offspring from two parents by exchanging sets of genes. There are various crossover operators suggested in the literature [24]. Figure 2.1 illustrates one of the well-known crossover operator, one-point crossover, which exchanges the sets of genes of two parents starting from a randomly selected crossover point $r$.

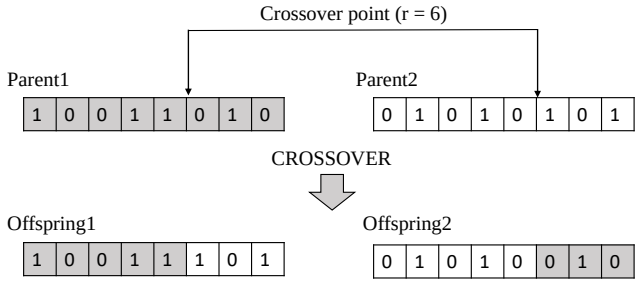



Figure 2.1: One-point crossover operator in genetic algorithms.

The mutation operator, illustrated in Figure 2.2, is used for exploration where genes in an offspring chromosome is flipped with a small probability.

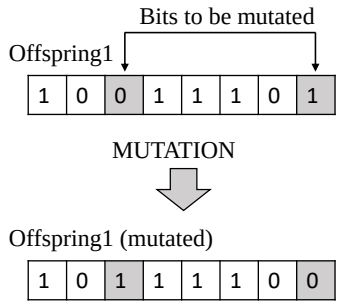



Figure 2.2: Mutation operator in genetic algorithms.

### 2.1.2 Differential Evolution

The DE algorithm is a population-based search algorithm proposed for continuous optimization [99]. A candidate solution set $\{x_1, x_2, \ldots, x_{NP}\}$ with a population size of $NP$ is represented as $D$-dimensional real-valued vectors $x_i \in \mathbb{R}^D, i = 1,2,\ldots,NP$. In the initialization phase of the algorithm, the candidate solutions are randomly sampled within the domain boundaries of each dimension $j = 1,2,\ldots,D$.

The algorithm employs a strategy composed of a mutation, crossover, and selection operators with their specified parameters. For each generation $g$, a candidate solution $x_i^g$, called *target vector*, is selected $\forall i \in \{1,2,\ldots,NP\}$. The mutation, crossover and selection operators are then applied to generate a *trial vector* $u_i^g$, and replace the target vector. The *mutation operator* generates a *mutant vector* $v_i^g$ by perturbing the target vector $x_i^g$ using the scaled differences of several distinct individuals selected randomly from the population. The *crossover operator* generates a trial vector $u_i^g$ by performing recombination between the target vector and the mutant vector. The *selection operator* replaces the target vector $x_i^g$ in the population with the trial vector $u_i^g$ if the fitness value of $u_i^g$ is better than or equal to $x_i^g$. This process is iteratively executed until a stopping criteria is met.

The mutation operator is controlled by the parameter *scale factor* ($F$) that is used to adjust the magnitude of the perturbation. There are various mutation operators suggested in the literature [17,68]. Four types of mutation strategies, referred as *"DE/rand/1"*, *"DE/rand/2"*, *"DE/rand-to-best/2"*, and *"DE/current-*

*to-rand/1*" are provided in Equations (2.1), (2.2), (2.3), and (2.4), respectively, see [77].

$$v_i^g = x_{r_1}^g + F \cdot (x_{r_2}^g - x_{r_3}^g) \tag{2.1}$$

$$v_i^g = x_{r_1}^g + F \cdot (x_{r_2}^g - x_{r_3}^g) + F \cdot (x_{r_4}^g - x_{r_5}^g) \tag{2.2}$$

$$v_i^g = x_i^g + F \cdot (x_{best}^g - x_i^g) + F \cdot (x_{r_1}^g - x_{r_2}^g) + F \cdot (x_{r_3}^g - x_{r_4}^g) \tag{2.3}$$

$$v_i^g = x_i^g + K \cdot (x_{r_1}^g - x_i^g) + F \cdot (x_{r_2}^g - x_{r_3}^g) \tag{2.4}$$

where $r_1, r_2, r_3, r_4$, and $r_5$ are mutually exclusive integers different from $i$, and selected randomly from the range $[1, NP]$; the parameter $K$ is a random number uniformly sampled in $(0, 1]$; $x_i^g$ is the target vector; $x_{best}^g$ is the best individual at generation $g$ in terms of fitness.

The crossover operator is used to recombine the target vector and the mutant vector with a certain rate, $CR$, to generate a trial vector $u_i^g$. The *binomial (uniform) crossover* operator is given in (2.5). There are several more existing crossover operators such as the *exponential crossover* [75].

$$u_{i,j}^g = \begin{cases} v_{i,j}^g, & \text{if } rand([0,1)) \leq CR \text{ or } j = randi([1,D]); \\ x_{i,j}^g, & \text{otherwise.} \end{cases} \tag{2.5}$$

where $j$ is an integer within the range $[1, D)$, functions $rand()$ and $randi()$ return a real and an integer value uniformly sampled from a defined range, respectively. The notation $x_{i,j}^g$ refers to the $j$th dimension of $i$th vector in the population at generation $g$.

If the value of the trial vector along the $j$th dimension exceeds the boundaries defined as $x_j^{min}$ and $x_j^{max}$, it is randomly and uniformly sampled within the domain boundary range [77], using a toroidal boundary condition [40].

The selection operator determines whether or not the trial vector is kept for the next generation $g+1$. If the fitness value of the trial vector is better than or equal to the target vector, then the target vector is replaced by the trial vector as it is shown in Equation (2.6), which assumes a minimization problem:

$$x_i^{(g+1)} = \begin{cases} u_i^g, & \text{if } f(u_i^g) < f(x_i^g); \\ x_i^g, & \text{otherwise.} \end{cases} \tag{2.6}$$

The selection phase can be performed synchronously or asynchronously. In synchronous selection, the selected trial vectors are stored in a temporary set, and replaced with target vectors after the selection process of all individuals is complete. In asynchronous selection, a selected trial vector is replaced directly with the target vector without waiting the selection procedure for all individuals. Asynchronous selection makes it possible to use a newly generated trial vector in the trial vector generation process of all the remaining target vectors within the same generation.

The settings of the parameters in DE plays an influential role in the behavior of the algorithm for balancing the trade-off between the exploration and exploitation [16, 68]. Neri and Tirronen surveyed the existing works in the literature and performed an empirical analysis of the strategies and parameters in DE; more recently, Das *et al.* reviewed the works in the literature on the self-adaptive parameter control in DE [17].

## 2.2 Artificial Neural Networks

Artificial neural networks (ANNs) are biologically inspired computational models successfully applied to many machine learning problems [19]. They consist of a number of artificial neurons typically arranged as layers with specific connectivity referred as *topology*. Among the ANNs, *feed forward neural networks (FFNNs)* arranged in certain number of consecutive layers of neurons where each neuron in each layer is directionally connected to all neurons within the next layer. The connections between neurons referred as *synapses*, and the neuron with incoming connection is called *post-synaptic*, and the neuron with an outgoing connection is called *pre-synaptic* neuron in respect to the synapse. The activation of a neuron is computed as using the equation given in Equation (2.7).

$$a_i = \psi\left(\sum_{j=0} w_{i,j} \cdot a_j\right) \tag{2.7}$$

where $a_i$ and $a_j$ are the post- and pre-synaptic neurons, $w_{i,j}$ is the connection weight from the pre- to post-synaptic neurons, $a_0$ is the bias of the post-synaptic neuron (usually kept constant as 1). Two example activation functions $\psi$ known as *hyperbolic tangent sigmoid* and *binary* activation functions are provided in (2.8) and (2.9) respectively.

$$\psi(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.8}$$

$$\psi(x) = \begin{cases} 1, & \text{if } x > 0; \\ 0, & \text{Otherwise.} \end{cases} \tag{2.9}$$

A more general model of ANNs, known as Recurrent Neural Networks (RNNs), include recurrent and feedback connections. The recurrent connections in the RNNs introduce the activations of the neurons from previous time step. Therefore, they exhibit memory capabilities. In a matrix form, the activations of the hidden and output layers $\boldsymbol{A}_h(t+1)$ and $\boldsymbol{A}_o(t+1)$ are updated at time step $t+1$ as:

$$\boldsymbol{A}_h(t+1) = \psi\Big(\boldsymbol{W}_{hi} \cdot \boldsymbol{A}_i(t+1) + \alpha_h \cdot \boldsymbol{W}_h \cdot \boldsymbol{A}_h(t) + \alpha_o \cdot \boldsymbol{W}_{ho} \cdot \boldsymbol{A}_o(t)\Big) \tag{2.10}$$

$$\boldsymbol{A}_o(t+1) = \psi\Big(\boldsymbol{W}_{oh} \cdot \boldsymbol{A}_h(t+1)\Big) \tag{2.11}$$

where:
1) $\boldsymbol{W}_{hi}$ and $\boldsymbol{W}_{oh}$ are feed-forward connections between input-hidden and hidden-output layers, $\boldsymbol{W}_h$ is the recurrent connection of the hidden layer, and $\boldsymbol{W}_{ho}$ is the feedback connection from the output layer to hidden layer. The recurrent and feedback connections provide inputs of the activations of the hidden and output neurons from the previous time step.
2) $\alpha_h$ and $\alpha_o$ are coefficient used to scale the recurrent and feedback inputs from the hidden and output layers respectively.

The ANNs are trained on a training dataset to minimize the error between the *target* (actual labels of the input data) and *predicted* outputs. During the training process of the ANNs, the connection weights between neurons (a.k.a. model parameters) are adjusted to minimize the difference between the target and prediction values.

One of the conventional ways of training the ANNs is the backpropagation (BP) algorithm with stochastic gradient descend (SGD) [52]. However, the number of layers and the number of neurons per each layer should be defined before the training. These parameters are referred as *hyper-parameters* of the ANNs models, and play important role in the performance of the networks. Although there are some "rule of thumb" guidelines established based mainly on empirical studies, the selection of a proper set of hyper-parameter settings may require lots of expert knowledge and/or design process that may involve trial-and-error.

## 2.3 Neuroevolution

Inspired by the evolutionary process of biological systems, the research field known as *Neuroevolution* (NE) employs evolutionary computing approaches to optimize ANNs [26, 96, 119]. The main goal of this research is to propose algorithms that can produce artificial neural networks that can provide solutions for certain tasks. Frequently used network training algorithms such as: backpropagation [52, 82], requires an error function that is differentiable. Evolutionary algorithms on the other hand, do not require gradient information and there is no need for the error function to be differentiable [119]. Moreover, it is also possible to optimize the topology of the networks during an evolutionary process.

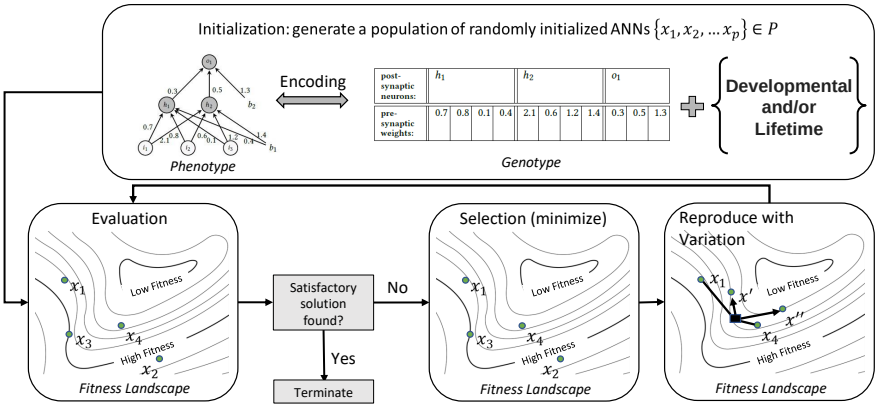



Figure 2.3: Neuroevolution: artificial evolution of artificial neural networks.

The evolutionary process of ANNs using Neuroevolution is depicted in Figure Figure 2.3. Adopting the terminology from biology, a population of individuals are represented as individual *genotypes* consisting of a finite number of *genes* to encode the parameters (i.e. topology, weights and/or the learning approaches) of the ANNs. In other words, individual genotypes can be considered as *blueprints* to construct different ANNs (i.e., corresponding *phenotypes*). Each individual is evaluated on a task, and is assigned a fitness value that measures

its performance. A *selection operator* is then used to select better individual —based on their fitness values— that will reproduce (i.e. generate new genotypes) by means of biologically inspired *crossover* and *mutation*. These allow a partially inheritance of genetic material, from parents to offspring, while also introducing variation [29]. By iterating this cycle over a certain number of *generations*, it is expected to find individuals that are better adapted to the task at hand.

One of the key aspects in NE is the approach used for encoding the ANNs. This, in turn, influences the so-called genotypes-phenotype mapping, i.e. the way a given genotype is used to build a certain phenotype [104]. Broadly speaking, there are two main kinds of encoding approaches: *direct* and *indirect* [26]. In direct encoding, the parameters of the networks (mainly weights and/or topology) are directly encoded into the genotype of the individuals [62, 117]; whereas, in indirect encoding, some form of specifications for development and/or training procedures are encoded into the genotype of the individuals [67, 70]. Based on biological evidence, indirect encoding approaches are biologically more plausible in terms of genotype-phenotype mapping, especially considering the large number of neurons and connections in BNNs and the relatively small number of genes in the genotype [48, 93]. Moreover, biological neural networks undergo changes throughout their lifetime without the need to change the genes involved in the expression of the neural networks.

### 2.3.1 Direct Encoding

Direct encoding is a representation type of an artificial neural network wherein the connection weights of the networks are directly specified in the genetic code of the individuals. One of the first examples of this method used genetic algorithms [5, 10, 106]. In standard genetic algorithms, each connection weight of an artificial neural network is represented by a bit string. The genetic code of an ANN is obtained by concatenating bit string representations of all the connection weights. Figure 2.4a illustrates binary encoding of an artificial neural network. Each connection is encoded by using 3 bits. There are methods that can be used for encoding real numbers with a given range and precision. However, they require prior knowledge on the range and precision of the real numbers. Moreover, the determination of the precision of binary representations becomes problematic because if the connection weights are represented using a few bits then the algorithm would not be able to find optimum weights; and if the weights are represented using many bits, then the length of the genetic code will be large that may cause the algorithm to converge slower [119].

It is possible to use real numbers directly without converting the binary representations (illustrated in Figure 2.4b). In this case, a real-valued vector is used for each genetic code. However, the classical evolutionary operators of crossover and mutation, cannot be applied. Montana and Davis [63] used this approach and designed evolutionary operators that can work with real numbers. They also compared their results with the backpropagation algorithm and found that the evolutionay algorthm performs better on the problems they studied.

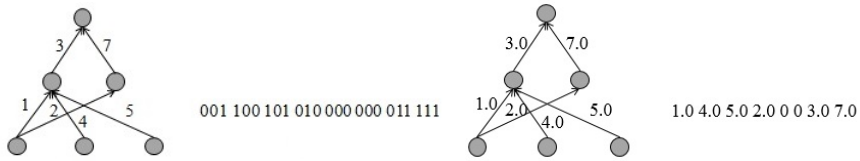



Figure 2.4: An example of a binary and real representation of an artificial neural network using direct encoding in Figures 2.4a and 2.4b respectively. In binary representation each connection is represented using 3 bits.

Another problem which is referred as competing conventions or the permutation problem. It is a situation where many different genotypes in the population decode the same artificial neural network. For example, if the hidden nodes of the ANN in Figure 2.4a are flipped, the genetic code will be different, however, the functionality of the networks would be the same. The result of the crossover operator of these two networks would have repeated structures and that would make it inefficient [2, 5, 85].

Angeline *et al.,* (1994) suggested an algorithm to evolve recurrent neural networks in order to reduce the effect of competing conventions and increase the continuity of the structural search [2]. They used evolutionary programming with a special structural mutation operator. Their structural mutation operator allows addition of new nodes or links; and deletion of existing nodes or links. Since the parameters and the structure of the networks would be optimized to some level, adding a link or deleting a link would radically change the behavior of the network, causing the fitness of the offspring to be lower than their parents. In order to prevent these jumps in the search space, they suggested adding new nodes or links with zero weight, and then leaving optimization to future modifications.

Neuroevolution of augmenting topologies (NEAT) is an evolutionary method to co-evolve the topology and the parameters of the networks [98]. This method was developed in order to reduce the negative impact of competing conventions by ensuring that the product of the crossover of two parents will be meaningful. To achieve this goal, the algorithm keeps track of the gene markings which indicate their genetic origins. Each genetic code encodes the connections and their weights of the network. The length of the genetic codes can be different. Each gene in the genetic code is associated with a binary number which indicates whether the gene is expressed in the phenotype and; a number which is referred to as the innovation number. When a new gene appears through mutation, a global counter is increased by one and assigned to that gene as an innovation number. Innovation number for genes do not change throughout the evolutionary process.

It is natural to use evolution strategies since they are designed to optimize real-valued vectors [42, 84, 107, 122]. These approaches provided results as good as competitive backpropagation algorithm.

In deep learning [52], the ANN architectures are often engineered for certain tasks. This process may take lots of trial and error. To find the optimum topology for the task, evolutionary computing approaches can be used to optimize only the topology of the networks, and the weight optimization process is performed by backpropagation algorithm [11, 61, 79].

### 2.3.2 Indirect Encoding

In indirect encoding the parameters of the networks are not directly specified in the genotype of the individuals. Instead, there are some form of intermediate rules to specify how the networks are to be developed or learn during their lifetime [48]. This approach may be more suitable for parametric and topological search for large scale artificial neural networks since there are less parameters to be optimized.

One of the earliest examples of indirect encoding was proposed by Kitano [46] that used a grammatical graph encoding method, based on graph rewriting rules represented as individuals' genotypes, to evolve the connectivity matrix of ANNs.

Koutnik *et al.* proposed using lossy compression techniques to reduce the high-dimensional parameters of the networks by transforming their parameters to the frequency domain, using transformation functions such as the Fourier Transform and the Discrete Cosine Transform. In this case, the evolutionary

process is performed on a few significant coefficients on the frequency domain [47].

Gruau suggested a developmental method that evolves tree-structured programs to specify the instructions to grow ANNs based on cell division and differentiation [32]. Nolfi *et al.* suggested a developmental encoding approach that includes phenotypic plasticity property in their ANN growth model [70]. Phenotypic plasticity allows the structure of the artificial neural network to change over time. The growth model and the plasticity properties are encoded in the genotype of an agent, where each gene in the genotype, specified the instructions that include the position of the neuron, its threshold expression, axonal growth angle, sement length and synaptic length.

Other works used Hebbian plasticity/ learning rules to perform synaptic changes during the lifetime of the networks [14, 67, 93]. Hebbian plasticity rules performs synaptic changes based on the local interactions of the neurons. Several authors suggested evolving the type and parameters of Hebbian learning rules using evolutionary computing. For instance, Floreano and Urzelai evolved four Hebbian learning rules and their parameters in an unsupervised setting, where synaptic changes are performed periodically during the networks' lifetime [27].

Niv *et al.,* [69] evolved the parameters of a Hebbian rule that defines a complex relation between the pre- and post-synaptic neuron activations on a reinforcement learning setting. Others suggested evolving complex machine learning models (i.e. ANNs) to replace Hebbian rules to perform synaptic changes [44, 83].

Orchard and Wang [72] compared evolved plasticity based on a parameterized Hebbian learning rule with evolved synaptic updates based on ANNs. However, they also included the initial weights of the synapses into the genotype of the individuals.

Risi and Stanley [81] used *compositional pattern producing networks* (CPPN) [95] to perform synaptic updates based on the location of the connections. Tonelli and Mouret [101] investigated the learning capabilities of plastic ANNs evolved using different encoding schemes.

In summary, neuroevolution is a biologically inspired approach to optimizing ANNs that does not require gradient information. Direct neuroevolution aims to optimize networks encoding their parameters directly in the genotype of the individuals. Since evolutionary approaches perform population based global search, this approach may suffer from scalability issues while the number of parameter increases. Indirect approaches to neuroevolution on the other hand, aims to evolve some sort of rules to specify developmental or lifetime learning

processes in ANNs which may be helpful in scaling the learning processes to large networks.

# Chapter 3

# Limited Evaluation and Cooperative Co-evolution for Large-scale Direct Neuroevolution

Many real-world control and classification tasks involve a large number of features. When ANNs are used for modeling these tasks, the network architectures tend to be large. Neuroevolution is an effective approach for optimizing ANNs [65, 84, 96]; however, there are two bottlenecks that make their application challenging in case of high-dimensional networks using direct encoding [117]. First, classic evolutionary algorithms tend not to scale well for searching large parameter spaces; second, the network evaluation over a large number of training instances is in general time-consuming. The Cooperative Co-evolution (CC) is an effective approach for optimizing large-scale problems [74]; and the Limited Evaluation (LE) is an advantageous method for reducing the number of instances of fitness evaluations [65]. In this chapter[1], we propose

[1]This chapter is integrally based on:
[117] Anil Yaman, Decebal Constantin Mocanu, Giovanni Iacca, George Fletcher, and Mykola Pechenizkiy. Limited evaluation cooperative co-evolutionary differential evolution for large-scale neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 569–576, New York, NY, USA, 2018. ACM

using CC and LE schemes to optimize connection weights of high-dimensional ANNs. We employ Differential Evolution (DE) algorithm (see Section 2.1.2) as an optimizer since we use real-valued connection weights in our ANN models [117]. We refer to the proposed algorithm as *"Limited Evaluation Coopertive Co-evolutuonary Differential Evolution (LECCDE)"*.

With respect to the previous works, the LECCDE contributes as follows: 1) it considers the post-synaptic neurons as the building blocks of an ANN, and performs the subcomponent decomposition of the CC scheme by assigning the pre-synaptic weights of each post-synaptic neuron into a subpopulation; 2) it demonstrates the effectiveness of the CC in optimizing large-scale ANNs, and compares with the standard Differential Evolution (DE) optimization; 3) it shows that the LE scheme enhanced with the CC achieves better accuracy results than standard DE for evolving large networks, while reducing the time required for the fitness evaluation.

Four datasets (including extended results on the MNIST) were chosen to evaluate the performance of the proposed algorithm on supervised learning tasks. We used a fully connected feed forward ANNs with one hidden layer, with a total number of parameters in the order of thousands. We refer to these ANNs as "large-scale" in the sense of NE with direct encoding, and to distinguish them from the specialized networks used in Deep Learning (DL) approaches [52].

The rest of this chapter is organized as follows: in Section 3.1, we provide the background knowledge specifically on the topics of CC and LE; in Section 3.2, we discuss the proposed algorithm in detail; in Section 3.3, we present the experimental setup; in Section 3.4, we provide the numerical results; and finally, in Section 3.5, we discuss the conclusions.

## 3.1 Background

The proposed method uses cooperative co-evolution and limited evaluation schemes. Following two subsections provide background knowledge on these two schemes.

### 3.1.1 Cooperative Co-evolution

While the dimensionality of a problem increases, the performance of the evolutionary algorithms tend to decrease [57, 58]. The CC schemes were proposed for scaling evolutionary algorithms to higher dimensions using a divide-and-conquer strategy. In the CC, the subcomponents of a large-scale problem is

decomposed and assigned to a number of subpopulations, that are evolved separately [74]. Cooperation in co-evolution arises during the fitness evaluation, where the subcomponents are merged together to assign a global fitness score to a candidate solution.

The three aspects that play a key role in CC are *problem decomposition*, *subcomponent evolution*, and *subcomponent co-adaptation* [118]. The maximum number of subpopulations can be generated by splitting a $D$-dimensional problem into $D$ subgroups, assigning each subcomponent (dimension) to one subpopulation. Alternatively, the number of subcomponents in each subpopulation can be chosen arbitrarily to make the evolutionary optimization process manageable by reducing the dimensionality per subgroup. However, an arbitrary assignment of subcomponents may not be effective for solving non-separable problems. Ideally, the problem should be decomposed in a way that the interdependency between the subcomponents in different subpopulations should be minimized.

The existing knowledge about the problem domain can be beneficial in the problem decomposition process. If the interdependencies of the subcomponents are known, the problem can be decomposed based on this knowledge. This also relates to the separability property of the problem. If the problem is separable, then the problem can be decomposed into its separable subcomponents. If there is no/uncertain knowledge of the problem domain, then automated methods can be used to identify the interactions of the subcomponents [71, 100].

The subcomponent evolution can be performed by using various kinds of evolutionary algorithms [58, 89].

Some of the works incorporate the CC scheme within Neuroevolution. The Symbiotic Adaptive Neuroevolution (SANE) evolves two separate populations, one for neurons and another for the network “blueprints". The evolved network blueprints are used to determine which combinations of the neurons to use from the neuron population to generate a network [64]. The Enforced SubPopulations (ESP) initiates a subpopulation for each neuron, and the genotype of these neurons encode the weights for incoming, outgoing and bias connections [31]; Cooperative Synapse Neuroevolution (CoSyNE) initiates a subpopulation for each connection [30].

### 3.1.2 Limited Evaluation

Most complex machine learning tasks require a large number of instances to train ANNs. The evaluation process in Neuroevolution consumes a lot of time because it requires evaluating the ANNs on all of these instances. The LE is

an evaluation scheme that proposes to split the training instances into batches (small subset of training instances) to speed up the evaluation process. Since the evaluation is performed on batches, it is required to keep track of the individuals that performed well on the evaluations on previous batches. Thus, the LE scheme aims to adjust the fitness of the offspring by taking into account the success of its parents by fitness inheritance. The *asexual* and *sexual* reproduction rules are provided in Equations (3.1) and (3.2) respectively [65].

$$f' = f_{parent} \cdot (1 - decay) + f \tag{3.1}$$

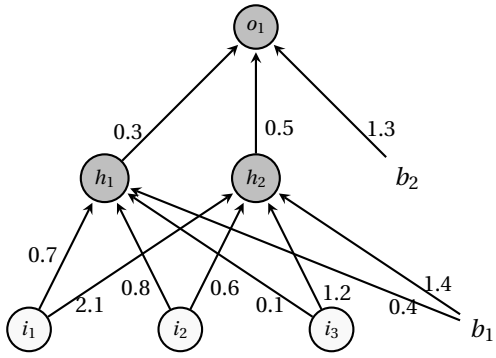
$$f' = \frac{f_{parent_1} + f_{parent_2}}{2} \cdot (1 - decay) + f \tag{3.2}$$

where, $f'$ is the adjusted fitness of the offspring, $f_{parent}$ is the parent of its parent, $f$ is the actual fitness of the offspring on current batch of the training instances, and $decay$ is a constant value for adjusting the weight of the previous fitness evaluations. The sexual reproduction method consists of two parents.

## 3.2 Limited Evaluation Cooperative Co-evolutionary Differential Evolution

The implementation details of the LECCDE algorithm are given in Algorithm 5. The algorithm is composed of the CC and LE schemes to decompose a large-scale continuous optimization task, and speed up the fitness evaluation process.

The CC scheme in LECCDE uses a heuristic to decompose the parameters of a high-dimensional ANN, i.e. the post-synaptic neurons are assumed to be the building blocks of the ANN, and are decomposed into subpopulations and evolved separately. Thus, the algorithm initiates $SP$ subpopulations for each post-synaptic neuron, where each subpopulation consists of $NP$ individuals. Each individual represents the pre-synaptic connection weights. Figure 3.1 illustrates our decomposition heuristic on an example where the parameters parameters of the network were split into three subpopulations that groups the parameters of incoming weights of two hidden neurons and one output neuron.




(a)

| post-synaptic neurons: | $h_1$ | | | | $h_2$ | | | | $o_1$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| pre-synaptic weights: | 0.7 | 0.8 | 0.1 | 0.4 | 2.1 | 0.6 | 1.2 | 1.4 | 0.3 | 0.5 | 1.3 |

(b)

Figure 3.1: (a) A fully-connected feed-forward ANN with one hidden layer, and (b) the representation of its genotype.

From the $SP$ subpopulations that contain $NP$ of individuals, there are $NP^{SP}$ ANNs that can be constructed. To find the average fitness of each individual, all possible network combinations need to be evaluated. Since this number is quite large, we randomly sample $trial \times NP$ times an individual from each subpopulation, construct a global network, evaluate it, and add the fitness value of the network to the fitness values of each individual that was part of the network [30]. At the end of this procedure, the fitness value of each individual is normalized to find the average fitness value, dividing by the number of time each individual is selected. The fitness of the individuals that were not selected during the sampling process set to 0. The individual with the maximum fitness from each subpopulation is then selected to construct the global ANN solution $\boldsymbol{X}$. Finally, the performance of the global solution on the validation instances is

found by evaluating $\boldsymbol{X}$ on the validation set.

The main loop of the algorithm iterates over all the batches. A *batch* is a small subset of the training instances used in the LE scheme [65]. In particular, $TrainingSize/BatchSize$ batches are generated by randomly assigning each training instance to a batch.

The fitness score of the target vector on the current batch is found by replacing it within its corresponding part in the global solution, and evaluating the global solution on the current batch. Subsequently, the fitness of the target vector is adjusted using the *asexual* reproduction rule (see Section 3.1.2).

The fitness of the trial vector is computed in a similar fashion, by first replacing its corresponding part within the global solution, and then evaluating the global solution on the current batch. Since the mutant vector is composed of three randomly selected individuals $\{\boldsymbol{x_{i,r_1}}, \boldsymbol{x_{i,r_2}}, \boldsymbol{x_{i,r_3}}\}$, the fitness value of the mutant vector is computed by taking their average. The fitness value of the trial vector is found using the *sexual* reproduction rule (see Section 3.1.2).

The selection operator copies the trial vector and its fitness to a temporary set if its fitness value is greater than or equal to the fitness value of the target vector; otherwise, the target value and its fitness are copied. After all the computations are completed for all individuals in the subpopulation, the subpopulation is updated simultaneously by copying back the individuals and their fitnesses from the temporary sets.

After each subpopulation update, the individual with the highest fitness value in the subpopulation is copied back to the corresponding part of the global solution $\boldsymbol{X}$. The global solution is evaluated on the validation set, and the one that performed the best is stored and provided as a the final result of the algorithm.

## 3.3 Experimental Setup

Our experimental setup is designed to focus on the following questions:

1. Do the ANNs that are evolved using the Cooperative Co-evolutionary DE algorithm with our subpopulation assignment heuristic achieve a better classification accuracy than the ANNs that are evolved by the standard DE algorithm?

2. Does the LE scheme applied to DE reduce the runtime of the algorithm, without decreasing the classification accuracy of the evolved ANNs?

To answer these questions, we compare the results of the ANNs optimized by four algorithms, DE, LEDE, CCDE, and LECCDE, on three datasets with various sizes. The details for the implementation of the LECCDE are given in Algorithm 5. The CCDE and LEDE are implemented in a similar way, but, without the batch loop and the subpopulations, respectively. In standard DE, both batch training and subpopulations are not used. The LE algorithms require two evaluation per generation (target and trial vectors are evaluated on the current batch), while the algorithms without LE require one evaluation per generation. Regardless of this fact, the algorithms were run for the same number of function evaluations (FEs) for each dataset. For all experiments, we used *"rand/1/bin*" (*"rand/1*" mutation with *binomial* crossover) strategy with empirically fixed the parameter settings of $F$ and $CR$ to 0.1 and 0.3, respectively. We used 20 individuals for the population size, except for one experiment that we performed on a larger population size consisting of 100 individuals (see below). We set $trial$ parameter to 5.

The three datasets used in the test process are listed in Table 3.1. These datasets were obtained from the Center for Machine Learning and Intelligent Systems dataset repository [56]. These datasets were chosen based on their number of features and instances, to show the relative performance of the algorithms in respect to the size of the dataset used. The Wisconsin breast cancer (WBC) dataset consists of 30 features, 2 classes, and 569 instances, the epileptic seizure recognition (ESR) consists of 178 features, 2 classes, and 4600 instances[2], and the human activity recognition (HAR) dataset consists of 561 features, 6 classes, and 7144 instances [3]. The instances in each dataset were split into three groups (training, validation, and test) with ratios 70%, 15%, and 15% respectively. The fitness evaluations and selection process were performed on the train instances. The network that performs the best on the validation set is provided as the output of the algorithm, and evaluated on the test set. The fitness evaluation is based on the classification accuracy of the ANNs which is calculated by the number of correctly classified instances divided by the total number of instances.

For all datasets, we used fixed-topology fully-connected feed forward ANNs with one hidden layer to perform the classification task. The number of neurons within the hidden layer was kept constant at 50 for all ANNs evolved for all

[2]The original epileptic seizure recognition dataset [1] consists of 5 classes (first class for the measurements of the patients who had epileptic seizure and the remaining 4 classes for the measurements of the patients who did not have epileptic seizure), and 11500 instances (2300 for each class). To reduce the complexity of the task we took only the instances from the first and second classes with 2300 instances from each, thus considering 4600 instances in total.

datasets. Based on the architecture of the ANNs and the number of features in the datasets, the total number of parameters evolved are 1652, 9052, and 28406 for the WBC, ESR, and HAR respectively.

We used a batch size of 100 instances for the WBC, 500 for the ESR, and 500 for the HAR. The decay value is set to 0.2, as suggested by Morse and Stanley [65]. The maximum number of FEs was set to 50000 for the WBC, 300000 for the ESR, and 500000 for the HAR, based on the number of their parameters.

Table 3.1: The specifications of the datasets used in the experiments.

| Datasets | Features | Classes | Instances | Parameters |
|---|---|---|---|---|
| WBC | 30 | 2 | 569 | 1652 |
| ESR | 178 | 2 | 4600 | 9052 |
| HAR | 561 | 6 | 7144 | 28406 |

## 3.4 Experimental Results

In this section, we present our experimental results. Each algorithm, with the specified settings, was run for 20 independent runs, and the median and the variance of train, validation and test accuracy were collected. All the accuracy results are shown with a precision of two digits.

Table 3.2 shows the results obtained from the WBC dataset. In this case, we could not observe a significant difference on the results of the ANNs evolved by the four algorithms. On the test data, the CCDE appears to be performing better than others. On the other hand, we observe a difference on the runtime of the algorithm ($t = 322$ sec, in our computing environment[3]). The algorithms that employ LE and CC are less computationally expensive and run faster. For example, the runtime of DE is more than twice as big as that of LECCDE. This difference is less significant for the other algorithms, due to the size of the dataset. Even though, all the algorithms are run for 50000 FEs for this dataset, the algorithms with LE performed evaluation on batches that are four times smaller than the whole set of training instances. However, since CCDE is run on the whole dataset, it appears that the CC improved its runtime possibly due to the computations of reduced-sized vectors within each subpopulation.

[3]All algorithms were run, in single-core, on an Intel Xeon E5 3.5GHz computer.

Table 3.2: The median of the accuracy results of the ANNs evolved using four variants of DEs on the WBC dataset.

| Alg. | Train | Validation | Test | Runtime |
|---|---|---|---|---|
| **DE** | 94.74 ± 2.2 | 97.65 ± 0.8 | 95.29 ± 2.2 | $2.12 \times t$ |
| **LEDE** | 95.99 ± 1.2 | 98.82 ± 0.6 | 95.29 ± 2.3 | $1.43 \times t$ |
| **CCDE** | 96.49 ± 1.3 | 97.65 ± 0.8 | 96.47 ± 1.8 | $1.10 \times t$ |
| **LECCDE** | 96.24 ± 1.9 | 97.65 ± 0.8 | 95.29 ± 2.8 | $t$ |

Table 3.3 presents the results obtained from the ESR dataset. Based on the test data, CCDE appears to show better performance than the rest of the algorithms, while LECCDE follows it very closely. We observe the best running time with LECCDE ($t = 2970$ sec).

Table 3.3: The median of the accuracy results of the ANNs evolved using four variants of DEs on the ESR dataset.

| Alg. | Train | Validation | Test | Runtime |
|---|---|---|---|---|
| **DE** | 90.50 ± 1.3 | 89.86 ± 1.2 | 89.57 ± 1.7 | $2.66 \times t$ |
| **LEDE** | 92.86 ± 0.9 | 92.25 ± 0.8 | 91.30 ± 0.9 | $1.26 \times t$ |
| **CCDE** | 93.94 ± 0.8 | 93.33 ± 0.3 | 92.17 ± 0.9 | $2.05 \times t$ |
| **LECCDE** | 93.98 ± 0.6 | 92.61 ± 0.5 | 91.88 ± 1.0 | $t$ |

Table 3.4 shows the results obtained from the HAR dataset. On this dataset, we observe a significant accuracy improvement when the CC scheme is used. The performance of CCDE and LECCDE are approximately %15-20 better than the algorithms that do not use the CC. Also, CCDE appears to be slightly better than LECCDE. On the other hand, we observe a significant runtime improvement when the LE scheme is used. The algorithms with the LE scheme run approximately four times faster than the algorithms that do not use the LE ($t = 6530$ sec). Also, LECCDE appears to produce the smallest variance on the train accuracy.

Finally, in Table 3.5, we report an additional experiment on the population size. In this case, we used a population size of 100 on the ESR dataset. When the population size increases (comparing to the Table 3.3), the accuracy results decrease. This may be due to the number of FEs needed for the convergence of the algorithm: in other words, when the population increases, the number of FEs needed for the convergence may increase. Moreover, we observe that CCDE and LECCDE perform significantly better than DE and LEDE. This may suggest that the CC increased the convergence speed. With respect to the running time

of the algorithms, we observe the similar pattern observed in Table 3.3 ($t = 2640$ sec).

Table 3.4: The median of the accuracy results of the ANNs evolved using four variants of DEs on the HAR dataset.

| Alg. | Train | Validation | Test | Runtime |
|---|---|---|---|---|
| **DE** | 70.06 ± 2.9 | 70.06 ± 2.7 | 68.38 ± 3.0 | $4.75 \times t$ |
| **LEDE** | 77.5 ± 5.2 | 77.99 ± 4.8 | 76.96 ± 4.8 | $1.28 \times t$ |
| **CCDE** | 94.01 ± 0.8 | 92.72 ± 0.7 | 92.4 ± 1.0 | $4 \times t$ |
| **LECCDE** | 93.58 ± 0.6 | 93.19 ± 0.5 | 92.16 ± 0.7 | $t$ |

Overall, CCDE appears to perform better than LECCDE, due to the fact, that it has the complete information for evaluating the individuals since it uses the entire set of training instances. However, CCDE comes with a larger runtime trade-off than LECCDE, which can make the difference with large datasets (e.g. for the HAR dataset the LECCDE runs on average four times faster). Also, increasing the number of evaluations or batch size can improve the performance of the LECCDE. For comparison, we performed two additional experiments with LECCDE, with the same settings used to produce the results in the Table 3.4, except the number of FEs and batch size. In the first experiment, we used 900000 FEs and observed that the ANNs the LECCDE optimize perform on training, validation and test sets on average 95.78, 94.31, and 93.28 respectively. This is almost %1 higher than the the performance observed in 3.4. On the other hand, the runtime of the algorithm is now $1.6 \times t$, which is still 1.8 times faster than the runtime of CCDE. In the second experiment, we used a batch size of 1000, and we observed that the algorithm performs on average 96.60, 93.84, and 93.38 on training, validation, and test datasets, with a runtime of $1.42 \times t$. These two additional experiments show an interesting trade-off between the batch size and the number of evaluations. Although the two additional experiments have similar runtime, the second experiment appears to produce better results.

