A Systematic Literature Review of the Successors of 'NeuroEvolution of Augmenting Topologies'

Papavasileiou, Evgenia; Cornelis, Jan Paul Herman; Jansen, Bart

Published in:
Evolutionary Computation

DOI:
10.1162/evco_a_00282

Publication date:
2021

License:
Other

Document Version:
Final published version

Link to publication

Citation for published version (APA):
Papavasileiou, E., Cornelis, J. P. H., & Jansen, B. (2021). A Systematic Literature Review of the Successors of 'NeuroEvolution of Augmenting Topologies'. *Evolutionary Computation*, *29*(1), 1-73. https://doi.org/10.1162/evco_a_00282

A Systematic Literature Review of the Successors of “NeuroEvolution of Augmenting Topologies”

Evgenia Papavasileiou epapavas@etrovub.be
Department of Electronics and Informatics (ETRO), Vrije Universiteit Brussel, Brussels, B-1050, Belgium
imec, Leuven, B-3001, Belgium

Jan Cornelis jpcornel@etrovub.be
Department of Electronics and Informatics (ETRO), Vrije Universiteit Brussel, Brussels, B-1050, Belgium

Bart Jansen bjansen@etrovub.be
Department of Electronics and Informatics (ETRO), Vrije Universiteit Brussel, Brussels, B-1050, Belgium
imec, Leuven, B-3001, Belgium

https://doi.org/10.1162/evco_a_00282

Abstract

NeuroEvolution (NE) refers to a family of methods for optimizing Artificial Neural Networks (ANNs) using Evolutionary Computation (EC) algorithms. NeuroEvolution of Augmenting Topologies (NEAT) is considered one of the most influential algorithms in the field. Eighteen years after its invention, a plethora of methods have been proposed that extend NEAT in different aspects. In this article, we present a systematic literature review (SLR) to list and categorize the methods succeeding NEAT. Our review protocol identified 232 papers by merging the findings of two major electronic databases. Applying criteria that determine the paper’s relevance and assess its quality, resulted in 61 methods that are presented in this article. Our review article proposes a new categorization scheme of NEAT’s successors into three clusters. NEAT-based methods are categorized based on 1) whether they consider issues specific to the search space or the fitness landscape, 2) whether they combine principles from NE and another domain, or 3) the particular properties of the evolved ANNs. The clustering supports researchers 1) understanding the current state of the art that will enable them, 2) exploring new research directions or 3) benchmarking their proposed method to the state of the art, if they are interested in comparing, and 4) positioning themselves in the domain or 5) selecting a method that is most appropriate for their problem.

Keywords

NeuroEvolution, genetic algorithms, artificial neural networks, topology evolution, encoding, systematic literature review.

1 Introduction

NeuroEvolution (NE) is a learning method that uses Evolutionary Algorithms (EA) to optimize the parameters of Artificial Neural Networks (ANNs) (Stanley et al., 2019). In early neuroevolutionary methods, evolutionary optimization methods (e.g., Genetic

Manuscript received: 25 March 2020; revised: 5 August 2020 and 23 October 2020; accepted: 27 October 2020.

Algorithms (GAs)) were used for learning the connection weights of fixed topology ANNs (Whitley et al., 1990; Montana and Davis, 1989; Gomez and Miikkulainen, 1997; Moriarty and Miikkulainen, 1996; Gomez and Miikkulainen, 1998). However, since the definition of a network's topology has a major effect on its performance, new methods appeared that optimize both the weights and the topology (Maniezzo, 1994; Liu and Yao, 1996; Yao and Liu, 1998; Moriarty and Miikkulainen, 1995; Stanley and Miikkulainen, 2002b). These methods are known as Topological and Weight Evolving Artificial Neural Networks (TWEANNs).

TWEANNs offer significant advantages over fixed topology ANNs, as finding the optimal topology of an ANN requires time-consuming evaluations of potential architectures. Especially, in complex problems, the number of neurons and connections that are required scales with the complexity of the problem (Manning and Walsh, 2012) and thus the manual definition of the optimal topology is even more difficult. Moreover, the topology defines the size of the search space: selecting a fixed topology smaller than the optimal means that the search is performed in a lower dimensional space and thus the optimal solution will not be found. On the other hand, selecting a bigger topology than the optimal one implies that the search is performed in an unnecessarily high dimensional space. In TWEANNs the problem of identifying the right ANN topology is tackled by their ability to automatically discover the optimal architecture.

In 2002, a revolutionary TWEANN method called NeuroEvolution of Augmenting Topologies (NEAT) was invented by Kenneth Stanley and Risto Miikulainen (Stanley and Miikkulainen, 2002b). The method provides solutions to problems encountered in NE by facilitating the crossover between individuals of different length, evolving networks by adding new structure when necessary, and protecting structural innovations by organizing them in species.

Today, 18 years later, the NEAT algorithm is still relevant as the large number of published papers extending NEAT or applying NEAT-based methods on challenging problems shows. To our knowledge, no systematic literature review (SLR) of NEAT-based successors exists. Literature review papers that present an overview of NE methods in general can be found (Yao, 1993, 1999; Floreano et al., 2008; Ding et al., 2013; Risi and Togelius, 2015; D'Ambrosio et al., 2014; Stanley et al., 2019). However, none of them uses a systematic review protocol. Instead, they report on the most prominent and important papers in the field. The reviews (Yao, 1993, 1999; Floreano et al., 2008) are very detailed papers that cover the field of NE until 2008. More recent review papers (e.g., Ding et al., 2013; Risi and Togelius, 2015) exist, but their scope is different than the current article, as Ding et al. (2013) describe basic theories and methods in NE and Risi and Togelius (2015) discuss NE's applications in games. A chapter in *Growing Adaptive Machines* by D'Ambrosio et al. (2014) gives an overview of the methods that followed HyperNEAT within five years. Finally, the most recent paper (Stanley et al., 2019) gives an overview of different aspects of recent NE techniques and discusses their potential for application in the field of deep learning.

In the current article, we present a new categorization of the NEAT successors based on criteria defined in this article. This article can serve as a *guidance manual* that allows researchers to *propose* a new NEAT-based method by starting from the most appropriate baseline method or the most recent method or to *select* the method that is more appropriate to solve their problem.

Toward this purpose, detailed tables presenting a fine-grained categorization of the methods according to three main criteria, as well as an illustration (see Figure 1 on p. 13) of the trajectory of the advances in NEAT-based methods are provided. With this new categorization scheme researchers can better position their work in the landscape of existing methods, or select an existing method that fits best the properties of their problem's landscape/search space, or that evolves ANNs with particular properties or that combines principles from different research fields. Moreover, researchers who are interested in benchmarking their method to the state of the art can consult the detailed tables in the Appendix regarding the employed datasets and performance metrics. All the NEAT successors identified by the review protocol are described in detail in the Appendix of this article. Finally, we discuss the findings and the shortcomings of existing approaches, the popularity of NEAT-based algorithms nowadays, and their future relevance in the era of deep learning.

The remainder of this article is organized as follows. Section 2 describes the methodology engaged for conducting the SLR and Section 3 presents the answers to the main research questions regarding the NEAT method. An extended list with detailed answers to all the research questions for all the identified methods can be found in the Appendix. Section 4 presents the proposed categorization clusters and detailed tables with the findings of the review protocol. Finally, Sections 5 and 6 contain the discussion and conclusion parts of this article.

2 Method

With this review article we want to present an objective historical account of NEAT successors. Toward this purpose, we raise the following question: "which methods are based on and succeeded NEAT within a period of 15 years?" A systematic literature review is a type of review that satisfies this objective (Bettany-Saltikov, 2010) as it allows for the findings to be presented in an objective and independent summary (Hemmingway and Brereton, 2009). Since a systematic review article is characterized by the development and use of a review protocol, we followed published guidelines (Keele, 2007; Kofod-Petersen, 2012) on "how to conduct a systematic literature review paper." For a comprehensive review on modern neuroevolution, not necessarily based on NEAT methods, but focusing on current trends, readers are advised to read a paper by Stanley et al. (2019).

Conducting a rigorous and repeatable SLR requires a review protocol (Bettany-Saltikov, 2010). We followed the guidelines according to Keele (2007) and Kofod-Petersen (2012). First, we defined the *objective* and the *research questions* that this review was going to address. Secondly, we designed the search strategy toward answering the research questions by defining the *search terms* and the *inclusion and exclusion criteria* for selecting the relevant literature. After a pilot study we refined both the search terms and the selection criteria. In the following phase, we created a *Quality Assessment check-list* to evaluate the selected literature. Finally, we determined a *data extraction form* inspired by the one in Wen et al. (2012) for documenting the main features of each method.

2.1 Research Questions

The objective of this SLR is to summarize, categorize, and clarify NEAT-based NE methods. These methods will be referred to as x-NEAT.[1] Toward this objective, we raised the following Research Questions (RQ):

1. RQ1: What are the main principles and features of each method? (Principles)

[1]The letter "x" is used to represent the various acronyms used in the names of NEAT's successors.

Table 1: Inclusion and exclusion criteria.

No.	Criterion
IC1	The study describes an extension of NEAT, (x-NEAT)
IC2	The study describes a hybrid model, for example, based on NEAT and another evolutionary or machine learning algorithm
EC1	The study is only an application of an x-NEAT method to a dataset or a new domain
EC2	The study describes an EA that is not a successor of NEAT but inspired by the NEAT principles (Section 3.1)
EC3	The study compares a not-NEAT-based NE method with NEAT, or existing x-NEAT methods among each other
EC4	The study is an older/conference version of a relevant journal paper

IC: Inclusion Criterion, EC: Exclusion Criterion.

2. RQ2: How are the ANNs encoded into the genome? (Encoding) The purpose is to identify the different encoding schemes that are used for representing the ANNs phenotypes in the GAs' genome.
3. RQ3: How does the x-NEAT method perform compared to others? Which performance metrics are used to make this comparison? (Performance)

2.2 Search Strategy

2.2.1 Search Terms

In order to identify the relevant papers succeeding NEAT (Stanley and Miikkulainen, 2002b), we had to define the search terms by identifying the major terms and alternative spellings and synonyms. We used the Boolean OR to combine synonyms and alternative spellings and the Boolean AND to link the major terms. The search was performed on the 18 April 2018 and we chose to include all the papers published until the end of 2017. After refinement, the resulting search term is the following ((neuro?evolution OR neuroevolution* OR evolv* neural networks) AND (*neat OR augment* topologies)) AND (EXCLUDE (PUBYEAR, 2018)).

2.2.2 Literature Resources

With the search term, we searched for published journal and conference papers in the electronic databases of Web of Science (WoS) (all databases) and Scopus. The search resulted in 225 publications from all the databases of WoS and 121 publications from Scopus. We searched each database separately and gathered the papers together. Excluding the common papers between the two databases resulted in 232 publications which had to be evaluated on their relevance and quality.

2.2.3 Study Selection

To select the relevant papers we read all the abstracts and we applied the inclusion (IC) and exclusion (EC) criteria defined in Table 1. Forty papers were selected based on IC1, 25 based on IC2, and 5 based on both IC1 and IC2. Sixty-two papers were excluded because of EC1, 7 because of EC2, 42 based on EC3, 39 based on EC4, 1 based on EC1 and EC3, and 1 based on EC2 and EC3.

2.2.4 Study Quality Assessment

The quality of each relevant paper was evaluated according to the Quality Assessment Criteria (QAC) defined in the format of questions and presented in Table 2. If a question

Table 2: Quality assessment questions with possible answers: "Yes," "Partly," or "No."

	QA questions concerning the x-NEAT's principles
QA1	Are the aims of the research clearly defined?
QA2	Are the main aspects of the proposed method explained in detail?
QA3	If the method introduces a new encoding scheme, is it described clearly? If the same encoding as in previous methods is used, is it understandable from the paper?
	QA questions regarding the experimental procedure
QA4	Is the experimental procedure clearly described?
QA5	Is the method evaluated on sufficient number of datasets? (number of datasets $\geq$3: yes, 2: partly, 1: no)
QA6	If the study involves a custom artificial dataset, is its construction method adequately described? If it cannot be described for example, in case of a video game, is the task clearly explained?
QA7	Are the parameters of the NE algorithm clearly described?
QA8	Are the metrics used for measuring the algorithm's performance clearly defined?
QA9	Is each experiment run for an adequate number of repetitions? (Yes: $\geq$20, Partly: [10,20), No: [0,10))
QA10	Is there a statistical test to test if a statistical difference in the compared performances exists?
QA11	Is the proposed method compared to the state of the art of NE methods?
QA12	Is the proposed method compared to other machine learning/EC algorithms?
	QA questions regarding the reception of the paper from the community
QA13	Does the study have an adequate number of citations per year?[2] (Yes: $\geq$1, Partly: [0.5,1), No: [0,0.5))

was addressed by the paper, it could take one of the answers: "yes," "partly," and "no," which received one, one-half, and zero points, respectively. If the question was not addressed at all by the paper, then it received the answer "non-applicable" excluding it from the calculation of the final result. In this way, a paper's final score was calculated by averaging the score of each question. Nine papers had a score $\in [0, 0.5]$ and they were excluded from the study.

2.2.5 Data Extraction

Application of the IC, EC, and the QAC resulted in 61 papers whose acronyms and obtained scores are presented in Table 4. From these, we collected the necessary data to answer the RQs. To keep the most important information from each paper, we created cards as shown in Table 3 similar to Wen et al. (2012).

3 NEAT—Answers to the Research Questions

NeuroEvolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen, 2002b) is a TWEANN method that enables the learning of the structure of ANNs at the same time it optimizes their connectivity weights. When NEAT was proposed in 2002, it provided solutions to important research questions of that time. The proficiency of the method is attributed to the three main innovations for which ablation studies showed

[2]The number of citations per year (the last quality assessment criterion (QA13)) is calculated by taking the average number of citations of the concerning paper by both the Web of Science and Scopus and dividing by the number of years after the paper's publication until 2017.

Table 3: Data extraction card.

Data Item	Value
Data extractor's name and date	
Data checker's name and date	
Study ID / Article type	
Publication year	
Names of the authors of the study	
Article title	
Name of the x-NEAT method	
Characteristics of the version	
Encoding scheme	
Performance metrics used	

that each of the introduced components is crucial to the NEAT's performance. The three main features of NEAT are described in Section 3.1.

3.1 Research Question 1: Principles

Historical Markings to Deal with the Competing Conventions Problem. One crucial issue in NE is the Competing Conventions Problem, also known as the Permutation Problem (Radcliffe, 1993) that appears when there is more than one way to represent an ANN and because structures evolve independently in different networks. In particular, the problem occurs during the crossover of two genomes that are encoded differently even though they represent the same solution. As a consequence, the resulting offspring might suffer from loss of functionality. To overcome this problem, the system should be able to identify identical structures. NEAT deals with this issue by means of *historical markings*, that is, by assigning an increasing innovation number when a new gene is added to the genome. A new gene is introduced in the system by structural mutations that add a new connection or a new node in the network. The historical markings act as chronological indicators that facilitate crossover by identifying homologous sections between different networks. The innovation number of each gene is inherited by the offspring, facilitating the retaining of its historical origin throughout evolution.

Speciation to Protect Innovation. NEAT protects topological innovations through *speciation* in order to give time to new structures to optimize. The individuals compete within their own niche instead of the entire population. In general, when a new connection is added, a random weight is assigned to it. In order to converge to its optimal value, a number of generations is required. Without speciation, the new individual would have to compete with the entire population. In that case, there is a high probability that the individual would be replaced before it gets optimized, because of its poor performance compared with the other already more optimized networks. On the other hand, with speciation, the network competes within its own niche, so it is given time to optimize its weights before having to compete with the entire population. The networks are grouped in species based on their topological similarities. This is defined by aligning the genomes based on the historical markings and determining the matching, disjoint and excess genes among the individuals. The nonmatching genes between the two individuals that are located in the middle of the genomes are called disjoint genes, while the nonmatching genes in the end of the genomes are called excess genes. This topological similarity is calculated through a measure called compatibility distance, as

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

where E is the number of excess genes, D the number of disjoint genes, $\overline{W}$ the average weight differences of matching genes, N the number of genes in the larger genome, and c_1, c_2, c_3 the coefficients which define the importance of the three factors. In practice the division with the number of genes N is omitted and N is set to 1.

Biasing the Search toward Smaller Structures. NEAT starts the evolution with a uniform population of minimal structures, that is, fully connected networks with no hidden nodes and it evolves more complex networks (*complexification*) by introducing new nodes and connections through structural mutations. These topological innovations are maintained only if they are found to be useful after fitness evaluation, that is, if they can increase the network's performance. In this way, NEAT tends to evolve smaller structures.

3.2 Research Question 2: Encoding

NEAT uses direct encoding to encode the phenotypes (ANNs) in the genotype. In direct genetic encoding, there is a one-to-one mapping between the networks' phenotypes and genotypes. One genome is used for encoding the nodes and a second genome for encoding the connections. The networks' nodes are encoded with a set of node genes while the connections among the nodes are encoded with a set of connection genes. The node genes have fields that indicate the type of the node, that is, whether the node is an input, an output, or a hidden node, whereas the connection genes indicate the connections established between the nodes in the node genome. Each connection gene consists of 5 fields: the id of the node receiving the connection (in-node), the id of the node from where the connection begins (out-node), the weight of the connection, an enabled bit that specifies whether the connection is enabled or not, and an innovation number which allows finding corresponding genes during mating.

3.3 Research Question 3: Evaluation

NEAT is tested on the XOR problem and its performance is evaluated with the average number of generations that are required to solve it. Moreover, NEAT is compared to four other NE systems on the double pole balancing problem (a benchmark Reinforcement Learning (RL) task), in terms of number of generations, their ability to generalize and the size of the evolved networks. NEAT converges faster, but its ability to generalize is not statistically different from other systems.

4 Findings and Categorization

In this section, we give a summary of the findings of the review protocol. The 61 methods that fulfill the inclusion criteria and pass the quality assessment threshold are presented in Table 4 along with the obtained scores. Detailed answers to the three research questions for all the methods are presented in the Appendix.

4.1 Categorization Based on the Encoding

To begin with, in Table 5, we present a categorization of NEAT's successors based on the employed encoding scheme, that is, the way the mapping between GA's genotype

Table 4: Quality scores of selected studies.

Study	Ref	Score	Study	Ref	Score
1. Coevolutionary NEAT	(Stanley and Miikkulainen, 2004)	0.92	32. Adaptive ES-HyperNEAT	(Risi and Stanley, 2012b)	0.65
2. ModularNEAT	(Reisinger et al., 2004)	0.81	33. NEATfields	(Inden et al., 2012)	0.92
3. FS-NEAT	(Whiteson et al., 2005)	0.81	34. SNAP-NEAT	(Kohl and Miikkulainen, 2012)	0.81
4. rtNEAT	(Stanley et al., 2005)	0.71	35. SUPG-HyperNEAT	(Morse et al., 2013)	0.85
5. rtNEATv2	(D'Silva et al., 2005)	0.73	36. MSS-HyperNEAT	(Pugh and Stanley, 2013)	0.85
6. CPPN-NEAT	(Stanley, 2006)	0.58	37. Adaptive HyperNEATv2	(Gallego-Durán et al., 2013)	0.58
7. (Online-) NEAT+Q	(Whiteson and Stone, 2006a)	0.81	38. NEAR	(Chatzidimitriou and Mitkas, 2013)	0.88
8. LAPCA-NEAT	(Monroy et al., 2006)	0.65	39. Layered NEAT	(Wang et al., 2013)	0.62
9. L-NEAT	(Chen and Alahakoon, 2006)	0.54	40. FNS-NEATFields	(Inden et al., 2013)	0.88
10. nnrg.hazel	(Reisinger et al., 2007)	0.65	41. Phased NEAT	(Tan et al., 2013)	0.77
11. KO-NEAT	(Zhao et al., 2007)	0.62	42. HyperNEAT-CCT	(Huizinga et al., 2014)	0.85
12. NEAT-CTRNN	(Miguel et al., 2008)	0.69	43. Seeded (Adaptive) HyperNEAT	(Risi and Stanley, 2014)	0.69
13. HyperNEAT	(Stanley et al., 2009)	0.85	44. NS-FE-CPPN-NEAT	(Methenitis et al., 2015)	0.62
14. TL-CPPN-NEAT	(Bahçeci and Miikkulainen, 2008)	0.69	45. Deep HyperNEAT	(Verbancsics and Harguess, 2015)	0.77
15. Multiagent HyperNEAT	(D'Ambrosio and Stanley, 2008)	0.92	46. PIGEON	(Stein et al., 2015)	0.85
16. cgNEAT	(Hastings et al., 2009)	0.54	47. PFS-NEAT	(Loscalzo et al., 2015)	0.81
17. RL-SANE	(Wright and Gemelli, 2009)	0.62	48. odNEAT	(Silva et al., 2015)	0.92
18. FD-NEAT	(Tan et al., 2009)	0.81	49. odNEATv2	(Silva et al., 2016)	0.92
19. RBF/Cascade-NEAT	(Kohl and Miikkulainen, 2009)	0.81	50. τ-NEAT	(Caamaño et al., 2016)	0.73
20. MO-NEAT	(Haggett and Chu, 2009)	0.73	51. MAP-Elites CPPN	(Tarapore et al., 2016)	0.73
21. Adaptive HyperNEAT	(Risi and Stanley, 2010)	0.73	52. NEAT-LSTM, NEAT-LSTM-IM	(Rawal and Miikkulainen, 2016)	0.73
22. Online (rt-) NEAT	(Cardamone et al., 2010)	0.85	53. MM-NEAT	(Schrum and Miikkulainen, 2016)	0.83
23. Recurrent CPPN-NEAT	(Auerbach and Bongard, 2011)	0.85	54. MB-HyperNEAT	(Schrum et al., 2016)	0.81
24. Multiagent HyperNEATv2	(D'Ambrosio et al., 2011)	0.77	55. PLPS-NEAT-TL	(Hardwick-Smith et al., 2017)	0.79
25. HyperNEAT-LEO	(Verbancsics and Stanley, 2011)	0.85	56. TMO-NEAT	(Marzullo et al., 2017)	0.75
26. IFSE-NEAT	(Wright et al., 2012)	0.57	57. τ-HyperNEAT	(Silva et al., 2017)	0.58
27. NoveltyNEAT	(Lehman and Stanley, 2011a)	0.88	58. EXACT	(Desell, 2017a)	0.63
28. SwitchHybrID	(Clune et al., 2011)	0.85	59. HA-NEAT	(Hagg et al., 2017)	0.90
29. MFF-NEAT	(Manning and Walsh, 2012)	0.62	60. NEAT-RAC-PGS	(Peng et al., 2017)	0.88
30. ES-HyperNEAT(-LEO)	(Risi and Stanley, 2012a)	0.92	61. NEAT-FLEX	(Grisci and Dorn, 2017)	0.58
31. DynNEAT	(Krčah, 2012)	0.73			

Table 5: Encoding used by each method.

Direct Encoding	(Stanley and Miikkulainen, 2002b, 2004; Reisinger et al., 2004; Whiteson et al., 2005; Stanley et al., 2005; D'Silva et al., 2005; Stanley, 2006; Whiteson and Stone, 2006a; Monroy et al., 2006; Chen and Alahakoon, 2006; Reisinger et al., 2007; Zhao et al., 2007; Miguel et al., 2008; Bahçeci and Miikkulainen, 2008; Hastings et al., 2009; Wright and Gemelli, 2009; Tan et al., 2009; Kohl and Miikkulainen, 2009; Haggett and Chu, 2009; Cardamone et al., 2010; Auerbach and Bongard, 2011; Wright et al., 2012; Lehman and Stanley, 2011a; Manning and Walsh, 2012; Krčah, 2012; Kohl and Miikkulainen, 2012; Chatzidimitriou and Mitkas, 2013; Wang et al., 2013; Inden et al., 2013; Tan et al., 2013; Methenitis et al., 2015; Stein et al., 2015; Loscalzo et al., 2015; Silva et al., 2015, 2016; Caamaño et al., 2016; Rawal and Miikkulainen, 2016; Schrum and Miikkulainen, 2016; Hardwick-Smith et al., 2017; Marzullo et al., 2017; Desell, 2017a; Hagg et al., 2017; Peng et al., 2017; Grisci and Dorn, 2017)
Indirect Encoding	(Stanley et al., 2009; D'Ambrosio and Stanley, 2008; Risi and Stanley, 2010; Auerbach and Bongard, 2011; D'Ambrosio et al., 2011; Verbancsics and Stanley, 2011; Risi and Stanley, 2012a, 2012b; Inden et al., 2012; Morse et al., 2013; Pugh and Stanley, 2013; Gallego-Durán et al., 2013; Huizinga et al., 2014; Risi and Stanley, 2014; Verbancsics and Harguess, 2015; Tarapore et al., 2016; Schrum et al., 2016; Silva et al., 2017)
Hybrid	(Clune et al., 2011)

and ANN's phenotype happens. We identify three types of encoding: direct, indirect, and hybrid.

Direct encoding refers to a one-to-one mapping between the genotype and the phenotype, such as NEAT's node and connection genes described in Section 3.2. On the other hand, in indirect encoding a set of rules is used to map the genotype to the phenotype. Compositional Pattern Producing Networks (CPPNs) (Stanley, 2007), details presented in the Appendix, are an example of indirect encoding (Stanley et al., 2019). CPPNs, which behave as ANNs, can be evolved by NEAT to output a spatial pattern in the hyperspace which corresponds to a connectivity pattern in a substrate of nodes. Hybrid encoding refers to an alternation between direct and indirect encoding.