Table 3.5: The median of the accuracy results of the ANNs evolved using four variants of DEs on the ESR dataset using population size of 100.

| Alg. | Train | Validation | Test | Runtime |
|---|---|---|---|---|
| **DE** | 81.99 ± 1.4 | 82.61 ± 1.2 | 80.65 ± 1.9 | $2.62 \times t$ |
| **LEDE** | 80.25 ± 1.4 | 81.59 ± 0.8 | 80.14 ± 2.2 | $1.30 \times t$ |
| **CCDE** | 91.65 ± 1.3 | 91.45 ± 0.7 | 90.29 ± 0.9 | $2.01 \times t$ |
| **LECCDE** | 91.27 ± 0.8 | 90.58 ± 0.7 | 88.99 ± 1.1 | $t$ |

Figure 3.2 shows the overall comparison of the runtime of LEDE, CCDE, and DE relative to LECCDE on the three datasets. The $x$-axis shows the dataset, and the $y$-axis shows the increase in the runtime of the algorithm. The LEDE is relatively stable across experimented datasets. On the other hand, the runtime of the CCDE and DE increases when the number of instances increases. This is because the algorithms with LE perform the same number of function evaluations, on a smaller number of instances, which produces a clear advantage in terms of total runtime.

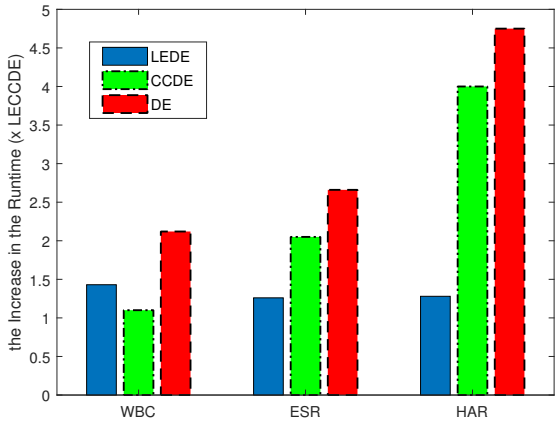



Figure 3.2: The increase in the runtime of each algorithm relative to the LECCDE on the three datasets.

Figure 3.3 shows the accuracy trend of the ANNs on the training, validation, and test instances during one example run of the optimization process performed by LECCDE (only the range $[0.8, 1]$ is shown on the $y$-axis, for the sake of clarity). The data collected from this specific run shows that the accuracy on the training data is almost always the highest. The accuracy results of the test data closely follows the validation accuracy, and it is even higher for some of generations.

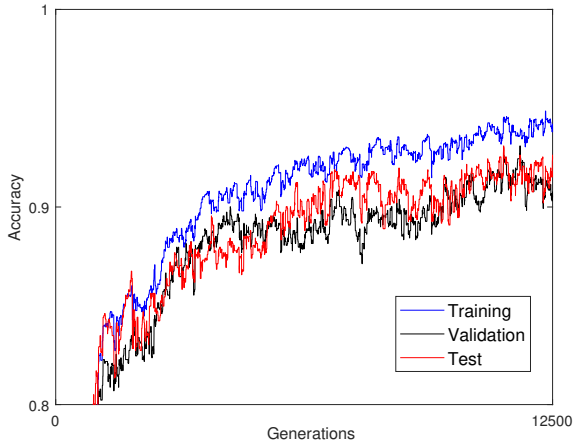



Figure 3.3: The change of the accuracy results of the ANNs on the training, validation, and test instances while the LECCDE algorithm is running.

Figure 3.4 shows the change of the validation accuracy during the evolutionary process of four algorithms on a single run (only the range $[0.6-1]$ is shown on the $y$-axis). Firstly, the lines that represent the results of the LEDE and LECCDE are shorter than those of the other algorithms because they consume the same number of FEs within a half number of generations, since they perform two FEs (trial and target vectors) per generation. We observe that LEDE improves the DE in terms of validation accuracy and convergence speed; however, it suffers from the lack of diversity within the population (for a population size of 20), which prevents it from finding better solutions after about 80000 FEs are consumed. On the other hand, CCDE appears not to suffer from the early convergence problem observed in the LEDE, while LECCDE appears to improve the speed of CCDE.

To summarize, our empirical analysis suggests positive answers to the questions posed in Section 3.3: (1) It appears more significantly on large datasets (in Table 3.4), or with a large population size (in Table 3.5), than the ANNs that are evolved using the CC scheme using our heuristic achieve a better classification accuracy than the ANNs that are evolved by the standard DE algorithm; and (2) all experiments on the three datasets (most significantly on the largest dataset in Table 3.4), show that the LE scheme applied to DE reduce the runtime of the algorithm considerably, without causing a degradation on the classification

accuracy of the evolved ANNs.

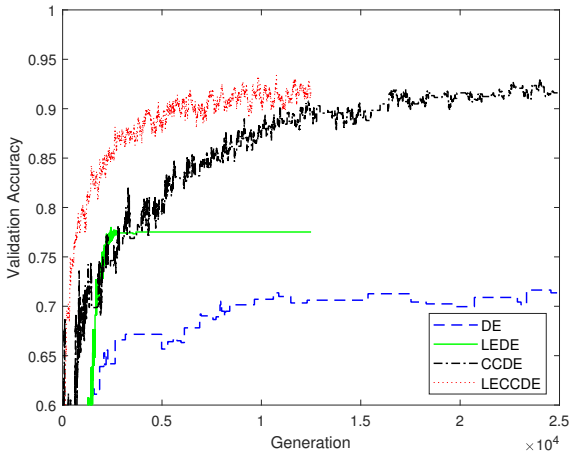



Figure 3.4: A single run of the change of the validation accuracy during the evolutionary process of four algorithms on the HAR dataset.

### Evaluation on the MNIST Dataset

To further assess the scalability of the proposed algorithm, we performed an additional experiment on the MNIST dataset [53]. The MNIST dataset consists of 60000 samples of 28 by 28 grayscale image instances of handwritten numbers between 0-9. Thus, the size of the input and output are 784 and 10, when each image pixel and its class label are considered as an input and output respectively.

We used the same architecture of the artificial neural networks that were used for the experiments performed on the other datasets (feed forward artificial neural networks with one hidden layer consisting of 50 neurons). Thus, the total number of parameters of the networks optimized for the MNIST is 47710. The parameters of the Differential Evolution algorithm are also initialized, using the same settings used for the other experiments, except for batch size, number of individuals in each subpopulation and the maximum number of function evaluations. Since MNIST is larger than the tested other datasets, we used a a batch size of 1000, a population size of 60 and a maximum number of evaluations set to $2.16e+6$.

Table 3.6 shows the training, validation and test accuracy results of the ANNs trained for the MNIST dataset. Each variant of the algorithm was executed for the same number of function evaluations. The total time required for computing every other algorithm is shown in relation to the computing time required for the LECCDE where $t = 6.6e+5$ seconds, that is approximately 19 hours on a single-core Intel Xeon E5 3.5GHz computer. Due to time constraints, we were able to perform 3 independent runs for the LEDE and LECCDE, and a single partial run for the DE and CCDE.

Table 3.6: The accuracy the ANNs evolved for the MNIST dataset, and the runtime of the algorithms.

| Alg. | Train | Validation | Test | Runtime |
|---|---|---|---|---|
| **DE** | 85.46 | 84.70 | 84.78 | $27.2 \times t$ |
| **LEDE** | 82.68 ± 0.36 | 82.01 ± 0.75 | 82.23 ±0.25 | $1.1 \times t$ |
| **CCDE** | 91.92 | 90.64 | 90.36 | $25.3 \times t$ |
| **LECCDE** | 91.79 ± 0.28 | 91.01 ± 0.63 | 90.80 ± 0.15 | $t$ |

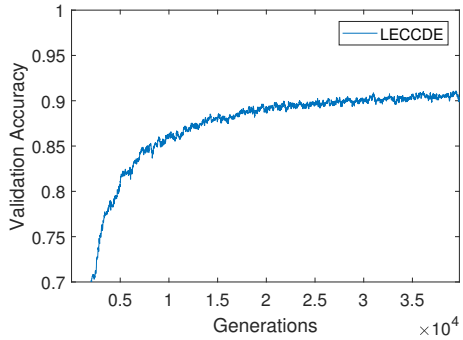



Figure 3.5: The change of the validation accuracy of the ANNs evolved using the LECCDE on MNIST dataset (only $[0.7, 1]$ range is shown on the $y$-axis).

We observe a significant advantage in using the LE scheme on MNIST from the computing time point of view: indeed, the DE and CCDE implementations of the algorithm require a computing time that is 25 times bigger than the computing time required by the corresponding algorithms that make use of the LE scheme.

Figure 3.5 illustrates the change of the validation accuracy of the evolved

ANNs using the LECCDE during an evolutionary process. The speed of the accuracy improvements slows down around 88% - 90% level. The best validation accuracy achieved during this evolutionary run was 91.62%.

## 3.5 Conclusion and Future Work

In this chapter, we proposed the LECCDE algorithm that employs the LE and CC schemes to improve the accuracy and the runtime of the standard DE algorithm for large-scale NE with direct encoding.

We performed experiments on four datasets, including the MNIST dataset. Our results show that the CC scheme improves the performance of DE on the tested classification tasks. Moreover, we used the LE scheme to further improve the scalability of the method. Our results show that the LE scheme reduces the runtime of the algorithms, without affecting the performance. This reduction is due to the fact that the evaluation is performed on a small number of instances.

We used a heuristic in the CC scheme that decomposes the problem at the level of post-synaptic neurons. Thus, we evolve all the pre-synaptic weights of the post-synaptic neurons in different subpopulations. This decomposition approach aims to reduce the parameter size per subpopulation. For large datasets on the other hand, the number of parameters per subpopulation may still be large. Although this heuristic worked well, there may also be other decomposition heuristics that can be more effective. Alternatively, automatic methods can also be used for this purpose.

Another possibility for improving the results can be achieved by performing a sensitivity analysis. In this work, we did not experiment on the strategy and the parameters settings of the DE algorithm. Self-adaptive parameter control approaches can also be investigated to improve the performance of the results since these approaches can adjust the balance between the exploration and exploitation during the search process [17, 112].

The parameters of the ANNs evolved using the direct encoding approaches, such as the approach proposed in this chapter, remains fixed. Thus, the ANNs do not have any learning and adaptation mechanisms unless the evolutionary process is resumed. On the other hand, biological neural networks have the capability of changing their configuration during their lifetime. Indirect approaches to neuroevolution may be able to produce the lifetime learning and adaptation capabilities in artificial neural networks.

# Chapter 4

# Evolving Plasticity in Artificial Neural Networks

In the previous chapter, we proposed an approach for two bottlenecks of neuroevolution with direct encoding. ANNs trained with direct encoding approaches lack the capability of lifetime learning, which is a fundamental aspect of learning in biological neural networks. In this chapter[1], we focus on lifetime learning in ANNs.

Biological neural networks have the plasticity property that enables them to change their configuration (i.e. *topology* and/or *connection weights*) and learn to perform certain tasks during their lifetime [102]. The learning process involves searching through the possible configuration space of the networks until a configuration that achieves a satisfactory performance is reached. Modelling plasticity, or rather *evolving* it, has been a long-standing goal in Neuroevolution, a research field that aims to design artificial neural networks (ANNs) using evolutionary computing approaches [26, 48, 97].

Evolving plastic artificial neural networks (EPANNs) [14,93] implement plasticity by modifying the networks' configuration based on some plasticity rules which are activated throughout the networks' lifetime. These are encoded within the genotype of a population of individuals, in order to optimize the

[1]This chapter is integrally based on:
[116] Anil Yaman, Decebal Constantin Mocanu, Giovanni Iacca, Matt Coler, George Fletcher, and Mykola Pechenizkiy. Evolving plasticity for autonomous learning under changing environmental conditions. *(Under review)*, 2019

learning procedure by means of evolutionary computing approaches. One possible way of modelling plasticity rules in EPANNs is by means of *Hebbian learning*, a biologically plausible mechanism hypothesized by Hebb [35] to model synaptic plasticity [51]. According to Hebbian learning, the synaptic efficiencies between pre- and post-synaptic neurons are strengthened/weakened if the neurons' activation states are positively/negatively correlated. However, the basic formalization of Hebbian learning can suffer from instability as it introduces an indefinite increase/decrease of the synaptic efficiencies [103]. Several other modified variants have been proposed to reduce this effect [8, 87, 103]. Nevertheless, these plasticity rules may still require further optimization to properly capture the dynamics needed for adjusting the network parameters.

A number of previous works suggested optimizing the parameters of some form of Hebbian plasticity rules using evolutionary approaches [27, 69, 91]. However, these attempts may involve a high degree of complexity that may not be able to deliver insights into the dynamics of the plasticity property [72, 81]. For instance, some of the existing works evolve the initial synaptic weights and/or the connectivity of the networks in addition to the plasticity rule. However, evolving the initial synaptic weights of the networks increases the number of parameters to evolve and, in principle, can be decoupled from the evolution of plasticity rules (except for tasks where there is a need to adapt to changing conditions); on the other hand, evolving the connectivity of the networks may overfit the networks to a certain task, making it difficult to evolve adaptive behavior.

This chapter proposes a novel approach to produce plasticity property in ANNs for a continuous learning scenario with changing environmental conditions [116]. We employ Genetic Algorithms to evolve discrete plasticity rules to determine how synaptic weights are adjusted based on the activation states of the connected neurons, and a reinforcement signal received from the environment. One of the main advantages of our approach is that the plasticity rules it evolves can provide interpretable results, as an alternative to the other plasticity rules proposed in the literature.

We use ANNs consisting of one hidden layer, and introduce local weight competition to allow self-organized adaptation of synaptic weights. We demonstrate the lifetime learning and adaptation capabilities of plastic ANNs on a foraging task where an agent is required to learn to navigate within an enclosed environment, and collect/avoid specific types of items. Starting from the randomly initialized values, weights are updated after each action step, based on the reinforcement signals received from the environment. We test the adaptation capabilities of the networks by switching the types of items to be collected/avoided

after a certain number of action steps. We show that the evolved plasticity rules are capable of producing stable learning, and autonomous adaptation capabilities under changing environmental conditions. This form of learning can be seen as a distributed, self-organized continuous learning process that can be carried on without the need for global evaluation or validation.

The rest of the chapter is organized as follows: in Section 4.1, we provide the background for Hebbian learning, and discuss the literature on the evolution of learning. In Section 4.2, we introduce our approach to evolving plasticity rules and provide the details of the genetic algorithm. In Section 4.3, we present the experimental setup. In Section 4.4, we provide the results for the learning and adaptation capabilities of the networks modified by the evolved plasticity rules; finally, in Section 4.5 we conclude by recapitulating our main results and highlighting possible future works.

## 4.1 Background

In this section, we provide background knowledge in Hebbian learning and discuss its use in the context of evolution of learning.

### 4.1.1 Hebbian Learning

In its general form, Hebbian learning rule is formalized as:

$$w_{i,j}(t+1) = w_{i,j}(t) + m(t) \cdot \Delta w_{i,j} \tag{4.1}$$

where the synaptic efficiency $w_{i,j}$ at time $t+1$ is updated by the change $\Delta w_{i,j}$ that is a function of $a_i$ and $a_j$:

$$\Delta w_{i,j} = f(a_i, a_j) \tag{4.2}$$

A modulatory signal, $m(t)$, is used to determine the sign of the Hebbian learning. If $m(t)$ is positive and there is a positive correlation between the activations of neurons, the synaptic efficiency between them is strengthened; whereas if their activations are not correlated then the synaptic efficiency between them is weakened. The negative sign of $m(t)$ reverses to sign of the Hebbian learning (which in this case is also known as *anti-Hebbian learning*), by strengthening the synaptic efficiencies between neurons with uncorrelated activations, and weakening the synaptic efficiencies of neurons with correlated activations. The

modulatory signal is usually equivalent to the reward received from the environment, unless other kinds of modulatory signaling mechanisms are used, such as neuromodulation [91].

A "plain" Hebbian rule formalizes $\Delta w_{i,j}$ as a product of the activations of pre- and post-synaptic activations, i.e.:

$$\Delta w_{i,j} = \eta \cdot a_i \cdot a_j \tag{4.3}$$

The plain Hebbian rule strengthens a synaptic efficiency when the signs of the pre- and post-synaptic neuron activations are positively correlated, weakens the synaptic efficiency when the signs are negatively correlated, and keeps the same efficiency when one of the two activations is zero [8]. A constant $\eta$ is used as a learning rate to scale the magnitude of the synaptic change.

As we stated in the previous section, one of the undesired effects of the plain Hebbian rule is that it may lead to an indefinite synaptic increase/decrease, since a synaptic change in one direction encourages further synaptic change in the same direction. A number of variants of the plain Hebbian rule have been proposed to stabilize the behavior [8, 103]. These rules can roughly be categorized into two groups: activity and threshold based. The activity based rules perform updates based on the activation correlations between pre- and post-synaptic neurons. The threshold-based methods perform updates when the activation correlations are above/below a given threshold, that can also be adaptive.

Finally, it is worth mentioning a further generalization of the plain Hebbian rule given by:

$$\Delta w_{i,j} = \eta \cdot [A \cdot a_i \cdot a_j + B \cdot a_j + C \cdot a_i + D] \tag{4.4}$$

This rule, usually referred to as the *ABCD* rule [69, 91], parameterizes the relationship between $a_j$ and $a_i$ by using four separate coefficients $A, B, C, D$.

### 4.1.2 Evolution of Learning

The back-propagation algorithm is one of the conventional methods to train ANNs; however, it is considered as a biologically implausible method since it requires propagating back the error through synapses [51]. The biological evidence supports instead forms of Hebbian learning [6, 8, 87]. Thus, Hebbian learning is often used as a learning mechanism for self-adapting ANNs based on supervision, and/or reinforcement signals, or in an unsupervised fashion [27, 37, 69, 93].

A near-optimum Hebbian rule and optimal learning parameters can depend on the task and the data. Furthermore, the parameters of the learning algorithm can affect the stabilization of the Hebbian rule and, ultimately, the performance of the ANN [103]. Therefore, some previous works have proposed the use evolutionary computing to optimize the parameters of Hebbian learning rules [14,27,69,72,81,101]. These works are briefly discussed in Section 2.3.2.

Others used neuromodulated learning where a number of special neurons within the network are used for signaling synaptic updates to the other neurons in the network [83]. Soltoggio *et al.,* [91] evolved network topologies and connection weights involving neuromodulated neurons and using the ABCD learning rule to perform synaptic updates.

## 4.2 Evolution of Plasticity Rules

We use the binary activation function given in Equation (2.9) to convert the activations of post-synaptic neurons into binary values. Thus, the activations of the neurons can only be in one of two possible states which allows to reduce the possible states of pair of neurons. In addition, we make use of reinforcement signals as modulatory signals to guide the learning process. These modulatory signals are received from the environment, and can take one of three possible values as given in Equation (4.5).

$$m = \begin{cases} +1, & \text{if desired output (reward);} \\ -1, & \text{if undesired output (punishment);} \\ 0, & \text{otherwise (neutral).} \end{cases} \tag{4.5}$$

According to the modulatory sugnals, if the network produces a "desired" outcome then $m$ is set to $+1$; if the network produces an "undesired" outcome then $m$ is set to $-1$; otherwise $m$ is set to $0$. The desired and undesired outcomes are defined by a *reward function* which depends on the task, according to the desired/undesired associations between sensory inputs and behavioral outputs. The reward functions we used in our experiments are discussed in Section 4.3.

The initial values of the synaptic efficiencies are randomly sampled from the range of $[-1,1]$ with uniform probability. After each network computation at time $t$, the synaptic efficiencies between post- and pre-synaptic neurons at time $t+1$ are updated based on the following rule:

$$w'_{i,j}(t+1) = w_{i,j}(t) + \eta \cdot f(a_i, a_j, m) \tag{4.6}$$

where $f()$ is a "composed" Hebbian plasticity rule that determines how each synaptic efficiency $w_{i,j}$ is updated depending on the activations of the pre- and post-synaptic neurons, $a_j$ and $a_i$, and the modulatory signal $m$. Differently from the existing approaches described in Section 4.1.2, here we evolve *discrete rules* to specify the synaptic modifications corresponding to each possible combination of the pre- and post-synaptic activations and the modulatory signal, as shown in the table illustrated in Figure 4.2. It is important to note that it is possible to enumerate all the possible discrete rules as we chose a binary neuron model, see Equation (2.9), and a discrete modulatory signal, see Equation (4.5).

Finally, after each update provided by Equation (4.6), the weights are scaled as follows:

$$w'_{i,j} = \frac{w'_{i,j}}{||\boldsymbol{w}'_{\boldsymbol{i}}||_2} \tag{4.7}$$

where the row vector $\boldsymbol{w}'_{\boldsymbol{i}}$ encodes all incoming weights to a post-synaptic neuron $i$. This scaling allows to have a unit length using the Euclidean norm $||\cdot||_2$. Furthermore, this normalization process prevents indefinite synaptic growth, and helps connections' specialization by introducing local synaptic competition, a concept recently observed also in biological neural networks [25]. Alternatively, a decay mechanism and/or saturation limits for weights can be introduced to prevent an indefinite increase/decrease of the weights [92].

### Details of the Genetic Algorithm

We employ a *Genetic Algorithm (GA)* to optimize the plasticity rules [29]. The genotype of the individuals, illustrated in Figure 4.2, encodes the learning rate $\eta \in [0,1)$, and one of three possible outcomes $L = \{-1,0,1\}$ (corresponding to *decrease*, *stable*, and *increase*, respectively) for each of the plasticity rules defined by $f(a_i, a_j, m)$. Since we use binary activations for neurons, there are 4 possible activation combinations for $a_i$ and $a_j$. Furthermore, we take into account only positive and negative modulatory signal states ignoring $m = 0$ because the synaptic update is performed only when $m = +1$ or $m = -1$. Consequently, there are 8 possible input combinations, and therefore 8 possible plasticity rules defined by $f(a_i, a_j, m)$. The size of the search space of the plasticity rules is then $3^8$, excluding the real-valued learning rate parameter.

A graphical illustration of the GA is provided in Figure 4.1. In the initialization step of the algorithm, we randomly initialize a population of 9-dimensional individual vectors $\boldsymbol{x}_i$, where $i = (1,\ldots,N)$ (in our experiments, $N$ was set to 30)

to encode synaptic update rules. Each dimension of the individuals are uniformly sampled within their domain depending on their data type.

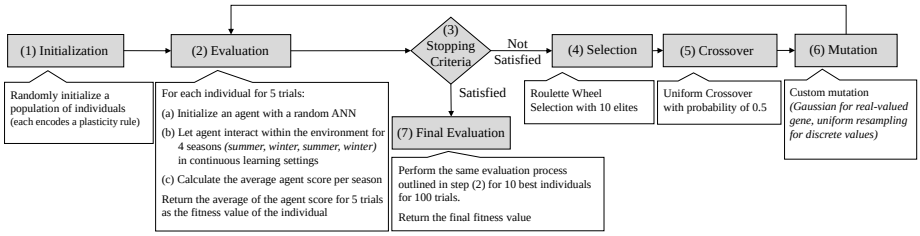



Figure 4.1: Graphical illustration of the steps of the Genetic Algorithm used to evolve the plasticity rules.

The evaluation process of an individual starts with the initialization of an agent with a random ANN configuration. Here, we use fixed topology fully connected feed forward neural networks, and sample the connection weights from a uniform distribution in [-1,1]. The agent is allowed to interact with the environment by performing actions based on the output of its controlling ANN. After each action step, the weights of the ANN are updated based on the synaptic update table. This table is constructed by converting the vector representation of the individual plasticity rules to specify how synaptic weights are modified based on the pre-, post-synaptic, and modulatory signals (see Figure 4.2). After every agent's action, a reinforcement signal is received from the environment and used directly as the modulatory signal.

We define a certain number of actions to allow the agent to interact with the environment. This process is divided into four periods of equal lengths, which we refer to as "seasons". We calculate the average of the agent's performance score, per each season, by subtracting the number of incorrectly collected items from the number of correctly collected items. Due to the stochasticity of this process (because of the random network initialization), we perform this process, starting from the same initial conditions, for five independent trials. Thus, the fitness value of an individual plasticity rule is given by:

$$fitness = \frac{1}{s \cdot t} \sum_{k=1}^{s} \sum_{l=1}^{t} (c_{k,l} - i_{k,l}) \tag{4.8}$$

where $t$ and $s$ are the number of trials and seasons, respectively, and $c_{k,l}$ and

$i_{k,l}$ are the number of correctly and incorrectly collected items in each season $k$ of each trial $l$.

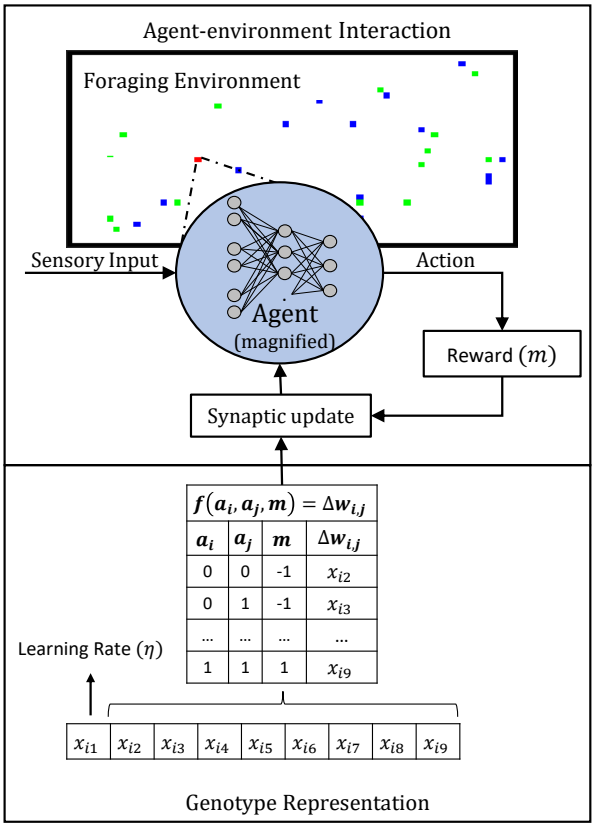



Figure 4.2: Genotype representation and agent-environment interaction. The genotypes of the individuals encode the learning rate and synaptic update outcomes for 8 possible states of $a_i$, $a_j$ and $m$. The agent (color coded as red) is evaluated on a foraging task where it is expected to learn to collect/avoid correct types of items (color coded as blue and green). An artificial neural network is used to perform the actions of the agent. The initial weights of the ANN are randomly initialized, and are updated after each action step based on the evolved plasticity rules and a reinforcement signal received from the environment.

The evaluation process of the GA is based on a continuous learning setting, where the weights of the ANNs are updated constantly. Thus, the algorithm does not involve any validation to store the best network configurations. However, in our experiments we additionally tested the evolved plasticity rules with validation settings, in order to show that validation improves the performance of the agents. On the other hand, this additional step is computationally expensive since it requires the agents to be tested for a certain number of further action steps.

We use an *elitist roulette wheel selection* operator to determine parents of the next population. The top 10 *elites* (best individuals) are copied to the next generation without any change. The rest of the offspring are generated using a *uniform crossover* operator with a probability of 0.5. As for the mutation operator, we perturb the real-valued component by a small value sampled from a Gaussian distribution with 0 mean and 0.1 standard deviation, and re-sample the discrete components with a probability of 0.15.

The evolutionary process is executed until there is no more improvement in terms of best fitness for a certain number of evaluations. At the end of the evolutionary process, the top 10 elite individuals in the population are evaluated for 100 trials, and the average statistics of this process is provided as the final result of the Genetic Algorithm. We execute the GA for 30 independent runs.

## 4.3 Experimental Setup

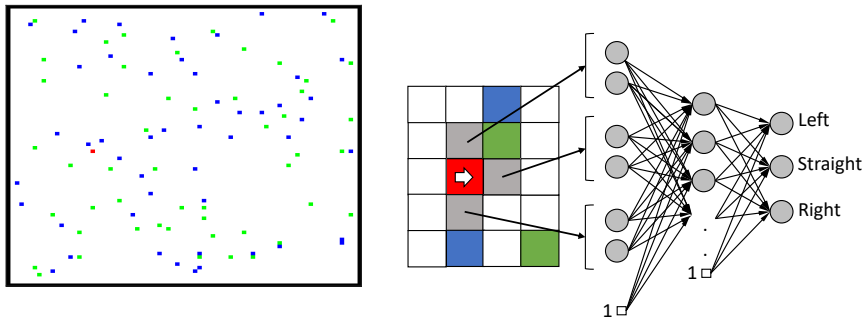
We test the learning and adaptation capabilities of the plastic ANNs with evolved plasticity rules on an agent-based foraging task within a reinforcement learning setting inspired by Soltoggio and Stanley [92]. In this task, an artificial agent, operated by a plastic ANN, is required to learn to navigate within an enclosed environment and collect/avoid correct types of items placed in the environment, based only on the reinforcement signals received in response to its actions.

The foraging environment contains two types of food items and is enclosed by a wall. To test the adaptation abilities of the networks, we define two reward functions that we refer to as two seasons: *summer* and *winter*. In both seasons, the agent is expected to explore the environment and avoid collisions with the walls. During the summer season, the agent is expected to learn to collect and avoid specific types of objects, while the expectation for the types of objects is swapped during the winter season. The details of the foraging task and environment are provided in the following section.

### Foraging Task and Environment

A visualization of the simulated environment used in the foraging task is provided in Figure 4.3a. We initialize a 100x100 grid enclosed by a wall. In the following, we refer to the green and blue items, and the wall, as “G”, “B” and “W” respectively. In the initialization phase of an experiment, an agent, 50 “G" and 50 “B" items are randomly placed on the grid.

The architecture of the ANNs used to control the agent is shown in Figure 4.3b. The agent is located in a cell, and has a direction to indicate its orientation on the grid. It is equipped with three sensors that can take inputs from the nearest cell on the left, in front, and on the right. Since there are four possible states for each cell (nothing, “W", “G", “B"), we represent the sensor reading of each cell with two bits, as $[(0,0),(1,1),(1,0),(0,1)]$. The agent performs one of the three possible actions as “Left”, “Straight”, and “Right” based on the output layer of its ANN. The output neuron with the maximum activation value is selected to be the action of the agent. In cases of “Left” and “Right” the agent’s direction is changed accordingly, and the agent is moved one cell in the corresponding direction. In case of “Straight”, the direction of the agent is kept constant and the agent is moved one cell along its original direction.




(a) Foraging environment (b) Agent architecture

Figure 4.3: (a) Simulation environment used in the agent-based foraging task experiments. The location of the agent, two types of items and the wall are color coded with red, green, blue and black respectively. (b) Agents’ position/direction (red cell) and sensory inputs from left, front and right cells (in gray) within a foraging environment, and the ANN controller, with a hidden layer, 6 inputs (2 per cell), and 3 outputs (left/straight/right).

We use fully connected feed forward networks with one hidden layer with 6 input, 20 hidden and 3 output neurons. We also perform additional experiments for networks consisting of various numbers of hidden neurons (see next section for details). Additional bias neurons are added to the input and hidden layers.

The statistics of the agent while interacting with the environment are collected. If the agent visits a cell occupied by a “G" or “B" item, the item is collected and recorded. A collected item disappears from its original location and reappears in another location selected at random. If the agent steps into a cell occupied by a wall, the action is ignored and recorded as a hit on the wall. After each action, a reinforcement signal is received from the environment, and used as a modulatory signal, see Equation (4.5).

Table 4.1: Associations of the Sensor and Behavior states to the reinforcement signals for Summer and Winter seasons.

| ID. | Sensor | Behavior | Summer | Winter |
|---|---|---|---|---|
| **1** | nothing | straight | 1 | 1 |
| **2** | nothing | left or right | -1 | -1 |
| **3** | W straight | left or right | 1 | 1 |
| **4** | W straight | straight | -1 | -1 |
| **5** | W on left | right | 1 | 1 |
| **6** | W on left | left or straight | -1 | -1 |
| **7** | W on right | left | 1 | 1 |
| **8** | W on right | right or straight | -1 | -1 |
| **9** | G straight | straight | 1 | -1 |
| **10** | G straight | left or right | -1 | 0 |
| **11** | G on left | left | 1 | -1 |
| **12** | G on left | straight or right | -1 | 0 |
| **13** | G on right | right | 1 | -1 |
| **14** | G on right | straight or left | -1 | 0 |
| **15** | B straight | straight | -1 | 1 |
| **16** | B straight | left or right | 0 | -1 |
| **17** | B on left | left | -1 | 1 |
| **18** | B on left | straight or right | 0 | -1 |
| **19** | B on right | right | -1 | 1 |
| **20** | B on right | straight or left | 0 | -1 |

The complete list of sensory states, behaviors and reinforcement signal asso-

ciations that we used in our experiments is provided in Table 4.1. In the table, the columns labeled as "Sensor" and "Behavior" show the sensor states and actions of the network, respectively. The reward functions corresponding to the two seasons, "Summer" and "Winter", are also shown. It should be noted that for some sensory states, multiple reward function associations may be triggered. In these cases, the associations that concern the behaviors to collect/avoid items are given priority. We should also note that the reward functions described here do not specify the reinforcement signal outcomes for *all* possible sensor-behavior combinations. Indeed, in total there are 192 possible sensor-behavior associations (resulting from 3 possible behavior outcomes for each of $2^6$ possible sensor states).

The first two reward associations are defined to encourage the agent to explore the environment. Otherwise, the agent may get stuck in a small area, for example by performing only actions such as going left or right when there is nothing present. Sensor state labeled as "nothing" refers to an input to the network equal to $[0,0,0,0,0,0]$.

Reward associations 3 through 8 specify the reinforcement signals for the behaviors in relation to the wall, and are the same in both summer and winter seasons. It is expected that the agent learns to avoid the wall. Therefore, we define positive reward signals for the states where there is a wall, and the agent picks a behavior that avoids a collision (IDs: 3, 5, 6). Conversely, we define punishment signals for the sensor-behavior associations where the agent collides to the wall (IDs: 4, 6, 8). For instance, the sensor state of the association given in ID. 3 refers to the input to the network as $[0,0,1,1,0,0]$, and provides reward if the agent decides to go left or right (and, this behavior is desired in both summer and winter seasons).

Reward associations 9 through 14 and 15 through 20 define sensor-behavior and reinforcement signal associations regarding the "G" and "B" types of items respectively. The reinforcement signals of the seasons are reversed for these two seasons. In summer, the agent is expected to collect "G" items and avoid "B" items, whereas in winter the agent is expected to collect "B" items and avoid "G" items.

We perform an experiment by first placing into the environment an agent, initialized with a random ANN, and starting with the summer season. We then allow the agent to interact in a continuous learning setting for 5000 action steps. At the end of the summer season, we let the agent continue its interaction with the environment for another 5000 action steps by keeping its network configuration and only changing the season to winter. At the end of the given number of action steps, we perform two more seasonal changes in the same fashion.

The performance of the agent is the average of the performances over the four seasons.

For all experiments, the agents are set to pick a random action with a probability of 0.02 regardless of the actual output of their ANNs. Random behaviors are introduced to avoid getting stuck in a behavioral cycle. Such random behaviors, though, are not taken into account in the synaptic update procedure.

## 4.4 Experimental Results

In this section, we discuss the results of the evolved plasticity rules evolved in our experiments. We then show the learning and adaptation processes of the best performing evolved plasticity rule during a foraging task in continuous learning settings, with and without validation. Finally, we show additional experimental results reporting the best performing evolved plasticity rules used with ANNs with various number of hidden neurons.

For comparison, we provide the results of two algorithms: the results of agents with ANNs controllers that are optimized by using the Hill Climbing (HC) algorithm [19], and the results of a rule-based agent that is controlled by hand-coded rules without using an ANN.

In the case of the HC algorithm, we start with an ANN (the same architecture as used in the plasticity experiments) where all of its weights are randomly initialized between $[-1, 1]$, evaluated on our foraging task (using the same evaluation settings for 5000 action steps) for the first summer season to find its fitness, and assigned as the best network. We then iteratively generate a candidate network by randomly perturbing all the weights of the best network using Gaussian mutation with zero mean 0.1 standard deviation, evaluate on our foraging task for a given season, and replace the best network with the candidate network if its fitness value is better than the fitness of the best network. We repeat this process consecutively for summer, winter, summer and winter seasons for 1000 iterations each, in total of 4000 iterations combined. When a seasonal change happens, we keep the best network and continue the optimization procedure as specified.

Figure 4.4 shows the average results of 100 runs ($100 \times 4000$ iterations) of the HC algorithm for four seasons. The average values and standard deviations of fitness values, collected number of Green and Blue items, and Wall hits of the 10th, 500th and 1000th iterations for each season are given in Table 4.2. The values for Summer1, Winter1, Summer2 and Winter2 presented in table corresponds to the iteration numbers in ranges 0-1000, 1001-2000, 2001-3000

and 3001-4000 on $x$-axis of Figure 4.4 respectively.

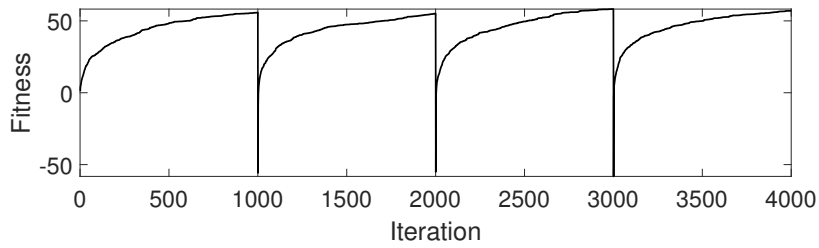



Figure 4.4: The average fitness value during 100 runs of the offline optimization process of the agents using the HC algorithm without the plasticity property. The process starts with a randomly initialized ANN which is trained on our foraging task starting with the summer season, and every 1000th iteration, the seasons is changed.

Table 4.2: The average and standard deviations of the performance statistics of 100 runs of the offline optimization procedure of the agents using the Hill Climbing algorithm.

| Result | Iteration | Summer1 | Winter1 | Summer2 | Winter2 |
|---|---|---|---|---|---|
| **Fitness** | **10th** | $9.20 \pm 10.12$ | $9.79 \pm 11.83$ | $8.74 \pm 10.24$ | $10.41 \pm 12.04$ |
| | **500th** | $48.34 \pm 24.82$ | $47.42 \pm 19.15$ | $49.59 \pm 18.02$ | $49.94 \pm 17.04$ |
| | **1000th** | $55.81 \pm 23.04$ | $55.08 \pm 20.00$ | $58.21 \pm 17.08$ | $56.95 \pm 18.35$ |
| **G** | **10th** | $11.61 \pm 12.37$ | $5.02 \pm 7.63$ | $13.32 \pm 12.22$ | $4.48 \pm 6.46$ |
| | **500th** | $49.36 \pm 24.48$ | $1.58 \pm 4.87$ | $51.71 \pm 17.43$ | $1.59 \pm 4.62$ |
| | **1000th** | $56.56 \pm 22.88$ | $0.85 \pm 3.29$ | $59.65 \pm 16.89$ | $1.38 \pm 4.09$ |
| **B** | **10th** | $2.41 \pm 5.59$ | $14.81 \pm 14.40$ | $4.58 \pm 6.66$ | $14.89 \pm 14.05$ |
| | **500th** | $1.02 \pm 3.23$ | $49.00 \pm 19.06$ | $2.12 \pm 4.73$ | $51.53 \pm 16.33$ |
| | **1000th** | $0.75 \pm 1.43$ | $55.93 \pm 19.88$ | $1.44 \pm 3.47$ | $58.33 \pm 17.48$ |
| **W** | **10th** | $614.09 \pm 919.95$ | $693.78 \pm 854.64$ | $770.47 \pm 1101.77$ | $702.46 \pm 862.67$ |
| | **500th** | $170.26 \pm 463.02$ | $157.15 \pm 446.59$ | $146.47 \pm 417.37$ | $183.05 \pm 485.90$ |
| | **1000th** | $114.92 \pm 375.96$ | $88.30 \pm 351.09$ | $115.72 \pm 398.26$ | $140.97 \pm 439.384$ |

We observe that it takes about 1000 iteration for the networks to reach 56.51 fitness value on average at the end of each season. Since the networks are well optimized for the task, a sudden decrease in their performance at the beginning of each seasonal change is observed. In about 10 generations, the fitness values

are increased around 9 in each season. We see clear increasing and decreasing trends in number of collected items depending on the season while the number of iterations increase. Moreover, the number of wall hits are also reduced although it is relatively high even at the end of the iterations for each season.

This agent that is controlled by hand-coded rules without ANNs has a "perfect knowledge" of which items to collect/avoid in each season throughout its evaluation process. Also, it moves straight if there is nothing around, and makes a turn when it encounters a wall (this behavior is expected to improve the exploration of the environment). The mean and standard deviation of the rule-based agent for 100 trials are given in Table 4.6. Since, there is no learning or optimization process involved in this case, we only show its results at the end of a Summer and Winter seasons.

Table 4.3: Statistics of the complete list of distinct evolved plasticity rules found by the GA, ranked by their median fitness. The columns "Number of Evolved Rules" shows the number of evolved rules found for each distinct rule; "Median", "Std.D", "Max", "Min" show their median, standard deviation, max and minimum fitness; "$\eta$ Mean", "$\eta$ Std.D." show their average learning rate and standard deviations, respectively. The rest of the columns, encoded in 2-bits, represent the activation states of pre- and post-synaptic neurons when modulatory signal is $-1$ or 1.

| | | | | | | | | $m\ -1$ | | | | $m\ 1$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ID.** | **Number of Evolved Rules** | **Median** | **Std.D.** | **Max** | **Min** | **$\eta$ Mean** | **$\eta$ Std.D.** | **00** | **01** | **10** | **11** | **00** | **01** | **10** | **11** |
| **1** | 164 | 48.63 | 0.83 | 49.96 | 42.95 | 0.0375 | 0.008 | 0 | 0 | 1 | -1 | 0 | 0 | 0 | 0 |
| **2** | 23 | 44.23 | 1.05 | 45.88 | 41.55 | 0.0167 | 0.004 | -1 | 1 | 1 | -1 | 0 | 0 | 0 | 0 |
| **3** | 3 | 42.35 | 2.65 | 46.45 | 41.48 | 0.0192 | 0.003 | 1 | 0 | 1 | -1 | 0 | 0 | 0 | 0 |
| **4** | 19 | 28.35 | 0.51 | 29.05 | 27.22 | 0.0488 | 0.009 | -1 | 1 | 1 | -1 | 1 | 0 | -1 | 0 |
| **5** | 1 | 27.28 | 0 | 27.28 | 27.28 | 0.0182 | 0 | -1 | -1 | 1 | 0 | 1 | 0 | -1 | 0 |
| **6** | 9 | 26.70 | 1.48 | 27.91 | 22.80 | 0.0118 | 0.003 | -1 | 1 | 1 | -1 | -1 | -1 | 1 | 1 |
| **7** | 16 | 26.54 | 0.89 | 28.13 | 24.65 | 0.0092 | 0.002 | 0 | 1 | 1 | -1 | -1 | -1 | 1 | 1 |
| **8** | 2 | 25.91 | 0.07 | 25.97 | 25.86 | 0.0096 | 0.0008 | 1 | 1 | 1 | -1 | -1 | -1 | 1 | 1 |
| **9** | 1 | 23.55 | 0 | 23.55 | 23.55 | 0.0273 | 0 | -1 | 1 | 1 | -1 | 0 | -1 | 0 | 1 |
| **10** | 2 | 22.11 | 1.26 | 23 | 21.21 | 0.0052 | 0.003 | 1 | 1 | 1 | -1 | 0 | -1 | 0 | 1 |
| **11** | 10 | 20.96 | 0.33 | 21.59 | 20.45 | 0.0198 | 0.003 | 0 | 1 | 1 | -1 | 1 | -1 | -1 | -1 |
| **12** | 20 | 20.63 | 0.40 | 21.34 | 19.81 | 0.061 | 0.022 | -1 | 1 | 1 | 0 | 1 | -1 | -1 | -1 |
| **13** | 1 | 12.41 | 0 | 12.41 | 12.41 | 0.0799 | 0 | 0 | 0 | 0 | -1 | 1 | 1 | 0 | -1 |
| **14** | 19 | 10.30 | 0.55 | 10.87 | 8.36 | 0.0662 | 0.018 | 0 | 0 | 0 | -1 | 0 | 1 | 1 | -1 |
| **15** | 10 | 8.82 | 0.54 | 9.59 | 7.97 | 0.0301 | 0.009 | 1 | 1 | -1 | -1 | 1 | 0 | -1 | 1 |

Table 4.4: The statistics (fitness values, standard deviations of the fitness values, number of correctly and incorrectly collected items and their standard deviations, and number of wall hits and its standard deviation) of some rules defined by hand over 100 trials. Columns encoded in 2-bits represent the activation states of pre- and post-synaptic neurons.

| | | | | | | | | | | $m\ -1$ | | | | $m\ 1$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ID.** | **Fitness** | **Std.F.** | **Correct** | **Std.C.** | **Incor.** | **Std.I.** | **Wall** | **Std.W.** | $\eta$ | **00** | **01** | **10** | **11** | **00** | **01** | **10** | **11** |
| **16** | 41.68 | 18.26 | 47.63 | 14.3 | 5.95 | 7.17 | 14.3 | 54.59 | 0.04 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 |
| **17** | 5.6 | 9.28 | 25.2 | 6.88 | 19.7 | 5.28 | 120 | 140.2 | 0.01 | 0 | 0 | 1 | -1 | 0 | -1 | 0 | 1 |
| **18** | 0.2 | 6.13 | 18.3 | 4.48 | 18.1 | 4.27 | 602 | 117.5 | 0.01 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 1 |

Table 4.5: Statistics of the best evolved rule on continuous learning for each season.

| | **Summer 1** | | | | **Winter 1** | | | | **Summer 2** | | | | **Winter 2** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Fit.** | **G** | **B** | **W** | **Fit.** | **G** | **B** | **W** | **Fit.** | **G** | **B** | **W** | **Fit.** | **G** | **B** | **W** |
| **Mean** | 50.81 | 53.37 | 2.56 | 5.53 | 48 | 4.13 | 52.1 | 3.6 | 50.37 | 53.71 | 3.3 | 3.18 | 50.61 | 3.7 | 54.33 | 4.7 |
| **Std. Dev.** | 10 | 9.5 | 1.8 | 3.5 | 9.2 | 2.2 | 8.4 | 3.6 | 10.2 | 9.64 | 1.7 | 3.4 | 10.2 | 1.9 | 9.5 | 6.3 |

### 4.4.1 Evolved versus Defined Plasticity Rules

We collected a total of 300 evolved plasticity rules by performing 30 independent GA runs. The max, min, median, mean and standard deviation of the average fitness values of all evolved plasticity rules are 49.96, 7.97, 47.37, 37.89 and 13.98 respectively.

Table 4.6: The statistics of the hand-coded rule-based agent with perfect knowledge of the task in each season.

| | **Summer** | | | | **Winter** | | | |
|---|---|---|---|---|---|---|---|---|
| | **Fitness** | **G** | **B** | **W** | **Fitness** | **G** | **B** | **W** |
| **Mean** | 67.2 | 67.2 | 0 | 0 | 67.8 | 0 | 67.8 | 0 |
| **Std. D.** | 8.6 | 8.6 | 0 | 0 | 7.9 | 0 | 7.9 | 0 |

We identified in total 15 distinct rules, distinguished only by their discrete part (i.e., without considering the specific value of the learning rate). A complete list of these rules is provided Table 4.3. The first and second column show the distinct rule identifier (ID) and the number of rules found for each distinct rule, respectively. The columns labeled as "Median", "Std.D.", "Max", "Min", "$\eta$ Mean", and "$\eta$ Std.D." show the median, standard deviation, max and min values of the fitness, and the average and standard deviations of the learning rates per each distinct rule respectively. In the remaining columns, we report the activation states of pre- and post-synaptic neurons $a_j$ and $a_i$ (encoded as 2-bits) when $m = -1$ and $m = 1$. The most successful rules, shown in the first three rows, constitute 63% of all collected rules, and achieve a fitness values greater than 40. They perform synaptic updates only when $m = -1$. The rest of the rules perform synaptic updates also when $m = 1$. However, their performance is significantly worse in terms of their fitness values. For instance, the rules given in the fourth row achieve the best fitness value of 29.05 which is 20 points lower than the best rule.

The overall best performing evolved plasticity rule (the best of 164 rules with ID. 1 shown in Table 4.3) achieved a fitness value of 49.96 with a standard deviation of 9.97, corresponding to an average of 53.39 correctly collected and 3.43 incorrectly collected items, with standard deviations of 9.31 and 2.04 items respectively, and on average incurred 4.27 wall hits, with a standard deviation of 4.46 hits. The learning rate of this rule is $\eta = 0.039$. This rule performs synaptic updates in two cases only, i.e. when the network produces undesired

behavior ($m = -1$). In the first case, the plasticity rule increases the synaptic weights between $a_j$ and $a_i$ when $a_j$ is active but $a_i$ is not. This may facilitate finding new connections. The second case implements an anti-Hebbian learning where the plasticity rule decreases the synaptic weights when both $a_j$ and $a_i$ are active.

For comparison, Table 4.4 provides the statistics of some plasticity rules that we defined by hand. The columns "Fitness", "Std.F.", "Correct", "Incor.", "Std.C." and "Std.I." show the fitness, standard deviation of the fitness, average number of correctly and incorrectly collected items, and their standard deviations, respectively. The remaining columns show the details of the plasticity rule.

The rule with ID. 16 was defined by taking the best performing evolved rule (ID. 1) and replacing the part corresponding to $m = -1$, activations=10 with 0. After this change, the rule performs synaptic updates only when pre- and post-synaptic neurons are active and the network produces an undesired outcome ($m = -1$). When this rule is used, the performance of the networks are significantly better than other rules defined by hand. However, the performance is worse than the rule without this replacement. Surprisingly though, the GA did not find this rule, as shown in the distinct rule list given in Table 4.3, even though it performs better than most of all the other rules. This may be due to the convergence of the evolutionary process to the rules that perform better, specifically the rules given in the first three rows.

The rules given in row IDs. 17 and 18 were also defined by hand, but showed the worst performances. In particular, the rule given in row ID. 18 performs Hebbian/anti-Hebbian learning as it increases/decreases the synaptic weights between neurons when they are both active and the network produces a desired/undesired outcome. Instead, the rule given in row ID. 17 performs synaptic updates on two additional activation combinations w.r.t. rule with ID. 18, in order to facilitate the creation of new connections. Even though this latter rule performs better than the rule ID. 18, its performance is not better than the evolved plasticity rule with the worst performance.

The reason why the performance of the plasticity rules that perform synaptic updates also when $m = 1$ is worse than the ones that perform synaptic update only when $m = -1$ is likely due to the design of the task and the reward function. Since the reward function keeps providing rewards while a network is achieving the desired outcomes, the accumulation of these rewards can "overcharge" the synaptic weights and may cause forgetting after a certain number of synaptic updates. The issue of forgetting already known knowledge/skills due to the acquisition of new knowledge/skills in ANNs is usually referred as "catastrophic

forgetting" [73].

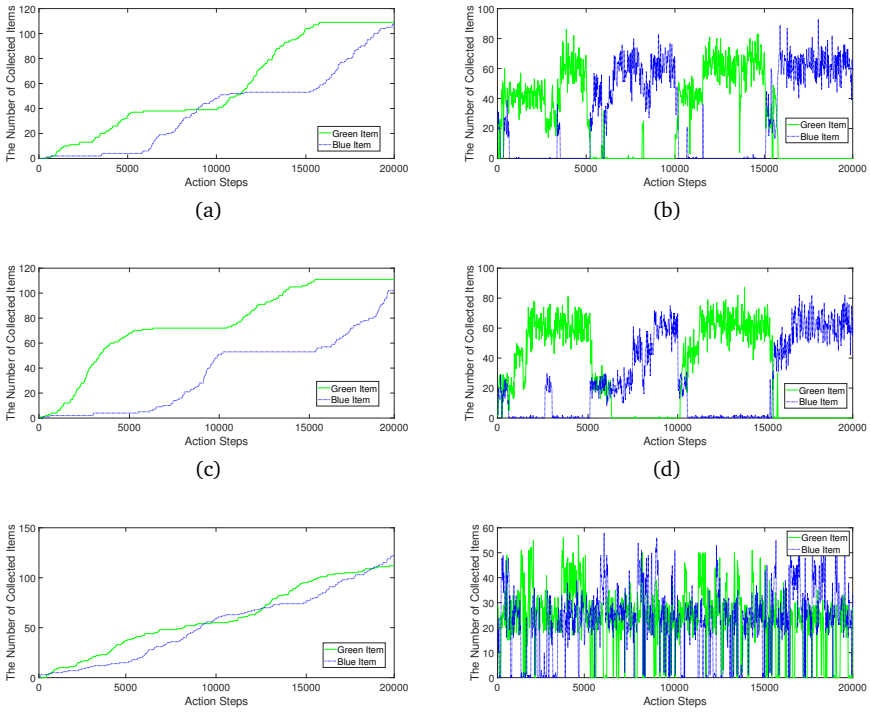
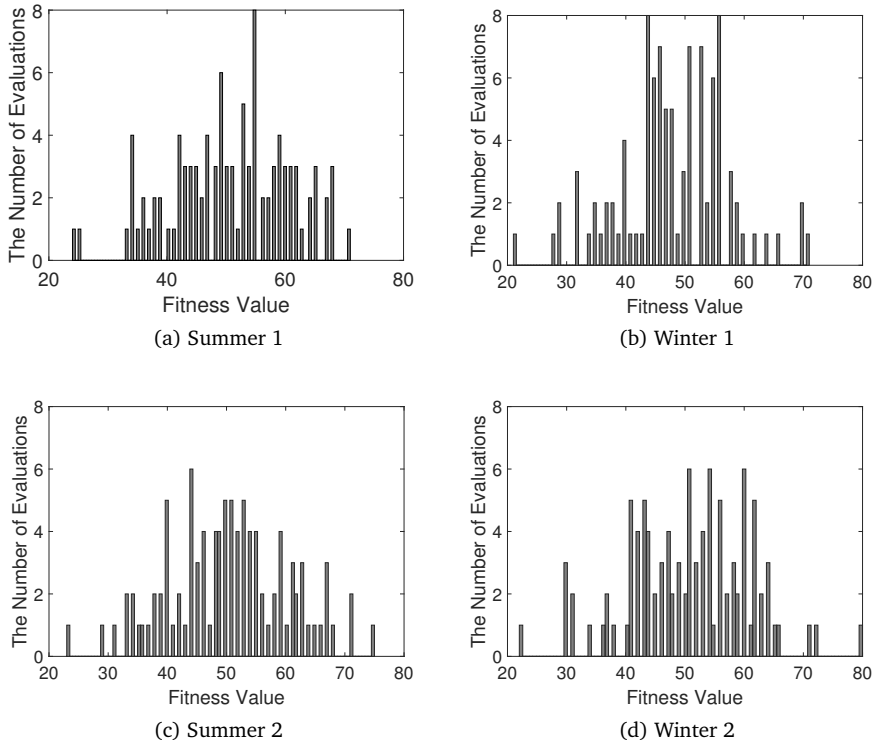



Figure 4.5: The statistics of three selected plasticity rules during a single run of a foraging task over four seasons. Each row of the figures provides the results of the best plasticity rules from the rule types with IDs. 1, 2, and 6 respectively. Figures in the first column show cumulative results of the number of items collected, whereas figures in the second column show the number of items collected when the agent is tested independently for 5000 action steps using the configurations of its ANN at the time of the measurement.

### 4.4.2 Performance of the Networks During a Foraging Task

Figure 4.5 shows the statistics of some selected plasticity rules during a single run of a foraging task over four seasons. The overall process lasts 20000 action

steps in total, consisting of four seasons of 5000 action steps each. The foraging task starts with the summer season and switches to the other season every 5000 action steps. Measurements were sub-sampled at every 20th action step to allow better visualization. The figures given in the first column show the cumulative number of items of both types collected throughout the process, whereas each measurement given in the figures in the second column shows the number of items collected when the agent is tested independently for 5000 action steps using its ANN configuration at the time of the measurement.




(a) Summer 1 (b) Winter 1

(c) Summer 2 (d) Winter 2

Figure 4.6: The distribution of fitness values of the best evolved rule from ID.1 in Table 4.3 over four seasons. The $x$- and $y$-axes of each subfigure show the fitness value and number of evaluations, respectively.

Figures 4.5a and 4.5b show the results of the best rule from ID. 1, Figures

4.5c and 4.5d show the results of the best rule from ID. 2, and Figures 4.5e and 4.5f show the results of the best rule from ID. 6. In the case of the rules from IDs. 1 and 2, the agent learns quickly to collect the correct type of items in each season. The number of correctly collected items from the previous season stabilizes, and the number of incorrectly collected items from the previous season increases in the next season after a seasonal change occurs. We observe in Figures 4.5b and 4.5d that there are distinct and stable seasonal trends for each season. The noise in the measurements is due to the sub-sampling and stochasticity of the evaluation process.

In the case of rule from ID. 6, the agent keeps collecting both kinds of items; however, the number of correctly collected type of items is larger than incorrectly collected items in each season. We can observe in Figure 4.5f that the testing performance is not stable throughout the process. The fluctuations show that the configuration of the ANN seems to be frequently shifting the states between learning and forgetting.

Table 4.5 and Figure 4.6 show, respectively, the average statistics and the distribution of the fitness of the best performing evolved plasticity rule in each season over 100 trials. The agent achieves an average fitness value of about 50 in all the seasons except the first winter season where it achieves a fitness value of 48.

Table 4.7: The statistics of the best performing evolved rule with validation.

| | **Summer** | | | | **Winter** | | | |
|---|---|---|---|---|---|---|---|---|
| | **Fitness** | **G** | **B** | **W** | **Fitness** | **G** | **B** | **W** |
| **Mean** | 61.0 | 61.0 | 0 | 0 | 63.0 | 0 | 63.0 | 0 |
| **Std. D.** | 8.62 | 8.62 | 0 | 0 | 8.19 | 0 | 8.19 | 0 |

Table 4.7 shows the average results of the best ANN configuration found through validation. During a foraging task process, the agent is tested independently at every 20th action step, and the configuration of the ANN that achieved the highest fitness value was stored as the best ANN configuration.

To assess the statistical significance of the difference of the results produced by different agents, we use the Wilcoxon rank-sum test [108]. The null-hypothesis, i.e. that the mean of the results produced by two agents (thus, their behavior) are the same, is rejected if the $p$-value is smaller than $\alpha = 0.05$. We perform pairwise comparisons of three sets of results of fitness values for sum-

mer and winter seasons obtained from three agents evaluated over 100 evaluations. More specifically, we compare the results of the hand-coded rule-based agent, with those of the agents that use the best performing evolved plasticity rule in continuous learning settings, with and without validation. Based on the pairwise comparison results, the hand-coded rule-based agent is significantly better than the other two agents with and without validation, with significance levels of $1.9 \cdot 10^{-28}$ and $8.2 \cdot 10^{-05}$ respectively. Furthermore, the results of the agent with validation are significantly better than those without validation, with a significance level of $9.9 \cdot 10^{-14}$.

To gain further insight into these results, we have recorded the behavior of the agent during a foraging task with four seasons each consisting of 3000 action steps[2]. The demonstration shows that the agent is capable of efficiently adapting to the environmental conditions imposed by each season. Even though the desired behavior with respect to the wall should be constant across the different seasons, the agent makes a few mistakes at the beginning of each season by hitting the wall. This may be due to the change of the synaptic weights that affect the behavior of the agent with respect to the wall.

### 4.4.3 Sensitivity Analysis on the Number of Hidden Neurons

Finally, we performed a sensitivity analysis of our results with respect to the number of hidden neurons. In Figure 4.7, we show the average results of 100 trials each performed using the best performing evolved plasticity rule on the ANNs, with various number of hidden neurons. We test networks with 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50 hidden neurons. The results show that there is a 13 point increase on the average fitness value when the number of hidden neurons is increased from 5 to 20. There is also a slight upward trend when larger than 20 hidden neurons are used, which results in a 3 point average fitness gain when the number of hidden neurons is increased from 20 to 50. Moreover, the standard deviations are also slightly reduced while the number of hidden neurons increases, showing that the use of more hidden neurons tends to make the agent's behavior more consistent across different trials.

[2]An online video demonstration of the behavior of an agent for four seasons each consisting of 3000 action steps: `https://youtu.be/9jy6yTFKgT4`.

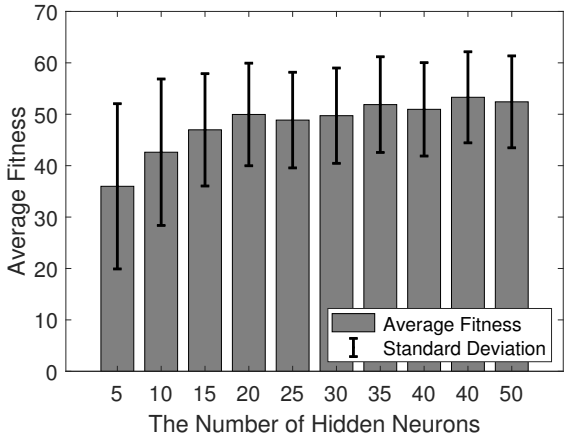
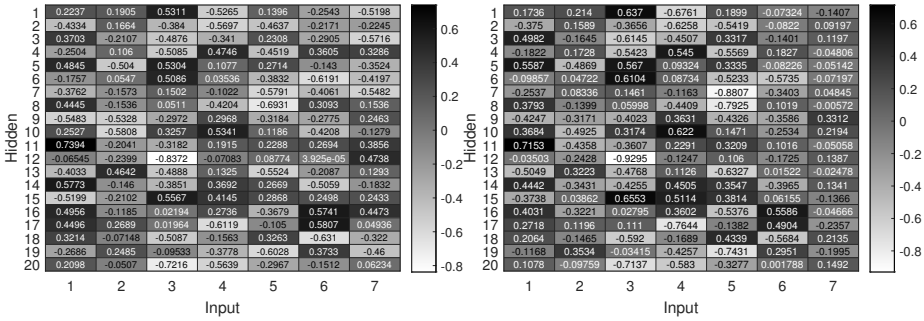



Figure 4.7: The average fitness values of the ANNs with various number of hidden neurons.

### 4.4.4 Change of the Synaptic Weights

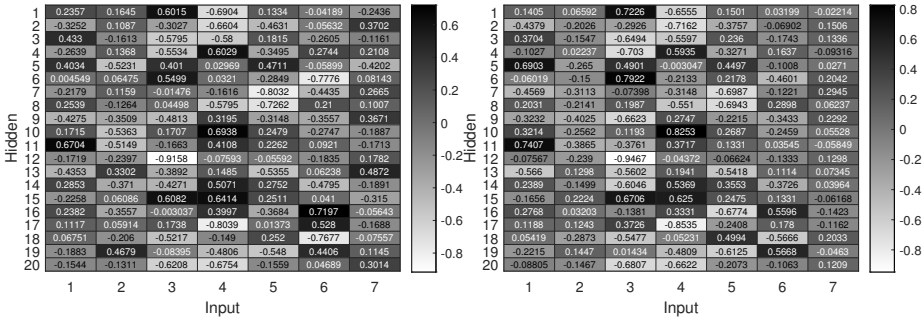
Figures 4.8 and 4.9 show the actual values of the connection weights between input and hidden, and between hidden and output layers, in a matrix form. Each column and row in the figures correspond to a neuron in the input, hidden and output layers. In particular, Figures 4.8a and 4.9a show the initial connection weights between input and hidden, and between hidden and output layers, sampled randomly. We performed three seasonal changes in the following order: summer, winter, summer. We used the best performing evolved plasticity rule to perform synaptic updates during the seasons. We provide the rest of the figures to show the connection weights after each consecutive season.

The connection weights between input and hidden layers appear to be distributed in the range [-1,1]; on the other hand, the connection weights between hidden and output layers tend to be distributed within the range [0,1] at the end of each season. This may be due to the activation function of the output neurons, where only the neuron with the maximum activation value is allowed to fire.




(a) Randomly sampled initial connection weights.

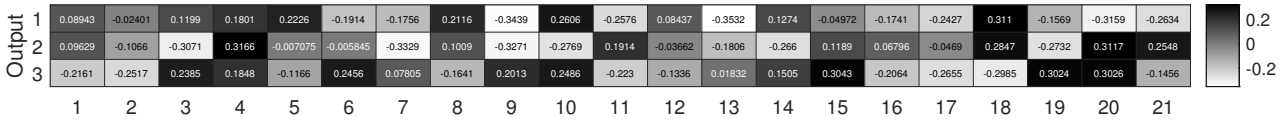
(b) Connection weights after the first summer season.



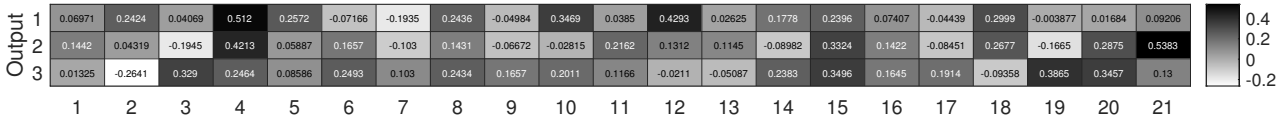

(c) Connection weights after the first winter season.

(d) Connection weights after the second summer season.

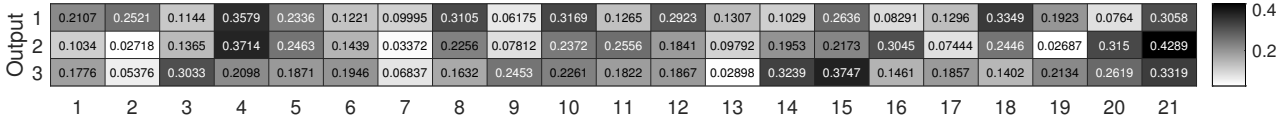
Figure 4.8: Heat map of the intensities of the connection weights between the input and hidden neurons during a single run using the best evolved plasticity rule. The $x$ and $y$-axes show the input and hidden neuron indices respectively (7th column shows the biases). Each connection on the heat map is color coded based on its intensity from $-1$ to 1. Figure 4.8a shows the initial state of the connection weights that are assigned randomly. Figures 4.8b, 4.8c and 4.8d show the weights after summer, winter and summer seasons.




(a) Connection weights after the initial random sampling.




(b) Connection weights after the first summer season.




(c) Connection weights after the first winter season.


(d) Connection weights after the second summer season

Figure 4.9: Heat map of the intensities of the connection weights between the hidden and output neurons during a single run using the best evolved plasticity rule. The $x$ and $y$-axes show the hidden and output neuron indices respectively (21st column shows the biases). Each connection on the heat map is color coded based on its intensity from $-1$ to 1. Figure 4.9a shows the initial state of the connection weights that are assigned randomly. Figures 4.9b, 4.9c and 4.9d show the weights after summer, winter and summer seasons.

## 4.5 Conclusions and Future Work

The plasticity property of biological and artificial neural networks enables learning by modifying the networks' configurations. These modifications take place at individual synapse/neuron level, based on the local interactions between neurons.

In this chapter, we proposed an evolutionary approach to produce autonomous learning in ANNs in a task with changing environmental conditions. We devised plasticity rules to conduct synaptic adjustments inspired by the local plasticity property of the BNNs. The plasticity rules operate at individual synapse level, based on the interactions between artificial neurons. We employed Genetic Algorithms (GA) to explore the search space of the possible plasticity rules for all possible states of the post- and pre-synaptic neuron activations and reinforcement signals.

We evaluated the evolved plasticity rules on an agent-based foraging task, focusing specifically on the adaptation capabilities of the ANNs under changing environmental conditions. In our test scenario, an agent starts with a randomly initialized ANN configuration, and is required to learn collecting/avoiding correct types of items while interacting with the environment. After a certain number of action steps, the types of items to collect/avoid are switched, and the agent is expected to adapt to the new conditions.

The results of the multiple runs of the GA produced in total 300 rules, that we have grouped in 15 distinct plasticity rules (based only on the discrete parts of the rule), among all $3^8 = 6561$ possible rules. Interestingly, more than half of all plasticity rules (164 of 300) converged on a single type of rule that showed the best performance.

To set an upper and a lower bound for the performance of the ANNs with evolved plasticity rules, we performed a set of separate experiments with hand-coded rule-based agents with perfect knowledge and agents with ANN controllers that are optimized using the HC algorithm respectively. We also observed that the fitness of an agent that takes decisions randomly at each action step is zero, as expected, since in this case the agent does not have any intelligent mechanism to distinguish between items to collect; thus, we did not report the performance results of this experimental configuration.

Comparison with a hand-coded rule-based agent with a perfect knowledge of the task showed instead that the performance of the best evolved rule produced agents that can perform the task very well (as good as about 74% of the performance of the hand-coded agent), considering a continuous learning setting where there are no separate resources for training and testing. In addition,

we showed the efficiency of the evolved plasticity rules in searching the configuration space of the networks by performing validation during the learning process. With validation, the best networks could achieve 94% of the performance of the hand-coded rule-based agent (this performance difference may be due to the specific design of the reward function).

The results of the agents with ANNs that are trained using the HC algorithm shows that the average fitness value at 1000th iteration is better than the results of the agents with ANNs trained with the plasticity rules in continuous learning settings without validation. However, the results of the HC has a higher standard deviation. This is reasonable since in continuous learning settings, the learning process (thus the mistakes the agents make) are also involved into the evaluation process. On the other hand, the agents that are trained with plasticity rules with validation performed better than the agents that are trained using the HC.

Surprisingly, the best evolved rule performed synaptic updates only when the network produced undesired output (negative reinforcement signal). This may be due to the specific reward functions we used in the experimentation, which were designed to provide constant reward/punishment while the networks produced desired/undesired outcomes. Intuitively, after the networks learn to perform the task successfully, keep performing synaptic updates may cause degradation in the synaptic weights and result in forgetting.

The approach presented in this chapter is capable of performing synaptic modifications if the reinforcement signals are available after every action. In most complex real-world problems, the reinforcement signals are available after certain period of time, when an agent achieves certain task by performing a sequence of actions. Next chapter, extends on this work to allow learning in these complex cases.

# Chapter 5

# Evolving Plasticity for the Distal Reward Problem

Reinforcement signals are used as modulatory signals to guide the learning process by signaling how and when the synaptic modifications are performed [22, 91, 116]. In the previous chapter, we studied the case where the reinforcement signals were required after each output of the network, to help associating the activation patterns with desired outputs. If the reinforcement signals are available only after a certain period of time, but not immediately after each action step, it may not be possible to directly associate the activation patterns of the neurons to the reinforcement signals to perform necessary synaptic changes. This is known as the *distal reward problem* [43, 94], and it cannot be addressed using Hebbian learning in its basic form, but requires a modified learning model to take into account the neuron activations over a certain period of time.

In this chapter[1], we modify Hebbian learning mechanism proposed in Chapter 4 and propose *delayed synaptic plasticity (DSP)* for enabling plasticity in ANNs in tasks with distal rewards [114]. We introduce the use of *neuron activation traces (NATs)*, i.e. additional data storage in each synapse that keep track of the average pre- and post-synaptic neuron activations. We use discrete DSP rules to perform synaptic updates based on the NATs and a reinforcement

[1]This chapter is integrally based on:
[114] Anil Yaman, Giovanni Iacca, Decebal Constantin Mocanu, George Fletcher, and Mykola Pechenizkiy. Learning with delayed synaptic plasticity. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, pages 152–160, New York, NY, USA, 2019. ACM

signal provided after a certain number of activation steps, that we refer to as an *episode*. The reinforcement signals are based on the performance of the agent relative to the previous episode (i.e. if the agent performs better/worse relative to the previous episode, a positive/negative reinforcement signal is provided). We further introduce competition in incoming synapses of neurons to stabilize delayed synaptic updates and encourage self-organization in the connectivity. As such, the proposed DSP scheme is a distributed and self-organized form of learning which does not require global information of the problem.

We employ *genetic algorithms (GA)* to evolve DSP rules to perform synaptic changes on *recurrent neural networks (RNNs)* that have to learn to navigate in a triple T-maze to reach a goal location. To test the robustness of the evolved DSP rules, we evaluate them for multiple trials with various goal positions. We then note how the process of training RNNs for a task using DSP can be seen as analogous to optimizing them using a single-solution metaheuristic, except for the fact that in contrast to general-purpose metaheuristics, the DSP rules take into account the (domain-specific) knowledge of the local neuron interactions for updating the synaptic weights. Therefore, to assess the effect of the domain knowledge introduced with DSP, we compare our results with a classic *hill climbing (HC)* algorithm. Our results show that DSP is highly effective in speeding up the optimization of RNNs relative to HC in terms of number of function evaluations needed to converge. On the other hand, the NATs data structure introduces an additional computational complexity.

The rest of the Chapter is organized as follows: in Section 5.1, we discuss Hebbian learning and the distal reward problem; in Section 5.2, we introduce our proposed approach for DSP and provide a detailed description of the evolutionary approach we used to optimize DSP; in Section 5.4, we present our experimental setup; in Section 5.5, we provide a comparison analysis of our proposed approach and the baseline HC algorithm; finally, in Section 5.6, we discuss our conclusions.

## 5.1 Distal Reward Problem

When reinforcement signals are available after a certain period of time, it may not be possible to associate the neuron activations that contributed to receiving the reinforcement signals. This is referred to as the *distal reward problem*, and has been studied in the context of time dependent plasticity [28, 43, 94].

From a biological viewpoint, *synaptic eligibility traces (SETs)* have been suggested as a plausible mechanism to trace the activations of neurons over a cer-

tain period of time [28] by means of chemicals present in the synapses. According to this mechanism, co-activations of neurons may trigger an increase in the SETs, which then decays over time. Therefore, their level can indicate a recent co-activation of neurons when a reinforcement signal is received, and as such be used for distal rewards.

## 5.2 Learning with Delayed Synaptic Plasticity

In this chapter, we focus on a maze-navigation task in a triple T-maze environment (see Section 5.4). Since this task requires memory capabilities, we use a RNN models to control the behavior of the agents in maze.

The learning process of an RNN is a search process that aims to find an optimal configuration of the network (i.e., its synaptic weights) that can map the sensory inputs to a proper sequence of actions in order to achieve a given task. We use the proposed DSP rules, and the HC algorithm [19] independently, to perform the learning the RNNs for the specific tripe T-maze navigation task, and compare their results. Both of these approaches perform synaptic updates based on the progress of the performance of an individual agent in consecutive episodes; however, the DSP rules incorporate the knowledge of the local neuron interactions, while HC uses a certain random perturbation heuristic that does not incorporate any knowledge. Here, we assume that the progress of the performance of an agent can be measured relative to its performance in a previous episode. We refer to the measure of the performance of an agent in a single episode as "episodic performance" (EP). We should note that we formalize our task as a minimization problem. Therefore, in our experiment an agent with a lower EP is better.

The illustration of the optimization processes of the RNNs using the DSP rules and the HC algorithm are given in Figures 5.1a and 5.1b respectively. Both algorithms run for a pre-defined number of episodes ($N_{episodes}$), starting from an RNN with randomly initialized synaptic weights, and record the best encountered RNN throughout the optimization process.

In the DSP-based algorithm illustrated in Figure 5.1a, the synaptic updates are performed after each episode —thus, the synaptic weights of the RNN during an episode are fixed— using DSP rules which take the RNN, NATs, and a modulatory signal as inputs. The NATs provide the average interactions of post- and pre-synaptic neurons during an episode. The structure of the NATs alongside with the DSP rules are explained in Section 5.3 in detail. The modulatory signal is used as a reinforcement which depends on the performance of

the agent in the current episode relative to its performance during the previous episode. If the current episodic performance $EP_e$ is lower than the previous episodic performance $EP_{e-1}$, then the modulatory signal $m$ is set to 1 (reward), otherwise $m$ is set to $-1$ (punishment).