Analyzing how the ANNs are encoded in the genotype, we found that approximately two-thirds of the methods use direct encoding, while one method proposes a hybrid encoding. Most of the methods with direct encoding use NEAT's encoding without modifications. However, there are methods that modify it by adding new fields according to the characteristics of the proposed method. The methods using indirect encoding either propose a new type of encoding or use HyperNEAT's encoding. When necessary, modifications on HyperNEAT's encoding are made to meet the requirements of the proposed method, for example, by changing the initial topology of the CPPNs to include more input, output, or hidden nodes.

4.2 Proposed Categorization Scheme

Although NE methods can be traditionally categorized based on the encoding used for mapping the genotype to the phenotype, as presented in Section 4.1, in this article we perform a more fine-grained classification. Based on three criteria presented next, the NE methods are classified into three clusters, each of which consists of a number of subclusters that result in 18 subclusters in total. We started the categorization from subclusters that considered research questions we wanted to address and then we regrouped them in a higher level of clusters to increase the readability and get a more structured cluster description.

- Cluster 1: methods that consider issues relevant to the *search space* or the *fitness landscape*. This cluster is about properties of the search space and the fitness landscape and not about descriptive features of a certain method.
- Cluster 2: *hybrid* methods, that is, methods that employ principles from NE and another field of EC or Machine Learning (ML).
- Cluster 3: methods that evolve *ANNs with particular properties*.

4.2.1 Cluster 1: Issues Specific to the Search Space and the Landscape

NE methods are classified into the following subclusters based on which search space/landscape issues they solve, which include:

- a problem/search space with *multiple objectives*. A multiobjective optimization task is a domain where the simultaneous optimization of multiple conflicting objectives is required (Branke et al., 2008).
- a search space characterized by many *irrelevant* or *redundant* features. A feature is a measurable property or characteristic that is used to describe a task (Bishop, 2006). In this way, the instances of a problem are represented as multidimensional points in an n-dimensional space (where n is the number of features describing the problem), which is difficult to search.
- a *deceptive* landscape. A landscape is deceptive when the population of the GA is misguided away from the objective (Horn and Goldberg, 1995). In this case, a fitness function deceives the search by pointing the wrong way.
- a landscape characterized by *uncertainty*. An uncertain landscape is one where the fitness functions' optimum changes rapidly between generations (Krčah, 2012).
- a search space of an *open-ended* problem, i.e. a problem whose final solution is not finite (Stanley and Miikkulainen, 2004) and for which more complex (Maley, 1999) and novel (Standish, 2003) solutions are continuously generated.
- a space where evolution takes place *online* or in *real time*. In online evolution an ANN is evolved spontaneously without having prior information of the whole environment or task. This is in contrast with offline evolution when all the information is available from the beginning and the ANN is evolved before its final application to the target task. In real-time evolution, a constraint of the time for generating a solution exists. According to Cardamone et al. (2010), online evolution focuses on maximizing an agent's performance during the whole learning process, whereas real-time evolution aims to find a group of agents that perform well as fast as possible.

4.2.2 Cluster 2: Hybrid NE Methods

These methods combine NE principles with methods from the following fields:

- *Evolutionary Computation (EC)*
- *Backpropagation (BP)*
- *Reinforcement Learning (RL)*
- *Unsupervised Learning (UL)*

4.2.3 Cluster 3: Evolving ANNs with Particular Properties

NE methods that can evolve ANNs with particular characteristics belong to this cluster. These properties include:

- *modularity*. A network is modular if it contains "highly connected clusters of nodes that are sparsely connected to nodes in other clusters" (Clune et al., 2013). A modular network consists of independent functional structures that can be separately optimized (Kashtan and Alon, 2005), can operate on separate inputs to perform a sub-task and are organized by an intermediary to produce the network's output (Azam, 2000).
- *plasticity*. An ANN is plastic when its connections' weights do not remain static during their lifetime but can change in response to the changing activation levels in the neurons they connect (Risi and Stanley, 2010).
- *transfer learning* ability, that is, transferring knowledge that is learned on one task to another one (Bahçeci and Miikkulainen, 2008).
- automatic *ANN substrate configuration*. This refers to methods that evolve not only the weights of a large scale ANN substrate, but also its topography, that is, the density and placement of neurons (Risi and Stanley, 2012a). Substrate is a specific terminology introduced in HyperNEAT (Stanley et al., 2009). For a detailed description please see Section A.13 in the Appendix.
- different *types of nodes*, that is, ANNs with nodes that are different than the sigmoid neuron units.
- *large scale topologies*, evolving large ANNs that define a high-dimensional search space. In this category we also include ANNs with a large number of input and output nodes to solve problems with high-dimensional input/output space.
- *deep architectures*. We refer to methods optimizing parameters of deep ANNs (DNNs).
- *memory capacity*. Augmenting an ANN with memory is essential for sequential problems requiring long-term memory.

4.3 The x-NEAT Methods

The classification of the x-NEAT methods found by the review protocol is presented in Table 6; only two methods, TMO-NEAT (Marzullo et al., 2017) and Cascade-NEAT (Kohl

Table 6: Proposed categorization of the x-NEAT methods.

Methods	Reference	Cluster 1:Search Space/Landscape Issues						Cluster 2: Hybrid NE with				Cluster 3: Evolving ANNs with							
		Deceptive landscape	fitness uncertainty	Open ended	Multiple objectives	Irrelevant features	Online/ real time evolution	EC	BP	RL	UL	Different node types	Modularity	Plasticity	Transfer learning	Automatic substrate configuration	Large Topologies	Deep architectures	Memory capacity
A. Coevolutionary NEAT	(Stanley and Miikkulainen, 2004)			✓															
B. Modular NEAT	(Reisinger et al., 2004)												✓				✓		
C. FS-NEAT	(Whiteson et al., 2005)					✓													
D. rtNEAT	(Stanley et al., 2005)						✓												
E. rtNEATv2	(D'Silva et al., 2005)						✓												
F. CPPN-NEAT	(Stanley, 2006)											✓							
G1. NEAT+Q	(Whiteson and Stone, 2006a)							✓	✓	✓									
G2. Online NEAT+Q	(Whiteson and Stone, 2006a)						✓	✓	✓	✓									
H. LAPCA-NEAT	(Monroy et al., 2006)			✓															
I. L-NEAT	(Chen and Alahakoon, 2006)								✓										
J. nnrg.hazel	(Reisinger et al., 2007)			✓															
K. KO-NEAT	(Zhao et al., 2007)						✓	✓		✓									
L. NEAT-CTRNN	(Miguel et al., 2008)											✓							✓
M. HyperNEAT	(Stanley et al., 2009)																✓		
N. TL-CPPN-NEAT	(Bahçeci and Miikkulainen, 2008)											✓			✓				
O. Multiagent HyperNEAT	(D'Ambrosio and Stanley, 2008)																✓		
P. cgNEAT	(Hastings et al., 2009)						✓												
Q. RL-SANE	(Wright and Gemelli, 2009)									✓									
R. FD-NEAT	(Tan et al., 2009)					✓													
S1. RBF-NEAT	(Kohl and Miikkulainen, 2009)											✓							
S2. Cascade-NEAT	(Kohl and Miikkulainen, 2009)																		
T. MO-NEAT	(Haggett and Chu, 2009)				✓									✓					
U. Adaptive HyperNEAT	(Risi and Stanley, 2010)													✓			✓		
V1. Online NEAT	(Cardamone et al., 2010)						✓			✓					✓				
V2. Online rtNEAT	(Cardamone et al., 2010)						✓			✓					✓				
W. Recurrent CPPN-NEAT	(Auerbach and Bongard, 2011)											✓							
X. Multiagent HyperNEATv2	(D'Ambrosio et al., 2011)																✓		
Y. HyperNEAT-LEO	(Verbancsics and Stanley, 2011)												✓				✓		
z. IFSE-NEAT	(Wright et al., 2012)					✓													
AA. NoveltyNEAT	(Lehman and Stanley, 2011a)	✓		✓															
AB. Switch HybrID	(Clune et al., 2011)																✓		
AC. MFF-NEAT	(Manning and Walsh, 2012)												✓						
AD1. ES-HyperNEAT	(Risi and Stanley, 2012a)															✓	✓		
AD2. ES-HyperNEAT-LEO	(Risi and Stanley, 2012a)												✓			✓	✓		
AE. DynNEAT	(Krčah, 2012)		✓																
AF. Adaptive ES-HyperNEAT	(Risi and Stanley, 2012b)													✓		✓	✓		
AG. NEATFields	(Inden et al., 2012)																✓		
AH. SNAP-NEAT	(Kohl and Miikkulainen, 2012)											✓							
AI. SUPG-HyperNEAT	(Morse et al., 2013)											✓					✓		
AJ. MSS-HyperNEAT	(Pugh and Stanley, 2013)															✓	✓		
AK. Adaptive HyperNEATv2	(Gallego-Durán et al., 2013)													✓			✓		
AL. NEAR	(Chatzidimitriou and Mitkas, 2013)							✓		✓		✓							✓
AM. Layered NEAT	(Wang et al., 2013)					✓													
AN.FNS-NEATFields	(Inden et al., 2013)	✓						✓									✓		
AO. Phased NEAT	(Tan et al., 2013)					✓													
AP. HyperNEAT-CCT	(Huizinga et al., 2014)												✓				✓		
AQ1. Seeded HyperNEAT	(Risi and Stanley, 2014)																✓		
AQ2. Seeded Adaptive HyperNEAT	(Risi and Stanley, 2014)													✓			✓		
AR. NS-FE-CPPN-NEAT	(Methenitis et al., 2015)	✓						✓				✓							
AS. Deep HyperNEAT	(Verbancsics and Harguess, 2015)								✓									✓	
AT. PIGEON	(Stein et al., 2015)							✓											
AU. PFS-NEAT	(Loscalzo et al., 2015)					✓													
AV. odNEAT	(Silva et al., 2015)						✓												
AW. odNEATv2	(Silva et al., 2016)						✓												
AX. τ-NEAT	(Caamaño et al., 2016)																		✓
AY. MAP-Elites CPPN	(Tarapore et al., 2016)	✓						✓				✓							
AZ1. NEAT-LSTM	(Rawal and Miikkulainen, 2016)											✓							✓
AZ2. NEAT-LSTM-IM	(Rawal and Miikkulainen, 2016)	✓									✓	✓							✓
BA. MM-NEAT	(Schrum and Miikkulainen, 2016)				✓								✓						
BB. MB-HyperNEAT	(Schrum et al., 2016)												✓				✓		
BC. PLPS-NEAT-TL	(Hardwick-Smith et al., 2017)														✓				
BD. TMO-NEAT	(Marzullo et al., 2017)																		
BE. τ-HyperNEAT	(Silva et al., 2017)																✓		✓
BF. EXACT	(Desell, 2017a)								✓			✓						✓	
BG. HA-NEAT	(Hagg et al., 2017)											✓							
BH. NEAT-RAC-PGS	(Peng et al., 2017)									✓									
BI. NEAT-FLEX	(Grisci and Dorn, 2017)										✓								

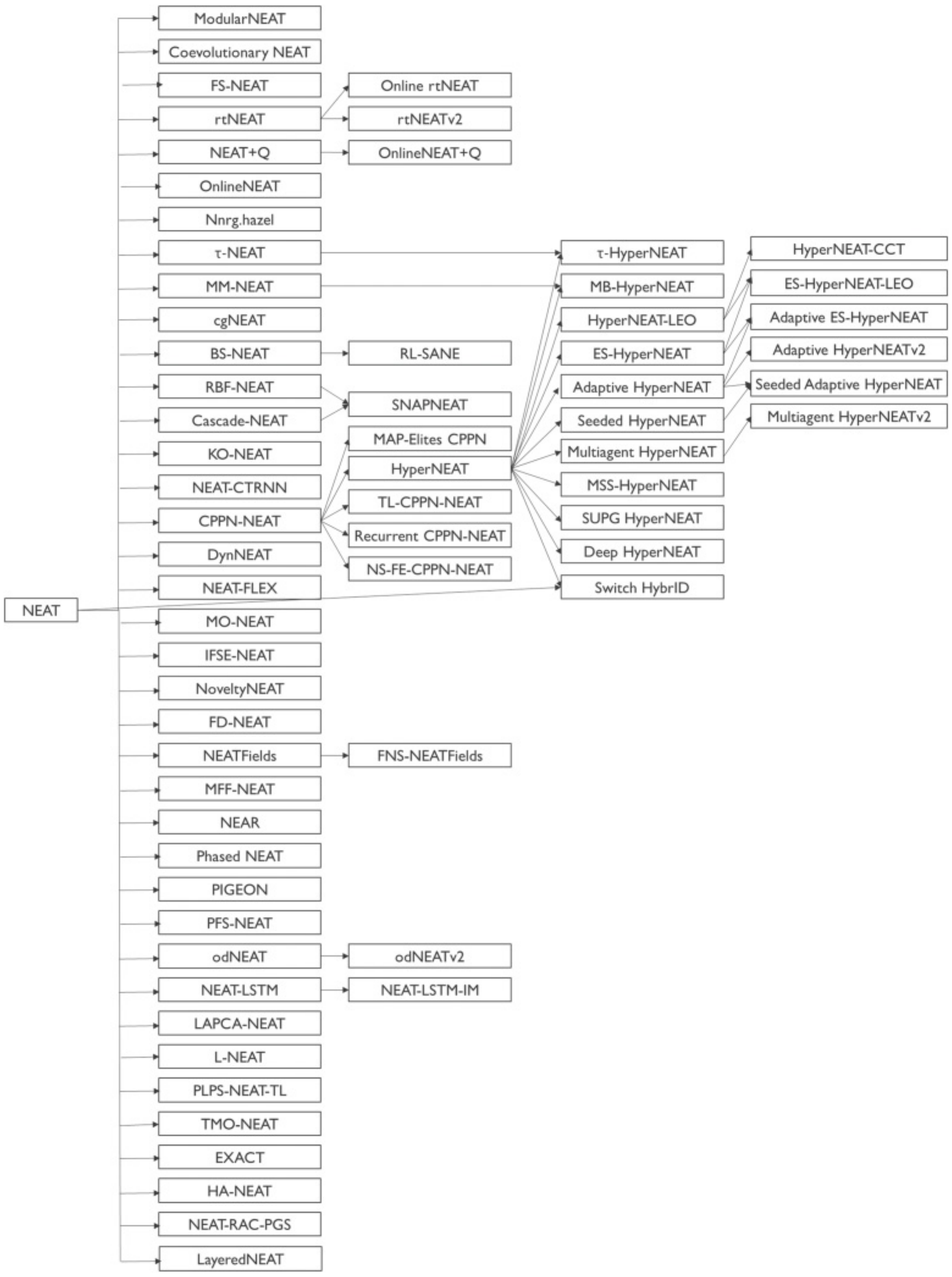

Figure 1: The trajectory of evolution of x-NEAT methods.

and Miikkulainen, 2009), could not be categorized into any of the proposed clusters. It should be noted that this table is not an exhaustive list of all the methods in the field, just the ones that are returned by the inclusion and selection criteria of the protocol. The table is preferably read column-wise: a single tick in a column does not at all mean that the indicated paper is the only paper addressing the category. In the Appendix, we present detailed answers of the three RQs (principles, encoding, and evaluation) for each method. Figure 1 presents the trajectory of the x-NEAT methods' evolution. This is a graphical illustration of how x-NEAT methods emerged from the original NEAT method, that is, which method is being extended by another.

4.4 Reference Tables for Benchmarking

Tables 8, 9, and 10 in the Appendix present the domains/datasets/tasks on which the x-NEAT methods of this review article are tested and the used performance metrics. Each dataset has one or more references that correspond to the x-NEAT method that employs it. In some cases, the original source of the dataset was available and this is included as the last citation in the list of provided citations. These tables can help researchers to pick up the method that is most suitable for their problem and find its detailed description in the Appendix. Also, it facilitates comparing and benchmarking new or existing NE methods by choosing the dataset and the performance metric used in the state of the art.

5 Discussion, Open Issues, and Future Perspectives

5.1 Why NEAT Inspired the Different Extensions

After the invention of NEAT, many methods have appeared that extend its functionality in various ways. We believe that the reason for this expansion is because NEAT is an easily understandable algorithm that works well on difficult problems and many researchers made its implementation publicly available. Moreover, we attribute this expansion to the fundamental importance of the NEAT principles, detailed in Section 3. NEAT incorporates specific biological principles (Stanley and Miikkulainen, 2002b) that have contributed to its success. The evolutionary process in NEAT resembles the process of natural evolution, as the evolved networks become more complex during their optimization. As in NEAT, the genome in nature does not have a fixed length, but new genes have been added during evolution through a process known as gene amplification (Darnell and Doolittle, 1986; Watson, 2004). Moreover, the solution that NEAT provides to the competing conventions problem is also inspired by natural evolution and in particular by the synapsis process, that is, the process of lining up homologous genes before crossover (Radding, 1982; Sigal and Alberts, 1972). Similarly, the speciation mechanism is inspired from nature's speciation mechanism that groups together individuals that share a common characteristic and implicitly protects innovation since the different species compete within their own niche (Stanley and Miikkulainen, 2002b). Although speciation as a notion was known in GAs, it was NEAT which introduced it in the TWEANNs, since NEAT offered a solution to the competing conventions problem with the use of historical markings. Before this solution, the definition of a compatibility metric, necessary for assigning individuals into species, was difficult to define. As far as the notion of evolving minimal architectures is concerned, before NEAT there were already attempts to control the size of the evolved topologies by including penalties in the fitness function. However, these attempts had the disadvantages of having to predefine the penalty and risking to have a different performance from the original non-penalized fitness function (Stanley and Miikkulainen, 2002b). By keeping the topologies minimal throughout evolution, NEAT tends to minimize the search space and to speed up the learning process (Stanley and Miikkulainen, 2002b). Before NEAT, starting with a uniform minimal population was not possible. Instead, TWEANNs started the evolution with topologically diversified networks because without speciation topological innovations could not survive (Stanley and Miikkulainen, 2002b). On the other hand, speciation in NEAT enabled the evolution to start minimally and diversify along evolution, since topological innovations would be protected. In conclusion, NEAT tackled problems known in the community in a unified and fundamentally strong manner.

A clear indication of how fundamentally important and globally applicable the NEAT principles are, can be found in the existence of a large number of papers whose method is inspired by the NEAT principles in, for example, NE with ontogeny (Inden, 2008), Genetic Programming (Drchal and Šnorek, 2013; Drchal and Snorek, 2012; Trujillo et al., 2016), and molecular programming (Dinh et al., 2014). Neat-GP (Trujillo et al., 2016) is an example of a method that adopts NEAT's principles (minimal topology initialization, speciation with fitness sharing and crossover by identifying the matching topologies) for evolving programs in Genetic Programming (GP) for regression and classification tasks. Neat-GP was able to achieve or improve the test fitness performance of standard GP while evolving smaller programs and reducing the computational cost and it shows that NEAT principles are fundamental and can be extended beyond neural networks.

5.2 Findings of the Review and Recommendations

From the tables provided in the Appendix of this review article as well as from Table 7, we can conclude that comparing the performance of different x-NEAT methods is difficult, as even methods belonging to the same subcluster are not all evaluated on the same datasets and do not employ the same performance metrics.

Different Performance Metrics. The majority of the methods report on the algorithm's performance using the obtained fitness value. Similar metrics include the accuracy, the average error and the absolute error. Most of the methods obtain the mean value of these metrics calculated over the different algorithmic runs. Another metric is the ratio of runs when the algorithm was successful in solving the problem. Also, the majority of the methods report metrics to describe the algorithm's efficiency such as the number of generations and the computational time. Metrics describing the size of the evolved networks are also very common, for example, the number of evolved nodes and connections. Some methods are evaluated based on qualitative metrics, i.e. based on the user's interpretation of the evolved patterns and the agents' evolved behavior or morphology. Other papers evaluate the generalization ability of the proposed algorithm by testing the best ANN on a different task with different initial conditions. Finally, metrics to evaluate a specific characteristic of a method are proposed, such as the ratio between relevant and irrelevant features or the sum of absolute weights for methods performing feature selection, the number of agents that are dominated for coevolutionary methods and others.

Performance Comparisons. Even though a newly proposed method that brings new insights in the field has a great value in its own, we observed that many research papers also compare their methods to other ones using a wide set of datasets and performance metrics. We believe that in order to facilitate comparisons, a good practice would be to report performance on a minimal set of universal performance metrics. Regarding the algorithms' performance, the accuracy, precision, recall, and confusion matrix for classification tasks and the accumulated reward for identical reinforcement learning

Table 7: Table summarizing an x-NEAT method's categorization to clusters, the method it extends, and its comparisons to the state of the art (either to an existing x-NEAT algorithm or to an algorithm from the fields of ML/NE/EC).

			Compared to	
Method	Cluster	Extends	x-NEAT	ML/NE/EC
RBF-NEAT	Different node types	NEAT	NEAT	
SNAP-NEAT	Different node types	RBF-NEAT Cascade-NEAT	NEAT RBF-NEAT Cascade-NEAT	√
NoveltyNEAT	Deceptive landscape Open ended	NEAT	NEAT NEAT-CTRNN	
DynNEAT	Fitness uncertainty	NEAT	NEAT	√
NEAT-LSTM-IM	Deceptive landscape UL Different node types Memory Capacity	NEAT-LSTM	NEAT-LSTM NEAT-RNN	
Coevolutionary NEAT	Open ended	NEAT	Fixed Topology Coevolutionary NEAT Simplifying Coevolutionary NEAT	
LAPCA-NEAT	Open ended	NEAT	NEAT-HOF	
nnrg.hazel	Open ended	NEAT		√
MO-NEAT	Multiple objectives Plastic NNs	NEAT		√
MM-NEAT	Multiple objectives Modular NNs	NEAT		√
FS-NEAT	Irrelevant features	NEAT	NEAT	
FD-NEAT	Irrelevant features	NEAT	NEAT FS-NEAT	
IFSE-NEAT	Irrelevant features	NEAT	NEAT FS-NEAT	
Layered NEAT	Irrelevant features	NEAT	NEAT FD-NEAT	
PFS-NEAT	Irrelevant features	NEAT	NEAT FS-NEAT FD-NEAT SAFS-NEAT	
Phased NEAT	Irrelevant features	NEAT	NEAT FD-NEAT	
rtNEAT	Online/Real time	NEAT		
rtNEATv2	Online/Real time	rtNEAT	rtNEAT	
Online NEAT+Q	Online/Real time EC BP RL	NEAT+Q	NEAT NEAT+Q softmax NEAT+Q softmax NEAT	

Table 7: Continued.

			Compared to	
Method	Cluster	Extends	x-NEAT	ML/NE/EC
KO-NEAT	Online/Real time EC RL	NEAT	NEAT	
cgNEAT	Online/Real time	NEAT		
Online NEAT	Online/Real time RL NNs with Transfer Learning	NEAT	NEAT rtNEAT Online rtNEAT	
Online rtNEAT	Online/Real time RL NNs with Transfer Learning	rtNEAT	NEAT rtNEAT Online NEAT	
odNEAT	Online/Real time	NEAT	rtNEAT	√
odNEATv2	Online/Real time	odNEAT	odNEAT	√
HyperNEAT	Large Topologies	CPPN-NEAT	PNEAT	
Switch HybrID	Large Topologies	FT-NEAT HyperNEAT	FT-NEAT HyperNEAT	
NEAT+Q	EC BP RL	NEAT	NEAT	√
NEAR	EC RL Different node types NNs with memory	NEAT	NEAT	√
FNS-NEATFields	Deceptive landscape EC Large Topologies	NEATFields	NEATFields	√
NS-FE-CPPN-NEAT	Deceptive landscape EC Different node types	CPPN-NEAT	NS-CPPN-NEAT CPPN-NEAT	√
PIGEON	EC	NEAT PSO	NEAT	√
L-NEAT	BP	NEAT	NEAT	
DeepHyperNEAT	BP Deep NNs	HyperNEAT	HyperNEAT	√
EXACT	BP Different node types Deep NNs	NEAT	EXACTv1	√
RL-SANE	RL	BS-NEAT	RL-SANE	
NEAT-RAC-PGS	RL	NEAT	NEAT	
NEAT-FLEX	UL	NEAT		√
ES-HyperNEAT	Automatic Substrate Configuration Large Topologies	HyperNEAT	HyperNEAT	

Table 7: Continued.