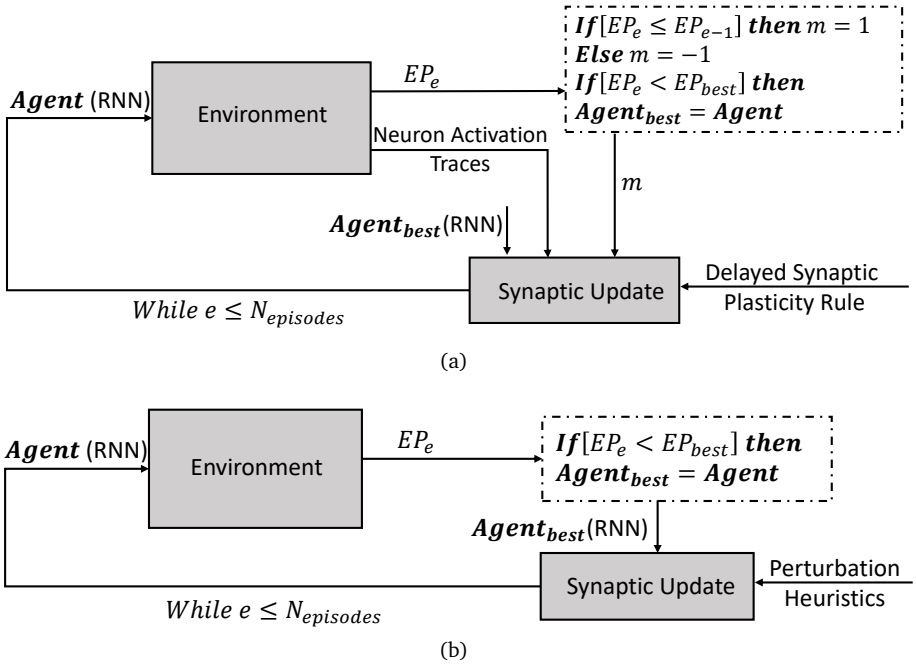



Figure 5.1: (a) The learning process by using the delayed synaptic plasticity, and (b) the learning process by optimizing the parameters of the RNNs using the hill climbing algorithm.

The HC algorithm that is illustrated in Figure 5.1b performs instead synaptic updates after each episode using a perturbation heuristic that does not assume any knowledge of the neuron activations. We use a Gaussian perturbation with a zero mean and unitary standard deviation and scale it with a parameter $\sigma$ to perturb all the synaptic weights of the RNN of the best agent. The synaptic update procedure generates a new candidate $\boldsymbol{Agent}$ that is tested in the environment for the next episode and replaced with the best RNN if it performs

better. Conventionally, in standard HC the measure of the performance of an agent in an episode would be called "fitness". However, we refer to it as EP, to make it analogous to the algorithm that uses DSP rules.

## 5.3 Evolving Delayed Synaptic Plasticity

We propose delayed synaptic plasticity to allow synaptic changes based on the progress of the performance of an agent relative to its performance during the previous episode, i.e.:

$$\Delta w_{i,j}(e) = DSP(NAT_{i,j}, m, \theta) \tag{5.1}$$

$$w'_{i,j}(e) = w_{i,j}(e) + \eta \cdot \Delta w_{i,j}(e) \tag{5.2}$$

where the synaptic change $\Delta w_{i,j}(e)$ between neurons $i$ and $j$ after an episode $e$ is computed based on their NATs, the modulatory signal $m$ (which can either be 1 or $-1$, see Section 5.2), and a threshold $\theta$ that is used to convert NATs into binary vectors. The resulting DSP synaptic changes can be of three types (*decreased*, *stable*, or *increased*), i.e. at each time step $DSP(NAT_{i,j}, m, \theta) \in \{-1, 0, 1\}$. In Equation (5.2), the synaptic change is scaled with a learning rate $\eta$. Subsequent to the synaptic updates of all synapses, the synaptic weights for the next episode $w_{i,j}(e+1)$ are scaled using equation given in Equation (4.7) to prevent an indefinite increase/decrease, and allowing self-organized competition between synapses.

The NAT data structure is illustrated in Figure 5.2. It keeps track of the frequencies of the pre- and post-synaptic neuron activation states throughout an episode. We use four-dimensional vectors for each synapse —since there are four possible states for the activations of pre- and post-synaptic neurons— to store the number of times the pre- and post-synaptic neurons were in one of the following states: 00, 01, 10, and 11, where the first and second bits represent the pre- and post-synaptic neuron activations, and 0 and 1 represent non-active and active states of neurons respectively. At the beginning of an episode, all NATs are initialized as zeros; and the end of an episode, they are divided by the number of total activations to convert them into frequencies.

The NATs are used in their binary forms to discretize the search space. Each frequency in a NAT is converted to either 0 or 1 based on a threshold $\theta$ (1 for the frequencies more than $\theta$, and 0 for the ones that are below $\theta$). Thus, a DSP rule,

as illustrated in Table 5.1, is a combined form of 32 synaptic change decisions provided for all possible states of the binary forms of the NATs and the signal $m$.

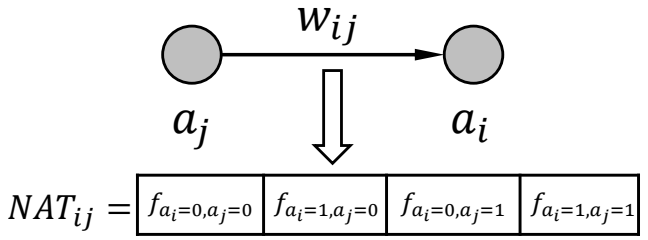



Figure 5.2: The neuron activation trace $NAT_{i,j}$ of the pre- and post-synaptic neuron activations $a_j$ and $a_i$.

We use genetic algorithms (GAs) to evolve three possible synaptic changes $\Delta w = \{-1, 0, 1\}$ (*decrease, stable, increase*) for each possible states of the NATs and $m$. In total, there are 32 possible states that can take three possible values. Thus, there is a total of $3^{32}$ number of possible DSP rules. In addition to the discrete part, we optimize four continuous parameters $(\eta, \theta, \alpha_h, \alpha_o)$ by including them into the genotype of the individuals. We used suitable evolutionary operators separately for discrete and continuous variables. The details of the GA are provided in Section 5.4.1.

Table 5.1: A delayed synaptic plasticity rule visualized in a tabular form. Depending on a certain threshold $\theta$ for converting the binary forms of specific NATs, the DSP rule provides the synaptic changes $x_1, x_2, \ldots, x_{32}$ for all possible combinations of binary NATs and $m$ states.

| $NAT^{\theta}$ | | | | $m$ | $\Delta w$ |
|---|---|---|---|---|---|
| **00** | **01** | **10** | **11** | | |
| 0 | 0 | 0 | 0 | $-1$ | $x_1$ |
| 0 | 0 | 0 | 0 | 1 | $x_2$ |
| … | … | … | … | … | … |
| 1 | 1 | 1 | 1 | 1 | $x_{32}$ |

## 5.4 Experimental Setup

In the following sections, we provide the details of our experimental setup including the test environment, the architecture of the agent, and the details of the GA.

### 5.4.1 Triple T-Maze Environment

A visual illustration of the triple T-maze environment is given in Figure 5.3. The triple T-maze environment consists of $29 \times 29$ cells. Each cell can be occupied by one of five possibilities: *empty, wall, goal, pit, agent*, color-coded in white, black, blue, green, red respectively. The starting position of the agent is illustrated in red. There are eight final positions, illustrated in blue or green. Among eight final positions, one of them is assigned as the goal position (in blue), and the rest as pits (in green). Starting from the initial position, an agent navigates the environment for a total of 100 action steps. To solve the task, it is required to find a set of connection weights of the recurrent neural network that can allow the agent to achieve the goal from the starting position.

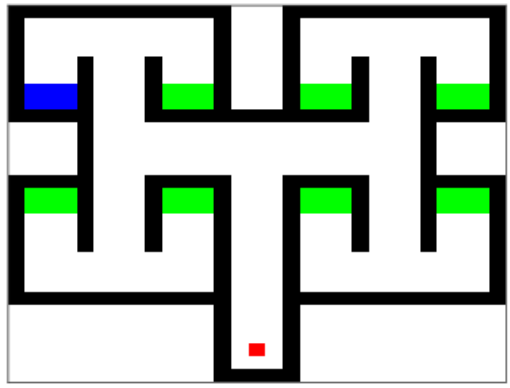


Figure 5.3: Triple T-maze environment. The walls, starting position of the agent, goal and pits are color-coded using black, red, blue and green.

### Agent Architecture

An illustration of the architecture of the agents used for the triple T-maze environment is provided in Figure 5.4. An agent has a certain orientation in the environment, facing one of the four possible directions along the $x$ and $y$ axes of the environment (i.e. north, south, west, east), and can only move one cell at a time horizontally or vertically. It can take sensory inputs from the nearest cells from its left, front and right, depending on its orientation. We let the agent sense only an empty cell, or a wall (i.e., the agent is not aware of the presence of the goal or pits), so the sensory inputs are encoded using one bit, representing an empty cell with 0 and a wall with 1. Thus, there are three bits as input, that the agent uses to decide one of four possible actions: *stop*, *left*, *right*, and *straight*. In case of stop, the agent stays in the same cell with the same orientation for an action step; in cases of left and right, the agent changes its orientation accordingly and proceeds one cell straight; in case of straight, the agent keeps its original orientation and proceeds one cell straight. If the cell the agent wants to move in is occupied with wall, then the agent stays in its original position.

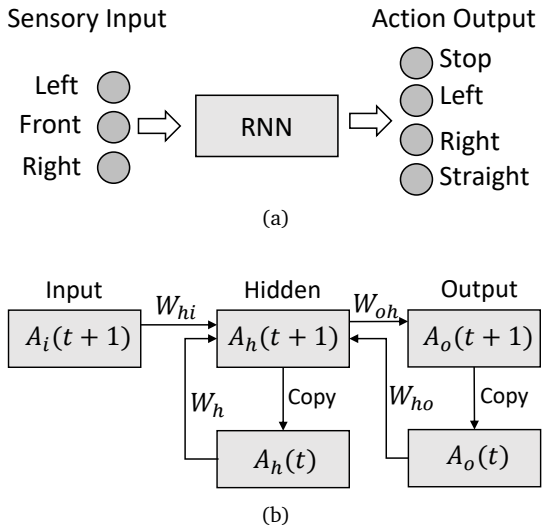



Figure 5.4: (a) The sensory inputs and action outputs of the recurrent neural networks and (b) the architecture of the networks that are used to control the agents.

We use RNNs to control the agents since our task requires memory capabilities (see 2.2), and may not be solved by a static architecture such as a feed-forward ANN. The recurrent and feedback connections between the hidden-hidden and output-hidden layers help agents to keep track of the sequences of actions they perform throughout an episode. The RNN networks used in all experiments consist of 20 hidden neurons unless otherwise is specified. Therefore, the networks consist of $(3+1)\times 20 = 80$ input to hidden connections, $20\times 19 = 380$ hidden to hidden connections (we do not allow self-recurrent connections in $\boldsymbol{W}_h$, thus the diagonal elements of $\boldsymbol{W}_h$ equal to zero) $(20+1)\times 4 = 84$ hidden to output connections, and $4\times 20 = 80$ output to hidden connections, in total of 624 connections between all layers (+1 refers to the bias).

### Genetic Algorithm

We use a standard GA to evolve DSP rules with the exception of a custom-designed mutation operator. The genotype of the DSP rules consists of 32 discrete ($\Delta w$ for all possible states of the NATs and $m$, see Section 5.3) and four continuous values ($\eta \in [0,1], \theta \in [0,1], \alpha_h \in [0,1], \alpha_o \in [0,1]$). Therefore, we encode a population of individual genotypes represented by 36-dimensional discrete/real-valued vectors.

The pseudo-code for the evaluation function is provided in Algorithm 6. To find the DSP rules that can robustly learn the triple T-maze navigation task independently of the goal position and starting from a random RNN state, each evaluation called during the evolutionary process consists of testing each rule for five trials for each of the eight possible goal positions (for total of 40 independent trials). The final positions (one goal in blue, seven pits in green) of the triple T-maze environment are shown in Figure 5.3. We switch the position of the goal with one of the pits to have eight distinct goal positions in total. For all distinct goal positions, the rest of the final positions are assigned as pits. Due to the computational complexity, the maximum number of episodes per trial is set to 100.

The $EP$ of an agent in an episode is computed as follows:

$$EP = \begin{cases} steps(Agent), & \text{if the goal is reached;} \\ N_{steps} + d(XY(Agent), XY(g)), & \text{Otherwise.} \end{cases} \tag{5.3}$$

For each episode, the agent is given 100 action steps ($N_{steps}$) to reach the goal. We aim to minimize the number of action steps that the agents take to reach the goal (in case they do reach it); otherwise, we aim to minimize their distance to the goal. Thus, if the agent reaches the goal, the EP is the number

of action steps the agent took to reach to the goal ($steps(Agent)$). Otherwise, the $EP$ is $N_{steps} + d(XY(Agent), XY(g))$, that is the maximum number of total action steps, plus the distance between the final agent's position and the goal's position. The distance between the agent and the goal is found by the A* algorithm [120], which finds the closest path (distance) taking into account the obstacles. Additionally, the EP of an agent is increased by 5 every time the agent steps into a pit (recalling that the $EP$ is minimized).

Finally, the fitness value (the lower the better) of a DSP rule is obtained by averaging $EP_{best}$ found in each trial (i.e. for 5 trials and 8 distinct goal positions, in total of 40 trials). We should note that the proposed design of the $EP$ and fitness is chosen to introduce a gradient into the evaluation process of agents. For instance, the selection process is likely to prefer agents that on average reach the goal with a smaller number of action steps, and with the least number of interactions with pits.

To limit the runtime of the algorithm, we use a small population size, 14 individuals in total. We employ a *roulette wheel selection* operator with an elite number of four, which copies the four individuals with the highest fitness values to the next generation without any change. The rest of the individuals are selected with a probability proportional to their fitness values. We use *1-point crossover* operator with a probability of 0.5. We designed a custom *mutation* operator which re-samples each discrete dimension of the genotype with a probability of 0.15, and performs a Gaussian perturbation with zero mean and 0.1 standard deviation for the continuous parameters. We run the evolutionary process for 300 generations and finally select at the end of the evolutionary process the DSP rule that achieved the best fitness value.

### 5.4.2 Hill Climbing

We use the HC algorithm as a baseline to compare the results of the DSP. The HC algorithm, visualized in Figure 5.1b, is a single-solution local search algorithm [19]. It is analogous to the DSP, except the fact that it is used as a direct encoding optimization approach where domain knowledge (i.e. knowledge of the neuron activations) is not introduced into the optimization process. Therefore, it provides a suitable baseline to assess the effectiveness of the domain-based heuristic used in the DSP.

A trial of the HC algorithm aims to find an optimum set of RNN weights that allows the agent to reach a given goal position, starting from the starting position. All the connection weights of the RNN (644 in total) are directly encoded into a single real-valued vector that represents an $\boldsymbol{Agent}$ (candidate

solution). At the beginning of the algorithm, each variable of the $\boldsymbol{Agent}$ is randomly initialized in the range $[-1,1]$ with uniform probability. The $\boldsymbol{Agent}$ is then evaluated and assigned as $\boldsymbol{Agent_{best}}$, with fitness equal to $EP_{best}$. After the evaluation, $\boldsymbol{Agent_{best}}$ is perturbed using a *perturbation heuristic* to generate a new $\boldsymbol{Agent}$ as follows:

$$\boldsymbol{Agent} = \boldsymbol{Agent_{best}} + \sigma \cdot \mathcal{N}(0,1) \tag{5.4}$$

where $\mathcal{N}$ is a random vector whose length is the same as that of $\boldsymbol{Agent_{best}}$, and each dimension is independently sampled from a Gaussian distribution with zero mean and unitary standard deviation, scaled by the parameter $\sigma$.

The $\boldsymbol{Agent}$ is evaluated and if its $EP_e$ is smaller than $EP_{best}$, $\boldsymbol{Agent_{best}}$ is replaced by the new $\boldsymbol{Agent}$. The perturbation and evaluation processes are performed iteratively for 100 episodes (evaluations). The overall performance of the HC algorithm is then obtained by averaging the $EP_{best}$ from 40 trials (consisting of 5 trials for 8 distinct goal positions), that is the same evaluation procedure used to evaluate DSP rules.

The performance of the HC algorithm may depend on the parameter of the perturbation heuristic. Thus, to provide a fair comparison with DSP, we optimize the parameter $\sigma \in [0,1]$ with respect to the hyper-parameters of the RNN model $\alpha_h \in [0,1]$ and $\alpha_o \in [0,1]$ by using a GA with the same settings used to optimize the continuous part of the DSP rules (see Section 5.4.1). We refer these three parameters as HC parameters.

## 5.5 Experimental Results

Figures 5.5a and 5.5b show the change of the best fitness value during 15 independent evolutionary optimization processes of the DSP rules and the HC parameters respectively. We run all the experiments on a single-core Intel Xeon E5 3.5GHz computer; therefore, we fix the number of generations to 300, to keep the runtime of the algorithm reasonably limited. In each generation, the best fitness value obtained by the DSP rules and the HC algorithm with evolved parameters are shown. A complete list of the evolved DSP rules can be found in Appendix C.

The initial DSP rules obtain an average fitness of 113.79, with a standard deviation of 4.30; at the end of the evolutionary processes, they achieve an average fitness of 81.39, with a standard deviation of 4.02. On the other hand, the initial HC parameters obtain an average fitness value of 98.71, with a standard deviation of 1.64; and at the end of the evolutionary process, they achieve

an average fitness of 93.50, with a standard deviation of 0.87. We used the Wilcoxon test to statistically assess the significance of the results produced by the evolutionary processes [108]. The null-hypothesis that the mean of the results produced by two processes are the same is rejected if the $p$-value is smaller than $\alpha = 0.05$. In our case, the results of the evolutionary processes of DSP rules are statistically different (better than the HC results) with a $p$-value of $3.3 \times 10^{-6}$.

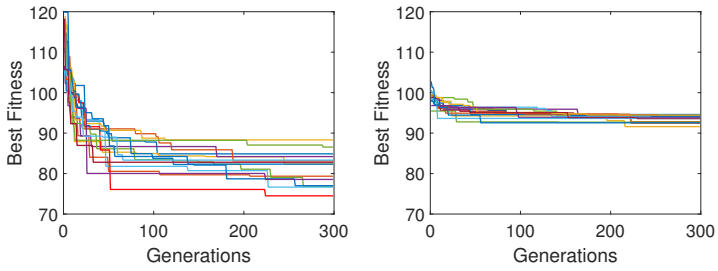



Figure 5.5: The change of the best fitness during 15 independent evolutionary processes for optimizing (a) the DSP rules and (b) the HC parameters. The $y$-axes of figures are scaled between 70–120 to allow a better visual comparison.

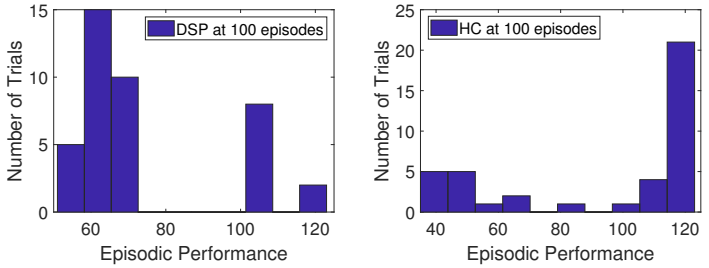



Figure 5.6: The distribution of the episodic performance of 40 trials using the best performing evolved DSP rule (a) and HC parameters (b) trained for 100 episodes.

The distribution of the episodic performance of the best evolved DSP rule and HC parameters are given in Figure 5.6. The trials with an episodic performance smaller than 100 indicate that the goal is achieved in that trial. Thus, 75% of the 40 trials reached the goal when the agents are trained with the DSP rules. On the other hand, only 35% of the trials reached the goal when the agents are trained using the HC. A Wilcoxon rank-sum test (calculated on the EPs) shows that the DSP rule is better with a $p$-value of 0.03.

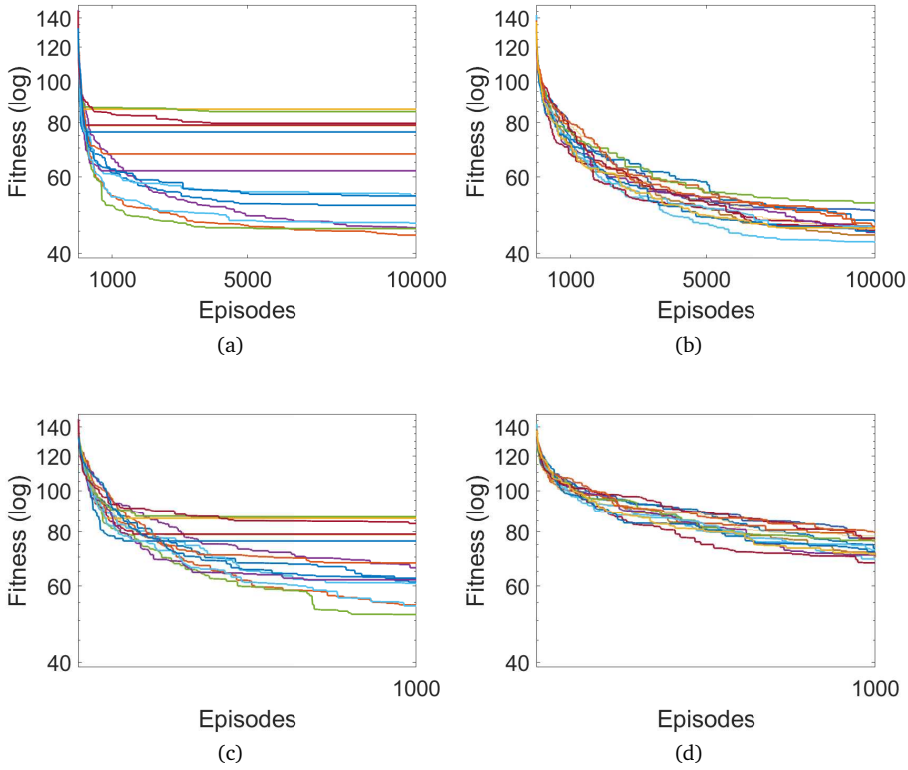



Figure 5.7: (a) The fitness values of the best evolved DSP rules, and (b) the fitness values of the best evolved HC parameters, both independently tested for 40 trials with 10000 episodes. Figures 5.7c and 5.7d are magnified views between 1–1000 episodes of (a) and (b) respectively.

The results show that the training process with 100 episodes does not seem to be sufficient for the HC algorithm to provide results as good as the results provided by the DSP rules. Moreover, it may be possible to improve the success percentage of achieving the goal using the DSP rule by increasing the number of episodes. To test this, we separately tested the DSP rules and the HC with evolved parameters provided by the multiple runs of the GA on the same task settings with 10000 episodes. The results are given in Figure 5.7 where we show the change of the fitness values (average of 40 trials) w.r.t. the episode number during the training of the DSP rules and the HC.

The best DSP rule achieved fitness of 54.27 and 44.10, and the HC with the best parameters achieved fitness of 69.35 and 42.5 in 1000 and 10000 episodes respectively. The results indicate that the DSP rules converge at a better fitness value faster than the HC. However, when the number of episodes is increased (for 10000), the HC achieves slightly (a fitness value difference of 1.6) better performance than the best DSP rule. The $p$-values for the Wilcoxon tests for the results at episodes 1000 and 10000 are 0.02 and 0.3; thus, at 1000 episodes DSP rule is significantly better than HC, whereas at 10000 there is no significant difference between their results.

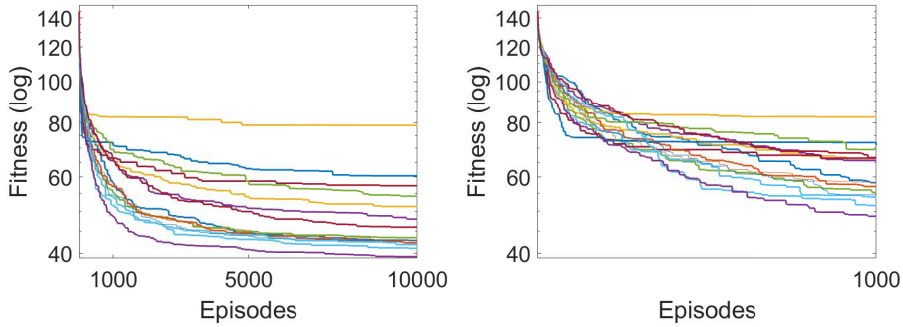



Figure 5.8: The fitness values of the best evolved delayed plasticity rules tested for 40 trials for 10000 episodes, with iterative re-sampling every 100 episodes. Figure 5.8b is a magnified view of the same results between episodes 1–1000.

We observe that some of the DSP rules seem to get stuck at a local optimum after around 100 episodes, which may be due to the fact that they were

optimized for 100 episodes. Thus, to reduce this effect we used an iterative re-sampling approach to randomly reset all the weights of the networks (from the initial domain) at every 100 episodes without resetting the best found fitness value. Thus, in the visualization, the fitness value does not appear worse than the best fitness due to each re-initialization. The training process generated by the iterative re-sampling based DSP rules is provided in Figure 5.8. The best DSP rule achieves fitness values of 48.72 and 39.32 at 1000 and 10000 episodes. A perfect agent (an average of the distances of starting and 8 goal positions found by the A* algorithm) achieves a fitness value of 38.5. For completeness, we also performed additional experiments for the HC using the iterative re-sampling approach. However, we observed in this case that their results were not better than the standard HC with the tested settings.

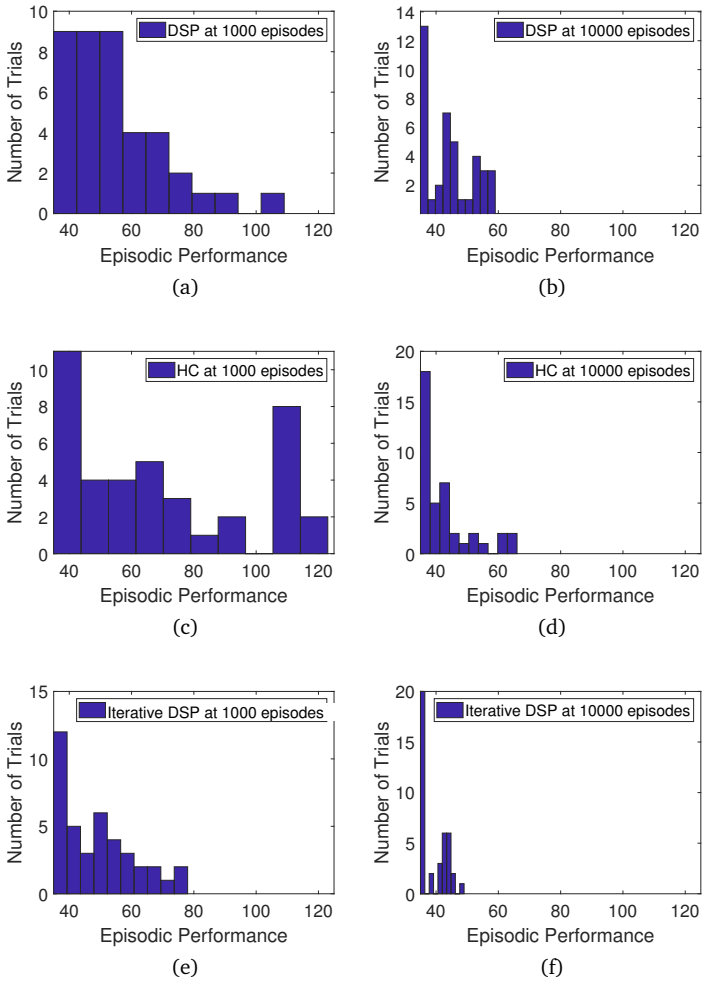



Figure 5.9: The distribution of the episodic performance of 40 trials using the best DSP rule, HC, and DSP rule with iterative re-sampling approach at episodes 1000 and 10000 are given in (a) (b), (c) (d), and (e) (f) respectively.

The distributions of the episodic performance of the best performing DSP rule, HC with best parameters and DSP with iterative approach at 1000 and

10000 episodes are given in Figure 5.9. We fixed the minimum and maximum values of the $x$-axes of all figures to 35–125 for a better visual comparison. At 1000 episodes, all of the agents in 40 trials reach the goal with the smallest number of steps when the DSP rule with iterative re-sampling is used. On the other hand, 39 and 30 of the agents in 40 trials reach the goal when we use the DSP rule and HC respectively. At 10000 episodes, all of the agents in 40 trials reach the goal when the best heuristic from each approach is used. However, in the case of the DSP rule with iterative re-sampling, the agents reach the goal with the smallest number of steps on average.

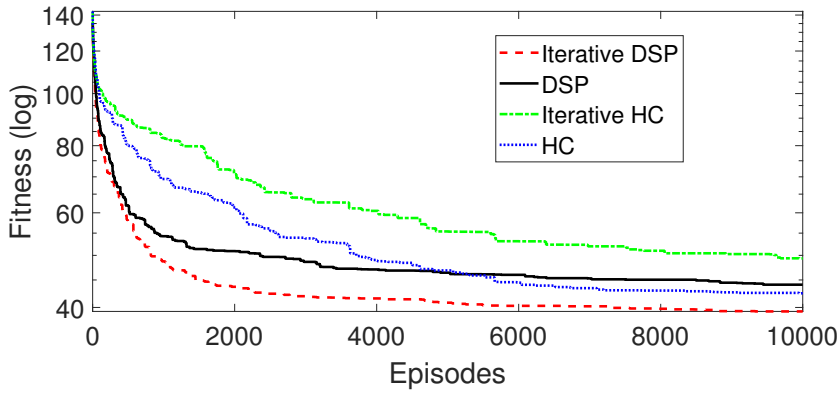



Figure 5.10: The fitness of the best evolved DSP (standard and iterative re-sampling variants) rule and the HC with best evolved parameters (standard and iterative re-sampling variants) tested on 10000 episodes.

Figure 5.10 shows the comparison of the best evolved DSP[2], iterative DSP, HC and iterative HC heuristics with best evolved parameters. Iterative HC performs worse than the HC. On the other hand, the HC was able to outperform the DSP at around 5000 generations, while it could not perform better than the iterative DSP.

[2] We have recorded the performance of the agents on the triple T-maze task after training with the best evolved DSP rule for 100, 1000 and 10000 episodes. A video of the experiment is available online at: `https://youtu.be/JOWYMrAMSdU`.

## 5.6 Conclusions and Future Work

When the reinforcement signals are available after a certain period of time, it may not be possible to associate the activations of the neurons during this period with the reinforcement signals, and perform synaptic updates using Hebbian learning. In this chapter, we proposed the NATs, i.e. additional data storage in each synapse to keep track of neuron activations. We used DSP rules that take into account the NATs and delayed reinforcement signals to perform synaptic updates. We used relative reinforcement signals that were provided after an episode based on the relative performance of the agent in a previous episode. Since the NATs introduce knowledge of neuron activations into the DSP rules, we compared their results to an analogous HC algorithm that performs random synaptic updates without any knowledge of the network activations.

We observed that the DSP rules were highly efficient at training networks with a smaller number of episodes compared to the HC, as they converge quickly at a better fitness than the HC. When they were tested on a larger number of episodes on the other hand, they seemed to be outperformed slightly by HC. We hypothesized that this could be due to the fact that the DSP rules were optimized for a relatively small number of episodes (100). When we tried a iterative re-sampling approach, the DSP rules provided the best results. On the other hand, the DSP rules introduce an additional complexity that requires storing/updating four parameters per synapse during the network computation, and looking-up the update rule based on the activation patterns for each synaptic update.

We aim to apply the DSP on different ANN network models (i.e. continuous neuron activations) with various sizes, and test it on tasks with various complexity. It would also be interesting to investigate the adaptation capabilities of the networks with DSP when the environmental conditions change.

# Chapter 6

# Evolving Plasticity for Producing Novelty

A learning process with the plasticity property often requires reinforcement signals to guide learning based on the reward and punishment signals received in response to the behavior of the agents. However, in some tasks (i.e. maze-navigation), the agents receive reinforcements only when they perform a correct sequence of actions that can solve the task. Thus, they do not receive any reinforcement during the process to indicate how close they are achieving the task. In these cases, there can only be a binary feedback that indicates whether or not the agent solved the task. The binary feedback may provide a reward after a successful behavior, and punishment for the failed behaviors. Usually, the number of successful behaviors would considerably be less than the number of failed behaviors. Since there is no information gained by performing failed behaviors (i.e. all failed behaviors have the same poor fitness value), it becomes challenging to find a successful behavior. Usually exhaustive search is required to find the successful behaviors. We refer to this problem as the "*needle in a haystack*" problem, "needle" is being a correct sequence of actions that can solve the problem, and "haystack" being all the other sequences of actions that fail to solve the problem [36].

In Chapters 4 and 5, we proposed evolving plasticity rules to provide learning in the cases of immediate and delayed reinforcement signals respectively. In

this chapter[1], we aim to model a plasticity based learning process in ANNs for the needle in a haystack problem where we introduce novelty producing synaptic plasticity (NPSP) that aims to evolve plasticity rules to produce as novel behaviors as possible without any reinforcement signals. We make use of the neuron activation traces (NATs) and delayed synaptic plasticity (DSP) framework introduced in Chapter 5 to provide information of the neuron activations and perform synaptic changes at the end of an episode. We evaluate the NPSP on deceptive maze tasks that require complex actions to complete. Deceptive maze problems usually require solving several sub-goals to achieve a final goal state. Thus, it is not straightforward to define a simple fitness value (i.e. Euclidean distance to the goal position in a maze). We propose NPSP because it may allow finding the solution by producing novel behaviors.

The rest of the sections are organized as follows: in Section 6.1, we introduce the needle in a haystack problem; in Section 6.2, we discuss the NPSP framework; in Section 6.3, we provide the details on evolving the NPSP rules; in Sections 6.4 and 6.5, we provide our experimental setup and discuss our results and in Section 6.6, we discuss our conclusions and future work.

## 6.1 The Needle in a Haystack Problem

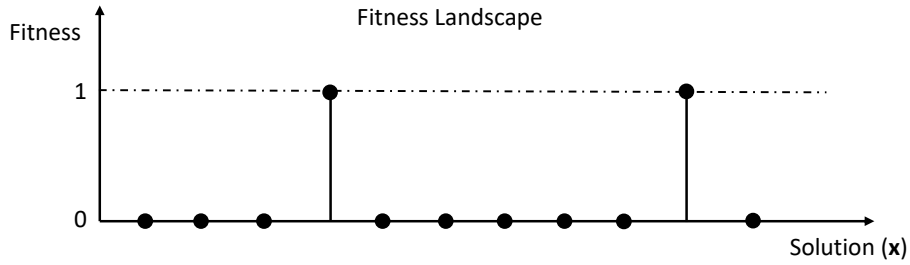



Figure 6.1: An illustration an hypothetical fitness landscape where there is only two possible discrete fitness values (i.e., 1: global optimum, 0: global minimum). Typically, only a small set of solutions associated with the high fitness value that indicate the solution to the problem.

[1]This chapter is integrally based on:
[113] Anil Yaman, Giovanni Iacca, Decebal Mocanu, George Fletcher, and Mykola Pechenizkiy. Novelty producing synaptic plasticity. *(in Preparation)*

Figure 6.1 illustrates an hypothetical case of the fitness landscape with a needle in a haystack problem. The $x$- and $y$-axes show the behaviors (solutions) and their fitness values respectively. The problem regarded as a "black box" which assumes that there is no available metric to measure the efficiency of behaviors in solving the task. Therefore, fitness values "1" and "0" indicate the fitness values of the successful and failed behaviors. Since there is no other fitness value, there is no difference among the behaviors within these two behavior types (successful, failed). There may be more than one behavior that can provide a solution to the problem; and the remaining majority of the behaviors in the behavior space fail to provide solution to the task.

## 6.2 Novelty Producing Plasticity

Figure 6.2c shows a learning process using the NPSP. An agent controlled by a randomly initialized RNN is introduced into an environment to perform a certain task for an episode (period of time). At the end of an episode, only episodic performance $EP = 1$ or $EP = 0$ received from the environment to indicate whether the agent was successful or not in solving the task. While $EP = 0$ and the number of episodes smaller than the maximum number of allowed episodes $N_{episodes}$, the synaptic weights of the network is updated based on the NPSP and NATs, and the agent is reintroduced into the environment to attempt to solve the problem for the next episode. Note that, there is no information about the performance of the agent to determine how close/far the behavior it produced for solving the task. Thus, it is not possible to incorporate this information into the process of learning/optimization of the weights of the RNN.

The learning process of the RNNs using random search and random walk are illustrated in figures 6.2a and 6.2b. These two processes are analogous to the learning process using the NPSP. In the case of random search, at the beginning of each episode, all synaptic weights of the RNNs are randomly sampled. In the case of random walk, after each episode, the synaptic weights of the RNNs are perturbed by a perturbation heuristic (i.e. Gaussian perturbation). However, it is not possible to perform perturbations on a best network that is encountered during the search process similar to what is done in Hill Climbing algorithm since there is no way of distinguishing the networks that do not solve the task because there is no fitness value.

Novelty search and MAP-Elites algorithms were proposed for deceptive tasks where the use of fitness values are often detrimental for the evolutionary process [54, 66]. These algorithms may be beneficial for solving problems with

needle in a haystack problems, however, they require external memory to store encountered solutions, and in the case of MAP-Elites, fitness values to map the solutions to a predefined feature space.

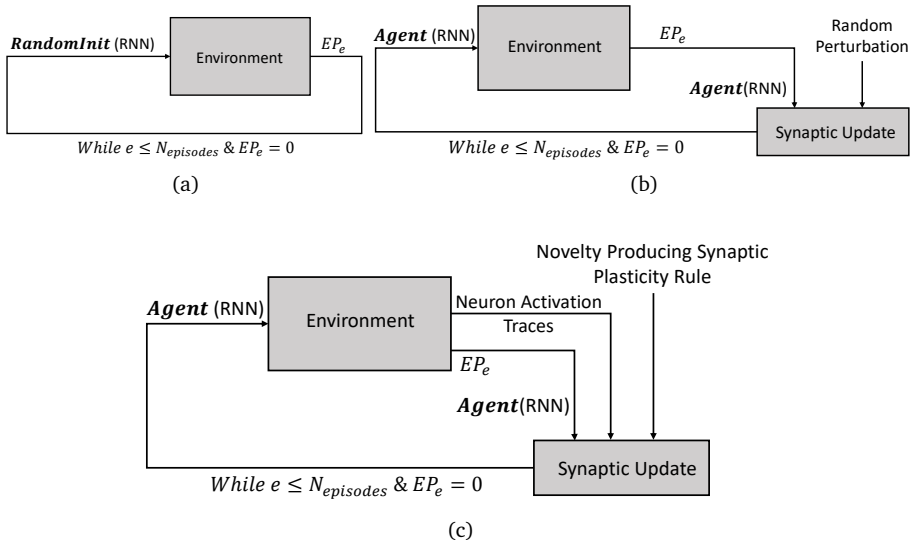



Figure 6.2: The learning process of the RNNs that controls the agents using (a) random search, (b) random walk, (c) novelty producing synaptic plasticity.