			Compared to	
Method	Cluster	Extends	x-NEAT	ML/NE/EC
ES-HyperNEAT-LEO	Automatic Substrate Configuration Large Topologies Modular NNs	HyperNEAT-LEO ES-HyperNEAT	HyperNEAT-LEO	
Adaptive ES-HyperNEAT	Automatic Substrate Configuration Large Topologies Plastic NNs	Adaptive HyperNEAT ES-HyperNEAT	ES-HyperNEAT	
MSS-HyperNEAT	Automatic Substrate Configuration Large Topologies	HyperNEAT	HyperNEAT	
Modular NEAT	Modular NNs Large Topologies	NEAT	NEAT	
HyperNEAT-LEO	Modular NNs Large Topologies	HyperNEAT	HyperNEAT	
MFF-NEAT	Modular NNs	NEAT	NEAT	
HyperNEAT-CCT	Modular NNs Large Topologies	HyperNEAT-LEO	HyperNEAT HyperNEAT-GS	√
MB-HyperNEAT	Modular NNs Large Topologies	HyperNEAT MM-NEAT	Multiagent HyperNEATv2 HyperNEAT	
NEAT-CTRNN	NNs with memory Different node types	NEAT	NEAT	√
τ-NEAT	NNs with memory	NEAT	NEAT	
τ-HyperNEAT	NNs with memory Large topologies	τ-NEAT	HyperNEAT	
NEAT-LSTM	NNs with memory Different node types	NEAT	NEAT-RNN NEAT-LSTM-IM	
TL-CPPN-NEAT	NNs with Transfer Learning Different node types	CPPN-NEAT	CPPN-NEAT	
PLPS-NEAT-TL	NNs with Transfer Learning	NEAT	Phased NEAT GPS-NEAT BS-NEAT	
Adaptive HyperNEAT	Plastic NNs Large Topologies	HyperNEAT		
Adaptive HyperNEATv2	Plastic NNs Large Topologies	Adaptive HyperNEAT	Adaptive HyperNEAT	
Seeded Adaptive HyperNEAT	Plastic NNs Large Topologies	Adaptive HyperNEAT Seeded HyperNEAT	Seeded HyperNEAT HyperNEAT	√
CPPN-NEAT	Different node types	NEAT		
Recurrent CPPN-NEAT	Different node types	CPPN-NEAT	CPPN-NEAT	

Table 7: Continued.

Method	Cluster	Extends	Compared to x-NEAT	Compared to ML/NE/EC
HA-NEAT	Different node types	NEAT	NEAT	
SUPG-HyperNEAT	Different node types Large Topologies	HyperNEAT	HyperNEAT	
MAP-Elites CPPN	Deceptive landscape Different node types EC	CPPN-NEAT	direct encoding MAP-Elites	
Multiagent HyperNEAT	Large Topologies	HyperNEAT	HyperNEAT	
Multiagent HyperNEATv2	Large Topologies	Multiagent HyperNEAT	Multiagent HyperNEAT	
NEATFields	Large Topologies	HyperNEAT	HyperNEAT	
Seeded HyperNEAT	Large Topologies	HyperNEAT	Seeded Adaptive HyperNEAT HyperNEAT	√

tasks should be reported and be accompanied by statistical tests to indicate whether differences in performance are significant or not. In the papers reviewed in this study, a statistical hypothesis test had been conducted in around 60% of the cases. Moreover, although it is a computationally demanding task, another good practice would be to evaluate the generalization ability of an algorithm on an independent test, either by performing k-fold cross validation in supervised learning tasks or testing on different versions of a reinforcement learning task and/or with different initial conditions. Reporting on the values of a fitness function might be valuable in specific cases but it is not necessarily useful in all scenarios, since different definitions of fitness functions can result in different outcomes. Regarding the efficiency of a method, the number of generations until convergence or until the best solution is found (maximal or minimal fitness value) should be reported as well. Reporting on the computational time can be informative but depends on the computational power of a computing machine as well as on the employed processing schemes and it does not allow for direct comparisons among different methods. x-NEAT methods that evolve the topology of an ANN, like the ones extending NEAT and the ones that automatically configure the substrate of the HyperNEAT-based methods should be additionally evaluated on the size of evolved networks, that is, the number of connections and hidden nodes. Agreeing on a set of performance metrics does not exclude the use of other extra metrics if necessary for characterizing the performance of a specific property of an algorithm. Although we stress the importance of comparisons and benchmarking, this does not mean that this is always the most important aspect of an algorithm (Stanley, 2018). Papers that propose new research directions or provide insights are of great value as well. As it is stated in Stanley (2018), "While a good algorithm is sometimes one that performs well, sometimes a good algorithm is instead one that leads to other algorithms and new frontiers."

Datasets. Although some datasets are commonly used among x-NEAT methods of the same or of different subclusters such as the XOR, the pole balancing, the retina classification problems, etc., in general x-NEAT methods belonging to the same cluster are compared on different problems. Combined with the different employed performance metrics as illustrated above, it is evident that direct comparisons among different methods become very difficult. More standardization and the use of benchmark problems is required. This article provides an overview of the datasets used in the methods included in the review but an analysis of the datasets and the proposal of a minimal set that is suitable for benchmarking should be addressed in future work.

Parameter Configuration. All x-NEAT methods depend on a large number of parameters. The majority of the methods presented in this review article report on the employed parameters, while the rest of the papers either include a small portion (e.g., the size of the population or the allowed number of generations) or do not report them at all. However, these parameters are usually configured by trial and error or by reusing previously defined default parameter settings and not by systematically searching the parameter space using intelligent optimization methods. Another issue concerns how parameters should be configured when a method is compared to the state of the art. Optimizing the parameters for the proposed method and then comparing to the old method may cause a bias toward favoring the new method. Optimizing the parameters of the old method and using these on the new method may cause less bias. On the other hand, optimizing the parameters of the two methods separately and comparing the methods on their best performance seems to be the right option. However, it is not very often observed in literature.

Comparison to the State of the Art. During the evaluation process of each paper, we observed that only one-quarter of the methods perform a (partial) comparison of the proposed NEAT-based method to another algorithm from EC or an algorithm from another field of computer science, such as ML. This is evident from Table 7. It clearly shows the trend that most of the papers follow: omitting the performance comparison of the proposed algorithm to other methods that may exist in other fields. In addition, when a method is proposed to solve a problem from a specific domain or a specific dataset, it should be compared to existing algorithms even though they belong to another category of methods, for example, to a classifier from ML or a GA from EC. Moreover, although a method is proposed to solve a specific task, it would be interesting to see how the method performs on other datasets. One of the main questions is how to select the appropriate baseline methods for comparison. The clustering proposed in this article might give an answer. Future studies that propose a new x-NEAT method should therefore use Figure 2 and Table 6 to position their new method in an existing subcluster and compare against the state-of-the-art methods of that subcluster. If a new x-NEAT method does not belong to an existing cluster, a new (sub)-cluster should be created. It is evident from Table 7 that until now many methods are proposed to deal with a problem that is tackled by another x-NEAT algorithm of the same subcluster, but no exhaustive comparisons are performed against the rest of the methods of this subcluster, only to NEAT or to the NEAT-based algorithm that is extended. This shows how important it is to be aware of the state of the art and of the clustering approach tackled by our article.

5.3 Current Perspectives

The main advantage of a systematic review is that it is performed based on a protocol that is reproducible. Moreover, deciding which papers should be included is based

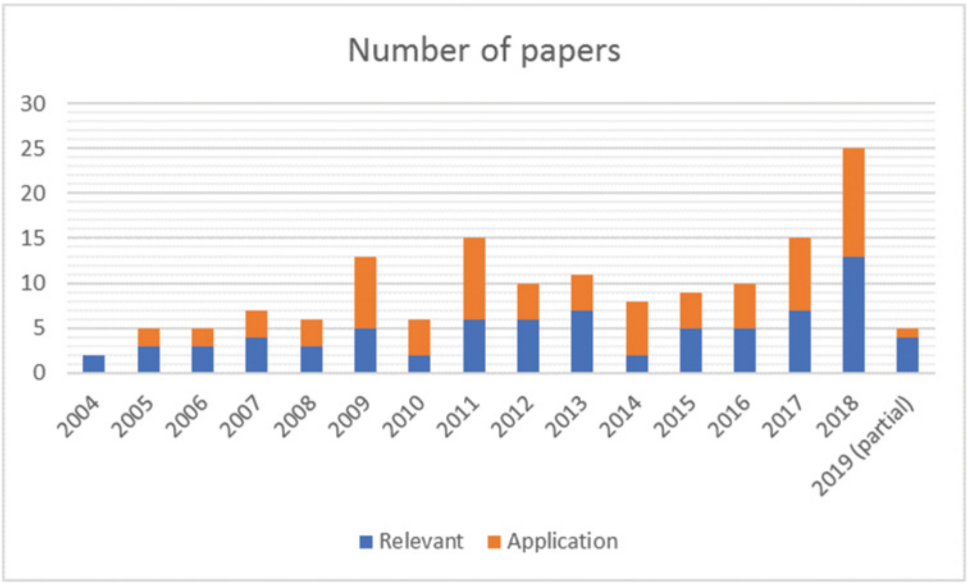

Figure 2: Histogram of the number of papers per year in WoS and Scopus.

on clearly defined criteria and evaluation scores and not on the authors' opinions. However, one limitation is that the decision of whether a paper should be excluded is based on its abstract. Therefore, papers whose abstracts or keywords did not include a reference to NEAT or even the fact that they are extending NEAT, are not included in this review. Moreover, some other methods can exist in databases other than the Web of Science or Scopus, although most of the high-quality papers are included in these two. We should therefore note that Table 6 does not include an exhaustive list of all the methods that belong to each cluster. Other methods may exist that can either be based on NEAT or employ algorithms independent of NEAT, but they are not included because they do not fulfill the criteria raised by the protocol. Additional methods have been published after the day we last searched the databases on 18 April 2018 and these are not included in this review article. A search performed in Web of Science and Scopus on the 18 July 2019 for papers published after January 2018, resulted into 35 publications. Applying only the inclusion/exclusion criteria and not the evaluation criteria resulted in 17 relevant papers. These concern x-NEAT methods belonging to the three clusters defined in the paper, for example, multiobjective optimization (Ihara and Kato, 2017; Künzel and Meyer-Nieberg, 2018; Chidambaran et al., 2018), evolving ANNs for sequential tasks with memory requirements (Merrild et al., 2018; ElSaid et al., 2019), hybrid NE methods (Nadkarni and Neves, 2018; Peng et al., 2018; McDonnell et al., 2018), optimizing ANNs with different types of nodes (Papavasileiou and Jansen, 2017; Sboev et al., 2018), optimizing deep learning structures (Desell, 2018), etc.

In the graph of Figure 2 we present the number of published papers in WoS and Scopus following NEAT in 2002. We chose to show not only the number of methods proposed to extend NEAT in the ways we present in this article and in the Appendix, but also the number of papers that are applications of an x-NEAT method to a challenging problem. Eighteen years after NEAT we can still see its relevance as the number of published papers extending or applying x-NEAT methods seems to increase.

5.4 Future Perspectives

We believe that especially nowadays in the era of deep learning, NEAT-based algorithms can contribute significantly to the configuration of the DNN. This potential is already evident in literature as hybrid methods of deep learning and NE have emerged. The papers (Schrum, 2018; Miikkulainen et al., 2019; Costa et al., 2019) are examples of x-NEAT methods for evolving DNNs. Especially, DeepNEAT and CoDeepNEAT (Miikkulainen et al., 2019) are two very promising x-NEAT methods (305 citations in Google Scholar) for optimizing the structure of DNNs. DeepNEAT follows the principles of standard NEAT, that is, an initial population of chromosomes, represented by graphs is complexified during evolution by gradually adding structure (edges and nodes) through mutation operators. Crossover is facilitated by historical markings and speciation is applied to protect innovation. The differences between NEAT and deepNEAT are: each gene in the chromosome represents a layer of a DNN, with a table describing its hyperparameters, instead of representing a node of an ANN. Moreover, the edges in the chromosome do not include weights but only indications of the layers' connectivity. During fitness evaluation the chromosome is decoded into a DNN that is trained for a fixed number of epochs. Coevolution DeepNEAT (CoDeepNEAT) is an algorithm optimizing the structure of DNNs inspired by principles of SANE (Moriarty, 1997), ESP (Gomez and Miikkulainen, 1999), and CoSyNE (Gomez et al., 2008). CoDeepNEAT generates repetitive modular structure by evolving two populations of modules and blueprints with coevolution. Each blueprint is a pointer to a module representing a small DNN. These DNNs are combined to create a larger network. We have the opinion that coevolution of two populations or within the individuals of one population is a powerful technique that NEAT-based successors should consider in the future, especially when aiming into reaching open-ended evolution (Stanley et al., 2017).

Quality Diversity (QD) algorithms also have potential applications to open-ended problems (Pugh et al., 2016; Wang et al., 2019). QD algorithms search a behavioral space to find a large set of solutions that are simultaneously diverse and high-performing (Cully and Demiris, 2017). We believe that novelty search with local competition (NSLC) (Lehman and Stanley, 2011b) and multidimensional archive of phenotype elites (MAP-Elites) (Mouret and Clune, 2015) are two QD algorithms that constitute another interesting research area where NEAT-based successors could focus on in the future.

NEAT is a complex method coming from intertwining three principles. Ablation studies showed that each component of NEAT is crucial to its performance (Stanley and Miikkulainen, 2002b). However, since then, research has provided many powerful works such as quality diversity methods (Lehman and Stanley, 2011b; Mouret and Clune, 2015), age fitness Pareto optimization (Schmidt and Lipson, 2011), and others that might raise the question of revisiting these ablation studies and investigating whether each component is really necessary when NEAT is applied in these types of cases. For example, in Mouret and Doncieux (2012), a NE algorithm that employs only the encoding part of NEAT, NEAT's structural mutation operations (no crossover) and NSGA-II (Deb et al., 2002) with an objective rewarding exploration, performs better.

6 Conclusion

Eighteen years after NEAT's invention, a plethora of successors have appeared. Our review protocol identified 60 methods that met the inclusion criteria and passed the quality assessment threshold. These methods allow us to identify relevant features, perform real time and/or online evolution, optimize multiple objectives, accelerate the computational performance, increase the interpretability of the evolved networks, and allow applications on search spaces with issues including high dimensionality and deceptive landscapes. NEAT is extended by modifications that include: introducing new mutation operators, evolving different types of neural nodes, adding new fields in NEAT's direct encoding, introducing new types of encoding (e.g., developmental), applying different selection strategies, changing the way by which the population is evolved (e.g., evolving subnets, coevolving two populations), etc. As supplementary material, readers can find a graphical tool on our website[3] that uses the 18 subclusters defined in this article to guide them in choosing a method that satisfies their needs. This article and the graphical tool can be therefore used as an index to the Appendix of this article where readers can find the description of the method and its reference to the original paper.

To facilitate benchmarking and comparisons, we believe that methods belonging to the same subcluster should be evaluated on the same tasks and should report on the same performance metrics. As a future work, it would be relevant to categorize the datasets/tasks of each subcluster to propose a set of benchmark tasks. Combined with the outcome of this article, researchers will then have all the relevant information to benchmark and compare a new method using the same state-of-the-art benchmark test sets and the same performance metrics. Finally, we hope that with this article we can stimulate similar review papers in other areas of EC, for example, in multiobjective optimization for NSGA-II (Deb et al., 2002) or in Evolutionary Strategy for CMA-ES (Hansen and Ostermeier, 2001).

Acknowledgment

Evgenia Papavasileiou was funded by a PhD grant from the Research Foundation Flanders (FWO) (scholarship numbers 1S34016N, 1S34018N).

References

Auerbach, J. E., and Bongard, J. C. (2010). Dynamic resolution in the co-evolution of morphology and control. In *Artificial Life XII: Proceedings of the Twelfth International Conference on the Synthesis and Simulation of Living Systems*, pp. 451–458.

Auerbach, J. E., and Bongard, J. C. (2011). Evolving complete robots with cppn-NEAT: The utility of recurrent connections. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pp. 1475–1482.

Azam, F. (2000). Biologically inspired modular neural networks. PhD thesis, Virginia Tech.

Bahçeci, E., and Miikkulainen, R. (2008). Transfer of evolved pattern-based heuristics in games. In *2008 IEEE Symposium on Computational Intelligence and Games*, pp. 220–227.

Bettany-Saltikov, J. (2010). Learning how to undertake a systematic review: Part 1. *Nursing Standard*, 24(50):47–55.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Berlin: Springer.

[3]www.neatreview.online

Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. (2015). *Time series analysis: Forecasting and control*. Hoboken, NJ: John Wiley & Sons.

Boyan, J. A., and Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pp. 369–376. Cambridge, MA: MIT Press.

Branke, J., Branke, J., Deb, K., Miettinen, K., and Słowiński, R. (2008). *Multiobjective optimization: Interactive and evolutionary approaches*, Vol. 5252. Berlin: Springer Science & Business Media.

Caamaño, P., Salgado, R., Bellas, F., and Duro, R. J. (2016). Introducing synaptic delays in the NEAT algorithm to improve modelling in cognitive robotics. *Neural Processing Letters*, 43(2):479–504.

Cardamone, L., Loiacono, D., and Lanzi, P. L. (2009). On-line neuroevolution applied to the open racing car simulator. In *2009 IEEE Congress on Evolutionary Computation*, pp. 2622–2629.

Cardamone, L., Loiacono, D., and Lanzi, P. L. (2010). Learning to drive in the open racing car simulator using online neuroevolution. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(3):176–190.

Chatzidimitriou, K. C., and Mitkas, P. A. (2013). Adaptive reservoir computing through evolution and learning. *Neurocomputing*, 103:198–209.

Chen, L., and Alahakoon, D. (2006). Neuroevolution of augmenting topologies with learning for data classification. In *International Conference on Information and Automation*, pp. 367–371.

Chidambaran, S., Behjat, A., and Chowdhury, S. (2018). Multi-criteria evolution of neural network topologies: Balancing experience and performance in autonomous systems. In *ASME 2018 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*. American Society of Mechanical Engineers Digital Collection.

Clune, J., Beckmann, B. E., Ofria, C., and Pennock, R. T. (2009). Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *2009 IEEE Congress on Evolutionary Computation*, pp. 2764–2771.

Clune, J., Beckmann, B. E., Pennock, R. T., and Ofria, C. (2009). Hybrid: A hybridization of indirect and direct encodings for evolutionary computation. In *European Conference on Artificial Life*, pp. 134–141.

Clune, J., Mouret, J.-B., and Lipson, H. (2013). The evolutionary origins of modularity. In *Proceedings of the Royal Society B: Biological Sciences*, 280(1755):20122863.

Clune, J., Stanley, K. O., Pennock, R. T., and Ofria, C. (2011). On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 15(3): 346.

Costa, V., Lourenço, N., Correia, J., and Machado, P. (2019). Coegan: Evaluating the coevolution effect in generative adversarial networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 374–382.

Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B. (2015). Robots that can adapt like animals. *Nature*, 521(7553):503–507.

Cully, A., and Demiris, Y. (2017). Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259.

D'Ambrosio, D. B., Gauci, J., and Stanley, K. O. (2014). HyperNEAT: The first five years. In *Growing adaptive machines*, pp. 159–185. Berlin: Springer.

D'Ambrosio, D. B., Lehman, J., Risi, S., and Stanley, K. O. (2011). Task switching in multirobot learning through indirect encoding. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2802–2809.

D'Ambrosio, D. B., and Stanley, K. O. (2008). Generative encoding for multiagent learning. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pp. 819–826.

Darnell, J. E., and Doolittle, W. (1986). Speculations on the early course of evolution. In *Proceedings of the National Academy of Sciences*, 83(5):1271–1275.

de Jong, E. D. (2004). Towards a bounded Pareto-coevolution archive. In *Congress on Evolutionary Computation*, Vol. 2, pp. 2341–2348.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Desell, T. (2017a). Developing a volunteer computing project to evolve convolutional neural networks and their hyperparameters. In *International Conference on e-Science (e-Science)*, pp. 19–28.

Desell, T. (2017b). Large scale evolution of convolutional neural networks using volunteer computing. Retrieved from arXiv:1703.05422.

Desell, T. (2018). Accelerating the evolution of convolutional neural networks with node-level mutations and epigenetic weight initialization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 157–158.

Ding, S., Li, H., Su, C., Yu, J., and Jin, F. (2013). Evolutionary artificial neural networks: A review. *Artificial Intelligence Review*, pp. 1–10.

Dinh, H. Q., Aubert, N., Noman, N., Fujii, T., Rondelez, Y., and Iba, H. (2014). An effective method for evolving reaction networks in synthetic biochemical systems. *IEEE Transactions on Evolutionary Computation*, 19(3):374–386.

Drchal, J., and Snorek, M. (2012). Distance measures for hyperGP with fitness sharing. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, pp. 545–552.

Drchal, J., and Šnorek, M. (2013). Genetic programming of augmenting topologies for hypercube-based indirect encoding of artificial neural networks. In *Soft computing models in industrial and environmental applications*, pp. 63–72. Berlin: Springer.

D'Silva, T., Janik, R., Chrien, M., Stanley, K. O., and Miikkulainen, R. (2005). Retaining learned behavior during real-time neuroevolution. In *AIIDE*, pp. 39–44.

ElSaid, A., Benson, S., Patwardhan, S., Stadem, D., and Desell, T. (2019). Evolving recurrent neural networks for time series data prediction of coal plant parameters. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pp. 488–503.

Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution: From architectures to learning. *Evolutionary Intelligence*, 1(1):47–62.

Gallego-Durán, F. J., Molina-Carmona, R., and Llorens-Largo, F. (2013). Experiments on neuroevolution and online weight adaptation in complex environments. In *Conference of the Spanish Association for Artificial Intelligence*, pp. 131–138.

Games, E. (2010). Galactic arms race. Retrieved from http://gar.eecs.ucf.edu

Gauci, J., and Stanley, K. (2007). Generating large-scale neural networks through discovering geometric regularities. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pp. 997–1004.

Genesereth, M., Love, N., and Pell, B. (2005). General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2): 62.

Gomez, F., and Miikkulainen, R. (1997). Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3–4):317–342.

Gomez, F., and Miikkulainen, R. (1998). 2-d pole balancing with recurrent evolutionary networks. In *International Conference on Artificial Neural Networks*, pp. 425–430.

Gomez, F., Schmidhuber, J., and Miikkulainen, R. (2008). Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9(May):937–965.

Gomez, F. J., and Miikkulainen, R. (1999). Solving non-Markovian control tasks with neuroevolution. In *IJCAI*, Vol. 99, pp. 1356–1361.

Green, C. (2004). Phased searching with NEAT: Alternating between complexification and simplification. Unpublished manuscript.

Grisci, B., and Dorn, M. (2017). NEAT-flex: Predicting the conformational flexibility of amino acids using neuroevolution of augmenting topologies. *Journal of Bioinformatics and Computational Biology*, 1750009.

Haasdijk, E., Eiben, A., and Karafotias, G. (2010). On-line evolution of robot controllers by an encapsulated evolution strategy. In *Congress on Evolutionary Computation*, pp. 1–7.

Hagg, A., Mensing, M., and Asteroth, A. (2017). Evolving parsimonious networks by mixing activation functions. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 425–432.

Haggett, S. J., and Chu, D. F. (2009). Evolving novelty detectors for specific applications. *Neurocomputing*, 72(10):2392–2405.

Hansen, N., and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195.

Hardwick-Smith, W., Peng, Y., Chen, G., Mei, Y., and Zhang, M. (2017). Evolving transferable artificial neural networks for gameplay tasks via NEAT with phased searching. In *Australasian Joint Conference on Artificial Intelligence*, pp. 39–51.

Hastings, E. J., Guha, R. K., and Stanley, K. O. (2009). Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263.

Hemmingway, P., and Brereton, N. (2009). What is a systematic review? *Hayward Medical Communications*, 4.

Horn, J., and Goldberg, D. E. (1995). Genetic algorithm difficulty and the modality of fitness landscapes. In *Foundations of genetic algorithms*, Vol. 3, pp. 243–269. Amsterdam: Elsevier.

Hornby, G. S., Lipson, H., and Pollack, J. B. (2003). Generative representations for the automated design of modular physical robots. *IEEE Transactions on Robotics and Automation*, 19(4):703–719.

Hosoya, T., Baccus, S. A., and Meister, M. (2005). Dynamic predictive coding by the retina. *Nature*, 436(7047): 71.

Huizinga, J., Clune, J., and Mouret, J.-B. (2014). Evolving neural networks that are both modular and regular: HyperNEAT plus the connection cost technique. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 697–704.

Ihara, K., and Kato, S. (2017). Neuro-evolutionary approach to multi-objective optimization in one-player mahjong. In *International Conference on Network-Based Information Systems*, pp. 492–503.

Inden, B. (2008). Neuroevolution and complexifying genetic architectures for memory and control tasks. *Theory in Biosciences*, 127(2):187–194.

Inden, B., Jin, Y., Haschke, R., and Ritter, H. (2012). Evolving neural fields for problems with large input and output spaces. *Neural Networks*, 28:24–39.

Inden, B., Jin, Y., Haschke, R., Ritter, H., and Sendhoff, B. (2013). An examination of different fitness and novelty based selection methods for the evolution of neural networks. *Soft Computing*, 17(5):753–767.

James, D., and Tucker, P. (2004). A comparative analysis of simplification and complexification in the evolution of neural network topologies. In *Proceedings of Genetic and Evolutionary Computation Conference*.

Kalyanakrishnan, S., Liu, Y., and Stone, P. (2006). Half field offense in robocup soccer: A multiagent reinforcement learning case study. In *Robot Soccer World Cup*, pp. 72–85. Berlin: Springer.

Karakovskiy, S., and Togelius, J. (2012). The Mario AI benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):55–67.

Kashtan, N., and Alon, U. (2005). Spontaneous evolution of modularity and network motifs. In *Proceedings of the National Academy of Sciences*, 102(39):13773–13778.

Keele, S. (2007). *Guidelines for performing systematic literature reviews in software engineering*. Technical Report Version 2.3. EBSE.

Kofod-Petersen, A. (2012). How to do a structured literature review in computer science. Version 0.1.

Kohl, N., and Miikkulainen, R. (2009). Evolving neural networks for strategic decision-making problems. *Neural Networks*, 22(3):326–337.

Kohl, N., and Miikkulainen, R. (2012). An integrated neuroevolutionary approach to reactive control and high-level strategy. *IEEE Transactions on Evolutionary Computation*, 16(4):472–488.

Kohl, N. F. (2009). Learning in fractured problems with constructive neural network algorithms. PhD thesis, Department of Computer Sciences, University of Texas at Austin.

Krčah, P. (2012). Effects of speciation on evolution of neural networks in highly dynamic environments. In *International Conference on Learning and Intelligent Optimization*, pp. 425–430.

Künzel, S., and Meyer-Nieberg, S. (2018). Evolving artificial neural networks for multi-objective tasks. In *International Conference on the Applications of Evolutionary Computation*, pp. 671–686.

Le Borgne, Y.-A., Santini, S., and Bontempi, G. (2007). Adaptive model selection for time series prediction in wireless sensor networks. *Signal Processing*, 87(12):3010–3020.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y., Cortes, C., and Burges, C. J. (1998). The MNIST database of handwritten digits, 1998. Retrieved from http://yann.lecun.com/exdb/mnist

Lehman, J., and Stanley, K. O. (2011a). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223.

Lehman, J., and Stanley, K. O. (2011b). Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pp. 211–218.

Liu, Y., and Yao, X. (1996). A population-based learning algorithm which learns both architectures and weights of neural networks. *Chinese Journal of Advanced Software Research*, 3:54–65.

Loscalzo, S., Wright, R., Acunto, K., and Yu, L. (2012). Sample aware embedded feature selection for reinforcement learning. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, pp. 887–894.

Loscalzo, S., Wright, R., and Yu, L. (2015). Predictive feature selection for genetic policy search. *Autonomous Agents and Multi-Agent Systems*, 29(5):754–786.

Maley, C. C. (1999). Four steps toward open-ended evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Vol. 2, pp. 1336–1343.

Mangasarian, O. L., and Wolberg, W. H. (1990). *Cancer diagnosis via linear programming*. Technical Report. University of Wisconsin–Madison Department of Computer Sciences.

Maniezzo, V. (1994). Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions on Neural Networks*, 5(1):39–53.

Manning, T., and Walsh, P. (2012). Automatic task decomposition for the neuroevolution of augmenting topologies (NEAT) algorithm. In *European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, pp. 1–12.

Marzullo, A., Stamile, C., Terracina, G., Calimeri, F., and Van Huffel, S. (2017). A tensor-based mutation operator for neuroevolution of augmenting topologies (NEAT). In *Congress on Evolutionary Computation*, pp. 681–687.

McDonnell, T., Andoni, S., Bonab, E., Cheng, S., Choi, J.-H., Goode, J., Moore, K., Sellers, G., and Schrum, J. (2018). Divide and conquer: Neuroevolution for multiclass classification. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 474–481.

Merrild, J., Rasmussen, M. A., and Risi, S. (2018). HyperNTM: Evolving scalable neural turing machines through HyperNEAT. In *International Conference on the Applications of Evolutionary Computation*, pp. 750–766.

Methenitis, G., Hennes, D., Izzo, D., and Visser, A. (2015). Novelty search for soft robotic space exploration. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 193–200.

Miguel, C. G., da Silva, C. F., and Netto, M. L. (2008). Structural and parametric evolution of continuous-time recurrent neural networks. In *Tenth Brazilian Symposium on Neural Networks*, pp. 177–182.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al. (2019). Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*, pp. 293–312. Amsterdam: Elsevier.

Monroy, G. A., Stanley, K. O., and Miikkulainen, R. (2006). Coevolution of neural networks using a layered Pareto archive. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pp. 329–336.

Montana, D. J., and Davis, L. (1989). Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Vol. 1, pp. 762–767.

Moriarty, D. E. (1997). Symbiotic evolution of neural networks in sequential decision tasks. PhD thesis, University of Texas at Austin.

Moriarty, D. E., and Miikkulainen, R. (1995). Discovering complex Othello strategies through evolutionary neural networks. *Connection Science*, 7(3-1):195–210.

Moriarty, D. E., and Miikkulainen, R. (1996). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22(1-3):11–32.

Morse, G., Risi, S., Snyder, C. R., and Stanley, K. O. (2013). Single-unit pattern generators for quadruped locomotion. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, pp. 719–726.

Mouret, J.-B., and Clune, J. (2015). Illuminating search spaces by mapping elites. arXiv:1504.04909.

Mouret, J.-B., and Doncieux, S. (2012). Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary Computation*, 20(1):91–133.

Nadkarni, J., and Neves, R. F. (2018). Combining neuroevolution and principal component analysis to trade in the financial markets. *Expert Systems with Applications*, 103:184–195.

Nelder, J. A., and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, 7(4):308–313.

Newman, M. E. (2006). Modularity and community structure in networks. In *Proceedings of the National Academy of Sciences*, 103(23):8577–8582.

Papavasileiou, E., and Jansen, B. (2017). The importance of the activation function in neuroevolution with FS-NEAT and FD-NEAT. In *2017 IEEE Symposium Series on Computational Intelligence*, pp. 1–7.

Peng, Y., Chen, G., Singh, H., and Zhang, M. (2018). Neat for large-scale reinforcement learning through evolutionary feature learning and policy gradient search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 490–497.

Peng, Y., Chen, G., Zhang, M., and Mei, Y. (2017). Effective policy gradient search for reinforcement learning through NEAT based feature extraction. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pp. 473–485.

Potter, M. A., and Jong, K. A. D. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29.

Prechelt, L. P., and Informatik, F. F. (1994). *A set of neural network benchmark problems and benchmarking rules*. Technical Report. Freie Universität, Berlin.

Pugh, J. K., Soros, L. B., and Stanley, K. O. (2016). Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40.

Pugh, J. K., Soros, L. B., Szerlip, P. A., and Stanley, K. O. (2015). Confronting the challenge of quality diversity. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 967–974.

Pugh, J. K., and Stanley, K. O. (2013). Evolving multimodal controllers with HyperNEAT. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, pp. 735–742.

Purdie, N., Lucas, E., and Talley, M. (1992). Direct measure of total cholesterol and its distribution among major serum lipoproteins. *Clinical Chemistry*, 38(9):1645–1647.

Radcliffe, N. J. (1993). Genetic set recombination and its application to neural network topology optimisation. *Neural Computing & Applications*, 1(1):67–90.

Radding, C. M. (1982). Homologous pairing and strand exchange in genetic recombination. *Annual Review of Genetics*, 16(1):405–437.

Rainey, K., and Stastny, J. (2011). Object recognition in ocean imagery using feature selection and compressive sensing. In *2011 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, pp. 1–6.

Rawal, A., and Miikkulainen, R. (2016). Evolving deep LSTM-based memory networks using an information maximization objective. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 501–508.

Reeder, J., Miguez, R., Sparks, J., Georgiopoulos, M., and Anagnostopoulos, G. (2008). Interactively evolved modular neural networks for game agent control. In *2008 IEEE Symposium on Computational Intelligence and Games*, pp. 167–174.

Reisinger, J., Bahceci, E., Karpov, I., and Miikkulainen, R. (2007). Coevolving strategies for general game playing. In *2007 IEEE Symposium on Computational Intelligence and Games*, pp. 320–327.

Reisinger, J., Stanley, K. O., and Miikkulainen, R. (2004). Evolving reusable neural modules. In *Genetic and Evolutionary Computation Conference*, pp. 69–81.

Risi, S., and Stanley, K. O. (2010). Indirectly encoding neural plasticity as a pattern of local rules. In *International Conference on Simulation of Adaptive Behavior*, pp. 533–543.

Risi, S., and Stanley, K. O. (2012a). An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial Life*, 18(4):331–363.

Risi, S., and Stanley, K. O. (2012b). A unified approach to evolving plasticity and neural geometry. In *2012 International Joint Conference on Neural Networks*, pp. 1–8.

Risi, S., and Stanley, K. O. (2014). Guided self-organization in indirectly encoded and evolving topographic maps. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 713–720.

Risi, S., and Togelius, J. (2015). Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*. Retrieved from arXiv:1410.7326.

Rosin, C. D., and Belew, R. K. (1997). Coevolutionary search among adversaries. PhD thesis, Citeseer.

Sboev, A., Serenko, A., Rybka, R., Vlasov, D., and Filchenkov, A. (2018). Estimation of the influence of spiking neural network parameters on classification accuracy using a genetic algorithm. *Procedia Computer Science*, 145:488–494.

Schmidt, M., and Lipson, H. (2011). Age-fitness Pareto optimization. In *Genetic programming theory and practice VIII*, pp. 129–146. Berlin: Springer.

Schrum, J. (2018). Evolving indirectly encoded convolutional neural networks to play tetris with low-level features. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 205–212.

Schrum, J., Lehman, J., and Risi, S. (2016). Using indirect encoding of multiple brains to produce multimodal behavior. Retrieved from arXiv:1604.07806.

Schrum, J., and Miikkulainen, R. (2016). Solving multiple isolated, interleaved, and blended tasks through modular neuroevolution. *Evolutionary Computation*, 24(3):459–490.

Sigal, N., and Alberts, B. (1972). Genetic recombination: The nature of a crossed strand-exchange between two homologous dna molecules. *Journal of Molecular Biology*, 71(3):789–793.

Silva, F., Correia, L., and Christensen, A. L. (2016). Leveraging online racing and population cloning in evolutionary multirobot systems. In *European Conference on the Applications of Evolutionary Computation*, pp. 165–180.

Silva, F., Urbano, P., Correia, L., and Christensen, A. L. (2015). odNEAT: An algorithm for decentralised online evolution of robotic controllers. *Evolutionary Computation*, 23(3):421–449.

Silva, O., Sigel, P., and Escobar, M. J. (2017). Time delays in a HyperNEAT network to improve gait learning for legged robots. In *2017 International Joint Conference on Neural Networks*, pp. 4222–4228.

Singh, S. P., and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1–3):123–158.

Smith, R. (2001). Open dynamics engine (ODE) physics simulator. Retrieved from http://www.ode.org

Sohn, H., Worder, K., and Farrar, C. R. (2001). *Novelty detection under changing environmental conditions*. Technical Report, Los Alamos National Lab, NM.

Standish, R. K. (2003). Open-ended artificial evolution. *International Journal of Computational Intelligence and Applications*, 3(02):167–175.

Stanley, K. O. (2006). Exploiting regularity without development. In *Proceedings of the AAAI Fall Symposium on Developmental Systems*, p. 37.

Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162.

Stanley, K. O. (2018). Art in the sciences of the artificial. *Leonardo*, 51(2):165–172.

Stanley, K. O., Bryant, B. D., and Miikkulainen, R. (2005). Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668.

Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35.

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212.

Stanley, K. O., Karpov, I., Miikkulainen, R., and Gold, A. (2006). The NERO video game. In *AIIDE*, pp. 151–152.

Stanley, K. O., Lehman, J., and Soros, L. (2016). The open racing car simulator. Retrieved from http://torcs.sourceforge.net

Stanley, K. O., Lehman, J., and Soros, L. (2017). Open-endedness: The last grand challenge you've never heard of. Retrieved from https://www.oreilly.com/radar/open-endedness-the-last-grand-challenge-youve-never-heard-of/

Stanley, K. O., and Miikkulainen, R. (2002a). The dominance tournament method of monitoring progress in coevolution. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, pp. 242–248.

Stanley, K. O., and Miikkulainen, R. (2002b). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

Stanley, K. O., and Miikkulainen, R. (2004). Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100.

Stein, G., and Gonzalez, A. J. (2010). Building high-performing human-like tactical agents through observation and experience. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(3):792–804.

Stein, G., Gonzalez, A. J., and Barham, C. (2015). Combining NEAT and PSO for learning tactical human behavior. *Neural Computing and Applications*, 26(4):747–764.

Stone, P., Kuhlmann, G., Taylor, M. E., and Liu, Y. (2005). Keepaway soccer: From machine learning testbed to benchmark. In *Robot Soccer World Cup*, pp. 93–105. Berlin: Springer.

Sutton, R. S., and Barto, A. G. (1998). *Reinforcement learning: An introduction*, Vol. 1. Cambridge, MA: MIT Press.

Tan, M., Deklerck, R., Cornelis, J., and Jansen, B. (2013). Phased searching with NEAT in a time-scaled framework: Experiments on a computer-aided detection system for lung nodules. *Artificial Intelligence in Medicine*, 59(3):157–167.

Tan, M., Hartley, M., Bister, M., and Deklerck, R. (2009). Automated feature selection in neuroevolution. *Evolutionary Intelligence*, 1(4):271–292.

Tan, M., Pu, J., and Zheng, B. (2014). Optimization of network topology in computer-aided detection schemes using phased searching with NEAT in a time-scaled framework. *Cancer Informatics*, 13:CIN–S13885.

Tarapore, D., Clune, J., Cully, A., and Mouret, J.-B. (2016). How do different encodings influence the performance of the map-elites algorithm? In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 173–180.

Taylor, M. E., Whiteson, S., and Stone, P. (2006). Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pp. 1321–1328.

Téllez, R. A., Angulo, C., and Pardo, D. E. (2006). Evolving the walking behaviour of a 12 dof quadruped using a distributed neural architecture. In *International Workshop on Biologically Inspired Approaches to Advanced Information Technology*, pp. 5–19. Berlin: Springer.

Thierens, D. (2005). An adaptive pursuit strategy for allocating operator probabilities. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, pp. 1539–1546.

Thoning, K. W., Tans, P. P., and Komhyr, W. D. (1989). Atmospheric carbon dioxide at Mauna Loa Observatory: 2. Analysis of the NOAA GMCC data, 1974–1985. *Journal of Geophysical Research: Atmospheres*, 94(D6):8549–8565.

Thrun, S. B., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K., Dzeroski, S., Fahlman, S. E., Fisher, D., et al. (1991). *The monk's problems—A performance comparison of different learning algorithms*. Technical Report, CMU-CS-91-197. Computer Science Department, Carnegie Mellon University.

Timin, M. E. (1995). The robot auto racing simulator. Retrieved from http://rars.sourceforge.net

Trujillo, L., Muñoz, L., Galván-López, E., and Silva, S. (2016). NEAT genetic programming: Controlling bloat naturally. *Information Sciences*, 333:21–43.

Verbancsics, P., and Harguess, J. (2015). Image classification using generative neuro evolution for deep learning. In *IEEE Winter Conference on Applications of Computer Vision*, pp. 488–493.

Verbancsics, P., and Stanley, K. O. (2011). Constraining connectivity to encourage modularity in HyperNEAT. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pp. 1483–1490.

Walsh, W. E., Tesauro, G., Kephart, J. O., and Das, R. (2004). Utility functions in autonomic systems. In *Proceedings of the International Conference on Autonomic Computing*, pp. 70–77.

Wang, G., Cheng, G., and Carr, T. R. (2013). The application of improved neuroevolution of augmenting topologies neural network in Marcellus Shale lithofacies prediction. *Computers & Geosciences*, 54:50–65.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. (2019). Poet: Open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 142–151.

Watson, J. D. (2004). *Molecular biology of the gene*. New Delhi: Pearson Education India.

Wen, J., Li, S., Lin, Z., Hu, Y., and Huang, C. (2012). Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, 54(1):41–59.

Whiteson, S., and Stone, P. (2006a). Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7(May):877–917.

Whiteson, S., and Stone, P. (2006b). On-line evolutionary computation for reinforcement learning in stochastic domains. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pp. 1577–1584.

Whiteson, S., Stone, P., Stanley, K. O., Miikkulainen, R., and Kohl, N. (2005). Automatic feature selection in neuroevolution. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, pp. 1225–1232.

Whitley, D., Starkweather, T., and Bogart, C. (1990). Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, 14(3):347–361.

Winter, D. (2005). Magnavox Odyssey: First home video game console. Retrieved from http://www.pong-story.com/odyssey.htm

Wright, R., and Gemelli, N. (2009). Adaptive state space abstraction using neuroevolution. In *International Conference on Agents and Artificial Intelligence*, pp. 84–96.

Wright, R., Loscalzo, S., and Yu, L. (2012). *Embedded incremental feature selection for reinforcement learning*. Technical Report. Air Force Research Lab, Rome, NY.

Xu, L., Yan, P., and Chang, T. (1988). Best first strategy for feature selection. In *Ninth International Conference on Pattern Recognition*, pp. 706–708.

Yao, X. (1993). Evolutionary artificial neural networks. *International Journal of Neural Systems*, 4(03):203–222.

Yao, X. (1999). Evolving artificial neural networks. In *Proceedings of the IEEE*, 87(9):1423–1447.

Yao, X., and Liu, Y. (1998). Towards designing artificial neural networks by evolution. *Applied Mathematics and Computation*, 91(1):83–90.

Zhao, Y., Cai, H., Chen, Q., and Hu, W. (2007). Kernel-based online NEAT for keepaway soccer. In *International Conference on Life System Modeling and Simulation*, pp. 100–107.

Appendix A Detailed Answers to the Research Questions

A.1 Coevolutionary NEAT *[4] (2004)

Principles. Coevolutionary NEAT (Stanley and Miikkulainen, 2004) is applicable to open-ended problems where the final solution is not known and more sophisticated

[4]The "*" symbol will be used next to a method's title if the paper did not provide an acronym for the proposed method. In these cases the abbreviations with this symbol are names provided by us.

strategies can be continuously generated using coevolution. A coevolutionary algorithm is an EA in which an individual's fitness cannot be evaluated based on an objective measure, but instead based on interactions with the other individuals of the population. It modifies the NEAT algorithm to evolve two separate populations that are evaluated against each other. Each population is evaluated against a sample of the opponent population consisting of the champions of the best four species and eight champion solutions of the past generations that are archived into a so-called "Hall of Fame." In this way, it is guaranteed that the opponent sample consists of good and diverse solutions and that the current solutions are competitive against older solutions. NEAT's complexification ability is leveraged to allow continual coevolution, that is, finding new solutions while maintaining existing capabilities, by elaborating on existing strategies. To achieve a sequence of increasingly more sophisticated strategies, a dominance tournament method is defined (Stanley and Miikkulainen, 2002a) so that champion strategies in every generation are evaluated against previous dominant strategies.

Encoding. NEAT's encoding is used.

Evaluation. The evolved strategies are evaluated on their complexity, that is, the number of nodes and connections that exist in the dominant network averaged over the different runs. A dominance tournament method (Stanley and Miikkulainen, 2002a) is defined to evaluate the progress of evolving more sophisticated strategies. The winner strategy of a generation is considered dominant if it defeats all the previous dominant strategies. Coevolutionary NEAT is evaluated against fixed topology coevolution and simplifying coevolutionary NEAT, by direct applications in a simulated robot duel. A simplifying coevolutionary algorithm works in a way opposite a complexifying one, that is, the evolution starts with a complex network and structure is gradually removed. The algorithms are compared using the highest achieved dominance level and the generation when dominance occurred. Moreover, the champion strategy of the simplifying and fixed-topology coevolutionary algorithms is evaluated relative to the performance of coevolutionary NEAT by comparisons against its entire dominance ranking. This metric is calculated by taking the dominance level of the highest strategy of coevolutionary NEAT that is defeated by a strategy from the champion of the other coevolutionary algorithm and dividing by the total number of dominance levels. Finally, the equivalent generation in which the latter happens is also considered a metric. It is concluded that coevolutionary NEAT with complexification finds more sophisticated strategies than fixed topology coevolution and simplifying coevolutionary NEAT.

A.2 Modular NEAT (2004)

Principles. Modular NEAT (Reisinger et al., 2004) is a TWEANN coevolutionary algorithm that enables the simultaneous evolution and modularization of network topologies. It breaks the problem into subproblems by evolving neural substructures (from one neuron to whole networks) that can be reused and combined together into more complex structures. The algorithm is suitable for high-dimensional search spaces.

Modular NEAT consists of two types of populations: a *population of modules* (NEAT networks) and a *population of blueprints*. Each module is a set of links connecting abstract input, output, and hidden nodes. The set of blueprints defines how the modules can be combined. Each blueprint consists of a fixed-size list of pairs of modules and mappings from the abstract nodes to the nodes of the solution, a process called binding. Both populations are evolved together symbiotically, such as in the SANE method (Moriarty, 1997). Symbiotic evolution is defined in Moriarty (1997) as "a type of coevolution where individuals explicitly cooperate with each other and rely on the presence of other individuals for survival." The populations of modules and blueprints can evolve in different time scales and in two sequential phases, each of which include procedures of selection, crossover, mutation, and fitness evaluation.

Encoding. The module population is encoded as in NEAT, while historical markings are also assigned on the blueprint population.

Evaluation. Modular NEAT is tested in an artificial board game with two functionally different regions and varying input/output dimensionality. Its ability to produce modular networks is measured by the percentage of crosslinks that connect an input from the one half of the board game to the output of its other half. Modular NEAT is compared to NEAT and it can obtain better networks with higher average accuracy achieved in less generations and with less crosslinks.

A.3 FS-NEAT (2005)

Principles. Feature Selective NEAT (FS-NEAT) (Whiteson et al., 2005) extends NEAT by performing feature selection in addition to the learning of the network's topology and weights. In FS-NEAT, evolution starts even more minimally than in NEAT, with networks with one random input connected to one random output. As evolution progresses, new inputs, connections and hidden nodes can be added as in NEAT. FS-NEAT performs implicit feature selection by maintaining those connections that initiate from the inputs which enhance the network's performance, that is, the relevant inputs.

Encoding. NEAT's encoding is used.

Evaluation. FS-NEAT is evaluated on a RL task of racing robot cars and it is compared to NEAT in terms of fitness performance, number of generations, and size of the evolved networks. It is found that FS-NEAT can learn better networks than NEAT in less generations especially when redundant or irrelevant inputs are present in the input dataset.

A.4 rtNEAT (2005)

Principles. Real-Time NEAT (rtNEAT) (Stanley et al., 2005) allows evolving video games' agents in real time. Each agent is represented by a NEAT-evolved ANN. In this way the agents can develop increasingly complex behavior which can be changed and be improved throughout the game, guided by the player itself. The main difference with NEAT lies in the reproduction cycle; in NEAT the whole population is replaced by a new population at each generation. But, if this would happen in a real-time environment, such as in a video game, all the agents would instantly change their behavior. In order to avoid this, rtNEAT replaces only one of the worst individuals of the population with the offspring of two of the best parents. This replacement occurs continuously and becomes unnoticeable by the player. To achieve speciation in real time, NEAT's reproduction cycle is changed and a new agent is introduced into the population every n ticks of the clock, to remove the agent with the worst adjusted fitness, given he has been alive for a minimum time.

Encoding. NEAT's encoding is used.

Evaluation. rtNEAT's abilities are illustrated on a military combat game called NERO in which the user can train robots to develop skills suitable for battle. Therefore, its performance is subject to the user's perception of when the trainable agents have reached the desired skills.

A.5 rtNEATv2* (2005)

Principles. rtNEATv2 (D'Silva et al., 2005) extends rtNEAT (Stanley et al., 2005) to train agents that should show different behaviors and perform multiple tasks. Milestoning refers to keeping a pool of old individuals, drawn from the population of agents at a specific time, that know how to perform a given task. With a specific probability and tournament selection, an individual, which at the point of the pool's creation had high fitness, is selected from the pool. Another individual is selected from the current population and the two are mated in order to produce a hybrid offspring. The generated offspring has the ability to perform both tasks, as it contains genes that come from both its ancestor and its modern parent.

Encoding. NEAT's encoding is used.

Evaluation. rtNEATv2 with different parameter settings regarding the size of the pool and the probability of creating a hybrid offspring are evaluated on the NERO task and compared against rtNEAT (Stanley et al., 2005) (Section A.4). The agents are trained on two tasks in two subsequent phases and they are tested on a third phase to evaluate whether they remember the first task or not. To evaluate their performance, the distance remaining from the target, as well as on the portion of agents from the population that solved each of the two tasks successfully are used.

A.6 CPPN-NEAT (2006)

Principles. CPPN-NEAT (Stanley, 2006) is a method using NEAT to evolve Compositional Pattern Producing Networks (CPPNs). CPPNs are proposed as a new abstraction of encoding natural development by simulating it through compositions of functions based on observed gradient patterns in natural embryos. CPPNs are connected graphs of functions, therefore they resemble ANNs with nodes of different activation functions (Gaussian, sigmoid, and periodic). Because CPPNs look like ANNs, NEAT can be used to complexify them by evolution. In CPPN-NEAT, evolution is interactive, that is, during the selection step the parent population is selected by the user. In contrast to NEAT, the population of CPPNs is initialized with one hidden layer of one or two hidden nodes and speciation's similarity metric is changed to take into account the different activation functions.

Encoding. CPPN-NEAT modifies NEAT's encoding by adding a field in the node genes to specify the nodes' activation functions.

Evaluation. CPPN-NEAT's performance is not evaluated based on a metric, but based on visual interpretation of the evolved patterns.