## 6.3 Evolving Plasticity for Producing Novelty

We propose evolving synaptic plasticity rules to produce novel behavior. This may especially be beneficial for the cases where there is no information (i.e. fitness values, reinforcement signals) about the problem to guide the learning process. We use the NATs (proposed and discussed in Chapter 5 in detail) to keep track of the pairwise activations of neurons during an episode. We perform synaptic updates $\Delta w_{i,j}$ between neurons $i$ and $j$ based on the NATs after an episode $e$ as follows:

$$\Delta w_{i,j}(e) = NPSP(NAT_{i,j}, \theta) \tag{6.1}$$

$$w'_{i,j}(e) = w_{i,j}(e) + \eta \cdot \Delta w_{i,j}(e) \tag{6.2}$$

where $\theta$ is a threshold that is used to convert NATs into binary vectors, and $\eta$ is a learning rate that scales the magnitude of the synaptic update. The result of an NPSP rule can produce one of three values as: 1, −1 or 0 which can *increase*, *decrease* or keep *stable* the synaptic strength between neurons $i$ and $j$ respectively. We use the normalization step given in Equation (4.7) to scale the synaptic weights and prevent an indefinite increase/decrease.

The NATs are stored as four-dimensional binary vectors for each synapse (see Chapter 5). Each of these four dimension can either be 0 or 1 that is discretized using threshold $\theta$. Therefore, there are $2^4 = 16$ possible NATs shown in Table 6.1.

Table 6.1: The complete list of all possible NATs states are visualized in a tabular form. Depending on a certain threshold $\theta$, NATs are converted to their binary forms. The synaptic changes $x_1, x_2, \ldots, x_{16}$ are performed based on the NATs.

| $NAT^{\theta}$ | | | | $\Delta w$ |
|---|---|---|---|---|
| **00** | **01** | **10** | **11** | |
| 0 | 0 | 0 | 0 | $x_1$ |
| 0 | 0 | 0 | 1 | $x_2$ |
| … | … | … | … | … |
| 1 | 1 | 1 | 1 | $x_{16}$ |

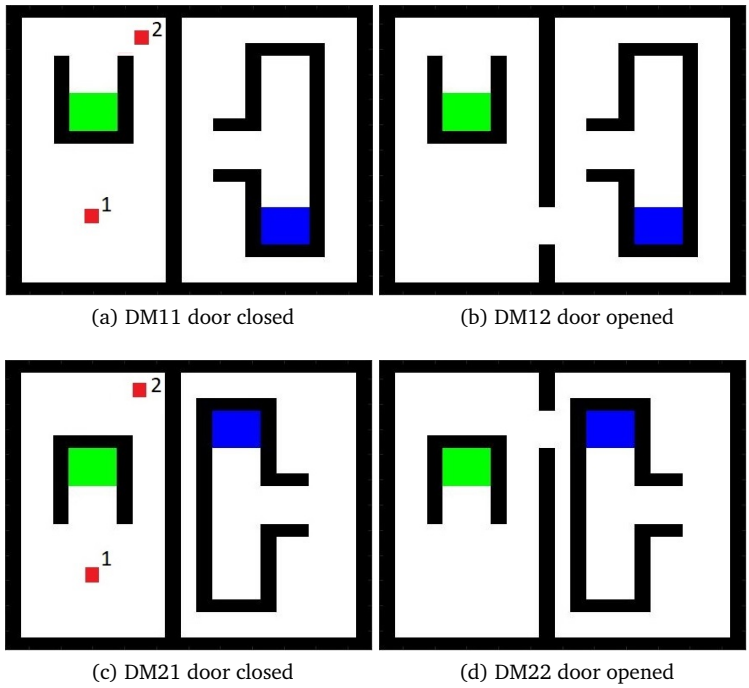
The corresponding synaptic changes $x_1, x_2, \ldots x_16$ to each possible NAT combination is found by a genetic algorithm. Each of these synaptic changes cant take one of three possibe values $\Delta w = \{-1, 0, 1\}$ (*decrease, stable, increase*). Since there are 16 possible states of NATs, there are $3^{16}$ possible NPSP rules.

## 6.4 Experimental Setup

In this section, we provide the details of our experimental setup including the test environment and the details of the GA.

### 6.4.1 Deceptive Maze Environment

We perform experiments on environments we refer as deceptive maze (DM) because it is not straightforward to specify a fitness function to solve these tasks. Moreover, the use of simple fitness functions is usually deceiving to solve the problem by getting stuck in a local optimum and thus preventing finding good solutions [4, 54].




(a) DM11 door closed (b) DM12 door opened

(c) DM21 door closed (d) DM22 door opened

Figure 6.3: An illustration of two deceptive maze environments. Figures 6.3a and 6.3b are two versions of the first environment, and Figures 6.3c and 6.3d are two versions of the second environment. The only difference between the two versions of the same environment is an opening on the middle wall that allows agents to travel from the left room to the right. Labels "1" and "2" show two independent starting positions the agent.

Visual illustrations of DM environments are shown in Figure 6.3. The en-

vironments consist of $23 \times 23$ cells. Each cell can be occupied by one of five possibilities: *empty, wall, goal, button, agent*, color-coded in white, black, blue, green, red respectively. The starting position of the agent is illustrated in red. There are two starting positions of the agent labelled as "1" and "2". These starting positions are tested separately.

Figures 6.3a and 6.3b show two versions of the same environment we refer as DM11 and DM12, and Figures 6.3c and 6.3d show two versions of the same environment we refer as DM21 and DM22. The only difference between these two versions of the environments is that there is an opening (door) in in the middle wall that allows an agent to travel from the left room to the right.

In the environments, there is a button area and goal area illustrated in green and blue. Starting from one of the starting positions, the agent is required to first go to the button area and perform a "press the button" action. In this case, the door in the middle of the wall opens. The agent is then required to pass through this opening and reach the goal position. We employ RNNs for controlling the agents to generate sequence of actions to allow agents to reach to the goal from the starting positions. To achieve that, it is required to find the connection weights of the RNNs that can generate correct sequences of actions.

### Agent Architecture

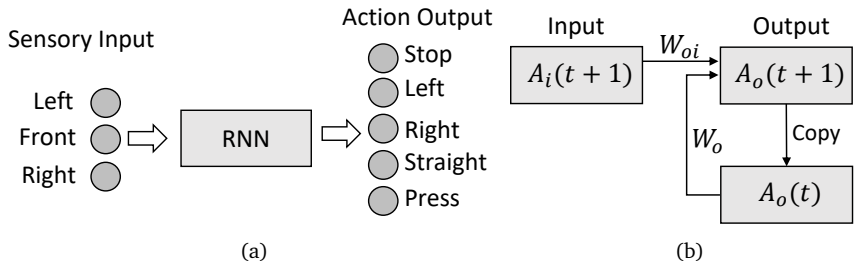



Figure 6.4: (a) The sensory inputs and action outputs of the recurrent neural networks without a hidden layer, and (b) the architecture of the networks that are used to control the agents.

An illustration of the architecture of the agents used for the deceptive maze tasks is given in Figure 6.4. We used similar simulation settings that were dis-

cussed in Section 5.4 in detail. Only an additional action "Press" is added to press the button to open the door.

We use RNNs to control the agents. To limit the computational requirements, we did not use a hidden layer. Thus, the RNNs consist of only input and output layers as shown in Figure 6.4b. The networks consist of $(3+1)\times 5 = 20$ input to output layer connections (+1 refers to the bias), $4\times 5 = 20$ output layer to output layer connections (except self), in total of 40 connections between all layers.

### Genetic Algorithm

A standard GA was used to evolve the discrete parts of the NPSP rules. The NPSP rules consist of discrete and continuous parts. The discrete parts of the genotypes consists of 16 genes, and were initialized randomly from $\{-1,0,1\}$. The continuous parts of the genotypes were initialized randomly from the ranges of $(\eta \in [0,1], \theta \in [0,1], \alpha_o \in [0,1])$. Thus, the genotype of the individuals represented as 19-dimensional discrete/real-valued vectors. We use custom-designed mutation operator (Gaussian mutation) for the continuous parts of the NPSP rules.

Algorithm 7 shows the evaluation of an individual NPSP rule. We evaluate each NPSP rule on two environments, illustrated in Figure 6.3, for two starting positions for three trials each. Thus, in total, we perform 12 independent trials of agent simulations in an environment. Each of these trials consists of 500 episodes of learning process, within each of these environment simulations in each episode, the agent is allowed to perform 250 action steps to reach to the goal.

$$fitness = \frac{\left|unique(\boldsymbol{B})\right|}{N_{trials}} \tag{6.3}$$

The fitness value of an individual NPSP rule is found by Equation (6.3) which is based on the average number of novel behavior the NPSP rule produces. To calculate that, we abstract and record the behavior of an agent during each episode and append to the behavior set $\boldsymbol{B}$, and find the average of the number of novel (unique) behavior per trial.

The environment was divided $3\times 3$ squares as shown in Figure 6.5, and each square is given two unique identifiers (ids) (i.e. "1" and "1*") to distinguish between two states of the agent: "located in the square" and "located in the square and pressed button". Inspired by Pugh *et al.,* [76], we abstract the behavior of an agent by recording its trajectory based on the locations visited, and save as a sequence of ids in a string form. We refer each of these strings as a behavior.

We collect the behavior in each episode and find how many novel behaviors the NPSP rule is able to produce during one trial (500 learning episodes). We aim to maximize number of novel behavior produced by the NPSP to be able to find the behavior that solves the task.

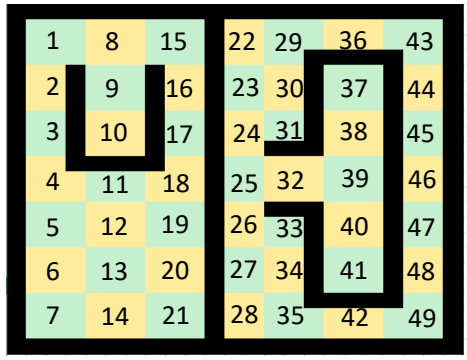
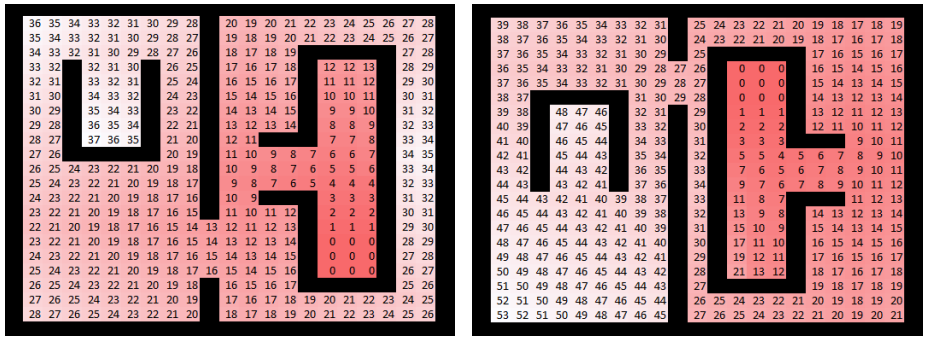



Figure 6.5: An example illustration of the environment representation that is used to abstract the behavior of the agents.



(a) (b)

Figure 6.6: The distances of each cells to the goal position in each environment are shown as heatmaps where the intensity of red color indicates lower distance.

The distances of each cell to the goal position in the environments are

measured as shown in Figure 6.6. During each episode, the closest distance $min(d_e(XY(Agent), XY(g)))$ to the goal position $XY(g)$ is also recorded.

The comparison is performed based on two performance measures: "novelty" and "distance". Novelty measure is the average of unique number of behaviors produced during an evaluation process and is also used as the fitness value for the selection process in the GA. Distance measure is the average of the smallest distances to the goals that an agent achieved during the evaluation. Both of these measures were scaled within a certain range to make it easy to perform comparison between the results of the different runs of different starting points of different environments. Thus, the novelty measurement was divided by the number of episodes to scale it between 0 and 1. So, the agent has a novelty score of 1 if it can produce a novel behavior in each episode. The distance measure is adjusted depending on whether the agent manages to pass through the door to the second room where the goal is located. If the closest distance of the agents to the goal is greater than the distance of the door to the goal (the agent was not able to go pass to the second room), its distance measure updated as:

$$dist_{agent} = 1 + \frac{min(d_e(XY(Agent), XY(g)))}{maxDist} \tag{6.4}$$

Otherwise (if the agent manages to go the second room where the goal is located), then its distance measure is updated as:

$$dist_{agent} = \frac{min(d_e(XY(Agent), XY(g)))}{maxDistSecondRoom} \tag{6.5}$$

where constants $maxDist$ and $maxDistSecondRoom$ are the maximum distance to the goal, and maximum distance to the goal in the second room. Thus, the updated distance measure is between 0 and 2. If it is greater than 1, it means that the agent was not able to pass to the second room; and if it is smaller than 1, it means that the agent managed to pass to the second room; and its magnitute indicate its distance to the goal position, smaller is the closer.

We use a population size of 14 and employ a *roulette wheel selection* operator with an elite number of four. We use *1-point crossover* operator with a probability of 0.5 and designed a custom *mutation* operator which re-samples each discrete dimension of the genotype with a probability of 0.15, and performs a Gaussian perturbation with zero mean and 0.1 standard deviation for the continuous parameters. We run the evolutionary process for 100 generations. In each generation during an evolutionary process, we store the NPSP

rules that produced the largest number of novel behavior, and the NPSP rules that achieved the minimum distance to the goal positions.

### 6.4.2 Benchmark Algorithms

We use two analogous algorithms, Random Search (RS) and Random Walk (RW), for comparison to the NPSP rules. The RS and RW algorithms use single solution to perform synaptic changes after every episode. However, they perform synaptic changes by random initialization and perturbation respectively, thus, do not use any domain knowledge on the activation of neurons as it is used in the NPSP rules.

The RS algorithm randomly initializes all the connection weights of the RNNs after each episode. Thus, it randomly searches the search space. In the case of the RW, the connection weights of the network is randomly perturbed by the Gaussian mutation as given in Equation (6.6). Thus, the RW aims to discover the neighboring points of an initial point in the search space.

$$\boldsymbol{Agent} = \boldsymbol{Agent} + \sigma \cdot \mathcal{N}(0,1) \tag{6.6}$$

## 6.5 Experimental Results

In this section, we present the results of the RS, RW and NPSP rules. The comparisons between the results of the algorithms are performed based on the updated novelty and distance measures that are explained in Section 6.4.

Table 6.2 shows the average results of the novelty and distance measures of the agents trained by RS, RW and evolved NPSP rules. During the evolutionary process of NPSP rules, the learning process is set to 500 episodes and for 12 trials in total (3 trials of 2 starting positions for 2 environments). The column labelled as NPSP_train presents the results of the best evolved NPSP rule during the evolutionary process whereas the column labelled as NPSP_test shows the results of the best evolved rule tested separately for 40 trials. We tested the RS and RW using the similar settings and presented in columns labelled as RS1 and RW1. Additionally, we tested RS and RW on 5000 episodes and presented the results under the columns labelled as RS2 and RW2. The rows labelled as "Second Room" and "Goal" indicate the number of times (out of total number of trials) the agent was able to go the the second room and reached to the goal respectively.

We observe that the RS produces more novel behavior relative to the RW. That could be expected since the RS randomly samples from the search space after each episode, whereas, the RW performs iterative perturbations on randomly initialized solutions, thus, it performs the search more locally. Consequently, RS leads to lower distance measure which indicates that producing as many novel behavior as possible is beneficial for finding a successful behavior.

| **Algorithm:** | **RS1** | **RS2** | **RW1** | **RW2** | **NPSP_train** | **NPSP_test** |
|---|---|---|---|---|---|---|
| **Episodes** | 500 | 5000 | 500 | 5000 | 500 | 500 |
| **Novelty** | 0.095 | 0.0344 | 0.018 | 0.058 | 0.3065 | 0.2973 |
| **Distance** | 1.3974 | 1.2635 | 1.5032 | 1.4219 | 0.9303 | 0.8701 |
| **Goal** | 0 | 0 | 0 | 0 | 1 | 3 |
| **Second Room** | 0 | 3 | 0 | 1 | 7 | 27 |
| **Total trial** | 12 | 12 | 12 | 12 | 12 | 40 |

Table 6.2: The average results of the novelty and distance measures of agents trained by RS, RW and evolved best performing NPSP rules.

When the number of episodes is increased to 5000, the RS2 and RW2 algorithms performs relatively better than the RS1 and RW1. The RS2 produced on average 170 novel behavior ($0.0344 \times 5000$) whereas the RS1 produced on average 47 novel behavior ($0.095 \times 500$). Although, the number of episodes increased 10 times, the novel behavior produced by RW2 is increased about 3.5 times. As a result, the agent trained by RS2 was able to access the second room three times. The RW2 produced on average 29 novel behavior ($0.0058 \times 5000$) whereas the RW1 produced on average 9 novel behavior ($0.018 \times 500$).

The results of the NPSP show that the best performing evolved NPSP rule is capable of producing about 150 ($0.3065 \times 500$ on train, $0.2973 \times 500$ on test) novel behavior on average. Consequently, the agents that are trained with the NPSP rule are capable of entering to the second room 67.5% of the test trials. One of the interesting observation is that the number of novel behavior produced by RS2 is slightly more than the novel behavior produced by the NPSP rule, however, the agents trained with the NPSP were able to explore the environment more. This indicates that the exploratory quality of the behaviors produced by the NPSP may be higher than the RS2.

During the evolutionary processes of the NPSP rules, we further stored the NPSP rule that achieved the smallest distance to the goals on average. This rule was able to achieve a novelty score: 0.1988 and a distance score: 0.5123, found the goal 4 times and entered to the second room 10 times out of 12 trials.

However, when this rule was tested separately for 40 trials, it achieved a novelty score: 0.1547 and a distance score: 1.1092, found the goal 1 time, and entered to the second room 17 times. These results show that selecting rules for novelty rather than closeness to the goal produces better performance on the test data which indicates that this approach is more generalizable.

Figures 6.7a and 6.7b provide the novelty and distance trends during 5 independent evolutionary runs respectively. Since the NPSP rules were selected based on their novelty, their distance trends are not decreasing all the time. Thus, some rules showed better distance with lower novelty score.

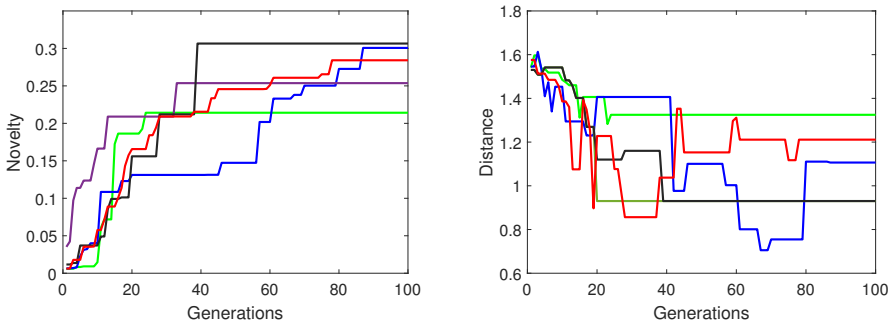
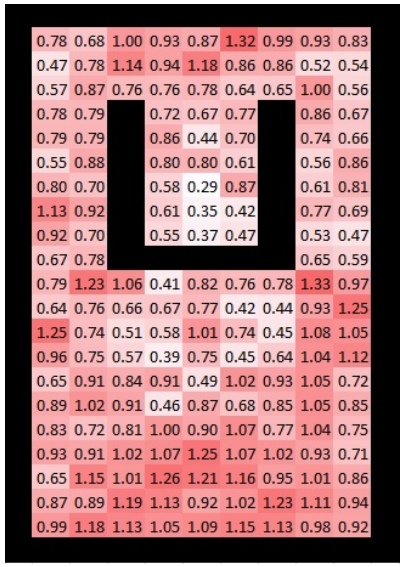
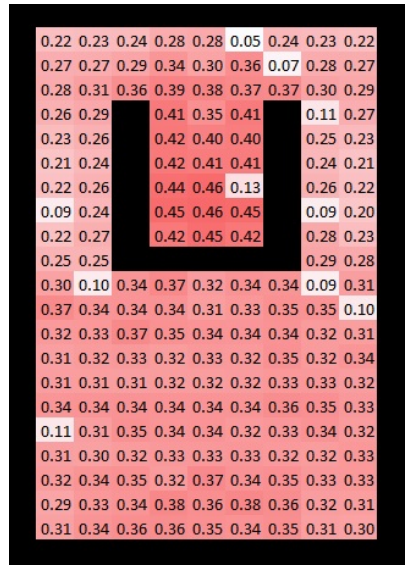



Figure 6.7: The novelty and distance trends of the NPSP rules given respectively in Figures 6.7a and 6.7b during 5 independent evolutionary runs.

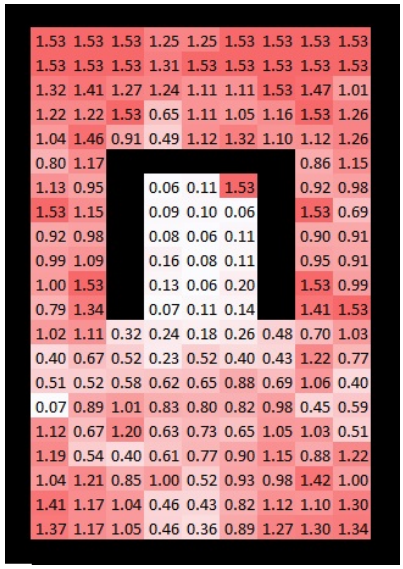
We assigned each cell in the first room of DM1 and DM2 as the starting point and used the best performing evolved NPSP rule to train agents for 10 independent trials (each starting from a randomly initialized RNN configuration). Note that, the NPSP rules were evolved based on two selected starting points. Here, we aim to assess the generalizability of the NPSP rule by measuring the distance and novelty measures of agents when they start any cell in DM1 and DM2. We show the results in Figure 6.8 where we assign the average of the distance and novelty measures to each cell when it is used as the starting point. We colour-coded the figures based on the magnitude of the values in each cell where the intensity of red indicate higher values.



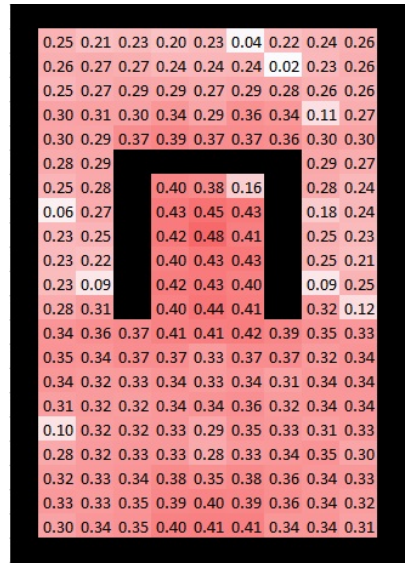

(a) DM1 Distance Measure




(b) DM1 Novelty Measure



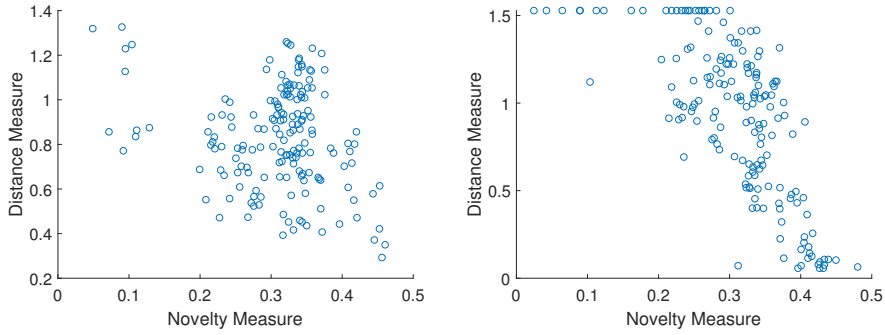

(c) DM2 Distance Measure




(d) DM2 Novelty Measure

Figure 6.8: The average distance and novelty measures of 10 independent trials of the agents trained by the evolved NPSP rule. The value in each cell indicate the result when it is set as the starting point. Only the first rooms of the environments are shown since the agents can only start from there. The intensity of the colour indicate the magnitude of the value in each cell.

Based on the distance measures shown in Figures 6.8a and 6.8c, we observe that the agents starting close the wall and behind the obstacle do not manage to get closer to the goal position. Correspondingly, Figures 6.8b and 6.8d show lower novelty measures in similar areas. Whereas, the agents that start from the middle area, and the area that face or within the button area, are capable of getting close to the goal. Moreover, they show higher novelty measure. For DM1, the average novelty measure of the cells that have the distance measure smaller than 0.5 is 0.3638, between 0.5 and 1 is 0.3034, and greater than 1 is 0.3057. For DM2, the average novelty measure of the cells that have the distance measure smaller than 0.5 is 0.3977, between 0.5 and 1 is 0.3141, between 1 and 1.5 is 0.3033, and greater than 1.5 is 0.1937. Therefore, we observe that, more clearly in DM2, there is inverse relation between novelty and distance measures. We show the pairwise distance and novelty measures of each cells in DM1 and DM2 in Figures 6.9a and 6.9b respectively.




(a) DM1 (b) DM2

Figure 6.9: The pairwise distance and novelty measures of each cells in DM1 and DM2.

Overall, we observe that agents started from 128 cells out of 172 (74.5%), and 87 out of 172 (50.5%) were able to access the second room in DM1 and DM2 respectively. This shows that the agents in DM1 were more successful in getting closer to the goal. This may indicate that the second environment is more difficult to solve, or environmental bias in the NPSP rule that may help capturing some features in DM1.

## 6.6 Conclusions and Future Work

In Chapters 4 and 5, we proposed evolutionary approaches to introduce plasticity in ANNs to allow learning in the cases with immediate and delayed reinforcement signals. In this chapter, we proposed an evolutionary based approach to produce synaptic plasticity to allow learning in ANNs for the cases with no reinforcement signals. We refer to these problems as the "needle in a haystack" where it is required to find a solution in a large search space without any information to distinguish between solutions (i.e. fitness value or reinforcement signal) to guide the learning process. We proposed novelty producing synaptic plasticity which aims to produce as many novel behavior as possible to find the behavior that can solve the problem. We used neuron activation traces to inform the NPSP rules to make synaptic changes. We compared the NPSP with random search and random walk algorithms that are analogous to the NPSP except they perform synaptic changes randomly. These algorithms use only a single solution. Our results show that the information about the pairwise activations of neurons introduced with the NATs, helps increase the number of novel behaviors relative to random search and perturbations.

We observed positive relation between novelty and finding a solution. We aimed also to assess the generalizability of the NPSP rule by testing for all possible starting positions in test environments. In some starting cells, the NPSP was not able to produce novel behaviors as well as others and thus solving the problem from those cells appeared to be difficult.

We used relatively small number of network size, population size and function evaluations during the evolutionary process to limit the computational requirements and runtime of the algorithms. Therefore, for the future work, we aim to perform experiments with large parameter settings for the possibility of finding better NPSP rules. We further aim to test the NPSP rules on larger networks.

# Chapter 7

# Conclusions and Future Work

Through the evolutionary process, biological systems and organisms have come to exhibit intelligent behavior to help them solve the problems they encounter in their environment, and ultimately survive. The intelligent behavior they exhibit, and the processes that brought about these behaviors have been one of the main sources of inspirations in generating intelligence in artificial systems.

Artificial neural networks are one of the biologically inspired computational models that aim to model the information processing behavior of BNNs. Using optimization approaches, the configuration of ANNs (i.e. topology, connections and their weights) are trained/optimized to map given inputs to desired outputs to solve certain tasks. The optimization process usually consists of training and validation phases to find the configuration that performs the best.

Inspired by the biological evolution of the BNNs, the field neuroevolution employs evolutionary computing approaches to optimize ANNs. However, while the dimensions of the problems increases, neuroevolution gets effected by two bottlenecks when the direct encoding approach is used. These bottlenecks concern the scalability of the evolutionary algorithms in performing efficient search in higher dimensional search spaces, and reducing the high computational cost required during the evaluation phase due to the sample or simulation size.

The parameters of the ANNs optimized by neuroevolution with direct encoding approaches remain fixed during their lifetime (unless the optimization process resumed or restarted). On the other hand, one of the fundamental aspect of learning in BNNs is their plasticity property that allows them to change their configuration during their lifetime. Although current physiological un-

derstanding indicates that these changes are performed in individual synapses based on the local interactions of the neurons, the emergence of a coherent global learning behavior of the whole network is not very well understood.

In this thesis, we presented our main contributions in direct and indirect neuroevolution approaches. In the following three sections, we discuss our conclusions, limitations and future work.

## 7.1 Conclusions

It is not straightforward to apply evolutionary optimization approaches to neuroevolution for optimizing ANNs with large number of parameters using direct encoding. In Chapter 3, we proposed limited evaluation and cooperative co-evolution schemes to improve the accuracy and runtime of standard differential evolution algorithm. The limited evaluation (LE) scheme allows the evaluation to be performed on a subset of samples similar to what is done in batch training; and, the cooperative co-evolution (CC) scheme allows to decompose the problem into smaller subproblems, and evolve them separately in different subpopulations. Here, the problem decomposition approach plays an important role in reducing the dependencies between the variables in different subpopulations. Thus, we treated each neuron as a building block of an ANN and optimized their parameters (incoming synaptic weights) separately in different subpopulations. Given the same number of function evaluations, our results showed that the LE scheme reduced the runtime, and the CC increased the accuracy of the results of the algorithm.

In Chapters 4, 5 and 6, we proposed an evolutionary approach for producing plasticity property in ANNs. Using this approach, we evolved discrete local learning rules to perform synaptic changes in each synapse based on the local interaction of neurons. This form of learning can exhibit distributed and self-organized properties, and the discrete learning rules can provide an interpretable alternative to other approaches proposed in the literature for synaptic plasticity. Moreover, discrete learning rules are able to specify an individual synaptic update rule for all possible pairwise (binary) interaction state of the neurons that may not be possible using continuous update functions.

We demonstrated the performance of the plasticity rules evolved with the proposed approach on three kinds of reinforcement learning tasks. In Chapter 4, we studied a case where the reinforcement signals are available immediately after each action of the agent. The immediate reinforcement signals are used to perform continuous lifetime learning. We evaluated the learning and adapta-

tion capabilities of feed forward ANNs trained by the evolved plasticity rules on a foraging task where an agent is required to learn to navigate within an enclosed environment and collect/avoid nutritious/poisonous types of food items starting from a random network configuration. To assess the adaptation capabilities of the agents, we periodically changed the environmental conditions by switching the reward/punishment values for the nutritious and poisonous food items. We observed that the agents that are trained by the evolved plasticity rules are able to adapt to the task when the reinforcement function changes. We obtained various evolved plasticity rules that show differences in their adaptation capabilities. Unlike classical Hebbian plasticity rules, the majority of the best performing plasticity rules carried out synaptic changes when the agent performed undesired behaviors. This would make sense because if the synaptic changes are performed continuously after each desired behavior, these changes may disrupt the configuration of the network causing the network to "forget" what was learned.

In Chapter 5, we studied the case where the reinforcement signals are not available after each action of the network, but received after a certain period of time. This case can often be encountered in tasks that require certain sequence of actions to achieve a certain goal. However, in this case, it becomes challenging to associate the activations of the neurons to delayed reinforcement signals. We proposed an additional data structure we referred as neuron activation traces to keep track of the pairwise activations of neurons within a period of time. We evolved delayed plasticity rules which are extensions of the discrete plasticity rules introduced in Chapter 4 to perform synaptic changes based on the NATs and a delayed reinforcement signal. We used a recurrent neural network model to control the agents and evaluated delayed plasticity rules on a triple T-maze navigation task where an agent is required to learn to navigate from a starting position to a goal position. We compared the proposed approach with the Hill Climbing algorithm that is analogous to DSP rules except it does not use the domain knowledge introduced with the NATs. The results showed that the heuristic introduced with the NATs significantly improves the learning process relative to the HC algorithm in smaller number of episodes.

In Chapter 6, we introduced novelty producing synaptic plasticity for the tasks where the reward signal is received after a certain period of time and only when the agent solves the task. In this case, the reward function provides the same feedback for the unsuccessful attempts of the agent in solving the task, and only provides positive feedback when the task is solved. Thus, there would not be any way of distinguishing the behavior of the agent in terms of their closeness to solving the problem. We referred to this problem as the "needle

in a haystack" problem and proposed evolving plasticity rules, we referred as the novelty producing synaptic plasticity rules, that can perform changes at the end of each episode using the NATs without making use of any reinforcement signal. The NPSP rules were evaluated based on how many novel behavior they can produce within a given number of episodes. Thus, this approach aims to find the behavior that can solve the task by producing as many novel behavior as possible. We observed that the NPSPs using the heuristic introduced with the NATs were capable of producing novel behaviors and thus finding the solutions to the problem in the absence of the reinforcement signals.

## 7.2 Limitations

The heuristic proposed in Chapter 3, decomposes the parameters of a large ANN based on the post-synaptic neurons where all the incoming synaptic weights of each post-synaptic neuron is assigned to a subpopulation. This heuristic aims to reduce the number of parameters per subpopulation. However, the number of parameters of a neuron may still be large if they have large number of incoming connections. Moreover, the number of subpopulations increases linearly depending on the number of neurons. The algorithm was designed to iterate over all of these subpopulations sequentially starting from the fist subpopulation to the last.

The indirect approach aims to evolve the learning algorithm instead of the parameters of the network. Our results indicated that the learning process was not affected by the size and initial configuration of the networks. However, it would be interesting to evaluate the learning process in tasks that require larger networks.

The NATs introduced in Chapter 5 introduces an additional level of complexity to store the pairwise activation states of the neurons. On the other hand, it allows training networks in shorter number of episodes relative to the HC algorithm. However, we observed that the networks trained by the HC algorithm showed a performance slightly better than the networks trained with the DSP rules. We reasoned that may be due to the fact that the DSP rules evolved on smaller number of episodes to reduce the computational complexity during the evaluation process. When they are tested on larger number of episodes, some of the rules did not show any improvement after about the same number of episodes they were evolved. Using the iterative DSP approach, it was possible to prevent getting stuck at a local optimum and perform better than the HC algorithm, and produce very close performance to the perfect results.

In Chapter 6, we evaluated the novelty producing plasticity for solving deceptive maze tasks. To limit the computational resources, we used small-scale parameter settings. Although two selected starting positions were used to evolve the NPSP rules, all possible starting positions in the environment used for the test. Although most of these starting positions showed a good performance, it was not equally possible to find solution for each of them.

## 7.3 Future Work

As a long-term future research direction, we believe that understanding learning, memory and problem-solving capabilities of Biological Neural Networks (BNNs) to mimic their behavior in artificial systems is crucial in developing control/decision-making systems that exhibit autonomous and continuous learning capabilities.

Neuroevolution is one of the powerful biologically inspired approaches to design ANNs that mimic the information processing behavior of BNNs. However, it is not straightforward to scale neuroevolution to optimize large networks. We believe that the use of additional heuristics can help scale evolutionary approaches. For instance, in CC scheme proposed in Chapter 3, it may be possible to use other problem decomposition heuristics to decompose ANNs with large number of parameters. Defining these decomposition heuristics based on expert knowledge may also be challenging. Therefore, automatic approaches that aim to identify the relations between variables/subcomponents of the problem can be used to decompose the problem. Moreover, instead of using random perturbations, the knowledge about the ANNs (such as the NATs introduced in Chapter 5) can be used to perform better parameter adjustments to increase the success of producing ANNs with better fitness values. Furthermore, it is possible to reduce the runtime of the algorithm by parallelizing the evolution of subpopulations.

The plasticity property allows ANNs to learn during their lifetime. However, lifetime learning may cause “catastrophic forgetting” that occurs when an ANN forgets learned knowledge while leaning a new one. An interesting direction would be to study approaches to prevent this effect. In the cases of delayed and novelty producing synaptic plasticity, it would be interesting to study the learning process of the networks to investigate how behavior changes between episodes. It is possible to further extend the environments/tasks that are used to evolve and test plasticity the synaptic plasticity rules in immediate, delayed and novelty producing plasticity cases to assess and improve their generalizability. In

some cases, we limited the evolutionary process to small number of generations to limit the runtime of the algorithms. It would be interesting to increase that limitation to be able to find possible rules that may perform better. It would also be interesting to extend the proposed approaches to work on larger networks and also on more biologically inspired artificial neural network modals such as Spiking Neural Networks.

Another research direction would be to test proposed algorithms on more complex agent-based tasks that may also include multiple agents to increase task complexity and study learning in social settings. It would also be interesting to apply the proposed approaches to robotics to demonstrate autonomous lifetime learning in real-life.

# Bibliography


[1] Ralph G Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and Christian E Elger. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. *Physical Review E*, 64(6):061907, 2001. (Cited on page 31.)

[2] Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1):54–65, 1994. (Cited on page 21.)

[3] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *ESANN*, 2013. (Cited on page 31.)

[4] Joshua E Auerbach, Giovanni Iacca, and Dario Floreano. Gaining insight into quality diversity. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 1061–1064. ACM, 2016. (Cited on page 92.)

[5] Richard K Belew, John McInerney, and Nicol N Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In *In*. Citeseer, 1990. (Cited on pages 20 and 21.)

[6] Elie L Bienenstock, Leon N Cooper, and Paul W Munro. Theory for the development of neuron selectivity: orientation specificity and binocular

interaction in visual cortex. *Journal of Neuroscience*, 2(1):32–48, 1982. (Cited on page 44.)

[7] Janez Brest, Sao Greiner, Borko Boskovic, Marjan Mernik, and Viljem Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE transactions on evolutionary computation*, 10(6):646–657, 2006. (Cited on page 129.)

[8] Thomas H Brown, Edward W Kairiss, and Claude L Keenan. Hebbian synapses: biophysical mechanisms and algorithms. *Annual review of neuroscience*, 13(1):475–511, 1990. (Cited on pages 42 and 44.)

[9] Fabio Caraffini, Giovanni Iacca, and Anil Yaman. Improving (1+ 1) covariance matrix adaptation evolution strategy: a simple yet efficient approach. In *LeGO 2018 - International Workshop on Global Optimization, 18-21 September, Leiden, The Netherlands*, 2018. (Cited on page 154.)

[10] Thomas P Caudell and Charles P Dolan. Parametric connectivity: Training of constrained networks using genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 370–374. Morgan Kaufmann Publishers Inc., 1989. (Cited on page 20.)

[11] Onur Çaylak, Anil Yaman, and Björn Baumeier. Evolutionary approach to constructing a deep feedforward neural network for prediction of electronic coupling elements in molecular materials. *Journal of Chemical Theory and Computation*, 15(3):1777–1784, 2019. PMID: 30753071. (Cited on pages 11, 22, and 153.)

[12] Praveen Chandar, Anil Yaman, Julia Hoxha, Zhe He, and Chunhua Weng. Similarity-based recommendation of new concepts to a terminology. In *AMIA Annual Symposium Proceedings*, volume 2015, page 386. American Medical Informatics Association, 2015. (Cited on page 155.)

[13] Darwin Charles. *On the origin of species by means of natural selection*. Murray, London, 1859. (Cited on page 11.)