A.7 NEAT+Q / Online NEAT+Q (2006)

Principles. NEAT+Q (Whiteson and Stone, 2006a) is an evolutionary function approximation algorithm, that is, a hybrid method using NEAT to configure the ideal ANN that can learn to represent the value estimates provided by Q-learning, a Time Difference

(TD) method. During NEAT's fitness evaluation, which is repeated for e episodes, the ANNs' weights are updated by backpropagation, with each episode starting with the weights of the last episode. Two approaches that enable the synergy between evolution and learning are applied: Lamarckian and Darwinian. With the former, any changes made during a generation are written back to the genome and inherited by the offsprings, while with the latter these changes are discarded.

NEAT+Q is further extended to online NEAT+Q for online scenarios where an agent aims at learning a good policy quickly while maximizing its reward (exploration and exploitation). For this purpose, two action selection mechanisms that can balance these two objectives are used: *ϵ-greedy* and *softmax*. These selection mechanisms are applied at the level of policies evaluation, aiming at maximizing the reward of the best policy and increasing the average reward, respectively.

Encoding. NEAT's encoding is employed.

Evaluation. The performance of NEAT+Q and online NEAT+Q is evaluated using the uniform moving average score per episode averaged over the different algorithmic runs. It was found that NEAT+Q outperforms TD methods, while online NEAT+Q improves EC's online performance.

A.8 LAPCA-NEAT (2006)

Principles. Layered Pareto Coevolution Archive NEAT (LAPCA-NEAT) (Monroy et al., 2006) is a Pareto coevolutionary algorithm that uses NEAT to evolve a population of individuals that competes against itself. Pareto coevolution is treated as a multiobjective optimization problem whose candidate solutions are known as learners and the individuals on which they are tested as testers or evaluators. In Pareto coevolution, the learners are compared in terms of Pareto dominance to a set of testers that are viewed as the objectives being optimized. LAPCA-NEAT's coevolutionary memory is a LAPCA (de Jong, 2004) that consists of nondominated learners organized in layers from past generations drawn from the species' champions set and testers that are able to discriminate between layers. This enables users to keep the number of evaluations that are required to establish a Pareto dominance low.

Encoding. NEAT's encoding is used.

Evaluation. LAPCA-NEAT is tested against other types of memories, for example, a Hall of Fame (HOF) as coevolutionary memory on the pong game. Two sets of comparisons are performed: the best of run and the best of generation to evaluate which coevolutionary memory evolves better players and which one results in better coevolutionary progress, respectively. The performance of a player is determined by the number of players it dominates, whereas the coevolutionary progress is evaluated based on the average number of wins, ties and losses. Finally, the two memories are compared based on the size of the archive they maintain. It is found that LAPCA uses less memory but it does not outperform a Hall of Fame memory in terms of the players' performance.

A.9 L-NEAT (2007)

Principles. Learning-NEAT (L-NEAT) (Chen and Alahakoon, 2006) is a hybrid learning algorithm applied to classification problems. It combines NEAT with backpropagation to benefit from global and local search, respectively. L-NEAT (Chen and

Alahakoon, 2006) employs a divide-and-conquer approach to divide each network of the population into fully connected subnetworks with only one output node corresponding to different subtasks. The final solution is generated by assembling the networks that perform best in each subtask. With this approach, the computational expensive problem of each individual's fitness evaluation is addressed. Moreover, NEAT's purpose is changed from searching the optimal solution to searching for networks that will correspond well to backpropagation, called *feasible points*. In this case, backpropagation takes place every few generations only on the NEAT-evolved networks with high fitness values.

Encoding. Since NEAT is employed during the global search, the same encoding as in NEAT is used.

Evaluation. The classification accuracy averaged over twenty runs is engaged for the evaluation of L-NEAT's (Chen and Alahakoon, 2006) performance. L-NEAT outperforms both NEAT and NEAT enhanced with a divide-and-conquer method.

A.10 Nnrg.hazel (2007)

Principles. Nnrg.hazel (Reisinger et al., 2007) is a coevolutionary algorithm based on NEAT for evolving strategies for General Game Playing (GGP). In GGP, agents can play a variety of unknown games described in first order logic. With coevolution the space of policies and multiple opponent strategies can be searched with little a priori knowledge. In nnrg.hazel, the ANNs that are evolved by NEAT are used as heuristic evaluators in a minimax partial game tree search. As in Stanley and Miikkulainen (2004), two populations of strategies are evolved and the fitness of an individual is evaluated against a group of opponents. To ensure that more sophisticated strategies will be discovered and avoid recycling behavior, the Covering Competitive Algorithm (CCA) (Rosin and Belew, 1997) is employed. This technique, similar to the dominance tournament algorithm in Stanley and Miikkulainen (2004), guarantees that monotonic progress toward some Pareto optimal solution will be made. During evolution, the individuals of the two populations are competing against a random set of opponents of the current generation but also against previous strategies. This is achieved through maintaining a set of previous dominating strategies called teachers. In each generation, each individual of a dominated population, that is, a population whose strategies are beaten by the opponent's teachers, is evaluated against the teachers of the other population in order to determine which population will dominate which.

Encoding. NEAT's encoding is used.

Evaluation. Nnrg.hazel is tested on a set of games from the GGP using only 1-ply search and it is compared against random search (also 1-ply). The performance metrics used include the average size of the teachers' set (representing the best fitness achieved) and the average match score plotted against the teacher's rank. To evaluate scalability of the 1-ply evolved heuristics to a deeper n-ply search, the evolved heuristics are used as n-ply evaluators ($n \in [1, 4]$) and their performance is measured by the average game score. Finally, the average amount of generations that each teacher with a specific role is not dominated is computed, to give an estimation of disengagement, that is, "when one population outperforms the other to the point that there is no longer a clear gradient that selection can flow" (Reisinger et al., 2007). It is found that nnrg.hazel outperforms random search but cannot scale in deeper search depths and it shows some degree of disengagement.

A.11 KO-NEAT (2007)

Principles. Kernel-based Online NEAT (KO-NEAT) (Zhao et al., 2007) is a hybrid NE algorithm for improving NEAT's online performance. It borrows an action selection mechanism from TD algorithms called Interval Estimation (Taylor et al., 2006) to take advantage of the fitness information gained in earlier generations so that more promising individuals will be assigned more episode evaluations. Moreover, KO-NEAT employs $\mu + \lambda$ evolutionary strategy to create the next generation's population from the combined μ parents and λ children based on their fitness. Finally, the offsprings' initial fitness is calculated using kernel function approximation.

Encoding. NEAT's encoding is used.

Evaluation. KO-NEAT (Zhao et al., 2007) is compared to NEAT on the keepaway soccer game (Stone et al., 2005) using the maximum fitness acquired by the best individual as performance metric. KO-NEAT was able to plateau sooner, accumulate more rewards during the same episodic evaluations and learn a better policy than NEAT. In conclusion, KO-NEAT improves NEAT's online performance and accelerates learning efficiency and speed.

A.12 NEAT-CTRNN (2008)

Principles. NEAT-CTRNN (Miguel et al., 2008) uses NEAT to evolve Continuous Time Recurrent Neural Networks (CTRNN). CTRNNs are a type of ANN with recurrent connections and nodes equipped with an internal time constant τ and whose response in the incoming neuron spike is based on differential equations. These types of nodes belong to a class of dynamic neuron models which are considered to be more realistic models of biological neurons.

Encoding. NEAT's encoding is used.

Evaluation. The algorithm is evaluated based on its generalization ability and the number of fitness evaluations. At the end of each generation, the best network is tested on a separate test with different initial conditions. The generalization score is defined as the fraction of times the network was able to solve the task. It is found that NEAT-CTRNN evolves smaller networks and requires fewer evaluations compared with other NE methods that evolve traditional ANNs.

A.13 HyperNEAT (2008)

Principles. HyperNEAT (Stanley et al., 2009) extends CPPN-NEAT (Section A.6) for designing large-scale ANNs of fixed topologies. It evolves the connectivity of ANNs by exploiting information of the geometry of the application domain. This can be achieved by defining the ANNs as substrates with input and output nodes placed in a way that is representative of their original positions. In this way, substrates with various topologies suitable for solving different problems can be defined, such as a (2D/3D) grid, a sandwich or a circular substrate.

Encoding. HyperNEAT uses developmental (indirect) encoding to map the regularities of the task's geometry onto the ANN's topology. Developmental encoding allows the reuse of the same genes for developing multiple structures. It is inspired by the biological encoding between the genome and the phenotype. NEAT is used to evolve the topology, weights, and activation functions of CPPNs which output a spatial pattern in the hyperspace. This pattern corresponds to a connectivity pattern in the substrate, that is, the CPPNs output the weight of the connection of the queried points of the substrate.

Evaluation. HyperNEAT's performance is compared to a modification of NEAT called Perceptron NEAT (PNEAT) (i.e., NEAT for evolving perceptrons) on a task of object recognition in a 2D space. As a metric, the average distance between the champion's output and the real position of the target at each generation, across all trials and runs is used. It is found that HyperNEAT is able to find better solutions and has better generalization ability than PNEAT. Moreover, HyperNEAT, in contrast to PNEAT, is able to scale to higher resolutions without the need of further evolution.

A.14 TL-CPPN-NEAT*2 (2008)

Principles. Transfer Learning CPPN-NEAT (TL-CPPN-NEAT) (Bahçeci and Miikkulainen, 2008) is a hybrid method that proposes evolving patterns as game heuristics for GGP tasks and transferring the learned patterns on other tasks. With transfer learning, the heuristics learned on a source task can be transferred to a similar, slightly more difficult target task. In TL-CPPN-NEAT, patterns are evolved by CPPN-NEAT (Section A.6) and transfer learning is achieved by using the final population evolved on a source task to initialize the population to be evolved on a target task. In contrast to CPPN-NEAT, the CPPN nodes include only sigmoid and Gaussian activation functions and a new mutation operator is introduced to change the activation function from Gaussian to sigmoid (or from sigmoid to Gaussian) without changing the innovation number of the node. This allows improving individuals that are near the optimal solution, but one of their nodes has the wrong activation function.

Encoding. The method is an extension of CPPN-NEAT, therefore it uses NEAT to evolve CPPNs and NEAT's direct encoding to encode the genome.

Evaluation. The method is tested on a single-player board game with one version serving as the source task and a more complex version as the target task. TL-CPPN-NEAT is evaluated against CPPN-NEAT (Section A.6) without transfer learning. The metrics used include the player's performance on the game, given by the ratio of successful checks and the total learning time, given by the number of generations. It is found that transfer learning is beneficial, improving the performance and the learning time.

A.15 Multiagent HyperNEAT (2008)

Principles. Multiagent HyperNEAT (D'Ambrosio and Stanley, 2008) extends HyperNEAT for applications in multiagent domains, where the agents specialize in specific tasks and therefore are characterized by separate policies. However, policies can also share common characteristics and be complementary. Multiagent HyperNEAT enables the automatic discovery of policies, as a pattern with common characteristics and variations. With multiagent HyperNEAT, there is no need to separately define each agent's policy nor rediscover the common features. This pattern of policies is produced as a function of the agents' position in the field through a single genome described in the following paragraph. Multiagent HyperNEAT is further modified by seeding the evolution with prior knowledge in the form of adding a good genome in the initial population.

Encoding. Multiagent HyperNEAT uses a modified version of CPPNs as indirect encoding, where two substrates need to be defined; the first one is an ANN substrate describing the geometry of the sensors of each agent while the second one describes the position of the agents in the field. In this way, the second substrate consists of a set of substrates whose connection weights are calculated as a function of their location within each network and within the larger substrate. Using a single genome, a heterogeneous team of agents, each represented by the same ANN substrate, can be evolved. As in HyperNEAT, a population of CPPNs is evolved that takes as inputs the positions x_i, y_i of each i agent. However, the CPPNs initial topologies are altered to include a hidden node $r(x)$ after the input node of the x_i coordinate. With this modification, the x coordinate frame is repeated for each agent in order to mark positions of each agent. In the end, a single CPPN can encode the outputs of the agents as patterns that share characteristics but also show variation.

Evaluation. Multiagent HyperNEAT with and without seeding that evolves heterogeneous groups of agents is evaluated against multiagent HyperNEAT that evolves homogeneous groups of agents (i.e. against HyperNEAT), with and without seeding. These algorithms are evaluated on a predator–prey task with a varying number of preys and a variety of topological prey formations repeated several times. The time remaining after the agents have captured all the preys is used as a performance metric, finding multiagent HyperNEAT with seeding to perform the best among the tested algorithms.

A.16 cgNEAT (2009)

Principles. Content-generating NEAT (cgNEAT) (Hastings et al., 2009) is an interactive EC method that uses a modified version of NEAT to evolve CPPNs for automatic content generation in video games, as they are played, based on each player's behavior. cgNEAT is based on NEAT but it differs in some points. To begin with, the population size is not a developer-defined parameter but it depends on the number of players in the game. Moreover, the selection of the "bad" behaving networks is done implicitly by the players' actions and when an offspring is created it is not placed immediately in the population but it stays in a temporary state from which the player can decide whether it will be added in the population or not. Finally, no speciation exists and diversity is maintained based on the player's preferences.

Encoding. The generated content includes weapons which are evolved through particle systems used in computer graphics for producing non-solid or fuzzy phenomena such as fire, smoke, water, explosions, etc. These particle systems are encoded by CPPNs and evolved with NEAT.

Evaluation. cgNEAT is evaluated on the Galactic Arms Race (GAR) multiplayer video game using statistics generated by the players and the evolved content. cgNEAT is able to evolve a variety of unique and tactically diverse weapons, but it is not compared with the state of the art.

A.17 RL-SANE (2009)

Principles. Reinforcement Learning using State Aggregation via NeuroEvolution (RL-SANE) (Wright and Gemelli, 2009) is a hybrid method combining NEAT and Sarsa(λ) (Sutton and Barto, 1998). It combines ANNs' advantage of being good function approximators, able to abstract knowledge and RL's ability to explore and exploit this knowledge. In particular, RL-SANE uses the custom ANNs, evolved by a NEAT version that allows random pruning [Blended Searching (BS-NEAT) (James and Tucker, 2004)] as a means to reduce the state space, by aggregating state information into a more compact representation, so that Sarsa(λ) can be applied to problems with large state space. The generated ANNs map inputs from a raw multidimensional space to a one-dimensional continuum consisting of different discrete states. The ANNs output the state identifier which is provided to Sarsa(λ) as an index to its Q-table which keeps track of the values of different actions taken in specific states.

Encoding. RL-SANE uses NEAT's encoding.

Evaluation. RL-SANE is compared to BS-NEAT (NEAT with structure pruning) (James and Tucker, 2004) based on the average number of time steps the most fit members of the population were able to solve two state-of-the-art RL problems. It is found that RL-SANE performs better than NEAT in both domains and evolves less complex networks. However, this is anticipated since in RL-SANE, NEAT evolves ANNs which perform only state aggregation whereas NEAT's ANNs perform both state abstraction and policy iteration which is a much more complicated task. These findings suggest that RL-SANE can scale to more complicated domains.

A.18 FD-NEAT (2009)

Principles. Feature De-selective NEAT (FD-NEAT) (Tan et al., 2009) extends NEAT in terms of performing feature selection simultaneously with optimizing the topology and the weights of ANNs. FD-NEAT differs from FS-NEAT (Section A.3) in the way feature selection is performed. FD-NEAT starts with the same minimal topology as NEAT, i.e. with all the input nodes connected to the output nodes. In order to perform feature selection, a new mutation operator is introduced that is able to drop connections initiating from the input layer. Implicit feature selection is therefore performed as the GA can drop connections from irrelevant or redundant inputs that decrease the performance and keep the relevant ones that increase it.

Encoding. NEAT's encoding is used.

Evaluation. FD-NEAT is compared to NEAT and FS-NEAT based on the maximum fitness achieved by the best network, the number of generations required to obtain this score and the number of evolved connections averaged over the different algorithmic runs. To assess the feature selection ability two metrics are used: the frequency with which the population selects the relevant inputs and the sum of the absolute values of the connections' weights that originate from each input in the population's best network. It is found that FD-NEAT achieves the maximum fitness score in less time than FS-NEAT and it offers the best compromise between the size of the evolved networks and their performance.

A.19 RBF-NEAT and Cascade-NEAT (2009)

Principles. Radial Basis Function NEAT (RBF-NEAT) and Cascade-NEAT (Kohl and Miikkulainen, 2009) are two algorithms extending NEAT, proposed to deal with fractured search spaces, where an agent's actions change discontinously among states, by capturing the problem's local features.

RBF-NEAT (Kohl and Miikkulainen, 2009) is proposed as a method that allows local adjustments to policies, therefore it is suitable for discovering solutions that are discontinuous and can change dramatically. RBF-NEAT extends NEAT by allowing two types of activation functions in the evolved ANNs: sigmoid and radial basis functions. To do that a new mutation operator called "add RBF node" is introduced which enables the introduction of nodes with Gaussian activation functions. The parameters of the Gaussian function are controlled together with all the other parameters by the GA of NEAT. The new algorithm enables exploration of the space through the normal NEAT mutation operators and in addition it introduces a bias toward local-processing structures through the RBF node mutations.

Cascade-NEAT (Kohl and Miikkulainen, 2009) constrains the evolved topologies into regular, cascade structures that can make local refinements, by adding a new mutation operator called "add cascade node." When such a node is introduced, it receives connections from all the inputs and the existing hidden nodes and is connected to all the outputs while it freezes all the preexisting connections to allow mutating only those connections that concern the recently added hidden node.

Encoding. NEAT's encoding is used.

Evaluation. The two proposed algorithms are compared against each other, against NEAT and a NEAT version that does not allow structural mutations on problems with variation. Variation is proposed (Kohl, 2009) as a means to estimate how fractured a problem is, by treating a solution as a function and calculating its variation. On a problem of generating solutions with maximal variations, Cascade-NEAT is able to produce significantly higher variation, while RBF-NEAT is more effective when the number of inputs is low. The algorithms are also compared on other benchmark supervised and RL problems in terms of fitness. It is observed that Cascade-NEAT and RBF-NEAT have significantly higher performance than NEAT; however, their scores decrease as the variation of the problem increases.

A.20 MO-NEAT (2009)

Principles. Multiobjective NEAT (MO-NEAT) (Haggett and Chu, 2009) is a NEAT version for dealing with problems with many conflicting objectives. MO-NEAT differs from NEAT as the fitness function assigns one value for each objective and fitness sharing is calculated for each objective. Moreover, it uses the NSGA-II algorithm (Deb et al., 2002) as the parent selection mechanism for reproducing, while the offsprings are assigned to species according to the value of the shared fitness of two objectives. MO-NEAT evolves ANNs to act as novelty detectors. Any significant deviation in the system's behavior is defined as novelty. The evolved ANNs adopt principles from Dynamic Predictive Coding (DPC) (Hosoya et al., 2005), to allow online learning by continuously updating the networks' weights. The hidden and output nodes can have two types of connections: connections with fixed weights determined by evolution and modifiable connections whose weights are modified by an anti-Hebbian rule determined by DPC parameters that are also subject to evolution and determined by a new mutation operator called "mutate Parameters."

Encoding. NEAT's encoding is altered by adding a new field "connectionType" to the connections' genes that determines whether a connection is modifiable or not.

Evaluation. MO-NEAT as a novelty detector is tested on two tasks which are successfully solved, while it outperforms the results in Le Borgne et al. (2007) and Sohn et al. (2001) based on metrics specific to the used datasets and on the percentage of true positives and false positives.

A.21 Adaptive HyperNEAT (2010)

Principles. Adaptive HyperNEAT (Risi and Stanley, 2010) extends HyperNEAT (Section A.13) to concurrently evolve patterns of connectivity and *learning rules* as a function of geometry, allowing the evolution of plastic ANNs. In this way, the synaptic weights of the ANNs' connections do not remain static during the networks' lifetime but they have the ability to change in response to the changing activation levels in the neurons they connect. The advantage of this method is that the rules are derived from the network as a function of geometry and they do not have to be explicitly defined.

Encoding. Adaptive HyperNEAT encodes the connectivity patterns and the learning rules with indirect encoding using three modified versions of CPPNs. In the first case, known as *iterated model*, three additional inputs are introduced in the input layer of the CPPN that represent the current connection weight w_{ij} between two nodes i and j, the presynaptic activity of node i, o_i, and the postsynaptic activity of node j, o_j. In the second case, known as *Hebbian ABC model*, the CPPN's output layer is augmented with four additional outputs in order to output not only the connection weight but also the learning rate η and three parameters A, B, C for defining the learning rule as $\Delta w_{ij} = \eta[Ao_io_j + \dot{B}o_i + Co_j]$. In the final case, only one additional output representing the learning rate η is placed on the CPPN's output layer and the rule of plasticity is given by $\Delta W_{ij} = \eta o_i\dot{o}_j$.

Evaluation. The performance of Adaptive HyperNEAT is evaluated on two navigation scenarios that require the agent to adapt in order to find the position of a reward. The employed evaluation metrics include the number of generations required to find a solution as well as the maximum fitness eventually reached. No comparisons exist between Adaptive HyperNEAT and other NE methods.

A.22 Online NEAT*2, Online rtNEAT*2 (2010)

Principles. Online NEAT (Whiteson and Stone, 2006b) and online rtNEAT (Reeder et al., 2008) are two methods for online evolution extending NEAT and rtNEAT, respectively. In contrast to offline evolution where agents are trained for once and then applied on a computer game, online evolution enables the evolution of agents with the ability to adapt. Moreover, online evolution focuses on maximizing the performance of an agent during the whole learning process in contrast to real-time evolution which focuses on quickly finding a group of agents that perform well. In onlineNEAT (Whiteson and Stone, 2006b) and online rtNEAT (Reeder et al., 2008), the fitness is evaluated on a number of episodes and it is assigned to each individual according to their performance so that better individuals are evaluated for more episodes. Toward this goal, different selection strategies from the RL domain are adopted including *ϵ-greedy search*, *softmax* and *interval-based* techniques. The authors of Cardamone et al. (2010) modify these algorithms by including an additional evaluation strategy called *ϵ-greedy improved* (Cardamone et al., 2009), that changes the fitness evaluation by averaging the fitness values over the different episodes and assigning small time slots to evaluations corresponding to only a portion of the problem instance. Moreover, in Cardamone et al. (2010), the influence of transfer learning by seeding the evolution with the best individuals evolved at different tracks of the game, is investigated.

Encoding. NEAT's encoding is used.

Evaluation. Online NEAT and online rtNEAT with different selection strategies are compared against each other, against offline NEAT and rtNEAT (Section A.4) on different tracks of a car simulator. Their evaluation is based on the average lap time of the best individuals at the end of evolution or the average lap time during the first and last laps. The champions are then tested on different tracks of two laps in total and their performance on the second lap is measured. While online rtNEAT seems to perform better than online NEAT in the beginning of evolution, at the end they do not differ statistically and onlineNEAT can generalize better. Moreover, seeding the population with good individuals accelerates the learning process and improves the overall performance.

A.23 Multiagent HyperNEATv2*2 (2011)

Principles. Multiagent HyperNEATv2 (D'Ambrosio et al., 2011) extends Multiagent HyperNEAT (D'Ambrosio and Stanley, 2008) so that agents in a multiagent environment can learn multiple policies as a function of their position (Policy Geometry, PG) and their current state (Situational Policy Geometry, SPG).

Encoding. Multiagent HyperNEATv2 uses CPPNs as indirect encoding with two different substrates; one ANN describing the individual agents and one describing the team of the agents. As in original multiagent HyperNEAT, the team substrate contains the individual substrates but now these are stacked in z dimension reflecting the position of the agent within the team (PG). Moreover, the team substrate is enhanced with another dimension S to model the different team policies (SPG). Therefore, the substrate consists of multiple stacks of agents, one for each policy. CPPNs' topologies are also changed to include two more input nodes: a node z describing the position of an agent within the team and a node S describing whether the agent should perform another task or not. By incorporating this node in the CPPN, the pattern that is generated is a function of the position of the agent within the team, its current state and its individual parameters.

Evaluation. Multiagent HyperNEATv2 is compared against its previous version (multiagent HyperNEAT) that evolves policies based on their policy geometry, on a difficult patrol and return task using both simulation and real robots. The performance of the two algorithms is evaluated in terms of whether the robots successfully fulfilled the task and the number of generations. It is found that Multiagent HyperNEATv2 was more consistent in finding solutions and robust enough to be transferred to real world scenarios.

A.24 Recurrent CPPN-NEAT* (2011)

Principles. Recurrent CPPN-NEAT (Auerbach and Bongard, 2011) is a NE method for designing the morphology of robots and the networks that control their policies. It extends CPPN-NEAT (Section A.6) by allowing recurrent connections in the evolved CPPNs, instead of only feedforward connections. Moreover, the method extends a previous CPPN-NEAT version (Auerbach and Bongard, 2010) that allows evolution to differentially optimize the size and the quantity of evolved robots' components. The CPPNs have outputs that determine the morphology of the components evolved (e.g., the radius of a spherical component and the presence of joints), the placement of sensors as well as the embedded closed loop ANNs (CTRNNs) that control the robot's motion and policies.

Encoding. NEAT's direct encoding is used to evolve the CPPNs that determine the robots' morphological components. However, the CPPNs are also used as generative encoding that output parameter values of the CTRNNs.

Evaluation. Recurrent CPPN-NEAT is compared to CPPN-NEAT without recurrency and it evolves more successful robots. The performance metrics used are the fitness function, the size of the evolved CPPNs (number of nodes, connections, forward connections, number of hidden nodes with a specific activation function), as well as several metrics and statistics concerning the evolved morphology (number of spheres, joints, distant sensors, touch sensors, etc).

A.25 HyperNEAT-LEO (2011)

Principles. HyperNEAT with Link Expression Output (HyperNEAT-LEO) (Verbancsics and Stanley, 2011) extends HyperNEAT to evolve large-scale ANNs that are characterized by modularity, on top of regularity and hierarchy, as a function of geometry.

Encoding. HyperNEAT-LEO uses CPPNs as indirect encoding, modified to have two outputs, one that delivers the weight pattern and one that determines whether the connection is expressed (LEO output). With this output, an expression pattern is generated independently of the weight patterns, separating network connectivity from weight patterns. Moreover, modularity can be supported by seeding the CPPNs' initial topologies with a hidden node of a Gaussian function that receives the difference of coordinates so that substrate connections close to each other can be encouraged.

Evaluation. HyperNEAT-LEO with and without seeded topology is tested against HyperNEAT with restricted connectivity. To achieve this, two types of thresholds are applied. When a uniform threshold is applied, a connection is expressed and assigned the weight given by the CPPN's output if the output's magnitude is higher than a predefined threshold. On the other hand, a dynamic threshold can be used. This is updated every time a connection is queried, based on the distance between the connection's corresponding nodes, encouraging locality (as remote connections should not be expressed). HyperNEAT and HyperNEAT-LEO are tested on a difficult classification problem requiring modularity and their performance is evaluated in terms of accuracy and ratio of successful algorithmic runs. It is found that HyperNEAT-LEO with and without seeded topologies outperforms HyperNEAT while it has the ability to evolve ANNs that show modularity. This is determined by visually inspecting the generated connectivity patterns of the ANNs and identifying that there are no interconnections between the different evolved modules.

A.26 IFSE-NEAT (2011)

Principles. Incremental Feature Selection Embedded in NEAT (IFSE-NEAT) (Wright et al., 2012) is a NEAT-based *embedded* feature selection method. In contrast to FS-NEAT, which searches the features' space randomly, IFSE-NEAT performs Sequential Forward Search, by adding relevant features to an ANN called *backbone*. Sequentially, each of the available features is connected to the backbone, which in the beginning has only output nodes, by zero weight direct connections to the outputs. Then, a population of ANNs with the same topology as the backbone, but with different weights, is evolved by NEAT for a specified number of generations. The champions generated for each of the available features are compared and the best one becomes the new backbone to which the remaining features will be sequentially added and NEAT will run again to evaluate their performance.

Encoding. NEAT's encoding is used.

Evaluation. ISFE-NEAT is evaluated on its ability to select a good policy, in terms of fitness function and its ability to select relevant features calculated as the ratio of the relevant features among the selected ones. It is compared against NEAT and FS-NEAT (Section A.3) on a RL problem with a higher ratio of irrelevant features. It is found that IFSE-NEAT converges to a higher fitness in fewer generations and includes a feature set with more relevant features than both NEAT and FS-NEAT. Nevertheless, IFSE-NEAT is computationally more expensive and no statistical test is performed to indicate whether the difference is significant.

A.27 NoveltyNEAT* (2011)

Principles. Novelty search-based NEAT (NoveltyNEAT) (Lehman and Stanley, 2011a) is an altered version of NEAT guiding the search by rewarding novel behavior, instead of better performance by means of fitness optimization. Difficult problems can be characterized by deceptive fitness landscapes, so modeling the fitness function to reward the intermediate stepping stones required to reach the objective can be difficult. On the other hand, search based on novelty is immune to deception and local optima, as it completely ignores the objective. NoveltyNEAT rewards a behavior whose functionality is significantly different from what has already been discovered. For this purpose, NoveltyNEAT keeps an archive of previous novel individuals to which new behaviors will be compared, driving novelty by coevolution.

A novel behavior is measured by the sparseness of a point in the behavior space defined as the average distance of that point to its k nearest neighbors. Individuals from sparse regions receive higher novelty scores. Finally, NEAT's complexification ability enables a principled search for novel behaviors and not a random walk.

Encoding. NEAT's encoding is used.

Evaluation. NoveltyNEAT performs statistically better compared to NEAT with fitness-based search and to NEAT with random selection. The performances of the algorithms are compared in terms of the average number of evaluations required to find a solution as well as the number of evolved connections at the end of the evolution.

A.28 Switch HybrID (2011)

Principles. HybrID (Clune et al., 2011) is a term used to describe a *hybridization* of indirect and direct encodings proposed to solve problems with different levels of regularity. Regularity is defined as the "compressibility of the information describing a structure" (Clune et al., 2011), which can involve symmetries and repeating modules or motifs (Hornby et al., 2003).

Encoding. Switch HybrID (Clune, Beckmann, Pennock, and Ofria, 2009) is a case of a HybrID encoding that runs HyperNEAT as indirect encoding that captures the problem's regularities and FT-NEAT as direct encoding that captures the irregularities. FT-NEAT is a simplified version of NEAT that evolves only the weights of ANNs. HyperNEAT runs for a number of generations and evolves regular ANNs which are converted to FT-NEAT's genome to be further evolved. Both HyperNEAT and FT-NEAT evolve the weights of fixed topologies ANNs, known as substrates.

Evaluation. Switch-HybrID is compared to HyperNEAT (Section A.13), FT-NEAT and NEAT on three problems characterized by different levels of regularity and irregularity. Depending on which datasets are used, different performance metrics are employed, for example, the average error, the fitness achieved by the champion network or the distance traveled by a quadruped robot. It is found that HyperNEAT performs better than the direct encodings on regular problems but not very well when the problem's irregularity increases. On the other hand, Switch-HybrID outperforms HyperNEAT on problems characterized by irregularities but matches its performance on regular problems.

A.29 MFF-NEAT (2012)

Principles. Modular Feed Forward (MFF) NEAT (MFF-NEAT) (Manning and Walsh, 2012) is a coevolutionary NEAT-based algorithm that evolves MFF Networks (MFFNs). MFFNs decompose a problem into subtasks to be solved by specialized subnetworks known as *expert networks*. Then, an intermediary called a *gating network* is used to combine the expert networks' outputs into MFFN-NEAT system's output. NEAT evolves a population of MFF-NEAT systems and a population of expert networks that are added to the MFF-NEAT systems over time. It keeps an archive of the existing expert species with a record of their fitness values, called "coverage vector," so that extinct species can be reevaluated and introduced again in the evolutionary process. New expert networks are integrated to MFF-NEAT systems by a novel *disassortative* evolutionary operator with which selection of parents with dissimilar phenotypic traits is done more frequently than would be expected in a random selection, to decrease genetic similarities within families.

Encoding. NEAT's encoding is used.

Evaluation. MFF-NEAT is tested on three classification tasks and it is compared to NEAT in terms of the best run's mean absolute error and the generation when the highest accuracy is achieved.

A.30 ES-HyperNEAT and ES-HyperNEAT-LEO (2012)

Principles. HyperNEAT (Stanley et al., 2009) determines the weights of an ANN substrate whose topology is user defined. The user not only places the input and output nodes to match the problem's geometry but also has to determine the number and location of hidden nodes. Evolvable Substrate HyperNEAT (ES-HyperNEAT) (Risi and Stanley, 2012a) is proposed to solve this issue by inferring the topography of the substrate, that is, the density and the placement of the nodes, by exploiting information stored in the connectivity pattern evolved in HyperNEAT's hypercube. ES-HyperNEAT exploits the fact that every 4D-point in the hypercube represents the weight of the connection between two 2D points on the substrate. So, by taking 2D cross sections of the hypercube and determining the connectivity of input and output nodes, ES-HyperNEAT can determine the placement of nodes. To choose which connections will be expressed and therefore which nodes will be included in the substrate, ES-HyperNEAT employs the quadtree algorithm based on the nodes' weights' *variance*. Regions of uniform weight encode little information, but areas with high variance contain more information and thus more connections should be expressed. The variance of the areas is determined by a quadtree decomposition of the weight-space that reaches higher resolution when the variance is higher. To determine the hidden nodes, ES-HyperNEAT determines the inputs' outgoing connections and outputs' ingoing connections. When all the hidden nodes are determined, only the nodes with connections to both inputs and outputs are kept and the rest are discarded.

Extending ES-HyperNEAT to ES-HyperNEAT-LEO is proposed to evolve large scale modular ANNs, by modifications in the CPPN encoding. Further extensions describe the possibility of seeding the initial topology of ES-HyperNEAT and ES-HyperNEAT-LEO with an initial good structure so that the algorithms would be less prone to local optima.

Encoding. ES-HyperNEAT's encoding is the same as HyperNEAT's. Concerning ES-HyperNEAT-LEO, CPPNs are augmented with an extra output that determines whether the CPPN's output will be expressed, as in HyperNEAT-LEO.

Evaluation. ES-HyperNEAT and its extensions are tested on a task where the agent has to switch behavior between tasks, a deceptive maze navigation problem and a problem that requires modularity. The algorithms are evaluated against HyperNEAT (Section A.13) and HyperNEAT-LEO (Section A.25) with fixed substrates of different topologies. As performance metrics the average fitness of the champion individual, the fraction of successful runs, the number of generations required to the solution as well as average number of evolved hidden nodes and connections are used. ES-HyperNEAT performs significantly better than both HyperNEAT and HyperNEAT-LEO and HyperNEAT-LEO performs better than HyperNEAT. Finally, on the difficult modular problem, ES-HyperNEAT-LEO with geometry seeding performs better than ES-HyperNEAT-LEO which outperforms ES-HyperNEAT.

A.31 DynNEAT (2012)

Principles. NEAT for Dynamic Fitness Functions (DynNEAT) (Krčah, 2012) is an extension of NEAT proposed for environments of high uncertainty, where the fitness functions' optimum is changing rapidly between generations. In NEAT the size of the species in the next generation depends on the average fitness of the individuals of the species in the current generation. However, DynNEAT takes into account the history of the species, by choosing the size of the species to be proportional of the maximum average fitness value of the individuals of the species obtained in the last t generations.

Encoding. NEAT's encoding is used.

Evaluation. DynNEAT is evaluated against NEAT, and nonspeciated GA with historical markings on three function approximation tasks of increasing uncertainty, using the maximum fitness achieved, the portion of successful runs as well as the number of species evolved as performance metrics. On the problems with static and slowly moving optimum, DynNEAT shows similar performance to NEAT; however, on the problems where the optimum is moving rapidly DynNEAT outperforms the two other algorithms.

A.32 Adaptive ES-HyperNEAT (2012)

Principles. Adaptive ES-HyperNEAT (Risi and Stanley, 2012b) combines and extends Adaptive HyperNEAT (Section A.21) and ES-HyperNEAT (Section A.30). It evolves ANNs substrates whose connectivity is a function of geometry (as in HyperNEAT), whose topography (nodes' location and density) is derived by the variance of the connectivity pattern itself (as in ES-HyperNEAT) and whose connections show plasticity, i.e. adaptation during their lifetime (as in Adaptive HyperNEAT). To sum up, Adaptive ES-HyperNEAT evolves ANNs with some properties resembling those of the natural brain: regularity, topography, heterogeneous plasticity and neuromodulation.

Encoding. As in Adaptive HyperNEAT and ES-HyperNEAT, Adaptive ES-HyperNEAT uses CPPNs as indirect encoding of the ANNs substrates. The CPPN output layer is augmented with five additional outputs in order to output a modularity output M, as well as the learning rate η and the parameters A, B, C describing the rule of plasticity, $\Delta w_{ij} = \eta[Ao_io_j + \dot{B}o_i + Co_j]$; that is, the model used is the Hebbian ABC model that was mentioned in Section A.21.

Evaluation. Adaptive ES-HyperNEAT is evaluated on a navigation task that requires adaptation and its performance is evaluated by taking the fitness averaged over different runs and the ratio of runs where a solution is found. It is shown that Adaptive ES-HyperNEAT has a better performance than ES-HyperNEAT. However, no comparisons to Adaptive HyperNEAT are performed.

A.33 NEATfields (2012)

Principles. NEATfields (Inden et al., 2012) is a multilevel NE method designed for problems with large input and output spaces, that evolves large ANNs by feeding evolution with externally specified design patterns. NEATfields evolves networks that have a three level architecture. In the highest level, a NEATfields network is NEAT-like network of fields that are connected as individual nodes are connected in a NEAT network. The intermediate level consists of fields which are grids of the basic field element and the lowest level consists of the basic field elements, that is, a 2D NEAT-evolved Recurrent Neural Network (RNN). Three types of connections can be established: local within a field element, lateral between field elements of the same field, and global between two fields.

Encoding. NEATfields uses indirect encoding to map the genome to the evolved NEATfield networks. The genotype consists of global chromosomes for describing the global connections, as well as chromosomes describing a field element's nodes and connections. The first gene in a chromosome contains information regarding the 2D size of the field and it is followed by genes describing a field element's nodes and connections. Every gene is characterized by a global reference number, as the innovation number in NEAT. The connection genes contain information of the connected nodes, the connections' weights, the reference numbers of the source and target nodes, a flag indicating whether the connection is active, a flag indicating whether a lateral connection is on or off, and a flag mapping to a specific design pattern.

Evaluation. NEATfields performance is evaluated on three high-dimensional pattern recognition problems in terms of the ratio of successful runs and the average number of evaluations in the successful runs. Different NEATfields versions with different design patterns including dehomogenization, lateral connections, and local feature identification are evaluated against each other. Finally, NEATFields, compared to HyperNEAT on two control tasks, performs better.

A.34 SNAP-NEAT (2012)

Principles. SNAP-NEAT (Kohl and Miikkulainen, 2012) is a method extending NEAT, RBF-NEAT and Cascade-NEAT (Kohl and Miikkulainen, 2009) (Section A.19) by integrating their different structural mutations. Based on *adaptive operator selection*, SNAP-NEAT allows evolution to choose whether it will use NEAT's structural mutations ("add-node" and "add connection"), RBF-NEAT's ("add RBF-node") or Cascade-NEAT's ("add cascade-node"). SNAP-NEAT modifies the Adaptive Pursuit Algorithm (APA) (Thierens, 2005), which is designed to define the probabilities of selecting an operator so that the probability of the operator with the maximal estimated reward is increased and other probabilities are decreased. In SNAP-NEAT, APA is modified to allow continuous updates on the values of the operators. This means that each time an individual is updated, regardless of whether it underwent a mutation or not, a reward signal for the operator that contributed last to the individual is generated, allowing to leverage a larger percentage of information. Moreover, since APA is sensitive to the initial conditions of the probabilities of each o_i operator (Po_i), another modification is proposed so that the probabilities Po_i are not updated in the beginning of evolution but they remain fixed and uniform. After this evaluation time passes, the information gained is used to estimate the values of each operator Qo_i which is then used to finally compute the initial operator probabilities Po_i.

Encoding. NEAT's encoding is used.

Evaluation. SNAP-NEAT is tested on problems that are highly fractured and require high-level strategies as well as on simpler problems that require reactive control. Its performance is compared to RBF-NEAT, Cascade-NEAT, NEAT and other state-of-the-art NE algorithms such as SANE (Moriarty, 1997), ESP (Gomez and Miikkulainen, 1997), CMA-ES (Hansen and Ostermeier, 2001), and others, in terms of the score metric of each problem. SNAP-NEAT's performance is comparable to the best algorithm on each domain, but it has the advantage of automatically selecting which structural mutations are best suited for each problem.

A.35 SUPG-HyperNEAT* (2013)

Principles. Single Unit Pattern Generator (SUPG)-HyperNEAT (Morse et al., 2013) is proposed to control the gaits of quadruped robots with long legs. It extends HyperNEAT in terms of evolving substrates of ANNs with a new type of neuron (SUPG). This outputs an activation pattern as a function of geometry and time that can be generated repetitively and produce oscillations. Moreover, the SUPG node consists of a timer that when triggered by an external event, resets the SUPG activation pattern so that gait imperfections in timing can be corrected. Two versions of the SUPG-HyperNEAT method are proposed depending on whether the evolution is driven by novelty search or by fitness objectives.

Encoding. SUPG uses indirect encoding of CPPNs that control the output of the SUPG node by providing the activation pattern, the connection weights, and an offset that is used in the beginning of the evolution to allow evolving a wide range of gaits. CPPNs also output parameters of the substrate's nodes, for example, the bias and receive as inputs, the location of the nodes in the substrate, and the time when the current cycle began.

Evaluation. SUPG-HyperNEAT is compared to two HyperNEAT-evolved substrates: a sine wave architecture and a CTRNN architecture, inspired by Clune, Beckmann, Ofria et al. (2009) and Téllez et al. (2006) that can evolve oscillatory gaits. The performance metrics used include the maximum distance traveled by the robot and the complexity of the evolved CPPNs (number of hidden nodes and connections). Even though all methods can evolve successful robot gaits, SUPG-HyperNEAT evolves more robust gaits that allow the robot to move for a longer period.

A.36 MSS-HyperNEAT (2013)

Principles. Multi-Spatial Substrate (MSS) HyperNEAT (Pugh and Stanley, 2013) is extending HyperNEAT (Section A.13) by determining the right configuration of the substrate via placing different input and output modalities on their own geometrically independent planes. When no geometric relationship between two groups of neurons exists, the neurons are placed into separate planes, whereas the logically related neurons are grouped together.

Encoding. MSS-HyperNEAT uses HyperNEAT's indirect endoding. However, the structure of the CPPNs is altered to include more outputs that express the connectivity among the different planes. In particular, the new CPPNs will consist of $M + N$ outputs where M is the number of planes with non-input neurons and N is the number of unique pairs between connected spaces.

Evaluation. MSS-HyperNEAT is evaluated on a maze exploration and group coordination task of multiple agents. Two MSS substrates on which the input and outputs are organized in different planes according to their functionality are compared against HyperNEAT with regular fixed substrates, called crowded substrates. The two methods are compared based on the average best fitness across different maze environments and across different runs as well as the proportion of runs that gave a good result. It is found that MSS-HyperNEAT outperforms crowded substrate HyperNEAT.

A.37 Adaptive HyperNEATv2* (2013)

Principles. Adaptive HyperNEATv2 (Gallego-Durán et al., 2013) extends Adaptive HyperNEAT (Section A.21) by defining a rule of plasticity that takes into account the presynaptic and post synaptic activities of n previous time steps.

Encoding. Adaptive HyperNEATv2 uses indirect encoding and especially in the format of the iterated CPPN model that was used in Adaptive HyperNEAT, but its architecture is modified in order to include new outputs that correspond to coefficients of the new plastic rule.

Evaluation. Adaptive HyperNEATv2 is compared to the three models used by Adaptive HyperNEAT: iterated, Hebbian ABC, and plain Hebbian. The algorithms are evaluated on a complex environment of simulated racing in terms of the race score and the number of CPU cycles on the same processor. Adaptive HyperNEATv2 is computationally less expensive than Adaptive HyperNEAT's iterated model, but it does not outperform it.

A.38 NEAR (2013)

Principles. Echo State Networks (ESNs) are a type of Recurrent Neural Networks (RNNs) from the field of Reservoir Computing. Originally, ESNs constituted a randomly generated reservoir of sparsely connected neurons. The connections among the inputs and the reservoir as well as the connections of the neurons of the reservoir are assigned once and remain fixed. The only trainable weights are the output weights which connect the reservoir to the output nodes. These can be determined through supervised learning (e.g., linear regression), RL (e.g., Sarsa temporal differences), or policy search (e.g., evolutionary computation). However, the random generation of the reservoir is considered insufficient whereas the design of a specific reservoir for a specific task seems more appropriate. Toward this purpose, Neuroevolution of Augmenting Reservoirs (NEAR) (Chatzidimitriou and Mitkas, 2013) was developed as a hybrid evolutionary-learning method which uses NEAT to evolve the reservoir of the ESNs and learning to learn the output layer. It is expected that the synergy between evolution and learning can yield better results since evolutionary computation can be used to locate a wider area wherein the global minimum is situated and local search can be used to refine the search and locate it.

NEAR is different from NEAT in terms of evolving ESNs instead of standard ANNs, starting the evolution with one neuron in the reservoir, having direct connections among the input and output nodes and performing speciation based on macroscopic features of the reservoirs instead of structural similarities. The networks are grouped in species based on a threshold on the value $\delta = \frac{c_\alpha|\rho-\rho_r|}{\rho_r} + \frac{c_\beta|D-D_r|}{D_r} + \frac{c_\gamma|N-N_r|}{N_r}$, where r corresponds to the representative of the species, ρ is the reservoir's spectral radius, N is the number of neurons in the reservoir, D is the reservoir's density, and $c_\alpha, c_\beta, c_\gamma$ predefined parameters.

At each generation, the genomes are converted to ESNs and they pass to the evaluation phase for a number of episodes. Then the output weights are adapted by using either TD learning or standard weight mutation, followed by evolution by NEAT. Both Lamarckian and Darwinian evolution are enabled. In Lamarckian evolution, the learned output weights are transferred back to the genome for further evolution whereas in Darwinian evolution the output weights are reinitialized at the beginning of each generation.

Encoding. NEAR uses direct encoding of a genome including the weights of the input connections, the output weights, the weights of the reservoir, as well as reservoir parameters such as the density and the spectral radius ρ used for the normalization of the reservoir weights.

Evaluation. NEAR is evaluated on time series and RL domains. On the time series data, two versions of NEAR, NEAR+LS and NEAR+MUT, are compared to standard ESNs with a fixed number of reservoir nodes and to a state-of-the-art algorithm for optimization of the reservoir's structure. NEAR+LS uses least squares for the learning of output weights whereas NEAT+MUT uses mutations. The performance comparisons are based on the NRMSE value on an unseen validation data. In the RL tasks, three versions of NEAR, NEAR+TD-L (TD learning and Lamarckian evolution), NEAR+TD-D (TD learning and Darwinian evolution), and NEAR+PS (policy search) are compared to ESN+TD (ESNs with TD learning) and NEAT. It is not clear from the paper whether NEAR+PS and NEAR+MUT refer to the same method since they both use mutation operators to adapt the output weights. As performance metrics, the average accrued reward over episodes by the champion network as well as the number of evaluations before a solution is found are used. The champion network is also evaluated on another set of episodes so that the generalization performance of the champion network can also be reported. In general, NEAR through learning and evolution was able to find ESNs that solved the problems, which is not always the case with ESNs with random generated reservoirs.

A.39 Layered NEAT (2013)

Principles. Layered NEAT (Wang et al., 2013) is a new algorithm that accelerates the computational time of NEAT by a) sorting of the evolved topology in layers, b) identification of recurrent connections, and c) increasing the size of the population over the course of generations. By sorting the nodes in layers, Layered NEAT introduces an efficient way of calculating the output of an ANN by first calculating the outputs of the nodes ("activating the nodes") placed in earlier layers. The identification of recurrent cycles also facilitates the aforementioned calculation as it allows for only an additional activation of the nodes that belong to the specific cycle. Moreover, Layered NEAT starts the evolution with a small population of networks whose size increases with the course of generations following a sigmoid-like function and a specified rate. Finally, Layered NEAT allows feature selection by starting the evolution with initial topologies of minimally connected inputs (in a specific dataset the number of initially connected inputs varied from 1 to 8) and allowing evolution to select the relevant inputs, which is similar to FS-NEAT (Section A.3).

Encoding. Layered NEAT changes NEAT's encoding by introducing two new fields in the node genes and one new field in the connection genes. In the node genes, the first field contains the location of a node that is updated as new nodes and connections are inserted, while the second field marks whether a node belongs to a recurrent circle. Similarly, the new field in the connection genes is used to indicate whether a connection that is added creates a cycle or not.

Evaluation. Layered NEAT is compared to NEAT on the well-known XOR problem by comparing the computational time required by the two algorithms on the same CPU. The comparison shows that when recurrent connections are allowed, Layered NEAT is four times faster than NEAT. Additionally, the average fitness value and the number of evolved connections and nodes are used as performance metrics on a multiclass classification problem, but no comparisons to NEAT are made.

A.40 FNS-NEATFields* (2013)

Principles. In Fitness- and Novelty-based Selection mechanisms-NEATfields (FNS-NEATFields) (Inden et al., 2013) the influence of different selection mechanisms on NEATfields (Section A.33) is investigated. These include tournament selection, selection mechanisms based on genotypic or behavioral similarities, and hybrid methods of both types of selection mechanisms.

Encoding. NEATfields encoding is used.

Evaluation. The different selection mechanisms are applied on benchmark problems and their performance is evaluated using the ratio of achieved fitness divided by the number of generations required to reach the fitness. Although a particular selection mechanism that outperforms all others is not found, generally, hybrid methods perform better than pure novelty-based or fitness-based methods.

A.41 Phased NEAT (2013)

Principles. Phased Searching with NEAT in a Time Scaled Framework (Phased NEAT) (Tan et al., 2013) extends NEAT by allowing both structure complexification and simplification during evolution. Phased Searching NEAT alternates between complexification (adding structure) and simplification (removing structure) that occur in every predefined number of generations. In this way, feature selection/deselection is also performed since input connections can be added and removed.

Encoding. NEAT's encoding is used.

Evaluation. Phased Searching NEAT is compared to NEAT and FD-NEAT (Section A.18) using the metrics of sensitivity and the size of the evolved networks (number of connections), both on the champion network and on the whole population at the end of the evolution. Phased searching NEAT does not perform significantly different than NEAT, but it can evolve significantly smaller and less complex networks.