[14] Oliver J Coleman and Alan D Blair. Evolving plastic neural networks for online learning: review and future directions. In *Australasian Joint Conference on Artificial Intelligence*, pages 326–337. Springer, 2012. (Cited on pages 23, 41, and 45.)

[15] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: a survey. *ACM Computing Surveys (CSUR)*, 45(3):35, 2013. (Cited on page 13.)

[16] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, 45(3):35, 2013. (Cited on page 17.)

[17] Swagatam Das, Sankha Subhra Mullick, and Ponnuthurai N Suganthan. Recent advances in differential evolution–an updated survey. *Swarm and Evolutionary Computation*, 27:1–30, 2016. (Cited on pages 15, 17, 39, and 128.)

[18] Leandro N De Castro and Fernando J Von Zuben. Learning and optimization using the clonal selection principle. *IEEE transactions on evolutionary computation*, 6(3):239–251, 2002. (Cited on page 134.)

[19] Leandro Nunes De Castro. *Fundamentals of natural computing: basic concepts, algorithms, and applications*. CRC Press, 2006. (Cited on pages 1, 17, 53, 71, 78, and 134.)

[20] Kenneth Alan De Jong. *Analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, 1975. (Cited on page 13.)

[21] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002. (Cited on page 132.)

[22] Peter Dürr, Claudio Mattiussi, Andrea Soltoggio, and Dario Floreano. Evolvability of neuromodulated learning for robots. In *Learning and Adaptive Behaviors for Robotic Systems, 2008. LAB-RS'08. ECSIS Symposium on*, pages 41–46, New York, 2008. IEEE. (Cited on page 69.)

[23] Ágoston E Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation*, 3(2):124–141, 1999. (Cited on page 13.)

[24] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003. (Cited on pages 11 and 14.)

[25] Sami El-Boustani, Jacque PK Ip, Vincent Breton-Provencher, Graham W Knott, Hiroyuki Okuno, Haruhiko Bito, and Mriganka Sur. Locally coordinated synaptic plasticity of visual cortex neurons in vivo. *Science*, 360(6395):1349–1354, 2018. (Cited on pages 3 and 46.)

[26] Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008. (Cited on pages 2, 19, 20, and 41.)

[27] Dario Floreano and Joseba Urzelai. Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, 13(4-5):431–443, 2000. (Cited on pages 8, 23, 42, 44, and 45.)

[28] Wulfram Gerstner, Marco Lehmann, Vasiliki Liakoni, Dane Corneil, and Johanni Brea. Eligibility traces and plasticity on behavioral time scales: Experimental support of neohebbian three-factor learning rules. *Frontiers in Neural Circuits*, 12:53, 2018. (Cited on pages 8, 70, and 71.)

[29] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. (Cited on pages 11, 14, 20, and 46.)

[30] Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9(May):937–965, 2008. (Cited on pages 8, 27, and 29.)

[31] Faustino John Gomez. *Robust non-linear control through neuroevolution*. PhD thesis, University of Texas at Austin USA, 2003. (Cited on pages 8 and 27.)

[32] Frédéric Gruau. Automatic definition of modular neural networks. *Adaptive behavior*, 3(2):151–183, 1994. (Cited on page 23.)

[33] Ahmed Hallawa, Anil Yaman, Giovanni Iacca, and Gerd Ascheid. A framework for knowledge integrated evolutionary algorithms. In Giovanni Squillero and Kevin Sim, editors, *Applications of Evolutionary Computation*, pages 653–669, Cham, 2017. Springer International Publishing. (Cited on pages 5, 11, 13, and 154.)

[34] Simon S Haykin, Simon S Haykin, Simon S Haykin, Kanada Elektroingenieur, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson education Upper Saddle River, 2009. (Cited on page 1.)

[35] Donald Olding Hebb. The organization of behavior: A neuropsychological theory, 1949. (Cited on pages 3 and 42.)

[36] Geoffrey E Hinton and Steven J Nowlan. How learning can guide evolution. *Adaptive individuals in evolving populations: models and algorithms*, 26:447–454, 1996. (Cited on pages 4 and 87.)

[37] Gregor M. Hoerzer, Robert Legenstein, and Wolfgang Maass. Emergence of Complex Computational Structures From Chaotic Neural Networks Through Reward-Modulated Hebbian Learning. *Cerebral Cortex*, 24(3):677–690, 2014. (Cited on page 44.)

[38] John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992. (Cited on pages 13 and 14.)

[39] G. Iacca, R. Mallipeddi, E. Mininno, F. Neri, and P. N. Suganthan. Superfit and population size reduction in compact differential evolution. In *2011 IEEE Workshop on Memetic Computing (MC)*, pages 1–8, April 2011. (Cited on page 128.)

[40] Giovanni Iacca, Fabio Caraffini, and Ferrante Neri. Multi-strategy coevolving aging particle optimization. *International journal of neural systems*, 24(01):1450008, 2014. (Cited on pages 16 and 138.)

[41] Giovanni Iacca, Ferrante Neri, Fabio Caraffini, and Ponnuthurai Nagaratnam Suganthan. *A Differential Evolution Framework with Ensemble of Parameters and Strategies and Pool of Local Search Algorithms*, pages 615–626. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. (Cited on page 128.)

[42] Christian Igel. Neuroevolution for reinforcement learning using evolution strategies. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, volume 4, pages 2588–2595. IEEE, 2003. (Cited on page 22.)

[43] Eugene M. Izhikevich. Solving the distal reward problem through linkage of stdp and dopamine signaling. *Cerebral Cortex*, 17(10):2443–2452, 2007. (Cited on pages 4, 8, 69, and 70.)

[44] Eduardo Izquierdo-Torres and Inman Harvey. Hebbian Learning using Fixed Weight Evolved Dynamical 'Neural' Networks. In *Artificial Life, 2007. ALIFE'07. IEEE Symposium on*, pages 394–401. IEEE, 2007. (Cited on page 23.)

[45] Giorgos Karafotias, Mark Hoogendoorn, and Ágoston E Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187, 2015. (Cited on page 13.)

[46] Hiroaki Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex systems*, 4(4):461–476, 1990. (Cited on page 22.)

[47] Jan Koutnik, Faustino Gomez, and Jürgen Schmidhuber. Evolving neural networks in compressed weight space. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 619–626. ACM, 2010. (Cited on page 23.)

[48] Taras Kowaliw, Nicolas Bredeche, Sylvain Chevallier, and René Doursat. Artificial neurogenesis: An introduction and selective review. In *Growing Adaptive Machines*, pages 1–60. Springer, 2014. (Cited on pages 2, 3, 20, 22, and 41.)

[49] John R Koza et al. *Genetic programming II*, volume 17. MIT press Cambridge, 1994. (Cited on page 13.)

[50] Oliver Kramer. *Self-adaptive heuristics for evolutionary computation*, volume 147. Springer, 2008. (Cited on page 13.)

[51] Eduard Kuriscak, Petr Marsalek, Julius Stroffek, and Peter G Toth. Biological context of Hebb learning in artificial neural networks, a review. *Neurocomputing*, 152:27–35, 2015. (Cited on pages 1, 42, and 44.)

[52] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015. (Cited on pages 18, 19, 22, and 26.)

[53] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. (Cited on page 37.)

[54] Joel Lehman and Kenneth O Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336, 2008. (Cited on pages 89 and 92.)

[55] JJ Liang, BY Qu, PN Suganthan, and Alfredo G Hernández-Díaz. Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization. *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report*, 201212:3–18, 2013. (Cited on page 136.)

[56] M. Lichman. UCI machine learning repository, 2013. (Cited on page 31.)

[57] Yong Liu, Xin Yao, Qiangfu Zhao, and Tetsuya Higuchi. Scaling up fast evolutionary programming with cooperative coevolution. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 2, pages 1101–1108. Ieee, 2001. (Cited on page 26.)

[58] Sedigheh Mahdavi, Mohammad Ebrahim Shiri, and Shahryar Rahnamayan. Metaheuristics in large-scale global continues optimization: A survey. *Information Sciences*, 295:407–428, 2015. (Cited on pages 26 and 27.)

[59] R. Mallipeddi, G. Iacca, P. N. Suganthan, F. Neri, and E. Mininno. Ensemble strategies in compact differential evolution. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 1972–1977, June 2011. (Cited on page 128.)

[60] Rammohan Mallipeddi, Ponnuthurai N Suganthan, Quan-Ke Pan, and Mehmet Fatih Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing*, 11(2):1679–1696, 2011. (Cited on pages 128 and 131.)

[61] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017. (Cited on page 22.)

[62] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9(1):2383, 2018. (Cited on page 20.)

[63] David J Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767, 1989. (Cited on page 21.)

[64] David Eric Moriarty. *Symbiotic evolution of neural networks in sequential decision tasks*. PhD thesis, University of Texas at Austin USA, 1997. (Cited on pages 8 and 27.)

[65] Gregory Morse and Kenneth O Stanley. Simple evolutionary optimization can rival stochastic gradient descent in neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 477–484. ACM, 2016. (Cited on pages 25, 28, 30, and 32.)

[66] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015. (Cited on page 89.)

[67] Jean-Baptiste Mouret and Paul Tonelli. Artificial evolution of plastic neural networks: a few key concepts. In *Growing Adaptive Machines*, pages 251–261. Springer, 2014. (Cited on pages 20 and 23.)

[68] Ferrante Neri and Ville Tirronen. Recent advances in differential evolution: a survey and experimental analysis. *Artificial Intelligence Review*, 33(1-2):61–106, 2010. (Cited on pages 15 and 17.)

[69] Yael Niv, Daphna Joel, Isaac Meilijson, and Eytan Ruppin. Evolution of Reinforcement Learning in Uncertain Environments: A Simple Explanation for Complex Foraging Behaviors. *Adaptive Behavior*, 10(1):5–24, April 2002. (Cited on pages 8, 23, 42, 44, and 45.)

[70] Stefano Nolfi, Orazio Miglino, and Domenico Parisi. Phenotypic plasticity in evolving neural networks. In *From Perception to Action Conference, 1994., Proceedings*, pages 146–157. IEEE, 1994. (Cited on pages 20 and 23.)

[71] Mohammad Nabi Omidvar, Xiaodong Li, Yi Mei, and Xin Yao. Cooperative co-evolution with differential grouping for large scale optimization. *IEEE Transactions on evolutionary computation*, 18(3):378–393, 2014. (Cited on page 27.)

[72] Jeff Orchard and Lin Wang. The evolution of a generalized neural learning rule. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 4688–4694. IEEE, 2016. (Cited on pages 8, 23, 42, and 45.)

[73] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 2019. (Cited on page 60.)

[74] Mitchell A Potter and Kenneth A De Jong. A cooperative coevolutionary approach to function optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 249–257. Springer, 1994. (Cited on pages 25 and 27.)

[75] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006. (Cited on page 16.)

[76] Justin K Pugh, Lisa B Soros, Paul A Szerlip, and Kenneth O Stanley. Confronting the challenge of quality diversity. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 967–974. ACM, 2015. (Cited on page 94.)

[77] A Kai Qin, Vicky Ling Huang, and Ponnuthurai N Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2):398–417, 2009. (Cited on pages 16, 129, 136, and 138.)

[78] A Kai Qin and Ponnuthurai N Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1785–1791. IEEE, 2005. (Cited on page 128.)

[79] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017. (Cited on page 22.)

[80] Ingo Rechenberg. Cybernetic solution path of an experimental problem. *Royal Aircraft Establishment Library Translation 1122*, 1965. (Cited on page 13.)

[81] Sebastian Risi and Kenneth O Stanley. Indirectly encoding neural plasticity as a pattern of local rules. In *International Conference on Simulation of Adaptive Behavior*, pages 533–543. Springer, 2010. (Cited on pages 8, 23, 42, and 45.)

[82] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986. (Cited on page 19.)

[83] Thomas Philip Runarsson and Magnus Thor Jonsson. Evolution and design of distributed learning rules. In *Combinations of Evolutionary Computation and Neural Networks, 2000 IEEE Symposium on*, pages 59–63. IEEE, 2000. (Cited on pages 23 and 45.)

[84] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017. (Cited on pages 22 and 25.)

[85] J David Schaffer, Darrell Whitley, and Larry J Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37. IEEE, 1992. (Cited on page 21.)

[86] Hans-Paul Schwefel. *Numerical optimization of computer models*. John Wiley & Sons, Inc., 1981. (Cited on page 13.)

[87] Terrence J Sejnowski and Gerald Tesauro. The Hebb rule for synaptic plasticity: algorithms and implementations. In *Neural models of plasticity*, pages 94–103. Elsevier, 1989. (Cited on pages 42 and 44.)

[88] Anando Sen, Andrew Goldstein, Shreya Chakrabarti, Ning Shang, Tian Kang, Anil Yaman, Patrick B Ryan, and Chunhua Weng. The representativeness of eligible patients in type 2 diabetes trials: a case study using gist 2.0. *Journal of the American Medical Informatics Association*, 25(3):239–247, 2017. (Cited on page 153.)

[89] Yan-jun Shi, Hong-fei Teng, and Zi-qiang Li. Cooperative co-evolutionary differential evolution for function optimization. In *International Conference on Natural Computation*, pages 1080–1088. Springer, 2005. (Cited on page 27.)

[90] Moshe Sipper, Eduardo Sanchez, Daniel Mange, Marco Tomassini, Andrés Pérez-Uribe, and André Stauffer. A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Transactions on Evolutionary Computation*, 1(1):83–97, 1997. (Cited on page 1.)

[91] Andrea Soltoggio, John A Bullinaria, Claudio Mattiussi, Peter Dürr, and Dario Floreano. Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In *Proceedings of the 11th international*

*conference on artificial life (Alife XI)*, pages 569–576. MIT Press, 2008. (Cited on pages 8, 42, 44, 45, and 69.)

[92] Andrea Soltoggio and Kenneth O Stanley. From modulated Hebbian plasticity to simple behavior learning through noise and weight saturation. *Neural Networks*, 34:28–41, 2012. (Cited on pages 46 and 49.)

[93] Andrea Soltoggio, Kenneth O Stanley, and Sebastian Risi. Born to learn: The inspiration, progress, and future of evolved plastic artificial neural networks. *Neural Networks*, 2018. (Cited on pages 20, 23, 41, and 44.)

[94] Andrea Soltoggio and Jochen J. Steil. Solving the distal reward problem with rare correlations. *Neural computation*, 25(4):940–978, 2013. (Cited on pages 4, 8, 69, and 70.)

[95] Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007. (Cited on page 23.)

[96] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019. (Cited on pages 2, 19, and 25.)

[97] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019. (Cited on page 41.)

[98] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002. (Cited on page 22.)

[99] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997. (Cited on pages 13 and 15.)

[100] Yuan Sun, Michael Kirley, and Saman K Halgamuge. A recursive decomposition method for large scale continuous optimization. *IEEE Transactions on Evolutionary Computation*, 2017. (Cited on page 27.)

[101] Paul Tonelli and Jean-Baptiste Mouret. On the relationships between generative encodings, regularity, and learning abilities when evolving plastic artificial neural networks. *PloS one*, 8(11):e79138, 2013. (Cited on pages 23 and 45.)

[102] Joshua T Trachtenberg, Brian E Chen, Graham W Knott, Guoping Feng, Joshua R Sanes, Egbert Welker, and Karel Svoboda. Long-term in vivo imaging of experience-dependent synaptic plasticity in adult cortex. *Nature*, 420(6917):788, 2002. (Cited on pages 3 and 41.)

[103] Zlatko Vasilkoski, Heather Ames, Ben Chandler, Anatoli Gorchetchnikov, Jasmin Léveillé, Gennady Livitz, Ennio Mingolla, and Massimiliano Versace. Review of stability properties of neural plasticity rules for implementation on memristive neuromorphic hardware. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2563–2569. IEEE, 2011. (Cited on pages 4, 42, 44, and 45.)

[104] Günter P Wagner and Lee Altenberg. Perspective: complex adaptations and the evolution of evolvability. *Evolution*, 50(3):967–976, 1996. (Cited on page 20.)

[105] Chunhua Weng, Anil Yaman, Kuo Lin, and Zhe He. Trend and network analysis of common eligibility features for cancer trials in clinicaltrials. gov. In *International Conference on Smart Health*, pages 130–141. Springer, 2014. (Cited on page 155.)

[106] Darrell Whitley, Timothy Starkweather, and Christopher Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel computing*, 14(3):347–361, 1990. (Cited on page 20.)

[107] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014. (Cited on page 22.)

[108] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945. (Cited on pages 62, 80, and 138.)

[109] Anil Yaman, Shreya Chakrabarti, Anando Sen, and Chunhua Weng. How have cancer clinical trial eligibility criteria evolved over time? *AMIA Summits on Translational Science Proceedings*, 2016:269, 2016. (Cited on page 154.)

[110] Anil Yaman, Ahmed Hallawa, Matt Coler, and Giovanni Iacca. Presenting the eco: Evolutionary computation ontology. In Giovanni Squillero and Kevin Sim, editors, *Applications of Evolutionary Computation*, pages 603–619, Cham, 2017. Springer International Publishing. (Cited on pages 5, 7, 11, 13, and 154.)

[111] Anil Yaman, Giovanni Iacca, and Fabio Caraffini. A comparison of three differential evolution strategies in terms of early convergence with different population sizes. In *LeGO 2018 - International Workshop on Global Optimization, 18-21 September, Leiden, The Netherlands*, 2018. (Cited on pages 5, 11, 14, and 154.)

[112] Anil Yaman, Giovanni Iacca, Matt Coler, George Fletcher, and Mykola Pechenizkiy. Multi-strategy differential evolution. In Kevin Sim and Paul Kaufmann, editors, *Applications of Evolutionary Computation*, pages 617–633, Cham, 2018. Springer International Publishing. (Cited on pages 5, 7, 11, 14, 39, 127, and 154.)

[113] Anil Yaman, Giovanni Iacca, Decebal Mocanu, George Fletcher, and Mykola Pechenizkiy. Novelty producing synaptic plasticity. *(in Preparation)*. (Cited on pages 7, 88, and 153.)

[114] Anil Yaman, Giovanni Iacca, Decebal Constantin Mocanu, George Fletcher, and Mykola Pechenizkiy. Learning with delayed synaptic plasticity. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, pages 152–160, New York, NY, USA, 2019. ACM. (Cited on pages 2, 4, 6, 8, 69, and 154.)

[115] Anil Yaman, Stephen Lucci, and Izidor Gertner. Evolutionary algorithm based approach for modeling autonomously trading agents. *Intelligent Information Management*, 6(02):45, 2014. (Cited on page 153.)

[116] Anil Yaman, Decebal Constantin Mocanu, Giovanni Iacca, Matt Coler, George Fletcher, and Mykola Pechenizkiy. Evolving plasticity for autonomous learning under changing environmental conditions. *(Under review)*, 2019. (Cited on pages 2, 4, 6, 8, 41, 42, 69, and 153.)

[117] Anil Yaman, Decebal Constantin Mocanu, Giovanni Iacca, George Fletcher, and Mykola Pechenizkiy. Limited evaluation cooperative coevolutionary differential evolution for large-scale neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 569–576, New York, NY, USA, 2018. ACM. (Cited on pages 2, 3, 5, 8, 20, 25, 26, and 154.)

[118] Zhenyu Yang, Ke Tang, and Xin Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985–2999, 2008. (Cited on page 27.)

[119] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999. (Cited on pages 2, 19, and 20.)

[120] Wei Zeng and Richard L Church. Finding shortest paths on real road networks: the case for a. *International journal of geographical information science*, 23(4):531–543, 2009. (Cited on page 78.)

[121] Jingqiao Zhang and Arthur C Sanderson. Jade: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation*, 13(5):945–958, 2009. (Cited on page 129.)

[122] Xingwen Zhang, Jeff Clune, and Kenneth O Stanley. On the relationship between the openai evolution strategy and stochastic gradient descent. *arXiv preprint arXiv:1712.06564*, 2017. (Cited on page 22.)

# Appendices

# Appendix A

# Multi-strategy Differential Evolution

In this chapter[1], we propose here a Multi-strategy Differential Evolution (MsDE) approach to self-adapt strategies and their parameters in DE during an evolutionary process. Most of the self-adaptive parameter control approaches aim to adapt algorithm parameters by including them within the genotype of the individuals and inheriting with the successful individuals during an evolutionary run. In MsDE, distinct from the inheritance based methods, an ensemble of search strategies are employed to operate on, and co-evolve with the candidate solutions. The strategies are referred as agents to distinguish them from the candidate solutions, and underline their function. The agent-based representation of the strategies provides the flexibility to apply a wide range of population adaptation mechanisms. We present three population adaptation schemes (sampling-based, clone-best and clone-multiple), to show how various self-adaptive agent-based approaches perform on the CEC2013 benchmark functions. Notably, the approach we propose here can be easily extended to any evolutionary algorithm to adapt their operators and parameters.

The rest of the paper is organized as follows: in Section A.1, we discuss strategy and parameter adaptation mechanisms proposed in the literature for

[1]This chapter is integrally based on:
[112] Anil Yaman, Giovanni Iacca, Matt Coler, George Fletcher, and Mykola Pechenizkiy. Multi-strategy differential evolution. In Kevin Sim and Paul Kaufmann, editors, *Applications of Evolutionary Computation*, pages 617–633, Cham, 2018. Springer International Publishing

the DE; in Section A.2, we describe our algorithm, MsDE, and present the three mechanisms for population adaptation; in Section A.3, we introduce our experiment setup, in Section A.4, we present our test results, with the different population adaptation schemes, on the CEC2013 benchmark functions; finally, in Section A.5 we provide the conclusions and future work.

## A.1 Strategy and Parameter Control in DE

In this section, we highlight the recent developments in strategy and parameter control for DE. Modern variants of DE aim to employ adaptive mechanisms to adjust the algorithm's parameters during an evolutionary run, or across different problems. Strategy and parameter control in DE can be examined in two broad classes: *both strategy and parameter* control, and *only parameter* control [17], where the parameters involved are $F$ and $CR$. There are also methods for adapting the population size $NP$, see e.g. [39]; however, in this work we limit our scope to the methods for adapting the strategies, and the parameters $F$ and $CR$. In the following we briefly describe four of the main DE variants falling in this category, namely EPSDE, SaDE, JADE and jDE.

In the Ensemble of Parameters and mutation Strategies Differential Evolution (EPSDE), mutation strategy and parameter pools are used [41, 59, 60]. Each individual in the candidate solution population is assigned with a strategy and a parameter setting from these pools. The strategies and their parameters are inherited from the target to trial vectors as long as they are successful in generating a better trial vector. Otherwise, the strategy and parameters that are associated with the target vector are reinitialized by either randomly sampling from their respective pools, or assigning a strategy and its parameters from the set where successful strategies and parameters are stored.

Self-adaptive differential evolution (SaDE) uses only two mutation strategies, namely *"DE/rand/l/bin*" and *"DE/rand-to-best/1*", and adapts the parameters $F$ and $CR$ [78]. The strategies and parameters are selected for their properties of generating diverse individuals and faster convergence rate respectively. For each generation, a mutation strategy is randomly selected based on its probability of generating a trial vector successfully. The success probability of the two mutation strategies is initialized uniformly and updated after each generation, based on the number of individuals generated successfully. The scale factor $F$ is randomly sampled, for each individual, from the normal distribution with mean 0.5 and standard deviation 0.3. The parameter $CR$ is initialized for each individual from a normal distribution with mean 0.5 and standard deviation 0.1.

The strategy pool of the SaDE has been later extended by Qin *et al.* [77].

The JADE algorithm introduces a new mutation strategy called “DE/current-to-pbest" with optional archive, and controls the parameters $F$ and $CR$ [121]. The optional archive keeps track of recently explored worse solutions, to provide additional information for the progression of the search. At each generation, the crossover rate $CR_i$ is independently initialized from a normal distribution. The mean of the normal distribution $\mu_{CR}$ is initialized as 0.5 in for the first generation, and updated based on the mean of the $CR_i$ of the trial vectors that are generated successfully. The mutation factor $F_i$ is generated and updated in similar fashion by using a Cauchy distribution.

Finally, in jDE [7] the mutation and crossover parameters $F$ and $CR$ are attached to the genotype of the individuals in the population. The algorithm is based on the idea that the parameters that survive with the individuals are likely to produce successful trial vectors; thus, the parameters of the target vectors are propagated to the successive trial vectors in the next generations.

## A.2 Multi-strategy Differential Evolution (MsDE)

The MsDE aims to self-adapt the strategy types (mutation and crossover operators) and their parameters ($F$ and $CR$) used in DE while solving the optimization problem. It employs an ensemble of strategies with certain parameter settings, and applies population adaptation schemes to construct and maintain the ensemble strategy set. Different from the established ensemble methods in the literature, the MsDE considers the strategies as agents that interact with the candidate solution set. The agent-based representation of the strategies provides the basis for an easy application of population adaptation approaches.

The pseudocode of MsDE (assuming an asynchronous population, see below) is provided in Algorithm 2. The algorithm takes $NP$ (number of solutions) and $m$ (number of strategies) as parameters. In addition, there are two thresholds we refer to as performance and maturation thresholds, $\tau$ and $\delta$, for determining the performance of a strategy and limiting the test phase of a strategy. The performance threshold $\tau$ is an adaptive threshold based on the average value of all the performances in the strategy ensemble. The maturation threshold $\delta$ is typically a small integer (e.g. 5) used to control how many algorithm iterations should be invested for the testing phase of new strategies.

The candidate solution set $X$ consisting of $NP$ $D$-dimensional real-valued vectors $x_i \in \mathbb{R}^D$ that represent a solution to the problem. The initial candidate solutions are randomly sampled in the domain range for each dimension. The

**Algorithm 2** Asynchronous MsDE

**procedure** MsDE($NP$, $m$)
  $g \leftarrow 0$ ▷ generation count
  initialize $X$ ▷ randomly initialize $NP$ solutions
  $\forall \sigma_j \in \Sigma, j = 1,2,\ldots,m; \sigma_j \leftarrow InitializeRandomStrategy()$ ▷ see Alg. 3
  $F \leftarrow evaluate(X)$
  **while** termination criterion is not satisfied **do**
    $\tau \leftarrow mean(P_\Sigma)$ ▷ $\tau$ is the average performance of the strategies
    **for each** $\sigma \in \Sigma$ **do**
      $targetVector \leftarrow randSelect(X)$ ▷ randomly select a target vector
      $mutantVector \leftarrow \sigma.mutate(targetVector)$
      $trialVector \leftarrow \sigma.crossover(targetVector, mutantVector)$
      $\sigma.totalActivation \leftarrow \sigma.totalActivation + 1$
      $F_{trial} \leftarrow evaluate(trialVector)$
      **if** $F_{trial} < F_{target}$ **then** ▷ selection operator (assuming minimization)
        $targetVector \leftarrow trialVector$
        $F_{target} \leftarrow F_{trial}$
        $\sigma.successfulActivation \leftarrow \sigma.successfulActivation + 1$
      **end if**
      $P_{\sigma_j} \leftarrow evaluate(\sigma)$ ▷ performance of a strategy
      **if** $P_{\sigma_j} < \tau$ and $\sigma.totalActivation > \delta$ **then**
        reinitialize $\sigma$
      **end if**
    **end for**
    $g \leftarrow g + 1$
  **end while**
**end procedure**

population size $NP$ is chosen during the initialization phase, and remains fixed throughout the run.

The ensemble strategy set $\Sigma$ consists of $m$ strategies $\sigma_1, \sigma_2, \ldots, \sigma_m \in \Sigma$. Each $\sigma_j$ defines a kind of mutation and crossover operator, with specified parameters $F$ and $CR$. In the initialization phase, each strategy is initialized by selecting a random mutation strategy with a type of crossover operator from a predefined set of strategies $S$, of size $l$. The parameters $F$ and $CR$ are randomly sampled

from a uniform distribution in $(0,1.2]$ and $[0,1]$, respectively. The upper limit of the scale factor is set to 1.2 because of the works that report the effective range for $F$ between $(0,1.2]$ [60]. The initialization procedure is illustrated in Algorithm 3.

**Algorithm 3** Initialize random strategy

```
1: function InitializeRandomStrategy()
2:     randomIndex ← randi[1, l]
3:     σ_random.type ← S[randomIndex]
4:     σ_random.F ← rand(0, 1.2]
5:     σ_random.CR ← rand[0, 1]
6:     return σ_random
7: end function
```

The main loop of MsDE repeats until a stopping criteria is reached. In each iteration, each strategy agent $\sigma_j$ is executed $\forall j \in (1,2,\ldots,m)$, such that: first, a target vector $x_i$ from $X$ is randomly selected; secondly, the mutation and crossover operators are applied to generate a trial vector $u_i$; finally, the selection operator is applied to replace the target vector with the trial vector if its fitness value is better or equal. The selection operator can be *synchronous* or *asynchronous* as discussed in Section 2.1.2.

The MsDE calculates a performance measure within the function $evaluate(\sigma)$ to evaluate each strategy. Based on this performance measure, the strategies are classified as successful or unsuccessful. To solve the problem efficiently, firstly successful or unsuccessful strategies should be identified as quickly as possible; and secondly, the number of successful strategies in the population should be maximized, or, vice versa, the number of unsuccessful strategies should be minimized. We discuss these two aspects in the following sections.

### Identifying successful strategies

Constructing and maintaining a successful set of strategies is crucial for the performance of the algorithm. The self-adaptive mechanism for ensemble construction and maintenance should be capable of managing the trade-off of exploring and exploiting successful strategies efficiently and adaptively. We use a performance measure to asses the quality of a strategy. We propose two measures $P_1$ and $P_2$ with different properties. The performance measure $P_1$, given in Equation (A.1), measures the ratio between the number of strategy activations

that led to a successful action and the total number of strategy activations:

$$P_1 = \frac{\sigma_j.successfulActivation}{\sigma_j.totalActivation} \tag{A.1}$$

where $\sigma_j.successfulActivation$ is the number of activations in which a strategy $\sigma_j$ produced a trial vector with better fitness than the target vector, and $\sigma_j.totalActivation$ is the number of total activations.

As we will demonstrate in Section A.3, a strategy selection criterion based on $P_1$ is likely to facilitate fast convergence; the drawback is a high probability of getting stuck onto a local optimum if the function is multimodal. This is because the strategies that are exploitative are more likely to score higher on $P_1$ than the exploratory strategies because small exploitative increments on the solutions are more likely to yield better solutions. To prevent the domination of exploitative strategies in the ensemble, the performance measure should be improved to promote also exploratory strategies. We measure the exploratory value of a strategy by its capability to produce diverse individuals with better fitness values. Such performance evaluation criteria would also encourage the diversity in the population; thus, it may be less susceptible to early convergence.

Methods used in multi-objective optimization, such as non-dominated sorting, can be used to select the strategies that have diverse trial vectors generation rate and high ratio of success [21]. On the other hand, these methods can increase the complexity of the algorithm. Thus, to avoid further complexity, we provide a performance measure $P_2$ for a single selection criterion to combine these two aspects implicitly in Equation (A.2), where we calculate the average differences between target and trial vectors for the last $\gamma$ activations in which the trial generation was successful.

$$P_2 = \begin{cases} \dfrac{1}{\sum_{a=TA_{\sigma_j}-\gamma+1}^{TA_{\sigma_j}} \psi_{\sigma_j}^{(a)}} \cdot \sum_{a=TA_{\sigma_j}-\gamma+1}^{TA_{\sigma_j}} \Delta_{\sigma_j}^{a} \cdot \psi_{\sigma_j}^{(a)}, & \text{if } TA_{\sigma_j} \geq \gamma; \\ \dfrac{1}{\sum_{a=1}^{TA_{\sigma_j}} \psi_{\sigma_j}^{(a)}} \cdot \sum_{a=1}^{TA_{\sigma_j}} \Delta_{\sigma_j}^{a} \cdot \psi_{\sigma_j}^{(a)}, & \text{otherwise.} \end{cases} \tag{A.2}$$

$$\Delta_{\sigma_j}^{(a)} = \sum_{d=1}^{D} \mid x_{i,d}^{(a)} - u_{i,d}^{(a)} \mid \tag{A.3}$$

$$\psi_{\sigma_j}^{(a)} = \begin{cases} 1, & \text{if } f(x_i^{(a)}) < f(u_i^{(a)}); \\ 0, & \text{otherwise.} \end{cases} \tag{A.4}$$

where $\Delta_{\sigma_j}^{(a)}$, defined by the distance metric given in Equation (A.3), is the sum of the absolute differences along each dimension between target and trial vectors, and $TA_{\sigma_j}$ represents $\sigma_j.totalActivation$. The parameter $\gamma$ is introduced into this measure to sum only the differences in the most recent history of activations, to have a self-adaptive property. If $\gamma$ is a large number then the measure may promote exploratory strategies that may have a few successful activations with large diversity.

**Maximizing the number of successful strategies**

For an efficient search, the strategies that are classified as successful should be kept in the ensemble as long as they remain successful. To identify the performance of a strategy, strategies are tested for a certain time. The testing phase consumes resources, namely function evaluations (FEs). Since the strategies activated during the testing phase may not be necessarily good, the resources consumed in this phase should be minimized.

To distinguish the successful and unsuccessful strategies, the average performance values of all strategies $\tau$ is used. If the performance value of a strategy $P_{\sigma_j} < \tau$, then it is considered to be unsuccessful. To collect necessary evidence on the performance of a strategy, a *maturation threshold* $\delta$ is used such that if a strategy does not exceed $\delta$ then it is neither classified as successful nor unsuccessful.

### A.2.1 Strategy Population Adaptation Schemes

We propose three mechanisms for strategy population adaptation: sampling-based, clone-best, and clone-multiple population adaptation schemes. These population adaptation schemes provide the logic for initiating new strategies into the ensemble, and removing existing strategies from the ensemble. The population adaptation methods are implemented in Line 21 of Algorithm 2.

**Sampling-based population adaptation**

The sampling-based adaptation scheme initiates new strategies based on random sampling. The sampling function given in Algorithm 3 is used to reinitialize

a mature unsuccessful strategy.

### Clone-best population adaptation

The clone-best adaptation scheme implements a clonal reproduction mechanism to replace unsuccessful strategies. The clone-best scheme is inspired by the clonal selection principle of the immune system theory [18, 19]. The main idea is to replace an unsuccessful strategy with a clone of the best performing strategy with a small perturbation.

**Algorithm 4** Strategy reinitialization by clone-best

```
1:  function Clone(σ_j^g)
2:      if rand(0,1) < φ then
3:          if rand(0,1) < η then
4:              σ_clone^g.type ← S[randomIndex]
5:          else
6:              σ_clone^g.type ← σ_j^g.type
7:          end if
8:          σ_clone^g.F ← σ_j^g.F + η · N(0,1)
9:          σ_clone^g.CR ← σ_j^g.CR + η · N(0,1)
10:     else
11:         σ_clone ← InitializeRandomStrategy()
12:     end if
13:     return σ_clone
14: end function
```

Algorithm 4 shows the function that is used to clone a strategy. In clone-best, the function takes $\sigma^g_{best}$ as an argument, where $\sigma^g_{best}$ is the best strategy in the current generation $g$. With probability $\phi$, the $type$, ($F$ and $CR$) of an unsuccessful strategy is replaced by the $type$ ($F$, and $CR$) of the best strategy in the ensemble, with a small perturbation with scale factor $\eta \in (0,1]$. In our experiments, we use $\eta = 0.1$. If the parameter boundaries are exceeded, they are reinitialized by a value close to the boundaries. If $rand(0,1) \geq \phi$ the strategy is reinitialized using the uniform sampling scheme given in Algorithm 3.

### A.2.2 Clone-multiple Population Adaptation

The clone-multiple adaptation is an extension of the clone-best adaptation where $m$ successful strategies are kept in a separate set referred to as memory strategies ($\Sigma$). If the limit of $\Sigma$ is not exceeded, the scheme aims to find more successful strategies to add to $\Sigma$ by going through the *cloning*, *selection*, *maturation* and *promotion* phases. These phases are described below:

1. ***Clonal expansion:*** $n$ best strategies from $\Sigma$ are selected and assigned into the best strategy set $B$. Each strategy in $B$ is cloned (with a small perturbation) proportional to their performance. The higher the performance of a strategy, the higher the number of clones generated. Algorithm 4, without $\phi$ parameter (or $\phi = 1$), is used for generating each clone for each $\sigma_j^g \in B$. Generated clones are added to a temporary candidate clone set $T$.

2. ***Clonal selection:*** $h$ ($h \leq n$) candidate clones from $T$ are selected based on their similarity to the strategies in $B$; and $v (v \leq h)$ strategies are generated randomly. The selected and randomly generated individuals are then added to the clone set $C$, that has size $h + v$. The similarity-based clone selection and the random strategy generation criteria are executed as follows:

   - $h$ individuals are selected as follows: for each $\sigma_i \in T, i = 1,2,\ldots,size(T)$ and $sigma_j \in B, j = 1,2,\ldots,n$, $d_{\sigma_i} = \sum_{j=1}^{n} dist(\sigma_i,\sigma_j)$ is calculated. The $h$ candidate clones with smallest $d_{\sigma_i}$ are added to the clone set $C$. The $dist(\sigma_i,\sigma_j)$ computes the Euclidean distance between the parameters ($F$ and $CR$) of $\sigma_i$ and $\sigma_j$;
   - $v$ random strategies are generated using Algorithm 3.

3. ***Maturation:*** each strategy in $C$ is tested for $\delta$ FEs.

4. ***Promotion:*** strategies in $C$ that are successful (i.e., that satisfy the performance threshold) are added to the memory set $\Sigma$. The same classification criterion for finding unsuccessful/successful strategies is used for finding successful strategies in $C$.

The logic behind the clone-multiple population adaptation scheme is such that it reduces the trial and error of newly generated strategies, by cloning successful strategies that are kept in a separate set. It also aims to find strategies

that are likely to perform well by making a similarity-based selection. Mutations during the cloning phase allow for exploration of different strategies with different parameter settings.

## A.3 Experimental Setup

In this section, we present our experimental results on the CEC2013 benchmark functions [55]. The objective of our experiments is threefold. First, we show the effect of the two strategy performance measures $P_1$ and $P_2$ proposed in Section A.2 on MsDE, with three population adaptation schemes. Second, we illustrate how the strategy adaptation dynamics compare between the sampling, clone-best and clone-multiple based population adaptation schemes during an evolutionary run. Finally, we compare the MsDE with basic DE and SaDE.

The types of the strategies and parameters of the algorithms are the same for all the experiments, unless otherwise specified. We employ four types of DE strategies referred to as *"DE/rand/1/bin*", *"DE/rand/2/bin*", *"DE/rand-to-best/2/bin*", and *"DE/current-to-rand/1*". The suffix *"bin*" refers to the binomial crossover. Note that *"DE/current-to-rand/1*" does not include a crossover operator. These strategies are selected on the basis of previous comparisons performed in the literature; furthermore, they are also used in SaDE [77]. Asynchronous selection is used for the selection operator.

All the experiments were performed using $NP = 100$ candidate solutions and $m = 50$ strategies. The algorithms were run for at most $5000 \times D$ function evaluations (FEs), where $D$ is the dimension of the problem. If the error between the best solution found and the global optimum is less than or equal to $1e-8$, we terminate the algorithm. Each algorithm was executed for 25 independent runs; the mean and standard deviation of minimum error $f(x_{best}) - f(x^*)$ achieved are presented.

The three strategy adaptation schemes (sampling-based, clone-best and clone-multiple) are referred to as *MsDE-Sam*, *MsDE-CB*, and *MsDE-CM*, respectively. For MsDE-CB, the probability for cloning the best strategy $\phi$ is set to 0.7. For MsDE-CM, the number of selected best strategies $n$ is set to 10, the max number of clones per strategy is set to 10 for the best strategy and reduced by 1 per each lower ranked strategy, the number of similar selected strategies is set to $h = 7$, and the number of randomly initialized strategies is set to $v = 3$; thus the number of strategies adds up to a total number of 10. For all algorithms, the maturation threshold and the history threshold $\gamma$ are set to 5 FEs and 10 activations, respectively.

## A.4 Experimental Results

In Table A.1, we compare the two different performance measures $P_1$ and $P_2$ for each kind of population adaptation scheme on the CEC2013 functions in 10 dimensions. The suffixes "-$P_1$" and "-$P_2$" indicate the performance measure used with a specific kind of population adaptation scheme. The best results for each performance measure for each algorithm setting is highlighted in bold. The results that do not have significant difference were not highlighted. The global best result for each function is marked by the symbol "*".

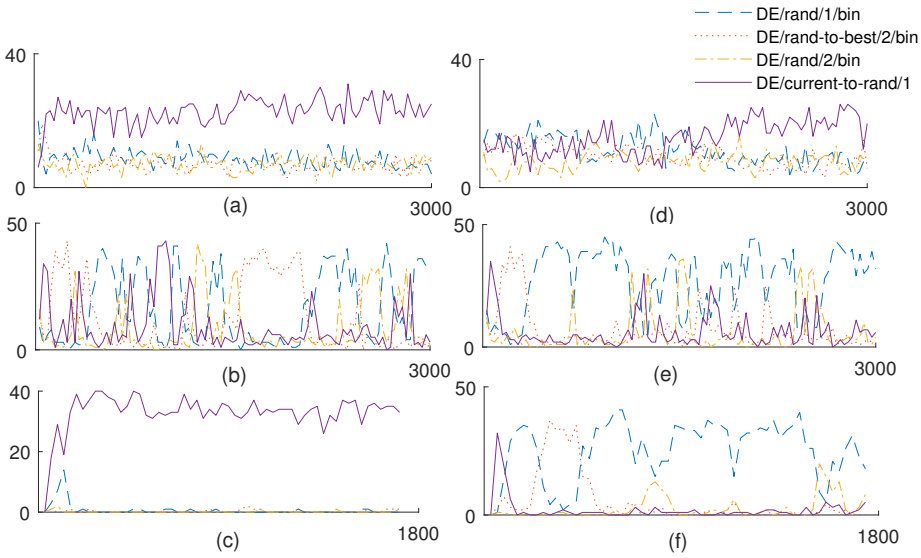
We observed that all three population adaptation schemes perform significantly better using $P_2$ on almost all benchmark functions. Since $P_1$ promotes the strategies based solely on the ratio of producing successful trial vectors, it is likely to promote exploitative strategies that can cause early convergence, or stalling the progress with small improvements. Performance measure $P_2$, on the other hand, promotes strategies that can produce diverse trial vectors successfully. The rest of the experiments are performed using $P_2$.

### A.4.1 Strategy ensemble adaptation dynamics

Next, we examine how the strategies adapt over time. In Figure A.1, we provide the results on $f_2$ (first column) and $f_6$ (second column) in 30 dimensions.

Each sub-figure in Figure A.1 shows how the distribution of strategies changes during an evolutionary run. Each line in the figures represents the number of strategies of a given type in the strategy population, at a given generation. Only the strategies that are mature and above the success threshold are counted. We observe that in MsDE-Sam (a) and (d), there is a baseline pool of random strategies that explores new strategies. The ratio of these pool is about %30 of the whole population. In MsDE-CB (b) and (e), this ratio is about %20; and in MsDE-CM (c) and (f), we observe that the random strategy pool is almost nonexistent, and there is usually one type of strategy that is dominant at each time. MsDE-Sam scheme constantly explores different strategies by keeping a small set of random strategies which can consume resources (number of function evaluations). On the other hand, MsDE-CB and MsDE-CM aims to exploit already found successful strategies by reintroducing them into the ensemble by reproduction.

Figure A.1: The distribution of strategies in the strategy population during an evolutionary run. The first and second columns show the results for $f_2$ and $f_6$ in 30 dimensions, while the rows show the results of MsDE-Sam, MsDE-CB, and MsDE-CM, respectively.




## A.4.2 Comparing with other algorithms

Finally, we test MsDE-Sam, MsDE-CB, MsDE-CM, basic DE and SaDE [77] on the CEC2013 benchmark functions in 30 dimensions. The parameters $F$ and $CR$ of the basic DE set are as 0.5 and 0.3 respectively. The results are given in Table A.2. The global best result for each function is highlighted in bold.

To assess the statistical significance of the results, we perform the Wilcoxon rank-sum test [108] based on the results provided in Table A.2. The Wilcoxon rank-sum test is a non-parametric test that does not assume normality condition [40, 108]. It is a pairwise test that aims to detect the significant difference between two different means that are the results of two algorithms. We reject the null-hypothesis, that is the behavior of the two algorithms are the same, if the $p$-value is smaller than $\alpha = 0.05$. We compare the MsDE-CM with the other algorithms using the best function error values of 25 independent runs

for each benchmark function. The results are given in Table A.2 next to the columns of the specified algorithms (except MsDE-CM, which is taken as the reference algorithm), where “=", “+" and “-" indicate no significant difference, significant difference in favor of MsDE-CM, and significant difference in favor of the specified algorithm. The results show that MsDE-CM is significantly better than basic DE and MsDE-CB, and on average better than SaDE and MsDE-Sam.

Table A.1: The experiment results with performance measures $P_1$ and $P_2$ on the CEC2013 benchmark problems in 10 dimensions.

| $f_i$ | MsDE-Sam-$P_1$ | MsDE-Sam-$P_2$ | MsDE-CB-$P_1$ | MsDE-CB-$P_2$ | MsDE-CM-$P_1$ | MsDE-CM-$P_2$ |
|---|---|---|---|---|---|---|
| $f_1$ | 8.47E-09±1.92E-09 | 8.74E-09±1.15E-09 | 9.04E-09±1.31E-04 | 8.52E-09±1.63E-09 | 8.49E-09±2.31E-02 | 8.77E-09±5.37E-09 |
| $f_2$ | 1.27E+03±4.20E+03 | **8.37E+02±3.20E+03** | 1.79E+05±3.34E+05 | **9.42E-09±3.52E-05*** | 1.00E+05±3.50E+06 | **8.16E-09±1.15E-06*** |
| $f_3$ | 5.77E+05±1.03E+07 | **2.08E+01±3.99E+05** | 1.72E+07±4.99E+08 | **4.95E-03±1.28E+00*** | 3.44E+07±1.06E+09 | **1.62E-01±1.25E+00** |
| $f_4$ | 8.19E-03±9.28E-01 | **2.51E-03±5.32E-02** | 5.45E+02±4.52E+03 | **8.83E-09±1.31E-09*** | 5.36E+01±6.69E+03 | **8.66E-09±9.76E-09*** |
| $f_5$ | 9.27E-09±1.07E-09 | 8.65E-09±1.11E-09 | 9.70E-09±7.56E-02 | 9.50E-09±6.92E-08 | 9.02E-09±8.79E-02 | 8.97E-09±1.16E-07 |
| $f_6$ | **9.53E-09±4.97E+00*** | 9.81E+00±4.81E+00 | 1.01E+01±8.31E+00 | **8.06E+00±3.67E+00** | 9.82E+00±2.66E+01 | **8.67E-09±2.72E+00*** |
| $f_7$ | 1.63E+01±1.61E+01 | **1.53E+00±8.36E+00** | 6.13E+01±2.80E+01 | **2.04E+01±1.21E+01** | 3.13E+01±4.49E+01 | **1.21E-03±5.52E-02*** |
| $f_8$ | 2.04E+01±7.74E-02 | 2.04E+01±9.58E-02 | 2.04E+01±8.67E-02 | **3.44E+00±7.08E-02** | 2.04E+01±1.22E-01 | **2.04E+01±9.15E-02*** |
| $f_9$ | 3.78E+00±1.27E+00 | **2.92E+00±9.42E-01** | 6.60E+00±1.39E+00 | **4.43E-02±1.28E+00*** | 7.41E+00±1.70E+00 | **1.96E+00±1.20E+00** |
| $f_{10}$ | 2.78E-01±2.71E-01 | **1.40E-01±6.72E-02** | **8.89E-01±2.73E+00** | 1.99E+00±4.68E-02 | 8.82E-01±1.62E+01 | **5.88E-08±2.99E-02*** |
| $f_{11}$ | 2.98E+00±3.52E+00 | **9.34E-09±6.47E-01*** | 1.39E+01±1.14E+01 | **1.09E+01±2.36E+00** | 6.59E+00±1.41E+01 | **9.15E-09±4.97E-01*** |
| $f_{12}$ | 1.59E+01±8.37E+00 | **1.09E+01±5.33E+00** | 3.28E+01±1.79E+01 | **2.44E+01±5.04E+00** | 1.88E+01±1.73E+01 | **3.98E+00±2.58E+00*** |
| $f_{13}$ | 2.65E+01±9.98E+00 | **1.58E+01±7.46E+00** | 4.66E+01±2.75E+01 | **4.33E+01±1.09E+01** | 3.57E+01±2.52E+01 | **5.05E+00±5.09E+00*** |
| $f_{14}$ | 1.51E+01±3.80E+01 | **3.54E+00±6.96E+00** | **6.17E+01±1.91E+02** | 7.68E+02±1.08E+02 | 6.48E+01±2.53E+02 | **3.12E-01±3.86E+00*** |
| $f_{15}$ | 7.96E+02±2.53E+02 | **7.02E+02±2.74E+02** | 8.19E+02±3.41E+02 | **4.24E-02±2.11E+02*** | 9.42E+02±4.02E+02 | **7.83E+02±2.65E+02** |
| $f_{16}$ | **9.49E-01±3.27E-01** | 1.07E+00±3.00E-01 | **4.91E-01±2.11E-01*** | 1.22E+01±5.89E-01 | **1.07E+00±4.45E-01** | 1.08E+00±6.94E-01 |
| $f_{17}$ | 1.04E+01±3.64E+00 | **1.02E+01±9.64E-02*** | 3.15E+01±1.31E+01 | **1.98E+01±1.21E+00** | 1.44E+01±6.61E+00 | **1.03E+01±6.07E-01** |
| $f_{18}$ | **2.09E+01±4.75E+00** | 2.14E+01±5.71E+00 | 4.02E+01±1.97E+01 | **7.42E-01±4.09E+00*** | 3.04E+01±1.32E+01 | **1.60E+01±2.63E+00** |
| $f_{19}$ | 6.00E-01±2.52E-01 | **3.68E-01±1.39E-01*** | **1.45E+00±1.28E+00** | 3.30E+00±2.30E-01 | 1.02E+00±6.46E-01 | **5.13E-01±1.70E-01** |
| $f_{20}$ | **2.43E+00±5.81E-01** | 2.78E+00±5.05E-01 | **3.89E+00±5.17E-01** | 4.00E+02±6.05E-01 | 3.55E+00±6.30E-01 | **2.40E+00±5.13E-01*** |
| $f_{21}$ | 4.00E+02±6.28E+01 | 4.00E+02±5.54E+01 | 4.00E+02±2.31E-01 | **8.44E+01±2.32E-13*** | 4.00E+02±8.73E+01 | 4.00E+02±7.08E+01 |
| $f_{22}$ | 7.54E+01±8.53E+01 | **3.12E+01±7.26E+01*** | **1.68E+02±2.35E+02** | 9.33E+02±1.69E+02 | 3.20E+02±2.79E+02 | **4.99E+01±5.09E+01** |
| $f_{23}$ | 9.89E+02±3.11E+02 | **8.33E+02±2.40E+02** | 7.92E+02±3.28E+02 | **2.06E+02±2.66E+02*** | 1.21E+03±3.45E+02 | **8.67E+02±3.07E+02** |
| $f_{24}$ | 2.12E+02±2.64E+01 | **2.09E+02±2.56E+01** | 2.21E+02±2.49E+01 | **2.00E+02±2.06E+01** | 2.18E+02±3.52E+01 | **2.00E+02±1.62E+01*** |
| $f_{25}$ | 2.11E+02±2.18E+01 | **2.02E+02±2.28E+01** | 2.20E+02±2.17E+01 | **1.19E+02±1.74E+01*** | 2.18E+02±1.98E+01 | **2.00E+02±1.78E+01** |
| $f_{26}$ | **1.56E+02±4.10E+01** | 2.00E+02±3.88E+01 | **1.32E+02±3.65E+01** | 3.00E+02±3.21E+01 | 1.51E+02±3.54E+01 | **1.06E+02±2.67E+01*** |
| $f_{27}$ | 4.00E+02±8.74E+01 | **3.00E+02±7.35E+01** | 4.58E+02±9.41E+01 | **3.00E+02±2.76E+01*** | 4.41E+02±1.09E+02 | **3.00E+02±2.00E+01** |
| $f_{28}$ | 3.00E+02±1.43E+02 | **3.00E+02±6.63E+01*** | **3.00E+02±2.15E+02** | 3.00E+02±9.80E+01 | **3.00E+02±1.86E+02** | 3.00E+02±4.00E+01 |

Table A.2: The experiment results of the selected algorithms on the CEC2013 benchmark problems in 30 dimensions.

| $f_i$ | MsDE-Sam | | MsDE-CB | | MsDE-CM | DE | | SaDE | |
|---|---|---|---|---|---|---|---|---|---|
| $f_1$ | 9.56E-09±7.88E-10 | = | 9.26E-09±9.58E-10 | = | 9.54E-09±6.05E-10 | 9.24E-09±5.51E-10 | = | 9.14E-09±1.14E-09 | = |
| $f_2$ | 1.40E+05±1.32E+05 | = | 1.71E+05±1.15E+05 | = | **1.19E+05±1.91E+05** | 2.21E+08±4.53E+07 | + | 1.76E+05±1.45E+05 | = |
| $f_3$ | 1.15E+05±4.06E+06 | = | 6.01E+06±1.37E+07 | + | **8.60E+04±1.59E+06** | 1.56E+09±8.56E+08 | + | 1.43E+05±1.72E+06 | = |
| $f_4$ | **1.22E+01±3.63E+01** | - | 1.73E+02±3.74E+02 | + | 4.40E+01±6.08E+01 | 1.16E+05±1.64E+04 | + | 3.52E+03±2.42E+03 | + |
| $f_5$ | 9.58E-09±4.97E-10 | = | 9.64E-09±7.46E-10 | = | 9.71E-09±3.71E-10 | 9.58E-09±5.56E-10 | = | 9.60E-09±6.51E-10 | = |
| $f_6$ | 9.67E+00±5.07E+00 | = | 9.41E+00±6.67E+00 | = | **9.17E+00±1.72E+01** | 2.63E+01±3.33E-01 | + | 1.22E+01±1.25E+01 | + |
| $f_7$ | 1.46E+01±1.03E+01 | - | 7.23E+01±1.64E+01 | + | 2.87E+01±1.40E+01 | 1.46E+02±1.48E+01 | + | **1.22E+01±6.62E+00** | - |
| $f_8$ | **2.10E+01±6.72E-02** | - | 2.10E+01±4.33E-02 | - | 2.10E+01±6.37E-02 | 2.10E+01±5.53E-02 | - | 2.10E+01±5.50E-02 | - |
| $f_9$ | **1.73E+01±3.08E+00** | - | 2.72E+01±4.73E+00 | + | 2.24E+01±5.74E+00 | 3.94E+01±1.14E+00 | + | 1.91E+01±2.32E+00 | - |
| $f_{10}$ | 3.45E-02±2.64E-02 | = | 5.17E-02±3.94E-02 | = | **3.45E-02±2.56E-02** | 1.27E+02±3.62E+01 | + | 4.93E-02±4.01E-02 | = |
| $f_{11}$ | 8.95E+00±4.54E+00 | - | 2.98E+01±1.16E+01 | + | 1.69E+01±8.36E+00 | 6.83E+01±5.55E+00 | + | **1.81E-05±5.57E+00** | - |
| $f_{12}$ | 4.58E+01±9.63E+00 | + | 7.16E+01±2.25E+01 | + | **3.18E+01±8.87E+00** | 2.03E+02±1.06E+01 | + | 3.48E+01±6.34E+00 | + |
| $f_{13}$ | 8.77E+01±2.31E+01 | + | 1.54E+02±4.15E+01 | + | **6.86E+01±2.37E+01** | 2.10E+02±1.50E+01 | + | 6.94E+01±2.64E+01 | = |
| $f_{14}$ | **2.90E+01±3.26E+01** | - | 9.99E+02±4.87E+02 | + | 6.11E+01±1.25E+02 | 3.93E+03±2.33E+02 | + | 1.81E+03±6.37E+02 | + |
| $f_{15}$ | 6.41E+03±1.15E+03 | + | 3.63E+03±5.78E+02 | = | 3.68E+03±1.49E+03 | 7.78E+03±3.10E+02 | + | **3.42E+03±5.60E+02** | = |
| $f_{16}$ | 2.59E+00±2.30E-01 | + | **2.03E-01±8.17E-01** | - | 2.33E+00±9.58E-01 | 2.71E+00±3.02E-01 | + | 2.51E+00±3.66E-01 | = |
| $f_{17}$ | **3.34E+01±1.61E+00** | - | 6.01E+01±1.11E+01 | + | 3.57E+01±4.18E+00 | 1.00E+02±6.01E+00 | + | 5.80E+01±9.32E+00 | + |
| $f_{18}$ | 1.70E+02±4.10E+01 | + | 8.28E+01±1.76E+01 | + | **5.15E+01±3.82E+01** | 2.32E+02±1.05E+01 | + | 5.55E+01±1.15E+01 | = |
| $f_{19}$ | 2.70E+00±1.04E+00 | + | 4.95E+00±2.64E+00 | + | **2.40E+00±5.24E-01** | 1.10E+01±6.77E-01 | + | 2.75E+00±9.47E-01 | + |
| $f_{20}$ | 1.14E+01±5.08E-01 | + | 1.18E+01±1.34E+00 | + | **1.03E+01±1.12E+00** | 1.37E+01±1.71E-01 | + | 1.04E+01±7.84E-01 | = |
| $f_{21}$ | **3.00E+02±8.32E+01** | = | 3.00E+02±8.77E+01 | = | 3.00E+02±8.04E+01 | 3.00E+02±4.26E+01 | = | 3.00E+02±6.78E+01 | + |
| $f_{22}$ | **1.40E+02±5.62E+01** | - | 1.29E+03±5.41E+02 | + | 1.66E+02±2.01E+02 | 4.55E+03±3.33E+02 | + | 1.47E+03±7.88E+02 | + |
| $f_{23}$ | 6.67E+03±1.24E+03 | + | 4.52E+03±7.48E+02 | = | 3.97E+03±1.14E+03 | 8.04E+03±2.95E+02 | + | **3.60E+03±6.42E+02** | = |
| $f_{24}$ | 2.19E+02±6.43E+00 | = | 2.40E+02±1.15E+01 | + | 2.16E+02±7.23E+00 | 3.01E+02±2.62E+00 | + | **2.09E+02±3.28E+00** | - |
| $f_{25}$ | **2.69E+02±8.14E+00** | = | 2.92E+02±1.11E+01 | + | 2.80E+02±3.02E+01 | 3.02E+02±2.63E+00 | + | 2.73E+02±6.39E+00 | = |
| $f_{26}$ | 2.00E+02±3.58E-03 | = | 2.00E+02±7.16E-03 | = | **2.00E+02±8.89E-03** | 2.79E+02±3.73E+01 | + | 2.00E+02±9.65E-03 | = |
| $f_{27}$ | 4.94E+02±6.56E+01 | = | 8.49E+02±1.27E+02 | + | 5.28E+02±1.03E+02 | 1.32E+03±1.91E+01 | + | **4.56E+02±7.11E+01** | = |
| $f_{28}$ | **3.00E+02±4.14E-13** | - | 3.00E+02±1.04E-10 | - | 3.00E+02±1.02E-07 | 3.00E+02±1.66E-08 | + | 3.00E+02±1.87E-09 | - |

## A.5 Conclusions and Future Work

In this work, we propose the Multi-strategy Differential Evolution (MsDE) algorithm to construct and maintain an ensemble set of strategies with various parameters. MsDE is capable of self-adapting the type of strategy and its parameters $F$ and $CR$. Different from the alternative approaches, MsDE represents the ensemble strategy population as agents that interact with the candidate solutions. The performance of the strategies is measured by a performance measure which is used to self-adapt the ensemble population.

We propose two performance measures, and compare their efficiency in constructing an ensemble of successful strategies. Our results show that favoring strategies that can produce diverse trial vectors successfully yields better than favoring them based solely on their ratio of producing successful trial vectors.

We propose three approaches for self-adapting the strategy population. The simplest approach is based on random sampling where new strategies are randomly introduced into, and the ones that do not satisfy a performance criterion are removed from the ensemble. Other two approaches use clonal selection mechanism to proliferate successful strategies in the ensemble. While, four different types of strategies with their continuous $F$ and $CR$ parameters are aimed to be optimized, the sampling based approach requires only a parameter for ensemble size, and a threshold for defining a successful strategy based on its performance metric. The clonal selection based algorithms introduce an additional parameter for perturbing strategies. In this work, we used asynchronous selection operator. Asynchronous selection can speed up the convergence and decrease the population diversity. We would like to examine the effect of synchronous/asynchronous update in the future work.

We compare the MsDE with basic DE and the SaDE algorithm with different combinations strategy performance measure and population adaptation schemes. Overall, our results show that the MsDE provides better results on CEC2013 benchmark functions. In future works, we will try to extend the MsDE approach to other evolutionary algorithms.

# Appendix B

# Algorithms

**Algorithm 5** LECCDE

**procedure** LECCDE($NP$, $F$, $CR$)
  Initialize $NP$ individuals in each subpopulation $P_i, i \in (1, SP)$
  Initialize $Ft_{i,j} \leftarrow 0$ ▷ Fitness of the $j$th individual in the $i$th subpopulation
  **for** $c = 1$ to $trial \times NP$ **do**
    Select a random individual $\boldsymbol{x_{i,r_j}}$ from each $P_i$ ▷ $r_j$ is a randomly generated integer index
    $\boldsymbol{X} \leftarrow \{\boldsymbol{x_{1,r_1}}, \boldsymbol{x_{2,r_2}}, \cdots, \boldsymbol{x_{SP,r_{SP}}}\}$
    $Ft_X \leftarrow evaluate(\boldsymbol{X}, b_1)$ ▷ $b_1$ is the first batch
    $\forall \boldsymbol{x_i} \in \boldsymbol{X}, Ft_{i,j} \leftarrow Ft_{i,j} + Ft_X$
  **end for**
  $\forall i \in (1, SP)$ and $j \in (1, NP), Ft_{i,j} \leftarrow normalize(Ft_{i,j})$
  $\boldsymbol{X} \leftarrow \{\boldsymbol{x_{1,max}}, \boldsymbol{x_{2,max}}, \cdots, \boldsymbol{x_{SP,max}}\}$
  $Ft_{validation} \leftarrow evaluate(\boldsymbol{X}, ValidationSet)$
  $bestValidation \leftarrow Ft_{validation}$
  $bestNetwork \leftarrow \boldsymbol{X}$
  **while** termination criterion is not satisfied **do**
    **for each** $b_k \in Batches$ **do**
      **for each** subpopulation $P_i$ **do**
        $P'_i \leftarrow P_i$
        $Ft'_i \leftarrow Ft_i$
        **for each** $\boldsymbol{x_{i,j}} \in P_i$ **do**
          $\boldsymbol{X_i} \leftarrow \boldsymbol{x_{i,j}}$
          $Ft_X \leftarrow evaluate(\boldsymbol{X}, b_k)$
          $Ft'_X \leftarrow Ft_{i,j} \cdot (1 - decay) + Ft_X$
          $\boldsymbol{v} \leftarrow mutate(\boldsymbol{x_{i,r_1}}, \boldsymbol{x_{i,r_2}}, \boldsymbol{x_{i,r_3}}, F)$
          $Ft_v \leftarrow (Ft_{i,r_1} + Ft_{i,r_2} + Ft_{i,r_3}) / 3$
          $\boldsymbol{u} \leftarrow crossover(\boldsymbol{x_{i,j}}, \boldsymbol{v}, CR)$
          $\boldsymbol{X_i} \leftarrow \boldsymbol{u}$
          $Ft_u \leftarrow evaluate(\boldsymbol{X}, b_k)$
          $Ft'_u \leftarrow ((Ft_{i,j} + Ft_v) / 2) \cdot (1 - decay) + Ft_u$
          **if** $Ft'_u > Ft'_X$ **then**
            $P'_{i,j} \leftarrow \boldsymbol{u}$
            $Ft'_{i,j} \leftarrow Ft'_u$
          **else**
            $P'_{i,j} \leftarrow \boldsymbol{x_{i,j}}$
            $Ft'_{i,j} \leftarrow Ft'_X$
          **end if**
        **end for**

```
                $P_i \leftarrow P'_i$
                $Ft_i \leftarrow Ft'_i$
                $\boldsymbol{X_i} \leftarrow \boldsymbol{x_{i,max}}$
                $Ft_{validation} \leftarrow evaluate(\boldsymbol{X}, ValidationSet)$
                if $Ft_{validation} > bestValidation$ then
                    $bestNetwork \leftarrow X$
                    $bestValidation \leftarrow Ft_{validation}$
                end if
            end for
        end for
    end while
end procedure
```

**Algorithm 6** Evaluation of an individual DSP rule

```
procedure Evaluate($\boldsymbol{x}, trials, N_{episodes}$)
    $fitness \leftarrow 0$
    for each $t \in trials$ do
        for each $g \in goalPositions$ do                ▷ $N_{goals}$ positions
            $EP_0 \leftarrow inf$, $EP_{best} \leftarrow inf$, $e \leftarrow 1$
            $RNN \leftarrow initializeRandom$
            while $e \leq N_{episodes}$ do
                $EP_e \leftarrow testNetwork(RNN)$
                $m \leftarrow -1$
                if $EP_e \leq EP_{e-1}$ then
                    $m \leftarrow 1$
                end if
                $EP_{best} \leftarrow min(EP_e, EP_{best})$
                $RNN \leftarrow synapticUpdate(RNN, \boldsymbol{x}, m)$
                $e \leftarrow e + 1$
            end while
            $fitness \leftarrow fitness + EP_{best}$
        end for
    end for
    return $fitness/(trials \cdot N_{goals})$
end procedure
```

**Algorithm 7** Evaluation of an individual NPSP rule

**procedure** Evaluate($\boldsymbol{x}, trials, N_{episodes}$)
  $fitness \leftarrow 0$
  **for each** $t \in trials$ **do**
    **for each** $s \in startPositions$ **do** ▷ $N_{start}$ positions
      $behaviors \leftarrow \{\}$
      $e \leftarrow 1$
      $RNN \leftarrow initializeRandom$
      **while** $e \leq N_{episodes}$ **do**
        $behavior \leftarrow testNetwork(RNN)$
        $behaviors \leftarrow behaviors \bigcup behavior$
        $RNN \leftarrow synapticUpdate(RNN, \boldsymbol{x})$
        $e \leftarrow e + 1$
      **end while**
      $fitness \leftarrow fitness + \frac{\left|unique(behaviors)\right|}{N_{episodes}}$
    **end for**
  **end for**
  **return** $fitness / (trials \cdot N_{start})$
**end procedure**

# Appendix C

# Evolved Delayed Synaptic Plasticity Rules

Tables C.1 and C.2 the continuous, and C.3 and C.4 provide the discrete parts of the 15 evolved DSP rules respectively. They are ranked based on their fitness values from the best (smallest) to worst (largest) corresponding to rule ids 1 to 15.

Table C.1: Rules from 1-7 of the 15 distinct evolved DSP rules and their fitness values after 10000 episodes. Their discrete parts can be found in Table C.3.

| **RuleID** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|---|
| $\eta$ | 0.0317 | 0.0754 | 0.0720 | 0.0470 | 0.0302 | 0.0152 | 0.0422 |
| $\theta$ | 0.2080 | 0.5574 | 0.3530 | 0.2763 | 0.1923 | 0.5547 | 0.5492 |
| $\alpha_h$ | 0.1931 | 0.1654 | 0.1523 | 0.2253 | 0.0985 | 0.0454 | 0.2291 |
| $\alpha_o$ | 0.2376 | 0.2255 | 0.6770 | 0.0214 | 0.0445 | 0.1633 | 0.0626 |
| **Fitness** | 44.10 | 45.65 | 45.83 | 46.98 | 51.65 | 54.28 | 54.35 |

Table C.2: Rules from 8-15 of the 15 distinct evolved DSP rules and their fitness values after 10000 episodes. Their discrete parts can be found in Table C.4.

| **RuleID** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|
| $\eta$ | 0.0927 | 0.0569 | 0.2396 | 0.2082 | 0.4536 | 0.0507 | 0.2315 | 0.9619 |
| $\theta$ | 0.1277 | 0.8238 | 0.0534 | 0.2351 | 0.3967 | 0.6040 | 0.3909 | 0.3672 |
| $\alpha_h$ | 0.1319 | 0.2833 | 0.0947 | 0.1272 | 0.4958 | 0.1334 | 0.1859 | 0.5027 |
| $\alpha_o$ | 0.4402 | 0.2538 | 0.0947 | 0.2613 | 0.1028 | 0.4862 | 0.2039 | 0.5758 |
| **Fitness** | 62.05 | 67.85 | 76.25 | 78.70 | 79.10 | 79.85 | 84.98 | 86.08 |

Table C.3: Rules from 1-7 of the 15 distinct evolved DSP rules given by the columns $\Delta w_1$ through $\Delta w_{15}$. Their continuous parts can be found in Table C.1. First four columns specify neuron activation traces where the first and second bits represent activations of pre- and post-synaptic neurons (i.e. 00 is when pre- and post-synaptic neurons are in a non-active state.), and the fifth column specify the modulatory signal $m$.

| $NAT^\theta$ | | | | $m$ | $\Delta w_1$ | $\Delta w_2$ | $\Delta w_3$ | $\Delta w_4$ | $\Delta w_5$ | $\Delta w_6$ | $\Delta w_7$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **00** | **01** | **10** | **11** | | | | | | | | |
| 0 | 0 | 0 | 0 | -1 | 1 | 1 | 0 | -1 | -1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | -1 | 0 | 1 |
| 0 | 0 | 0 | 1 | -1 | 0 | -1 | -1 | 0 | -1 | -1 | -1 |
| 0 | 0 | 0 | 1 | 1 | -1 | -1 | -1 | -1 | 0 | -1 | -1 |
| 0 | 0 | 1 | 0 | -1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | -1 | 0 | 0 | 0 | -1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | -1 | 1 | -1 | 0 | -1 | -1 | 1 |
| 0 | 1 | 0 | 0 | -1 | 0 | -1 | 0 | 1 | -1 | -1 | -1 |
| 0 | 1 | 0 | 0 | 1 | -1 | -1 | -1 | -1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | -1 | 0 | 0 | 0 | 1 | 0 | 1 | -1 |
| 0 | 1 | 0 | 1 | 1 | 1 | -1 | 1 | 1 | 1 | -1 | 1 |
| 0 | 1 | 1 | 0 | -1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | -1 | -1 | 1 | -1 | 0 | 1 |
| 0 | 1 | 1 | 1 | -1 | 0 | 1 | 1 | 1 | -1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | -1 | 0 | 1 | 1 | 0 | 0 | -1 |
| 1 | 0 | 0 | 0 | -1 | 0 | 1 | 1 | -1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | -1 | 0 | -1 | 0 | 1 | -1 | -1 | -1 |
| 1 | 0 | 0 | 1 | 1 | 1 | -1 | 0 | 1 | 0 | -1 | 1 |
| 1 | 0 | 1 | 0 | -1 | 0 | -1 | -1 | -1 | 0 | 0 | -1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | -1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | -1 | 0 | -1 | 0 | 1 | -1 | -1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | -1 | -1 | -1 | 1 | -1 | 1 |
| 1 | 1 | 0 | 0 | -1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | -1 | -1 | 0 | -1 |
| 1 | 1 | 0 | 1 | -1 | 0 | 1 | -1 | -1 | -1 | 0 | -1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | -1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | -1 | 0 | 0 | 0 | -1 | -1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | -1 | -1 | 0 | -1 | -1 | 0 |
| 1 | 1 | 1 | 1 | -1 | -1 | 0 | 0 | 1 | 1 | -1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | -1 | -1 | 0 | 0 | 1 |

Table C.4: Rules from 8-15 of the 15 distinct evolved DSP rules given by the columns $\Delta w_1$ through $\Delta w_{15}$. Their continuous parts can be found in Table C.2. First four columns specify neuron activation traces where the first and second bits represent activations of pre- and post-synaptic neurons (i.e. 00 is when pre- and post-synaptic neurons are in a non-active state.), and the fifth column specify the modulatory signal $m$.

| $NAT^{\theta}$ | | | | $m$ | $\Delta w_8$ | $\Delta w_9$ | $\Delta w_{10}$ | $\Delta w_{11}$ | $\Delta w_{12}$ | $\Delta w_{13}$ | $\Delta w_{14}$ | $\Delta w_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 10 | 11 | | | | | | | | | |
| 0 | 0 | 0 | 0 | -1 | -1 | 0 | 1 | 1 | 0 | 0 | 0 | -1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | -1 | 1 | -1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | -1 | -1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| 0 | 0 | 1 | 0 | -1 | -1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | -1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | -1 | 0 | 1 | 0 | -1 | -1 |
| 0 | 1 | 0 | 0 | -1 | 0 | 0 | -1 | -1 | 1 | -1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | -1 | -1 | 0 | 1 | -1 | 0 | 0 |
| 0 | 1 | 0 | 1 | -1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | -1 | 1 | 1 | 1 | 1 | -1 | 1 |
| 0 | 1 | 1 | 0 | -1 | -1 | 1 | 1 | 0 | 0 | 0 | 0 | -1 |
| 0 | 1 | 1 | 0 | 1 | -1 | 1 | 1 | 1 | -1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | -1 | -1 | 1 | -1 | -1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | -1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | -1 | 1 | -1 | -1 | -1 | -1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | -1 | -1 | 1 | -1 | -1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | -1 | 0 | 0 | -1 | -1 | -1 |
| 1 | 0 | 1 | 0 | -1 | 1 | 0 | -1 | 0 | -1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | -1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | -1 | -1 | 1 | 0 | 1 | 1 | 0 | 1 | -1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | -1 | -1 |
| 1 | 1 | 0 | 0 | -1 | 0 | -1 | -1 | -1 | -1 | 0 | -1 | -1 |
| 1 | 1 | 0 | 0 | 1 | 1 | -1 | -1 | 0 | -1 | -1 | 0 | 1 |
| 1 | 1 | 0 | 1 | -1 | 1 | 0 | 1 | -1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | -1 | 0 | 0 | 0 | -1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | -1 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | -1 |
| 1 | 1 | 1 | 0 | 1 | -1 | 0 | 1 | 1 | -1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | -1 | -1 | -1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |

# Curriculum Vitae

Anil Yaman was born in Tekirdağ, Turkey on the $16^{th}$ of May 1988. He obtained his B.S. degree from the Karadeniz Technical University in 2010. In October 2010, he moved to New York, United States. He received his master's degree in computer science from the City College of the City University of New York in 2014. During his last year of master's degree, he joined IBM Zurich lab for an internship.

From 2013 to 2015, Anil held a research position at the Department of Biomedical Informatics in Columbia University. In October 2015, he joined to the INCAS$^3$, an independent research institute founded in Assen, the Netherlands to participate in a Horizon2020-FETOPEN project entitled as "PHOENIX: Exploring the Unknown through Reincarnation and Coevolution". He simultaneously started his PhD Research in Mathetmatics and Computer Science Department at the Eindhoven University of Technology.

During his PhD, Anil has actively involved in educational and outreach activities. He has given several invited lectures for PDEng students at the Jheronimus Academy of Data Science and participated in coaching bachelor students. He also given public talks on Artificial Intelligence in various events in Eindhoven. An interview regarding his work "Learning with delayed synaptic plasticity" was published on TechXplore, a network that publishes news about technological advances.