A.42 HyperNEAT-CCT (2014)

Principles. HyperNEAT with Connection Cost Technique (HyperNEAT-CCT) (Huizinga et al., 2014) is extending HyperNEAT-LEO (Verbancsics and Stanley, 2011) (Section A.25) in order to produce ANNs that are modular and regular. Even though HyperNEAT-LEO can evolve modular ANNs, the authors argue that seeding the topology with the Gaussian function (HyperNEAT-GS) may not be beneficial in the long term, as it does not improve the fitness in the short term; therefore, the seed may be removed during the course of evolution. Thererefore, an extension of HyperNEAT-LEO is proposed to treat the problem as a multiobjective optimization problem. Three objectives need to be optimized; maximizing the networks' performance on a test problem, maximizing the behavioral diversity and minimizing the connectivity costs. The connection cost is calculated as the sum of the connections' squared lengths in the phenotype, a technique adopted from Clune et al. (2013). NEAT is altered to not perform crossover when evolving the CPPNs that encode the genome. Moreover, a speciation mechanism that encourages behavioral instead of genotypic diversity is used. To calculate the behavioral diversity of a network, the output for every possible input is stored in a binary vector and the average Hamming distance to the vectors of all the other networks is calculated.

Encoding. HyperNEAT-CCT uses indirect encoding by NEAT-evolved CPPNs, as described in Section A.25, without Gaussian seeding. The CPPNs output the weights and the biases of the ANN substrate as well as the pattern of connectivity expression by the LEO output whose activation function is changed to a hyperbolic tangent.

Evaluation. HyperNEAT-CCT is evaluated against HyperNEAT (Section A.13), HyperNEAT-GS, and Direct-CCT (Clune et al., 2013), on 1 to 3 problems with modularity and regularity challenges. Their ability to evolve structurally modular networks is evaluated in terms of the Q-score (Newman, 2006). To assess functional modularity two metrics are used (Clune et al., 2013): modular decomposition and number of solved subproblems. To calculate the former, the network is split up in as many splits as the maximum number of subproblems, and then whether inputs of different subproblems correspond to different modules is tested. To calculate the latter, it is tested whether for every subproblem a node exists that linearly separates its positive and negative classes. The weights and biases of the networks are encoded into an ASCII string and compressed using the Lempel–Ziv–Welch algorithm. Regularity is evaluated by the compression rate, and we evaluate it by how much fraction the string is compressed. HyperNEAT-CCT performs better than both HyperNEAT and HyperNEAT-GS and produces networks that are both modular and regular. Although Direct-CCT performs better than HyperNEAT-CCT, it lacks the ability to produce regular ANNS, which is an intrinsic property of HyperNEAT and therefore of HyperNEAT-CCT.

A.43 Seeded (Adaptive) HyperNEAT (2014)

Principles. Seeded HyperNEAT and Seeded Adaptive HyperNEAT (Risi and Stanley, 2014) are extending HyperNEAT (Stanley et al., 2009) (Section A.13) and Adaptive HyperNEAT (Risi and Stanley, 2010) (Section A.21) to evolve ANNs whose weights can be self-organized to form topographic maps of the input stimuli. The resulting ANNs exhibit the natural brain's feature of topographic maps, that is, neurons that are spatially organized to map the sensory input.

Encoding. Both algorithms use CPPNs as indirect encoding. CPPNs can be seeded with hidden nodes that output SOM connectivity and can be further evolved. CPPNs with seed can be queried to output the static weight of connections within the output substrate. CPPNs are also queried to output the weight of a connection between the input and output substrate together with its learning rate.

Evaluation. Seeded HyperNEAT and Seeded Adaptive HyperNEAT are evaluated against each other, against other seeded methods that set the weights between the input and output substrate in different ways and against unseeded HyperNEAT. The methods are tested on a line orientation task and the performance metrics used include the fitness function, the number of generations required to reach the best result, and a generalization metric testing the ratio of successfully finding the target weights within a predefined margin. It is found that the seeded methods perform better, accelerating the self-organization process and biasing it to a desired configuration. Moreover, seeded HyperNEAT performs better than seeded Adaptive HyperNEAT indicating the difficulty of learning simultaneously both the learning rate and the weight patterns.

A.44 NS-FE-CPPN-NEAT* (2015)

Principles. In Methenitis et al. (2015), two extensions of CPPN-NEAT (Section A.6) are proposed to evolve the morphology and the locomotion of soft robots for space exploration. Novelty-Search CPPN-NEAT (NS-CPPN-NEAT) guides the search by rewarding novel behaviors, while NS-FE-CPPN-NEAT combines Fitness-based Elitism (FE) with Novelty Search (NS). In this case, the best individual in terms of fitness is copied to the next generation and novel behaviors are sought.

Encoding. NS-FE-CPPN-NEAT is an extension of CPPN-NEAT, so it uses NEAT's direct encoding to evolve the CPPNs which are queried for the coordinates of each voxel of a 3D lattice describing a particular material type used to evolve the soft robots.

Evaluation. NS-FE-CPPN-NEAT is compared with CPPN-NEAT with fitness defined objectives, pure novelty search CPPN-NEAT, random search CPPN-NEAT, and a classic GA, using as metric the normalized displacement of a robot from its initial position. It is found that novelty search improves the performance and the diversity in the fitness space while contributing in a larger variety of morphologies.

A.45 Deep HyperNEAT* (2015)

Principles. In Verbancsics and Harguess (2015), two extensions of HyperNEAT (Section A.13) for application in deep classification problems are proposed. First, HyperNEAT evolves CPPNs of a feed forward ANN substrate. The difference with HyperNEAT is that in this case the outputs of the CPPNs play the role of the learned features that can be provided as inputs to a machine learning method (e.g., ANNs and KNNs) that will perform the final classification based on backward propagation. In the second case, HyperNEAT evolves CPPNs to learn Convolutional Neural Networks (CNNs) (Deep HyperNEAT) that have the topology of the LeNET5 network (LeCun et al., 1998) and will act as the deep learning classifier.

Encoding. HyperNEAT's encoding is used.

Evaluation. The method is tested on two image classification problems. Four versions of HyperNEAT are compared, HyperNEAT as a classifier that learns either ANNs or CNNs and HyperNEAT as a feature extractor using both ANNs and CNNs. For the comparisons on the MNIST dataset, the fitness function and the fraction of correct classifications are used. Moreover on the BCCT200 dataset (Rainey and Stastny, 2011) HyperNEAT as a feature extractor with CNN architecture is compared to other machine learning methods. It is found that the combination of NE with other machine learning methods is better compared to applying only NE or only machine learning algorithms.

A.46 PIGEON (2015)

Principles. Particle swarm Intelligence and Genetic algorithms for the Evolution and Optimization of Neural networks (PIGEON) (Stein et al., 2015) is a hybrid algorithm based on NEAT and Particle Swarm Optimization (PSO) for evolving agents that show tactical behavior, that is, behavior that resembles the humans' behavior when facing the same situation. In PIGEON, NEAT evolves the structure of the ANNs and PSO optimizes the connection weights based on the weights of the best individual in the species and the weights the individual had in its previous best performance. PSO is combined with NEAT in two ways: PIGEON-Chain and PIGEON-Alternate. In PIGEON-Chain, NEAT runs for a predefined number of generations evolving the structure and (suboptimal) weights that are later optimized by PSO. On the other hand, in PIGEON-Alternate, NEAT and PSO are applied in turns.

Encoding. The structure and the weights of the ANNs are encoded with NEAT encoding, while the weights optimized by PSO are represented in each particle of the PSO population by a list of real-valued numbers.

Evaluation. PIGEON is tested on two learning schemes used for developing tactical behavior: an observational and an experiential learning approach on three datasets. During observational learning, a human controls the way an entity performs a task. During this phase, data are collected so that they can be used to train the agent to mimic the human behavior. A similarity factor is calculated to evaluate how well the agent approximates the human behavior. Experiential learning resembles RL, where the agent learns how to behave based on which actions yielded higher performance.

PIGEON-Chain and PIGEON-Alternate are compared against each other and against NEAT and PSO. The similarity factor and the fitness value are used as performance metrics in observational and experiential learning, respectively. In general, PSO outperforms all the other algorithms in all the experiments of observational learning. On the other hand, NEAT and PIGEON-Alternate perform better than the other algorithms in the experiential learning tests. Although, a general conclusion cannot be drawn for all the domains; in complex domains PIGEON-Alternate performs better, faster and evolves smaller networks.

A.47 PFS-NEAT (2015)

Principles. Predictive Feature Selection NEAT (PFS-NEAT) (Loscalzo et al., 2015) is an *embedded* feature selection method for RL that uses NEAT as the genetic policy algorithm. The algorithm starts with a population of fully connected networks and identifies the relevant features by alternating NEAT-based evolution with feature selection. Feature selection is performed using the Best First subset search (Xu et al., 1988), which is a variant of Sequential Forward Search. This method is able to identify feature interaction and it is beneficial in cases where a feature is relevant only in combination with other features.

The algorithm gathers samples (states and actions) that are incorporated into a dataset D during the fitness evaluation of the whole population. The rewards obtained for each of the states are used as the class labels of these samples. Feature selection based on features scoring is then applied on the dataset D to identify the best features. The selected features are integrated in the topology of the best network through connections to all the outputs. These connections are assigned zero weights, so that the best identified policy will not degrade. The features are also integrated with the rest of the population through connections to the output nodes of random weights. Then, NEAT evolution follows for a number of generations until stagnation, that is, until the fitness of the best network does not improve above a threshold for a specified number of generations and then feature selection is applied again. The whole process repeats until a termination criterion is met.

Encoding. PFS-NEAT uses NEAT as a genetic policy search algorithm, so NEAT's encoding is used.

Evaluation. PFS-NEAT is significantly better than NEAT (Stanley and Miikkulainen, 2002b), FS-NEAT (Whiteson et al., 2005) (Section A.3), FD-NEAT (Tan et al., 2009) (Section A.18), and SAFS-NEAT (Loscalzo et al., 2012) in terms of the accumulated fitness value, the number of selected features and the ratio between relevant and irrelevant ones.

A.48 odNEAT (2015)

Principles. Online Distributed NEAT (odNEAT) (Silva et al., 2015) is a decentralized (distributed) NE algorithm for the online evolution of robots that can learn and self-adapt in long term. In online evolution the robots continuously optimize their behavior by executing the evolutionary algorithm during the execution of the task. In odNEAT two types of reproduction exist; inter-robot and intra-robot reproduction. During *inter-robot reproduction*, the robots can exchange candidate solutions by broadcasting a copy of their active genome of a competitive fitness score to their neighboring robots. When a robot receives the copy, it checks whether the genome already exists in a local list ("tabu" list), which contains the recently failed genomes. If the received genome is topologically similar to one of the genomes in this list, it does not get incorporated into the robot's internal population. If the received genome already exists in the population, then it is used for recalculating the average fitness, thus providing a more accurate estimate of the fitness. If the robot's internal population is complete, the received copy will replace a poorly performing genome. During the *intra-robot reproduction*, a top species is selected for traditional NEAT-based reproduction. In contrast to NEAT, this type of reproduction takes place only when a robot performs poorly, which is indicated by the robot's performance measure, called "virtual energy." Then, two parents selected from a top species are used to crossover and their offspring is mutated to generate the new genome. The new genome is assigned a maturation period of time which means that the new controller is given time to exhibit its performance in the deployed environment that may be characterized by different environmental conditions.

Encoding. odNEAT uses NEAT's encoding replacing the innovation markers with time stamps that function as chronological parameters, assigned to the genome when a new connection or a new node is introduced.

Evaluation. odNEAT is compared to modified versions of rtNEAT (Stanley et al., 2005) (Section A.4) and IM-($\mu + 1$) (Haasdijk et al., 2010). On the one hand, rtNEAT is a centralized method for online evolution of robotic controllers, whereas IM-($\mu + 1$) is a decentralized method not based on NEAT but following some of the NEAT principles such as speciation, complexification, and innovation markers. rtNEAT and IM-($\mu + 1$) are modified to generate an offspring only when their energy level reaches a predefined minimum threshold, and IM-($\mu + 1$)'s random assignments of innovation markers are replaced with odNEAT's timestamps.

The performances of odNEAT, rtNEAT, and IM-($\mu + 1$) are evaluated in four aspects: the number of evaluations, the fitness score, the total number of evolved connections, and nodes, which indicates the complexity of the solutions and the generalization ability. To measure the generalization ability, each task is restarted 100 times per run with different initial robots. The robotic controller is not allowed any further evolution and it is considered to generalize well if its virtual energy does not decrease to zero. In the majority of the tasks, it was found that odNEAT statistically outperforms rtNEAT and IM-($\mu + 1$) in most of the metrics, or performs as good as the other algorithms but resulting in less complex networks.

A.49 odNEATv2* (2016)

Principles. odNEAT with online racing and population cloning abilities (odNEATv2) (Silva et al., 2016) is proposed as an extension to odNEAT (Silva et al., 2015) (Section A.48) to accelerate online evolution in multirobot systems. Racing is proposed to reduce the time dedicated for evaluating poorly performing controllers. The evaluation time refers to the period of time assigned to a controller between its generation and its reproduction. With racing, controllers are evaluated in parallel so that the ones whose fitness is significantly lower than a dynamic threshold, known as minimal criterion, are discarded. Cloning is proposed to take advantage of the genetic information gathered by multiple robots by transferring a set of good performing individuals to nearby robots. When two robots approach, their internal population is compared based on the minimal criterion which represents the performance of the population. The "winner" robot transfers individuals with fitness greater than the minimal criterion to the "loser" robot. In another version of the cloning algorithm, the "loser" robot should "kill" those individuals whose fitness is lower than the minimal criterion.

Encoding. odNEATv2 is built on top of odNEAT, therefore odNEAT's encoding is used.

Evaluation. Different cases of odNEATv2 are evaluated against each other: odNEAT with racing, odNEAT with racing and cloning without individual extinction, odNEAT with racing and cloning with individual extinction, and original odNEAT on three multirobots tasks, using the mean fitness of the group of controllers in the end of evolution as well as the operational time of the controllers compared to the simulation time of the experiments. It is found that odNEATv2 (combining racing and population cloning) results in highly performing agents, with consistent group-wise performance as well as faster evolution.

A.50 τ-NEAT (2016)

Principles. τ-NEAT (Caamaño et al., 2016) is an extension of NEAT for modeling complex, temporal, dynamic systems. Even though NEAT enables the modeling of dynamic systems by recurrent connections, τ-NEAT extends this ability to model more complex time series by introducing a time delay in every connection of the evolved networks. This delay is implemented through a First In First Out (FIFO) buffer of size τ. The size of a buffer corresponds to the time of the delay imposed on the input of that connection and is mutable through a new mutation operator called *τ-mutation*. Moreover, τ-NEAT modifies NEAT's weight mutation operation, now called *Non-Uniform mutation* to decrease the mutation step as evolution progresses, allowing more refined search.

Encoding. τ-NEAT introduces a new field in the NEAT's chromosomes that corresponds to the synaptic delay, i.e. the size of the buffer τ.

Evaluation. The performance of τ-NEAT is compared to NEAT's using the Mean Square Error (MSE) and the Mean Absolute Percentage Error (MAPE). When τ-NEAT is applied on time series/temporal data, it performs better than NEAT achieving smaller error and predicting the complex time series with more precision. When it is applied to problems of cognitive behavior modeling, it is found that the delays introduced by τ-NEAT outperform the recurrent connections introduced by NEAT. However, no statistical hypothesis tests are performed and no standard deviations are reported.

A.51 MAP-Elites CPPN (2016)

Principles. In Tarapore et al. (2016), the authors investigate using CPPNs as the encoding for MAP-Elites. MAP-Elites is described as an illumination (Mouret and Clune, 2015) or quality diversity (Pugh et al., 2015) algorithm. In this article, MAP-Elites acts as the EA of the "Intelligent Trial & Error algorithm" (Cully et al., 2015) that allows robots to learn creative behaviors in order to adapt to unexpected situations such as damage. MAP-Elites maps high performing solutions, called the elites, to the points of a behavioral space. It has a large number of niches that represent different types of solutions and returns the highest performing solution of each (Tarapore et al., 2016). In MAP-Elites there is a pressure for both novelty and performance. Tarapore et al. use CPPNs to evolve Central Pattern Generators (CPG), ANNs (as a simpler version of HyperNEAT), SUPG, and constrained SUGP and they use three behavioral descriptors of 6 dimensions each.

Encoding. Indirect encoding of CPPNs is used. CPPNs are allowed to have the following activation functions: sine, Gaussian, sigmoid and linear.

Evaluation. The different encodings are compared among each other and against direct encoding on a legged locomotion task in simulation and real robots using quality metrics (Mouret and Clune, 2015) appropriate for the behavioral space: global fitness, coverage, global reliability, and precision. It is found that direct encoding outperforms the indirect encoding.

A.52 NEAT-LSTM (2016)

Principles. NEAT-LSTM (Rawal and Miikkulainen, 2016) is a hybrid method evolving Long Short Term Memory (LSTM) units for problems that require memory. A typical LSTM unit consists of a memory cell and three control gates: a write gate that controls the input that is directed to the memory, a forget gate that regulates how much of the stored memory should be transferred to the next time step, and the output gate that controls the output of the unit. Also, peep-hole connections exist that allow access to the values of the memory cell. NEAT evolves the number of LSTM units and the connections from the external logic (write, forget, and read) to the control gates and the weights of the peep-hole connections.

NEAT-LSTM is further extended to NEAT-LSTM with Information Maximization (NEAT-LSTM-IM) for application to deceptive optimization problems that require deeper memory. Instead of treating the problem as a multiobjective problem, a pre-training step of unsupervised learning is introduced. In this way, NEAT-LSTM is evolved in two steps: an unsupervised and a RL step. During the first phase, NEAT is used to evolve LSTM units that are able to capture highly informative and independent features by maximizing the information that is stored inside the memory cells. To do this, the features are partitioned into their histogram and they are compared using the values of entropy and mutual information. Highly informative, independent features are the ones with high individual entropy and low mutual information. In a second phase, the write, forget, and input logic gates are frozen and NEAT uses the already learned features to evolve the read logic and add more hidden layers on top of the existing LSTM network.

Encoding. NEAT is used to evolve a different type of network (i.e., LSTM networks), so we assume that there is no change in the encoding of NEAT.

Evaluation. NEAT-LSTM is compared to original NEAT that evolves RNNs (NEAT-RNN) and to NEAT-LSTM-IM. As a performance metric, the average success rate is used, that is, the ratio of trials that give a solution with a maximum fitness. In a simple sequential classification task, NEAT-LSTM significantly outperforms NEAT-RNN and in a sequence recall task, NEAT-LSTM-IM outperforms both NEAT-LSTM and NEAT-RNN.

A.53 MM-NEAT (2016)

Principles. Modular Multiobjective NEAT (MM-NEAT) (Schrum and Miikkulainen, 2016) extends NEAT to evolve agents for the Ms Pac Man video game that requires different behaviors on different tasks. The problem is treated as a multiobjective optimization problem and NSGA-II algorithm (Deb et al., 2002) is used. MM-NEAT evolves modular ANNs that consist of multiple output modules corresponding to different policies. To determine which module should be activated *preference neurons* are used in each module. In this way, a module is activated if its preference neuron's output has the highest value with respect to others. MM-NEAT can start the evolution with networks of one output module and add new output modules through a new mutation operator called *Module Mutation*. Different versions of the module mutation operator exist, based on how a new module is connected to a network's existing topology.

Encoding. NEAT's encoding is used.

Evaluation. MM-NEAT that evolves modular networks whose number of modules is predefined is evaluated against MM-NEAT with Modular Mutations and multitask learning (human-specified modularity). Their performance is evaluated in terms of the champion's average fitness value over the independent runs, a post-learning score computed by further evaluating each run's champion in additional evaluations, and the percentage of time that this champion employs its most used module. Generally, preference neurons in MM-NEAT with modular mutations or fixed number of modules perform better than multitask learning.

A.54 MB-HyperNEAT (2016)

Principles. Multi-Brain HyperNEAT (MB-HyperNEAT) (Schrum et al., 2016) is a term used to describe a set of algorithms for evolving agents that exhibit complex multimodal behavior. MB-HyperNEAT evolves agents with multiple "brains" (ANN substrates), each of which applies a different policy. In contrast to multiagent HyperNEATv2, the evolved policies do not depend on the agents' geometric relationships and the task division is not human defined. MB-HyperNEAT combines HyperNEAT's and MM-NEAT's benefits of indirect encoding and ability to evolve multimodal behaviors.

Three algorithms belong to the MB-HyperNEAT family: *multitask CPPNs*, *substrate brains with preference neurons*, and *CPPNs with module mutation*. In the first case, CPPNs have a predefined set of multiple output modules, like in multitask learning, each of which corresponds to a different brain. In the second case, the CPPNs' output modules and the substrate include a preference neuron that determines when a module should be activated. The number of modules can be predefined by the user or evolved by MM-NEAT's Module Mutation operator, which results in the third algorithm.

Encoding. HyperNEAT's encoding is used.

Evaluation. The different versions of MB-HyperNEAT (MultiTask CPPNs, CPPNs with preference neurons, and CPPNs with preference neurons and module mutation operators) are compared against each other, against multiagent HyperNEATv2 (D'Ambrosio et al., 2011) (Section A.23) and HyperNEAT with one module (Stanley et al., 2009) (Section A.13) on different RL problems that require multimodal behavior. The fitness of the champion network at the end of evolution averaged over the different runs is used as a performance metric. In all cases, multimodal networks outperform the standard HyperNEAT that uses one module. In general, it is observed that multitask CPPNs perform better than both multiagent HyperNEATv2 and CPPNs with preference neurons.

A.55 PLPS-NEAT-TL (2017)

Principles. Power-Law ranking probability-based Phased Searching NEAT for Transfer Learning (PLPS-NEAT-TL) (Hardwick-Smith et al., 2017) is an algorithm extending NEAT by proposing phased searching, that is, alternating between complexification and simplification. The goal is to produce effective and simple ANNs on a source task that can be transferred to a target task. The algorithm consists of two phases: a source task learning phase and a target task learning phase. During the first phase, PLPS-NEAT learns an ANN by alternating between complexification and simplification, by defining a probability of removing connections and hidden nodes. The difference with respect to other phased searching NEAT-based algorithms, for example, Tan et al. (2013), is that selecting structure (connections and hidden nodes) to be removed is not done randomly, but based on a ranking, so that structure that is not important to the network is removed. This ranking is performed, for the connections according to the absolute value of their weights and for the nodes based on the number of connections that are linked to them. At the end of the source task learning phase, transfer learning takes place by initializing a second learning phase with the best evolved ANN from the previous phase. In the target task learning, traditional NEAT (without phased searching) is used to learn a target task.

Encoding. NEAT's encoding is used.

Evaluation. PLPS-NEAT-TL is tested against other phased searching NE methods i.e. Phased NEAT (Tan et al., 2013, 2014), Green's Phased Searching (GPS-NEAT) (Green, 2004), and NEAT with Blended Searching (BS-NEAT) (James and Tucker, 2004) to evaluate the size of the evolved networks (number of connections and nodes) after the source task learning phase. Moreover, the transferability of learning is evaluated based on the fitness achieved by the best ANN at the end of the target task learning. PLPS-NEAT-TL shows competitive performance against other evaluated algorithms and it is capable of controlling the complexity while providing good performance, signifying the importance of controlling structure's removal during the simplification phase.

A.56 TMO-NEAT* 2017

Principles. Tensor-based Mutation Operator NEAT (TMO-NEAT) (Marzullo et al., 2017) is a NEAT-based algorithm that uses tensor-based analysis to adaptively compute the mutation rates for weight mutation in order for a better offspring to be created. It is based on the ability of an ANN to be represented as a graph whose vertices and edges correspond to the network's nodes connections respectively. The graph is represented by an adjacency matrix whose elements a_{ij} should be equal to 1 if the nodes i, j are connected through an edge, or 0 otherwise. Such a multidimensional array constitutes a tensor which is used to represent a population of chromosomes. NEAT is altered in such a way that every η generations a third order tensor T representing the best p% of the population is created. The tensor T has to be factorized into its basic factors; that is, $T = \alpha \circ \beta \circ \gamma$. The values for rate of future mutations of the connections' weights are given by the basic frame $F = \alpha \circ \beta$.

Encoding. Tensor-based analysis is used for adaptive calculation of the mutation rates, while NEAT's direct encoding of node genes and connection genes is applied for the reproduction step of the genetic algorithm.

Evaluation. TMO-NEAT is compared to original NEAT using the values of fitness, number of generations to reach the best value of the fitness and execution time. Regarding execution time, it is found that the adaptive method demands more time than NEAT but it finds the solution in less generations. Overall, statistical tests confirm that TMO-NEAT reaches higher fitness in less generations than NEAT.

A.57 τ-HyperNEAT (2017)

Principles. τ-HyperNEAT (Silva et al., 2017) extends HyperNEAT (Stanley et al., 2009) (Section A.13) by incorporating the notion of connection delay introduced by τ-NEAT (Caamaño et al., 2016) (Section A.50). τ-HyperNEAT outputs not only the weight of the connection between the substrate's nodes but also the delay τ of this connection. Similarly to τ-NEAT the delay is implemented through a buffer.

Encoding. The encoding is not described.