# List of Publications

Anil Yaman has the following publications:

## Journals Publications

1. Onur Çaylak, Anil Yaman, and Björn Baumeier. Evolutionary approach to constructing a deep feedforward neural network for prediction of electronic coupling elements in molecular materials. *Journal of Chemical Theory and Computation*, 15(3):1777–1784, 2019. PMID: 30753071

2. Anando Sen, Andrew Goldstein, Shreya Chakrabarti, Ning Shang, Tian Kang, Anil Yaman, Patrick B Ryan, and Chunhua Weng. The representativeness of eligible patients in type 2 diabetes trials: a case study using gist 2.0. *Journal of the American Medical Informatics Association*, 25(3):239–247, 2017

3. Anil Yaman, Stephen Lucci, and Izidor Gertner. Evolutionary algorithm based approach for modeling autonomously trading agents. *Intelligent Information Management*, 6(02):45, 2014

## Journals Publications (under review / in preparation)

4. Anil Yaman, Giovanni Iacca, Decebal Mocanu, George Fletcher, and Mykola Pechenizkiy. Novelty producing synaptic plasticity. *(in Preparation)*

5. Anil Yaman, Decebal Constantin Mocanu, Giovanni Iacca, Matt Coler, George Fletcher, and Mykola Pechenizkiy. Evolving plasticity for autonomous learning under changing environmental conditions. *(Under review)*, 2019

## Conference Proceedings

6. Anil Yaman, Giovanni Iacca, Decebal Constantin Mocanu, George Fletcher, and Mykola Pechenizkiy. Learning with delayed synaptic plasticity. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, pages 152–160, New York, NY, USA, 2019. ACM

7. Anil Yaman, Decebal Constantin Mocanu, Giovanni Iacca, George Fletcher, and Mykola Pechenizkiy. Limited evaluation cooperative co-evolutionary differential evolution for large-scale neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 569–576, New York, NY, USA, 2018. ACM

8. Anil Yaman, Giovanni Iacca, and Fabio Caraffini. A comparison of three differential evolution strategies in terms of early convergence with different population sizes. In *LeGO 2018 - International Workshop on Global Optimization, 18-21 September, Leiden, The Netherlands*, 2018

9. Fabio Caraffini, Giovanni Iacca, and Anil Yaman. Improving (1+ 1) covariance matrix adaptation evolution strategy: a simple yet efficient approach. In *LeGO 2018 - International Workshop on Global Optimization, 18-21 September, Leiden, The Netherlands*, 2018

10. Anil Yaman, Giovanni Iacca, Matt Coler, George Fletcher, and Mykola Pechenizkiy. Multi-strategy differential evolution. In Kevin Sim and Paul Kaufmann, editors, *Applications of Evolutionary Computation*, pages 617–633, Cham, 2018. Springer International Publishing

11. Anil Yaman, Ahmed Hallawa, Matt Coler, and Giovanni Iacca. Presenting the eco: Evolutionary computation ontology. In Giovanni Squillero and Kevin Sim, editors, *Applications of Evolutionary Computation*, pages 603–619, Cham, 2017. Springer International Publishing

12. Ahmed Hallawa, Anil Yaman, Giovanni Iacca, and Gerd Ascheid. A framework for knowledge integrated evolutionary algorithms. In Giovanni Squillero and Kevin Sim, editors, *Applications of Evolutionary Computation*, pages 653–669, Cham, 2017. Springer International Publishing

13. Anil Yaman, Shreya Chakrabarti, Anando Sen, and Chunhua Weng. How have cancer clinical trial eligibility criteria evolved over time? *AMIA Summits on Translational Science Proceedings*, 2016:269, 2016

14. Praveen Chandar, Anil Yaman, Julia Hoxha, Zhe He, and Chunhua Weng. Similarity-based recommendation of new concepts to a terminology. In *AMIA Annual Symposium Proceedings*, volume 2015, page 386. American Medical Informatics Association, 2015

15. Chunhua Weng, Anil Yaman, Kuo Lin, and Zhe He. Trend and network analysis of common eligibility features for cancer trials in clinicaltrials. gov. In *International Conference on Smart Health*, pages 130–141. Springer, 2014

# SIKS dissertations

## 2011

**2011-1** Botond Cseke (RUN), *Variational Algorithms for Bayesian Inference in Latent Gaussian Models.*

**2011-2** Nick Tinnemeier (UU), *Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language.*

**2011-3** Jan Martijn van der Werf (TUE), *Compositional Design and Verification of Component-Based Information Systems.*

**2011-4** Hado van Hasselt (UU), *Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference.*

**2011-5** Bas van der Raadt (VU), *Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.*

**2011-6** Yiwen Wang (TUE), *Semantically-Enhanced Recommendations in Cultural Heritage.*

**2011-7** Yujia Cao (UT), *Multimodal Information Presentation for High Load Human Computer Interaction.*

**2011-8** Nieske Vergunst (UU), *BDI-based Generation of Robust Task-Oriented Dialogues.*

**2011-9** Tim de Jong (OU), *Contextualised Mobile Media for Learning.*

**2011-10** Bart Bogaert (UvT), *Cloud Content Contention.*

**2011-11** Dhaval Vyas (UT), *Designing for Awareness: An Experience-focused HCI Perspective.*

**2011-12** Carmen Bratosin (TUE), *Grid Architecture for Distributed Process Mining.*

**2011-13** Xiaoyu Mao (UvT), *Airport under Control. Multiagent Scheduling for Airport Ground Handling.*

**2011-14** Milan Lovric (EUR), *Behavioral Finance and Agent-Based Artificial Markets.*

**2011-15** Marijn Koolen (UvA), *The Meaning of Structure: the Value of Link Evidence for Information Retrieval.*

**2011-16** Maarten Schadd (UM), *Selective Search in Games of Different Complexity.*

**2011-17** Jiyin He (UVA), *Exploring Topic Structure: Coherence, Diversity and Relatedness.*

**2011-18** Mark Ponsen (UM), *Strategic Decision-Making in complex games.*

**2011-19** Ellen Rusman (OU), *The Mind's Eye on Personal Profiles.*

**2011-20** Qing Gu (VU), *Guiding service-oriented software engineering - A view-based approach.*

**2011-21** Linda Terlouw (TUD), *Modularization and Specification of Service-Oriented Systems.*

**2011-22** Junte Zhang (UVA), *System Evaluation of Archival Description and Access.*

**2011-23** Wouter Weerkamp (UVA), *Finding People and their Utterances in Social Media.*

**2011-24** Herwin van Welbergen (UT), *Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior.*

**2011-25** Syed Waqar ul Qounain Jaffry (VU), *Analysis and Validation of Models for Trust Dynamics.*

**2011-26** Matthijs Aart Pontier (VU), *Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots.*

**2011-27** Aniel Bhulai (VU), *Dynamic website optimization through autonomous management of design patterns.*

**2011-28** Rianne Kaptein (UVA), *Effective Focused Retrieval by Exploiting Query Context and Document Structure.*

**2011-29** Faisal Kamiran (TUE), *Discrimination-aware Classification.*

**2011-30** Egon van den Broek (UT), *Affective Signal Processing (ASP): Unraveling the mystery of emotions.*

**2011-31** Ludo Waltman (EUR), *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality.*

**2011-32** Nees-Jan van Eck (EUR), *Methodological Advances in Bibliometric Mapping of Science.*

**2011-33** Tom van der Weide (UU), *Arguing to Motivate Decisions.*

**2011-34** Paolo Turrini (UU), *Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations.*

**2011-35** Maaike Harbers (UU), *Explaining Agent Behavior in Virtual Training.*

**2011-36** Erik van der Spek (UU), *Experiments in serious game design: a cognitive approach.*

**2011-37** Adriana Burlutiu (RUN), *Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference.*

**2011-38** Nyree Lemmens (UM), *Bee-inspired Distributed Optimization.*

**2011-39** Joost Westra (UU), *Organizing Adaptation using Agents in Serious Games.*

**2011-40** Viktor Clerc (VU), *Architectural Knowledge Management in Global Software Development.*

**2011-41** Luan Ibraimi (UT), *Cryptographically Enforced Distributed Data Access Control.*

**2011-42** Michal Sindlar (UU), *Explaining Behavior through Mental State Attribution.*

**2011-43** Henk van der Schuur (UU), *Process Improvement through Software Operation Knowledge.*
**2011-44** Boris Reuderink (UT), *Robust Brain-Computer Interfaces.*
**2011-45** Herman Stehouwer (UvT), *Statistical Language Models for Alternative Sequence Selection.*
**2011-46** Beibei Hu (TUD), *Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work.*
**2011-47** Azizi Bin Ab Aziz (VU), *Exploring Computational Models for Intelligent Support of Persons with Depression.*
**2011-48** Mark Ter Maat (UT), *Response Selection and Turn-taking for a Sensitive Artificial Listening Agent.*
**2011-49** Andreea Niculescu (UT), *Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality.*

## 2012

**2012-1** Terry Kakeeto (UvT), *Relationship Marketing for SMEs in Uganda.*
**2012-2** Muhammad Umair (VU), *Adaptivity, emotion, and Rationality in Human and Ambient Agent Models.*
**2012-3** Adam Vanya (VU), *Supporting Architecture Evolution by Mining Software Repositories.*
**2012-4** Jurriaan Souer (UU), *Development of Content Management System-based Web Applications.*
**2012-5** Marijn Plomp (UU), *Maturing Interorganisational Information Systems.*
**2012-6** Wolfgang Reinhardt (OU), *Awareness Support for Knowledge Workers in Research Networks.*
**2012-7** Rianne van Lambalgen (VU), *When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions.*
**2012-8** Gerben de Vries (UVA), *Kernel Methods for Vessel Trajectories.*
**2012-9** Ricardo Neisse (UT), *Trust and Privacy Management Support for Context-Aware Service Platforms.*
**2012-10** David Smits (TUE), *Towards a Generic Distributed Adaptive Hypermedia Environment.*
**2012-11** J.C.B. Rantham Prabhakara (TUE), *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics.*
**2012-12** Kees van der Sluijs (TUE), *Model Driven Design and Data Integration in Semantic Web Information Systems.*
**2012-13** Suleman Shahid (UvT), *Fun and Face: Exploring non-verbal expressions of emotion during playful interactions.*
**2012-14** Evgeny Knutov (TUE), *Generic Adaptation Framework for Unifying Adaptive Web-based Systems.*
**2012-15** Natalie van der Wal (VU), *Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.*
**2012-16** Fiemke Both (VU), *Helping people by understanding them - Am-*

*bient Agents supporting task execution and depression treatment.*

**2012-17** Amal Elgammal (UvT), *Towards a Comprehensive Framework for Business Process Compliance.*

**2012-18** Eltjo Poort (VU), *Improving Solution Architecting Practices.*

**2012-19** Helen Schonenberg (TUE), *What's Next? Operational Support for Business Process Execution.*

**2012-20** Ali Bahramisharif (RUN), *Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing.*

**2012-21** Roberto Cornacchia (TUD), *Querying Sparse Matrices for Information Retrieval.*

**2012-22** Thijs Vis (UvT), *Intelligence, politie en veiligheidsdienst: verenigbare grootheden?.*

**2012-23** Christian Muehl (UT), *Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction.*

**2012-24** Laurens van der Werff (UT), *Evaluation of Noisy Transcripts for Spoken Document Retrieval.*

**2012-25** Silja Eckartz (UT), *Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application.*

**2012-26** Emile de Maat (UVA), *Making Sense of Legal Text.*

**2012-27** Hayrettin Gurkok (UT), *Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games.*

**2012-28** Nancy Pascall (UvT), *Engendering Technology Empowering Women.*

**2012-29** Almer Tigelaar (UT), *Peer-to-Peer Information Retrieval.*

**2012-30** Alina Pommeranz (TUD), *Designing Human-Centered Systems for Reflective Decision Making.*

**2012-31** Emily Bagarukayo (RUN), *A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure.*

**2012-32** Wietske Visser (TUD), *Qualitative multi-criteria preference representation and reasoning.*

**2012-33** Rory Sie (OUN), *Coalitions in Cooperation Networks (COCOON).*

**2012-34** Pavol Jancura (RUN), *Evolutionary analysis in PPI networks and applications.*

**2012-35** Evert Haasdijk (VU), *Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics.*

**2012-36** Denis Ssebugwawo (RUN), *Analysis and Evaluation of Collaborative Modeling Processes.*

**2012-37** Agnes Nakakawa (RUN), *A Collaboration Process for Enterprise Architecture Creation.*

**2012-38** Selmar Smit (VU), *Parameter Tuning and Scientific Testing in Evolutionary Algorithms.*

**2012-39** Hassan Fatemi (UT), *Risk-aware design of value and coordination networks.*

**2012-40** Agus Gunawan (UvT), *Information Access for SMEs in Indonesia.*

**2012-41** Sebastian Kelle (OU), *Game Design Patterns for Learning.*

**2012-42** Dominique Verpoorten

(OU), *Reflection Amplifiers in self-regulated Learning.*
**2012-44** Anna Tordai (VU), *On Combining Alignment Techniques.*
**2012-45** Benedikt Kratz (UvT), *A Model and Language for Business-aware Transactions.*
**2012-46** Simon Carter (UVA), *Exploration and Exploitation of Multilingual Data for Statistical Machine Translation.*
**2012-47** Manos Tsagkias (UVA), *Mining Social Media: Tracking Content and Predicting Behavior.*
**2012-48** Jorn Bakker (TUE), *Handling Abrupt Changes in Evolving Time-series Data.*
**2012-49** Michael Kaisers (UM), *Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions.*
**2012-50** Steven van Kervel (TUD), *Ontologogy driven Enterprise Information Systems Engineering.*
**2012-51** Jeroen de Jong (TUD), *Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching.*

## 2013

**2013-1** Viorel Milea (EUR), *News Analytics for Financial Decision Support.*
**2013-2** Erietta Liarou (CWI), *MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing.*
**2013-3** Szymon Klarman (VU), *Reasoning with Contexts in Description Logics.*
**2013-4** Chetan Yadati (TUD), *Coordinating autonomous planning and scheduling.*
**2013-5** Dulce Pumareja (UT), *Groupware Requirements Evolutions Patterns.*
**2013-6** Romulo Goncalves (CWI), *The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience.*
**2013-7** Giel van Lankveld (UvT), *Quantifying Individual Player Differences.*
**2013-8** Robbert-Jan Merk (VU), *Making enemies: cognitive modeling for opponent agents in fighter pilot simulators.*
**2013-9** Fabio Gori (RUN), *Metagenomic Data Analysis: Computational Methods and Applications.*
**2013-10** Jeewanie Jayasinghe Arachchige (UvT), *A Unified Modeling Framework for Service Design.*
**2013-11** Evangelos Pournaras (TUD), *Multi-level Reconfigurable Self-organization in Overlay Services.*
**2013-12** Marian Razavian (VU), *Knowledge-driven Migration to Services.*
**2013-13** Mohammad Safiri (UT), *Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly.*
**2013-14** Jafar Tanha (UVA), *Ensemble Approaches to Semi-Supervised Learning Learning.*
**2013-15** Daniel Hennes (UM), *Multi-agent Learning - Dynamic Games and Applications.*

**2013-16** Eric Kok (UU), *Exploring the practical benefits of argumentation in multi-agent deliberation.*

**2013-17** Koen Kok (VU), *The PowerMatcher: Smart Coordination for the Smart Electricity Grid.*

**2013-18** Jeroen Janssens (UvT), *Outlier Selection and One-Class Classification.*

**2013-19** Renze Steenhuizen (TUD), *Coordinated Multi-Agent Planning and Scheduling.*

**2013-20** Katja Hofmann (UvA), *Fast and Reliable Online Learning to Rank for Information Retrieval.*

**2013-21** Sander Wubben (UvT), *Text-to-text generation by monolingual machine translation.*

**2013-22** Tom Claassen (RUN), *Causal Discovery and Logic.*

**2013-23** Patricio de Alencar Silva (UvT), *Value Activity Monitoring.*

**2013-24** Haitham Bou Ammar (UM), *Automated Transfer in Reinforcement Learning.*

**2013-25** Agnieszka Anna Latoszek-Berendsen (UM), *Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System.*

**2013-26** Alireza Zarghami (UT), *Architectural Support for Dynamic Homecare Service Provisioning.*

**2013-27** Mohammad Huq (UT), *Inference-based Framework Managing Data Provenance.*

**2013-28** Frans van der Sluis (UT), *When Complexity becomes Interesting: An Inquiry into the Information eXperience.*

**2013-29** Iwan de Kok (UT), *Listening Heads.*

**2013-30** Joyce Nakatumba (TUE), *Resource-Aware Business Process Management: Analysis and Support.*

**2013-31** Dinh Khoa Nguyen (UvT), *Blueprint Model and Language for Engineering Cloud Applications.*

**2013-32** Kamakshi Rajagopal (OUN), *Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development.*

**2013-33** Qi Gao (TUD), *User Modeling and Personalization in the Microblogging Sphere.*

**2013-34** Kien Tjin-Kam-Jet (UT), *Distributed Deep Web Search.*

**2013-35** Abdallah El Ali (UvA), *Minimal Mobile Human Computer Interaction.*

**2013-36** Than Lam Hoang (TUe), *Pattern Mining in Data Streams.*

**2013-37** Dirk B¨orner (OUN), *Ambient Learning Displays.*

**2013-38** Eelco den Heijer (VU), *Autonomous Evolutionary Art.*

**2013-39** Joop de Jong (TUD), *A Method for Enterprise Ontology based Design of Enterprise Information Systems.*

**2013-40** Pim Nijssen (UM), *Monte-Carlo Tree Search for Multi-Player Games.*

**2013-41** Jochem Liem (UVA), *Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning.*

**2013-42** Léon Planken (TUD), *Algorithms for Simple Temporal Reasoning.*
**2013-43** Marc Bron (UVA), *Exploration and Contextualization through Interaction and Concepts.*

## 2014

**2014-1** Nicola Barile (UU), *Studies in Learning Monotone Models from Data.*
**2014-2** Fiona Tuliyano (RUN), *Combining System Dynamics with a Domain Modeling Method.*
**2014-3** Sergio Raul Duarte Torres (UT), *Information Retrieval for Children: Search Behavior and Solutions.*
**2014-4** Hanna Jochmann-Mannak (UT), *Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation.*
**2014-5** Jurriaan van Reijsen (UU), *Knowledge Perspectives on Advancing Dynamic Capability.*
**2014-6** Damian Tamburri (VU), *Supporting Networked Software Development.*
**2014-7** Arya Adriansyah (TUE), *Aligning Observed and Modeled Behavior.*
**2014-8** Samur Araujo (TUD), *Data Integration over Distributed and Heterogeneous Data Endpoints.*
**2014-9** Philip Jackson (UvT), *Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language.*
**2014-10** Ivan Salvador Razo Zapata (VU), *Service Value Networks.*
**2014-11** Janneke van der Zwaan (TUD), *An Empathic Virtual Buddy for Social Support.*
**2014-12** Willem van Willigen (VU), *Look Ma, No Hands: Aspects of Autonomous Vehicle Control.*
**2014-13** Arlette van Wissen (VU), *Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains.*
**2014-14** Yangyang Shi (TUD), *Language Models With Meta-information.*
**2014-15** Natalya Mogles (VU), *Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare.*
**2014-16** Krystyna Milian (VU), *Supporting trial recruitment and design by automatically interpreting eligibility criteria.*
**2014-17** Kathrin Dentler (VU), *Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability.*
**2014-18** Mattijs Ghijsen (UVA), *Methods and Models for the Design and Study of Dynamic Agent Organizations.*
**2014-19** Vinicius Ramos (TUE), *Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support.*
**2014-20** Mena Habib (UT), *Named Entity Extraction and Disambiguation for Informal Text: The Missing Link.*
**2014-21** Kassidy Clark (TUD), *Negotiation and Monitoring in Open Environ-*

*ments.*

**2014-22** Marieke Peeters (UU), *Personalized Educational Games - Developing agent-supported scenario-based training.*

**2014-23** Eleftherios Sidirourgos (UvA/CWI), *Space Efficient Indexes for the Big Data Era.*

**2014-24** Davide Ceolin (VU), *Trusting Semi-structured Web Data.*

**2014-25** Martijn Lappenschaar (RUN), *New network models for the analysis of disease interaction.*

**2014-26** Tim Baarslag (TUD), *What to Bid and When to Stop.*

**2014-27** Rui Jorge Almeida (EUR), *Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty.*

**2014-28** Anna Chmielowiec (VU), *Decentralized k-Clique Matching.*

**2014-29** Jaap Kabbedijk (UU), *Variability in Multi-Tenant Enterprise Software.*

**2014-30** Peter de Cock (UvT), *Anticipating Criminal Behaviour.*

**2014-31** Leo van Moergestel (UU), *Agent Technology in Agile Multiparallel Manufacturing and Product Support.*

**2014-32** Naser Ayat (UvA), *On Entity Resolution in Probabilistic Data.*

**2014-33** Tesfa Tegegne (RUN), *Service Discovery in eHealth.*

**2014-34** Christina Manteli (VU), *The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.*

**2014-35** Joost van Ooijen (UU), *Cognitive Agents in Virtual Worlds: A Middleware Design Approach.*

**2014-36** Joos Buijs (TUE), *Flexible Evolutionary Algorithms for Mining Structured Process Models.*

**2014-37** Maral Dadvar (UT), *Experts and Machines United Against Cyberbullying.*

**2014-38** Danny Plass-Oude Bos (UT), *Making brain-computer interfaces better: improving usability through post-processing.*

**2014-39** Jasmina Maric (UvT), *Web Communities, Immigration, and Social Capital.*

**2014-40** Walter Omona (RUN), *A Framework for Knowledge Management Using ICT in Higher Education.*

**2014-41** Frederic Hogenboom (EUR), *Automated Detection of Financial Events in News Text.*

**2014-42** Carsten Eijckhof (CWI/-TUD), *Contextual Multidimensional Relevance Models.*

**2014-43** Kevin Vlaanderen (UU), *Supporting Process Improvement using Method Increments.*

**2014-44** Paulien Meesters (UvT), *Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.*

**2014-45** Birgit Schmitz (OUN), *Mobile Games for Learning: A Pattern-Based Approach.*

**2014-46** Ke Tao (TUD), *Social Web Data Analytics: Relevance, Redundancy, Diversity.*

**2014-47** Shangsong Liang (UVA), *Fusion and Diversification in Information Retrieval.*

## 2015

**2015-1** Niels Netten (UvA), *Machine Learning for Relevance of Information in Crisis Response.*

**2015-2** Faiza Bukhsh (UvT), *Smart auditing: Innovative Compliance Checking in Customs Controls.*

**2015-3** Twan van Laarhoven (RUN), *Machine learning for network data.*

**2015-4** Howard Spoelstra (OUN), *Collaborations in Open Learning Environments.*

**2015-5** Christoph B¨osch (UT), *Cryptographically Enforced Search Pattern Hiding.*

**2015-6** Farideh Heidari (TUD), *Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes.*

**2015-7** Maria-Hendrike Peetz (UvA), *Time-Aware Online Reputation Analysis.*

**2015-8** Jie Jiang (TUD), *Organizational Compliance: An agent-based model for designing and evaluating organizational interactions.*

**2015-9** Randy Klaassen (UT), *HCI Perspectives on Behavior Change Support Systems.*

**2015-10** Henry Hermans (OUN), *OpenU: design of an integrated system to support lifelong learning.*

**2015-11** Yongming Luo (TUE), *Designing algorithms for big graph datasets: A study of computing bisimulation and joins.*

**2015-12** Julie M. Birkholz (VU), *Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks.*

**2015-13** Giuseppe Procaccianti (VU), *Energy-Efficient Software.*

**2015-14** Bart van Straalen (UT), *A cognitive approach to modeling bad news conversations.*

**2015-15** Klaas Andries de Graaf (VU), *Ontology-based Software Architecture Documentation.*

**2015-16** Changyun Wei (UT), *Cognitive Coordination for Cooperative Multi-Robot Teamwork.*

**2015-17** André van Cleeff (UT), *Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs.*

**2015-18** Holger Pirk (CWI), *Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories.*

**2015-19** Bernardo Tabuenca (OUN), *Ubiquitous Technology for Lifelong Learners.*

**2015-20** Lois Vanhée (UU), *Using Culture and Values to Support Flexible Coordination.*

**2015-21** Sibren Fetter (OUN), *Using Peer-Support to Expand and Stabilize Online Learning.*

**2015-22** Zhemin Zhu (UT), *Co-occurrence Rate Networks.*

**2015-23** Luit Gazendam (VU), *Cataloguer Support in Cultural Heritage.*

**2015-24** Richard Berendsen (UVA), *Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation.*

**2015-25** Steven Woudenberg (UU), *Bayesian Tools for Early Disease Detection.*

**2015-26** Alexander Hogenboom (EUR), *Sentiment Analysis of Text Guided by Semantics and Structure.*

**2015-27** Sándor Héman (CWI), *Updating compressed colomn stores.*

**2015-28** Janet Bagorogoza (TiU), *Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO.*

**2015-29** Hendrik Baier (UM), *Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains.*

**2015-30** Kiavash Bahreini (OU), *Real-time Multimodal Emotion Recognition in E-Learning.*

**2015-31** Yakup Koç (TUD), *On the robustness of Power Grids.*

**2015-32** Jerome Gard (UL), *Corporate Venture Management in SMEs.*

**2015-33** Frederik Schadd (TUD), *Ontology Mapping with Auxiliary Resources.*

**2015-34** Victor de Graaf (UT), *Gesocial Recommender Systems.*

**2015-35** Jungxao Xu (TUD), *Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction.*

## 2016

**2016-1** Syed Saiden Abbas (RUN), *Recognition of Shapes by Humans and Machines.*

**2016-2** Michiel Christiaan Meulendijk (UU), *Optimizing medication reviews through decision support: prescribing a better pill to swallow.*

**2016-3** Maya Sappelli (RUN), *Knowledge Work in Context: User Centered Knowledge Worker Support.*

**2016-4** Laurens Rietveld (VU), *Publishing and Consuming Linked Data.*

**2016-5** Evgeny Sherkhonov (UVA), *Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers.*

**2016-6** Michel Wilson (TUD), *Robust scheduling in an uncertain environment.*

**2016-7** Jeroen de Man (VU), *Measuring and modeling negative emotions for virtual training.*

**2016-8** Matje van de Camp (TiU), *A Link to the Past: Constructing Historical Social Networks from Unstructured Data.*

**2016-9** Archana Nottamkandath (VU), *Trusting Crowdsourced Information on Cultural Artefacts.*

**2016-10** George Karafotias (VUA), *Parameter Control for Evolutionary Algorithms.*

**2016-11** Anne Schuth (UVA), *Search Engines that Learn from Their Users.*

**2016-12** Max Knobbout (UU), *Logics for Modelling and Verifying Normative Multi-Agent Systems.*

**2016-13** Nana Baah Gyan (VU), *The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach.*

**2016-14** Ravi Khadka (UU), *Revisiting*

*Legacy Software System Modernization.*

**2016-15** Steffen Michels (RUN), *Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments.*

**2016-16** Guangliang Li (UVA), *Socially Intelligent Autonomous Agents that Learn from Human Reward.*

**2016-17** Berend Weel (VU), *Towards Embodied Evolution of Robot Organisms.*

**2016-18** Albert Meroño Peñuela (VU), *Refining Statistical Data on the Web.*

**2016-19** Julia Efremova (Tu/e), *Mining Social Structures from Genealogical Data.*

**2016-20** Daan Odijk (UVA), *Context & Semantics in News & Web Search.*

**2016-21** Alejandro Moreno Célleri (UT), *From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground.*

**2016-22** Grace Lewis (VU), *Software Architecture Strategies for Cyber-Foraging Systems.*

**2016-23** Fei Cai (UVA), *Query Auto Completion in Information Retrieval.*

**2016-24** Brend Wanders (UT), *Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach.*

**2016-25** Julia Kiseleva (TU/e), *Using Contextual Information to Understand Searching and Browsing Behavior.*

**2016-26** Dilhan Thilakarathne (VU), *In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains.*

**2016-27** Wen Li (TUD), *Understanding Geo-spatial Information on Social Media.*

**2016-28** Mingxin Zhang (TUD), *Large-scale Agent-based Social Simulation - A study on epidemic prediction and control.*

**2016-29** Nicolas H¨oning (TUD), *Peak reduction in decentralised electricity systems - Markets and prices for flexible planning.*

**2016-30** Ruud Mattheij (UvT), *The Eyes Have It.*

**2016-31** Mohammad Khelghati (UT), *Deep web content monitoring.*

**2016-32** Eelco Vriezekolk (UT), *Assessing Telecommunication Service Availability Risks for Crisis Organisations.*

**2016-33** Peter Bloem (UVA), *Single Sample Statistics, exercises in learning from just one example.*

**2016-34** Dennis Schunselaar (TUE), *Configurable Process Trees: Elicitation, Analysis, and Enactment.*

**2016-35** Zhaochun Ren (UVA), *Monitoring Social Media: Summarization, Classification and Recommendation.*

**2016-36** Daphne Karreman (UT), *Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies.*

**2016-37** Giovanni Sileno (UvA), *Aligning Law and Action - a conceptual and computational inquiry.*

**2016-38** Andrea Minuto (UT), *Mate-*

*rials that Matter - Smart Materials meet Art & Interaction Design.*

**2016-39** Merijn Bruijnes (UT), *Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect.*

**2016-40** Christian Detweiler (TUD), *Accounting for Values in Design.*

**2016-41** Thomas King (TUD), *Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance.*

**2016-42** Spyros Martzoukos (UVA), *Combinatorial and Compositional Aspects of Bilingual Aligned Corpora.*

**2016-43** Saskia Koldijk (RUN), *Context-Aware Support for Stress Self-Management: From Theory to Practice.*

**2016-44** Thibault Sellam (UVA), *Automatic Assistants for Database Exploration.*

**2016-45** Bram van de Laar (UT), *Experiencing Brain-Computer Interface Control.*

**2016-46** Jorge Gallego Perez (UT), *Robots to Make you Happy.*

**2016-47** Christina Weber (UL), *Real-time foresight - Preparedness for dynamic innovation networks.*

**2016-48** Tanja Buttler (TUD), *Collecting Lessons Learned.*

**2016-49** Gleb Polevoy (TUD), *Participation and Interaction in Projects. A Game-Theoretic Analysis.*

**2016-50** Yan Wang (UVT), *The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains.*

# 2017

**2017-1** Jan-Jaap Oerlemans (UL), *Investigating Cybercrime.*

**2017-2** Sjoerd Timmer (UU), *Designing and Understanding Forensic Bayesian Networks using Argumentation.*

**2017-3** Dani¨el Harold Telgen (UU), *Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines.*

**2017-4** Mrunal Gawade (CWI), *Multi-core Parallelism in a Column-store.*

**2017-5** Mahdieh Shadi (UVA), *Collaboration Behavior.*

**2017-6** Damir Vandic (EUR), *Intelligent Information Systems for Web Product Search.*

**2017-7** Roel Bertens (UU), *Insight in Information: from Abstract to Anomaly.*

**2017-8** Rob Konijn (VU), *Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery.*

**2017-9** Dong Nguyen (UT), *Text as Social and Cultural Data: A Computational Perspective on Variation in Text.*

**2017-10** Robby van Delden (UT), *(Steering) Interactive Play Behavior.*

**2017-11** Florian Kunneman (RUN), *Modelling patterns of time and emotion in Twitter #anticipointment.*

**2017-12** Sander Leemans (TUE), *Robust Process Mining with Guarantees.*

**2017-13** Gijs Huisman (UT), *Social Touch Technology - Extending the reach of social touch through haptic technol-*

*ogy.*

**2017-14** Shoshannah Tekofsky (UvT), *You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior.*

**2017-15** Peter Berck (RUN), *Memory-Based Text Correction.*

**2017-16** Aleksandr Chuklin (UVA), *Understanding and Modeling Users of Modern Search Engines.*

**2017-17** Daniel Dimov (UL), *Crowdsourced Online Dispute Resolution.*

**2017-18** Ridho Reinanda (UVA), *Entity Associations for Search.*

**2017-19** Jeroen Vuurens (UT), *Proximity of Terms, Texts and Semantic Vectors in Information Retrieval.*

**2017-20** Mohammadbashir Sedighi (TUD), *Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility.*

**2017-21** Jeroen Linssen (UT), *Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds).*

**2017-22** Sara Magliacane (VU), *Logics for causal inference under uncertainty.*

**2017-23** David Graus (UVA), *Entities of Interest — Discovery in Digital Traces.*

**2017-24** Chang Wang (TUD), *Use of Affordances for Efficient Robot Learning.*

**2017-25** Veruska Zamborlini (VU), *Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search.*

**2017-26** Merel Jung (UT), *Socially intelligent robots that understand and respond to human touch.*

**2017-27** Michiel Joosse (UT), *Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors.*

**2017-28** John Klein (VU), *Architecture Practices for Complex Contexts.*

**2017-29** Adel Alhuraibi (UvT), *From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT.*

**2017-30** Wilma Latuny (UvT), *The Power of Facial Expressions.*

**2017-31** Ben Ruijl (UL), *Advances in computational methods for QFT calculations.*

**2017-32** Thaer Samar (RUN), *Access to and Retrievability of Content in Web Archives.*

**2017-33** Brigit van Loggem (OU), *Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity.*

**2017-34** Maren Scheffel (OU), *The Evaluation Framework for Learning Analytics.*

**2017-35** Martine de Vos (VU), *Interpreting natural science spreadsheets.*

**2017-36** Yuanhao Guo (UL), *Shape Analysis for Phenotype Characterisation from High-throughput Imaging.*

**2017-37** Alejandro Montes Garcia (TUE), *WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy.*

**2017-38** Alex Kayal (TUD), *Normative Social Applications.*

**2017-39** Sara Ahmadi (RUN), *Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR.*
**2017-40** Altaf Hussain Abro (VUA), *Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems.*
**2017-41** Adnan Manzoor (VUA), *Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle.*
**2017-42** Elena Sokolova (RUN), *Causal discovery from mixed and missing data with applications on ADHD datasets.*
**2017-43** Maaike de Boer (RUN), *Semantic Mapping in Video Retrieval.*
**2017-44** Garm Lucassen (UU), *Understanding User Stories - Computational Linguistics in Agile Requirements Engineering.*
**2017-45** Bas Testerink(UU), *Decentralized Runtime Norm Enforcement.*
**2017-46** Jan Schneider(OU), *Sensor-based Learning Support.*
**2017-47** Jie Yang (TUD), *Crowd Knowledge Creation Acceleration.*
**2017-48** Angel Suarez (OU), *Collaborative inquiry-based learning.*

## 2018

**2018-1** Han van der Aa (VUA), *Comparing and Aligning Process Representations.*
**2018-2** Felix Mannhardt (TUE), *Multi-perspective Process Mining.*
**2018-3** Steven Bosems (UT), *Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction.*
**2018-4** Jordan Janeiro (TUD), *Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks.*
**2018-5** Hugo Huurdeman (UVA), *Supporting the Complex Dynamics of the Information Seeking Process.*
**2018-6** Dan Ionita (UT), *Model-Driven Information Security Risk Assessment of Socio-Technical Systems.*
**2018-7** Jieting Luo (UU), *A formal account of opportunism in multi-agent systems.*
**2018-8** Rick Smetsers (RUN), *Advances in Model Learning for Software Systems.*
**2018-9** Xu Xie(TUD), *Data Assimilation in Discrete Event Simulations.*
**2018-10** Julienka Mollee (VUA), *Moving forward: supporting physical activity behavior change through intelligent technology.*
**2018-11** Mahdi Sargolzaei (UVA), *Enabling Framework for Service-oriented Collaborative Networks.*
**2018-12** Xixi Lu (TUE), *Using behavioral context in process mining.*
**2018-13** Seyed Amin Tabatabaei (VUA), *Computing a Sustainable Future.*

**2018-14** Bart Joosten (UVT), *Detecting Social Signals with Spatiotemporal Gabor Filters.*
**2018-15** Naser Davarzani (UM), *Biomarker discovery in heart failure.*
**2018-16** Jaebok Kim (UT), *Automatic recognition of engagement and emotion in a group of children.*
**2018-17** Jianpeng Zhang (TUE), *On Graph Sample Clustering.*
**2018-18** Henriette Nakad (UL), *De Notaris en Private Rechtspraak.*
**2018-19** Minh Duc Pham (VUA), *Emergent relational schemas for RDF.*
**2018-20** Manxia Liu (RUN), *Time and Bayesian Networks.*
**2018-21** Aad Slootmaker (OUN), *EMERGO: a generic platform for authoring and playing scenario-based serious games.*
**2018-22** Eric Fernandes de Mello Araujo (VUA), *Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks.*
**2018-23** Kim Schouten (EUR), *Semantics-driven Aspect-Based Sentiment Analysis.*
**2018-24** Jered Vroon (UT), *Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots.*
**2018-25** Riste Gligorov (VUA), *Serious Games in Audio-Visual Collections.*
**2018-26** Roelof Anne Jelle de Vries (UT), *Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology.*
**2018-27** Maikel Leemans (TUE), *Hierarchical Process Mining for Scalable Software Analysis.*
**2018-28** Christian Willemse (UT), *Social Touch Technologies: How they feel and how they make you feel.*
**2018-29** Yu Gu (UVT), *Emotion Recognition from Mandarin Speech.*
**2018-30** Wouter Beek, *The "K" in "semantic web" stands for "knowledge": scaling semantics to the web.*

## 2019

**2019-1** Rob van Eijk (UL), *Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification.*
**2019-2** Emmanuelle Beauxis Aussalet (CWI, UU), *Statistics and Visualizations for Assessing Class Size Uncertainty.*
**2019-3** Eduardo Gonzalez Lopez de Murillas (TUE), *Process Mining on Databases: Extracting Event Data from Real Life Data Sources.*
**2019-4** Ridho Rahmadi (RUN), *Finding stable causal structures from clinical data.*
**2019-5** Sebastiaan van Zelst (TUE), *Process Mining with Streaming Data.*
**2019-6** Chris Dijkshoorn (VU), *Nichesourcing for Improving Access to Linked Cultural Heritage Datasets.*
**2019-7** Soude Fazeli (TUD), *Recommender Systems in Social Learning Platforms.*
**2019-8** Frits de Nijs (TUD), *Resource-constrained Multi-agent Markov Decision Processes.*

**2019-9** Fahimeh Alizadeh Moghaddam (UVA), *Self-adaptation for energy efficiency in software systems.*

**2019-10** Qing Chuan Ye (EUR), *Multi-objective Optimization Methods for Allocation and Prediction.*

**2019-11** Yue Zhao (TUD), *Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs.*

**2019-12** Jacqueline Heinerman (VU), *Better Together.*

**2019-13** Guanliang Chen (TUD), *MOOC Analytics: Learner Modeling and Content Generation.*

**2019-14** Daniel Davis (TUD), *Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses.*

**2019-15** Erwin Walraven (TUD), *Planning under Uncertainty in Constrained and Partially Observable Environments.*

**2019-16** Guangming Li (TUE), *Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models.*

**2019-17** Ali Hurriyetoglu (RUN), *Extracting actionable information from microtexts.*

**2019-18** Gerard Wagenaar (UU), *Artefacts in Agile Team Communication.*

**2019-19** Vincent Koeman (TUD), *Tools for Developing Cognitive Agents.*

**2019-20** Chide Groenouwe (UU), *Fostering technically augmented human collective intelligence.*

**2019-21** Cong Liu (TUE), *Software Data Analytics: Architectural Model Discovery and Design Pattern Detection.*

**2019-22** Martin van den Berg (VU), *Improving IT Decisions with Enterprise Architecture.*

**2019-23** Qin Liu (TUD), *Intelligent Control Systems: Learning, Interpreting, Verification.*

**2019-24** Anca Dumitrache (VU), *Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing.*

**2019-25** Emiel van Miltenburg (VU), *Pragmatic factors in (automatic) image description.*

**2019-26** Prince Singh (UT), *An Integration Platform for Synchromodal Transport.*

**2019-27** Alessandra Antonaci (OUN), *The Gamification Design Process applied to (Massive) Open Online Courses.*

**2019-28** Esther Kuinderman (UL), *Cleared for take-off: Game-based learning to prepare airline pilots for critical situations.*

**2019-29** Daniel Formolo (VU), *Using virtual agents for simulation and training of social skills in safety-critical circumstances.*

**2019-30** Vahid Yazdanpanah (UT), *Multiagent Industrial Symbiosis Systems.*

**2019-31** Milan Jelisavcic (VU), *Alive and Kicking: Baby Steps in Robotics.*

**2019-32** Chiara Sironi (UM), *Monte-Carlo Tree Search for Artificial General Intelligence in Games.*

**2019-33** Anil Yaman (TUE), *Evolution of Biologically Inspired Learning in Artificial Neural Networks.*