Evaluation. τ-HyperNEAT is compared to HyperNEAT (Stanley et al., 2009) (Section A.13) in terms of quantitative and qualitative measures. Based on the mean and standard deviation of the fitness function and the distribution of the connection weights, the two methods do not differ. However, τ-HyperNEAT generates more natural and harmonic gaits.

A.58 EXACT (2017)

Principles. Evolutionary Exploration of Augmenting Convolutional Topologies (EXACT) (Desell, 2017a) evolves the structure of Convolutional Neural Networks (CNNs). EXACT is a hybrid algorithm which uses GAs to evolve the structure of the CNNs (in terms of filters' connectivity and size) and traditional backpropagation to learn the weights. Using Asynchronous Evolution Strategy, scalability to multiple hosts is enabled. In this way, worker hosts communicate with a master process to request genomes for evaluation (training) and return them together with their fitness value. Then, the master host puts them back in the population and evolves them.

EXACT introduces three new mutation operators for changing the size of a node, that is, the size of the filter, in both *x* and *y* dimensions, only in *x* or only in *y* dimension. In addition, a new "add Node" mutation operator is introduced for inserting a new node at a random depth together with 1 to 5 edges to previous and subsequent nodes, while its size is set equal to the average of the maximum input node size and minimum output node size. In contrast to NEAT, EXACT does not perform speciation and it allows "epigenetic weight initialization" as the weights obtained after the evaluation of the initial population can be transferred to their first offsprings to facilitate training through backpropagation. Finally, EXACT performs coevolution of the hyperparameters of the CNNs using the Nelder–Mead method of Simplex Hyperparameter Optimization (SHO) (Nelder and Mead, 1965).

Encoding. The encoding scheme of EXACT differs from NEAT's in terms of assigning an extra innovation marker to the node genes and a field with depth information.

Evaluation. EXACT is compared to its previous version (Desell, 2017b), which included comparison studies against three CNNs with predefined structure; a single and two layers CNN and a modified LeNET 5 (LeCun et al., 1998). The algorithms are compared based on the training and generalizing and testing errors on the best, worst and average genomes. The CNNs with structure evolved by EXACT seem to achieve better training and testing predictions, but no statistical tests were performed. Also, the size of the evolved CNNs is taken into account, finding that the latest version evolves smaller CNNs in fewer epochs and with better prediction abilities.

A.59 HA-NEAT (2017)

Principles. Heterogeneous Activation-NEAT (HA-NEAT) (Hagg et al., 2017) extends NEAT to evolve ANNs with mixed activation functions. It is shown that in order to approximate a function, more nodes of the same activation function are required compared to nodes using different activation functions. As a result, evolved heterogeneous networks are expected to be smaller and learn less parameters. In HA-NEAT the "add-node" mutation operator is altered to introduce a new node in the genome with a random activation function from a predefined list including the step, Rectifier Linear Unit (RELU), sigmoid, and Gaussian functions. Moreover, a new mutation operator, called "mutate activation function," is introduced to change the activation function of randomly selected node.

Encoding. NEAT's encoding is altered to include a new field in the node genes indicating the type of nodes' activation function.

Evaluation. HA-NEAT is compared to NEAT based on the median MSE values (with their 25% and 75% quantiles), the number of evolved nodes and connections, and the convergence speed. It is found that HA-NEAT performs as well as the best homogeneous NEAT in terms of MSE and convergence speed but it evolves smaller networks. Nevertheless, no statistical test was performed to indicate whether the difference is significant or not.

A.60 NEAT-RAC-PGS (2017)

Principles. NEAT with Regular Actor Critic (RAC) for Policy Gradient Search (PGS) (NEAT-RAC-PGS) (Peng et al., 2017) is a hybrid method for extracting features from states in a RL environment and finding good policies that solve the problem effectively. NEAT-RAC-PGS consists of a population of agents, each consisting of an ANN that acts as a feature extractor and a RAC model that optimizes the learning of policies using PGS. The ANN takes as inputs the environmental states, and outputs the features that are used to compute the value function of each agent optimized by TD. The RAC algorithm is then used to evaluate the fitness function of each individual. The algorithm has the advantages of using a population of diverse agents and automatically extracting features while optimizing the policy search process. These features are found to be reusable in other similar domains.

Encoding. NEAT's encoding is used.

Evaluation. NEAT-RAC-PGS is evaluated on two RL tasks against NEAT. Using the features extracted on one task, the RAC algorithm is evaluated against another feature extraction method with RAC on a similar problem. In both cases, the average number of steps required to solve the problem are used as performance metrics. NEAT-RAC-PGS is found successful in automatically extracting reusable features while solving the problems with better policies.

A.61 NEAT-FLEX (2017)

Principles. NEAT-FLEX (Grisci and Dorn, 2017) is a hybrid method developed for predicting the conformational flexibility of amino acids. It consists of two independent steps: a step of clustering and a step of classification. Hierarchical clustering that does not require a priori the knowledge of the number of clusters, is used to group protein data into groups (e.g., regular or irregular secondary structure). Then, NEAT evolves ANNs that learn to classify the amino acids of a segment and its secondary structure into the cluster it belongs.

Encoding. NEAT's encoding is used to represent the weights and the topology of the ANNs into the genome. To represent the output of the ANNs, one-hot encoding is used. This means that, for example, k classes exist, the output has the format of a k length string that consists of zeros except for the c position corresponding to class c.

Evaluation. NEAT-FLEX is compared to Multi-layer perceptrons of fixed structure consisting of one hidden layer and five hidden nodes trained with error backpropagation. The algorithms are compared on the best network resulting of multiple independent runs in terms of the size of the evolved network and the classification performance. NEAT finds similar classification rates as MLPs with smaller networks. In addition, NEAT and MLPs are compared on the angles' intervals and NEAT is found to deliver more precise angles than MLPs. However, no statistical test is performed to indicate whether the difference is significant or not.

Appendix B Datasets and Performance Metrics for Benchmarking

Table 8: Datasets and performance metrics of x-NEAT methods belonging to cluster 1.

Datasets	Performance metrics
Deceptive Landscape (NoveltyNEAT, FNS-NEATFields, NS-FE-CPPN-NEAT, MAP-Elites CPPN, NEAT-LSTM-IM)	
Maze navigation (Lehman and Stanley, 2011a; Inden et al., 2013)	Ratio of successful runs (Rawal and Miikkulainen, 2016)
Locomotion of a 3D biped robot (Lehman and Stanley, 2011a)	Average number of evaluations (Lehman and Stanley, 2011a)
The pole balancing (Inden et al., 2013)	Size of networks (Lehman and Stanley, 2011a)
Distinction of 4 visual patterns (Inden et al., 2013)	Achieved fitness divided by the number of generations (Inden et al., 2013)
Distinction of 2 visual patterns with variable position (Inden et al., 2013)	Global Fitness (Tarapore et al., 2016)
Robot locomotion (Tarapore et al., 2016)	Coverage (Tarapore et al., 2016)
Evolving the morphology and the locomotion of soft robots (Methenitis et al., 2015)	Global reliability (Tarapore et al., 2016)
Sequence classification and recall in T-maze (Rawal and Miikkulainen, 2016)	Precision (Tarapore et al., 2016)
	Fitness value (Methenitis et al., 2015)
Uncertain Landscape (DynNEAT)	
Function approximation problems (Krčah, 2012)	Fitness value (Krčah, 2012)
	Ratio of successful runs (Krčah, 2012)
	Number of evolved species (Krčah, 2012)
Open-Ended Problems (Coevolutionary NEAT, LAPCA-NEAT, nnrg.hazel, NoveltyNEAT)	
Robots' duel (Stanley and Miikkulainen, 2004)	Dominance metrics (Stanley and Miikkulainen, 2004; Monroy et al., 2006; Reisinger et al., 2007)
Pong game (Monroy et al., 2006; Winter, 2005)	Generation of achieved dominance (Stanley and Miikkulainen, 2004)
General Game Playing (Reisinger et al., 2007; Genesereth et al., 2005):	Size of networks (Stanley and Miikkulainen, 2004; Lehman and Stanley, 2011a)
i. connect four	Average number of wins/ties/losses (Monroy et al., 2006)
ii. chinese checkers	Game's score (Reisinger et al., 2007)
iii. crisscross	Memory's size (Monroy et al., 2006)
iv. blocker	Number of generations for which a set of dominated strategies is undominated (Reisinger et al., 2007)
v. two-board tic-tac-toe	Average number of evaluations (Lehman and Stanley, 2011a)
2D maze navigation task (Lehman and Stanley, 2011a)	
Locomotion of a 3D biped robot (Lehman and Stanley, 2011a)	
Multiple Objectives (MO-NEAT, MM-NEAT)	
Communication's optimization between a sensor node and a base station (Haggett and Chu, 2009)	Fitness Value (Schrum and Miikkulainen, 2016)
Anomaly detection in time series data (Haggett and Chu, 2009)	Percentage of True Positives and False Positives (Haggett and Chu, 2009)
Ms Pac Man (Schrum and Miikkulainen, 2016)	Post-learning score (Schrum and Miikkulainen, 2016)
	Percentage of time of mostly used module (Schrum and Miikkulainen, 2016)
	Data specific metrics (Haggett and Chu, 2009):
	-Percentage of transmitted packets

Table 8: Continued.

Datasets	Performance metrics
Irrelevant or Redundant Features (FS-NEAT, FD-NEAT, IFSE-NEAT, Layered NEAT, Phased NEAT, PFS-NEAT)	
Robot Auto Racing Simulator (RARS) (Whiteson et al., 2005; Tan et al., 2009; Loscalzo et al., 2015; Timin, 1995; Wright et al., 2012)	Fitness value (Whiteson et al., 2005; Tan et al., 2009; Wright et al., 2012; Wang et al., 2013; Tan et al., 2013; Loscalzo et al., 2015)
(2/5) XOR (Tan et al., 2009; Wang et al., 2013)	Number of generations (Whiteson et al., 2005; Tan et al., 2009)
The concentric spirals (Tan et al., 2009; Potter and Jong, 2000)	Computational time (Wang et al., 2013)
The surface plots (Tan et al., 2009)	Size of networks (Whiteson et al., 2005; Tan et al., 2009; Wang et al., 2013; Tan et al., 2013)
Marcellus Shale lithofacies prediction (Wang et al., 2013)	Feature selection metrics:
Lung Nodule classification (Tan et al., 2013)	-Frequency of selecting relevant inputs (Tan et al., 2009)
Double pole balancing (Loscalzo et al., 2015)	-Sum of absolute weights of input connections (Tan et al., 2009)
	-Ratio of relevant–irrelevant features (Wright et al., 2012; Loscalzo et al., 2015)
	-Number of selected features (Loscalzo et al., 2015)
Online Evolution (Online NEAT+Q, KO-NEAT, Online NEAT, Online rtNEAT, odNEAT, odNEATv2) Real Time Evolution (rtNEAT, rtNEATv2, cgNEAT, Online rtNEAT)	
The mountain car task (Whiteson and Stone, 2006a; Boyan and Moore, 1995)	Uniform Moving Average Score (Whiteson and Stone, 2006a)
Server Job Scheduling (Whiteson and Stone, 2006a)	Utility function (Whiteson and Stone, 2006a; Walsh et al., 2004)
Keepaway Soccer (Zhao et al., 2007; Stone et al., 2005)	Fitness Value (Zhao et al., 2007; Silva et al., 2015, 2016)
NERO video game (Stanley et al., 2005; D'Silva et al., 2005; Stanley et al., 2006)	Number of evaluations (Zhao et al., 2007; Silva et al., 2015)
Galactic Arms Race (Hastings et al., 2009; Games, 2010)	Size of networks (Silva et al., 2015)
The Open Racing Car Simulator (TORCS) (Whiteson and Stone, 2006b; Reeder et al., 2008; Cardamone et al., 2010; Stanley et al., 2016)	Data specific metrics:
Multiagent aggregation task (Silva et al., 2015)	-Subjective metric of user's perception (Stanley et al., 2005)
Multiagent integrated navigation & obstacle avoidance (Silva et al., 2015)	-Users' statistics (Hastings et al., 2009)
Foraging tasks (Silva et al., 2016)	-Remaining distance to the target (D'Silva et al., 2005)
Multiagent Phototaxis task (Silva et al., 2015, 2016)	-Portion of successful agents (D'Silva et al., 2005)
	-Evolved content (Hastings et al., 2009)
	-Average lap time in a car simulator (Cardamone et al., 2010)
	Controllers' operation time (Silva et al., 2016)

Table 9: Datasets and performance metrics of x-NEAT methods belonging to cluster 2.

Datasets	Performance metrics
Hybrid NE & BP (NEAT+Q, Online NEAT+Q, L-NEAT, EXACT, Deep HyperNEAT)	
The mountain car task (Whiteson and Stone, 2006a; Boyan and Moore, 1995)	Uniform Moving Average Score (Whiteson and Stone, 2006a)
Server Job Scheduling (Whiteson and Stone, 2006a)	Utility function (Whiteson and Stone, 2006a; Walsh et al., 2004)
IRIS flowers classification (Chen and Alahakoon, 2006)	Accuracy (Chen and Alahakoon, 2006; Verbancsics and Harguess, 2015)
Scale Balance Classification (Chen and Alahakoon, 2006)	Fitness Value (Verbancsics and Harguess, 2015)
MNIST handwritten digits (Verbancsics and Harguess, 2015; Desell, 2017a; LeCun, Cortes, et al., 1998)	Error Value (Desell, 2017a)
BCCT200 ship recognition dataset (Verbancsics and Harguess, 2015; Rainey and Stastny, 2011)	Number of epochs (Desell, 2017a)
	Size of networks (Desell, 2017a)

Table 9: Continued.

Datasets	Performance metrics
Hybrid NE & EC (NEAT+Q, Online NEAT+Q, KO-NEAT, NEAR, PIGEON, FNS-NEATFields, NS-FE-CPPN-NEAT, MAP-Elites CPPN)	
Chaser task simulation (Stein et al., 2015)	Similarity factor (Stein and Gonzalez, 2010; Stein et al., 2015)
Sheep task simulation (Stein et al., 2015)	Uniform Moving Average Score (Whiteson and Stone, 2006a)
Car task simulation (Stein et al., 2015)	Utility function (Whiteson and Stone, 2006a; Walsh et al., 2004)
The Mackey–Glass time series (Chatzidimitriou and Mitkas, 2013)	Fitness Value (Stein et al., 2015; Methenitis et al., 2015; Zhao et al., 2007)
The multiple superimposed oscillator (MSO) (Chatzidimitriou and Mitkas, 2013)	Achieved fitness divided by the number of generations (Inden et al., 2013)
The Lorentz attractor time series (Chatzidimitriou and Mitkas, 2013)	Accumulated reward (Chatzidimitriou and Mitkas, 2013)
The mountain car task (Chatzidimitriou and Mitkas, 2013; Singh and Sutton, 1996; Whiteson and Stone, 2006a; Boyan and Moore, 1995)	Number of evaluations (Chatzidimitriou and Mitkas, 2013; Zhao et al., 2007)
The pole balancing (Chatzidimitriou and Mitkas, 2013; Inden et al., 2013)	Error value (Chatzidimitriou and Mitkas, 2013)
Server Job Scheduling (Chatzidimitriou and Mitkas, 2013; Whiteson and Stone, 2006a)	Global Fitness (Tarapore et al., 2016)
Keepaway Soccer (Zhao et al., 2007; Stone et al., 2005)	Coverage (Tarapore et al., 2016)
Evolving the morphology and the locomotion of soft robots (Methenitis et al., 2015)	Global reliability (Tarapore et al., 2016)
Maze navigation (Inden et al., 2013)	Precision (Tarapore et al., 2016)
Distinction of 4 visual patterns (Inden et al., 2013)	
Distinction of 2 visual patterns with variable position (Inden et al., 2013)	
Robot locomotion (Tarapore et al., 2016)	
Hybrid NE & RL (NEAT+Q, Online NEAT+Q, KO-NEAT, RL-SANE, Online NEAT, Online rtNEAT, NEAR, NEAT-RAC-PGS)	
The mountain car task (Whiteson and Stone, 2006a; Wright and Gemelli, 2009; Chatzidimitriou and Mitkas, 2013; Peng et al., 2017; Boyan and Moore, 1995)	Uniform Moving Average Score (Whiteson and Stone, 2006a)
Server Job Scheduling (Whiteson and Stone, 2006a; Chatzidimitriou and Mitkas, 2013)	Utility function (Whiteson and Stone, 2006a; Walsh et al., 2004)
Double inverted pendulum (Wright and Gemelli, 2009; Gomez and Miikkulainen, 1999)	Fitness value (Zhao et al., 2007)
The Open Racing Car Simulator (TORCS) (Stanley et al., 2016; Whiteson and Stone, 2006b; Reeder et al., 2008; Cardamone et al., 2010)	Error value (Chatzidimitriou and Mitkas, 2013)
Keepaway Soccer (Zhao et al., 2007; Stone et al., 2005)	Number of (time) steps (Wright and Gemelli, 2009; Peng et al., 2017)
The Mackey–Glass time series (Chatzidimitriou and Mitkas, 2013)	Number of Evaluations (Chatzidimitriou and Mitkas, 2013; Zhao et al., 2007)
The multiple superimposed oscillator (MSO) (Chatzidimitriou and Mitkas, 2013)	Accumulated reward (Chatzidimitriou and Mitkas, 2013)
The Lorentz attractor time series (Chatzidimitriou and Mitkas, 2013)	Data specific metrics:
The pole balancing (Chatzidimitriou and Mitkas, 2013; Peng et al., 2017)	-Average lap time in a car simulator (Cardamone et al., 2010)
Hybrid NE & UL (NEAT-LSTM-IM, NEAT-FLEX)	
Sequence classification and recall in T-maze (Rawal and Miikkulainen, 2016)	Ratio of successful runs (Rawal and Miikkulainen, 2016)
Protein Data Bank (Grisci and Dorn, 2017)	Accuracy (Grisci and Dorn, 2017)
	Size of networks (Grisci and Dorn, 2017)
	Task-specific metrics (Grisci and Dorn, 2017)

Table 10: Datasets and performance metrics of x-NEAT methods belonging to cluster 3.

Datasets	Performance metrics
ANNs with different types of nodes (CPPN-NEAT, NEAT-CTRNN, TL-CPPN-NEAT, RBF-NEAT, Recurrent CPPN-NEAT, EXACT, SNAP-NEAT, SUPG-HyperNEAT, MAP-Elites CPPN, NEAT-LSTM, NEAT-LSTM-IM, NEAR, NS-FE-CPPN-NEAT, HA-NEAT)	
Pattern generation (Stanley, 2006)	Fitness value (Auerbach and Bongard, 2011; Methenitis et al., 2015)
Pole balancing (Miguel et al., 2008; Kohl and Miikkulainen, 2012; Chatzidimitriou and Mitkas, 2013)	Accumulated reward (Chatzidimitriou and Mitkas, 2013)
Board game (Bahçeci and Miikkulainen, 2008)	Number of fitness evaluations (Miguel et al., 2008)
Artificial data with maximal variations (Kohl and Miikkulainen, 2009)	Generalization score (Miguel et al., 2008)
Approximation of the $sin(\alpha x)$ function (Kohl and Miikkulainen, 2009)	Error value (Hagg et al., 2017)
The concentric spirals (Kohl and Miikkulainen, 2009, 2012; Potter and Jong, 2000)	Generalization metric:
The multiplexer (Kohl and Miikkulainen, 2009, 2012)	-Ratio of successful runs with different initial conditions (Miguel et al., 2008)
N-Point classification task (Kohl and Miikkulainen, 2012)	Ratio of successful runs (Rawal and Miikkulainen, 2016)
Communication's optimization between evolution of robots' morphology (Auerbach and Bongard, 2011)	Size of networks (Miguel et al., 2008; Auerbach and Bongard, 2011; Desell, 2017a; Morse et al., 2013; Hagg et al., 2017)
MNIST handwritten digits (Desell, 2017a; LeCun, Cortes, et al., 1998)	Game's score (Bahçeci and Miikkulainen, 2008)
Half-field Soccer (Kohl and Miikkulainen, 2012; Kalyanakrishnan et al., 2006)	Number of generations (Bahçeci and Miikkulainen, 2008)
Quadruped Robot Gait control (Auerbach and Bongard, 2011; Smith, 2001)	Number of evaluations (Chatzidimitriou and Mitkas, 2013)
Sequence classification and recall in T-maze (Rawal and Miikkulainen, 2016)	Metric of fracture: variation (Kohl and Miikkulainen, 2009)
Server Job Scheduling (Whiteson and Stone, 2006a)	Error Value (Kohl and Miikkulainen, 2009; Desell, 2017a; Chatzidimitriou and Mitkas, 2013)
The mountain car task (Chatzidimitriou and Mitkas, 2013; Singh and Sutton, 1996)	Number of epochs (Desell, 2017a)
The Mackey–Glass time series (Chatzidimitriou and Mitkas, 2013)	Data specific metrics:
The multiple superimposed oscillator (MSO) (Chatzidimitriou and Mitkas, 2013)	-Subjective metric of user's perception (Stanley, 2006)
The Lorentz attractor time series (Chatzidimitriou and Mitkas, 2013)	-Statistics of the evolved morphologies (Auerbach and Bongard, 2011)
Evolving the morphology and the locomotion of soft robots (Methenitis et al., 2015)	-Data specific scores (Kohl and Miikkulainen, 2012)
Cholesterol Level Indicators (Hagg et al., 2017; Purdie et al., 1992)	-Distance traveled by the robot (Morse et al., 2013; Methenitis et al., 2015)
Engine Torque and Emissions (Hagg et al., 2017; Gomez and Miikkulainen, 1998)	Global Fitness (Tarapore et al., 2016)
Wisconsin Breast Cancer Diagnosis Problem (Hagg et al., 2017; Mangasarian and Wolberg, 1990)	Coverage (Tarapore et al., 2016)
Robot locomotion (Tarapore et al., 2016)	Global reliability (Tarapore et al., 2016)
	Precision (Tarapore et al., 2016)
ANNs by Automatic Substrate Configuration (ES-HyperNEAT, ES-HyperNEAT-LEO, Adaptive ES-HyperNEAT, MSS-HyperNEAT)	
Retina classification problem (Risi and Stanley, 2012a; Kashtan and Alon, 2005)	Fitness Value (Risi and Stanley, 2012a, 2012b; Pugh and Stanley, 2013)
Dual Task (navigation and food gathering) (Risi and Stanley, 2012a)	Ratio of successful runs (Risi and Stanley, 2012a, 2012b; Pugh and Stanley, 2013)
Maze navigation (Risi and Stanley, 2012a, 2012b)	Number of generations (Risi and Stanley, 2012a)
Multiagent maze exploration and group coordination task (Pugh and Stanley, 2013)	Size of networks (Risi and Stanley, 2012a)

Table 10: Continued.

Datasets	Performance metrics
Modular ANNs (Modular NEAT, HyperNEAT-LEO, MFF-NEAT, ES-HyperNEAT-LEO, MM-NEAT, HyperNEAT-CCT, MB-HyperNEAT)	
Artificial board game (Reisinger et al., 2004)	Modularity metric: number of crosslinks (Reisinger et al., 2004)
Retina classification problem (Verbancsics and Stanley, 2011; Risi and Stanley, 2012a; Huizinga et al., 2014; Kashtan and Alon, 2005)	Accuracy (Reisinger et al., 2004; Verbancsics and Stanley, 2011)
The Monks problem (Manning and Walsh, 2012; Thrun et al., 1991)	Fitness Value (Risi and Stanley, 2012a; Schrum and Miikkulainen, 2016; Schrum et al., 2016)
Heart disease classification (Manning and Walsh, 2012; Prechelt and Informatik, 1994)	Average Error (Manning and Walsh, 2012)
Mass spectral classification (Manning and Walsh, 2012)	Post-learning score (Schrum and Miikkulainen, 2016)
5 independent XOR problems (Huizinga et al., 2014)	Number of generations (Reisinger et al., 2004; Risi and Stanley, 2012a; Manning and Walsh, 2012)
2 Hierachically nested XOR problems (Huizinga et al., 2014)	Percentage of time of mostly used module (Schrum and Miikkulainen, 2016)
Ms Pac Man (Schrum and Miikkulainen, 2016)	Ratio of successful runs (Verbancsics and Stanley, 2011; Risi and Stanley, 2012a)
Team patrol (Schrum et al., 2016)	Visual interpretation of crosslinks (Verbancsics and Stanley, 2011)
Lone patrol (Schrum et al., 2016)	Size of networks (Risi and Stanley, 2012a)
Dual Task (Schrum et al., 2016)	Q-score (Huizinga et al., 2014; Newman, 2006)
Two rooms task (Schrum et al., 2016)	Modularity metrics (Huizinga et al., 2014):
	- Modular decomposition
	-Number of solved subproblems
	Regularity metric (Huizinga et al., 2014)
DNNs (Deep HyperNEAT, EXACT)	
MNIST handwritten digits (Verbancsics and Harguess, 2015; Desell, 2017a; LeCun, Cortes, et al., 1998)	Accuracy (Verbancsics and Harguess, 2015)
BCCT200 ship recognition dataset (Verbancsics and Harguess, 2015; Rainey and Stastny, 2011)	Fitness value (Verbancsics and Harguess, 2015)
	Error Value (Desell, 2017a)
	Number of epochs (Desell, 2017a)
	Size of networks (Desell, 2017a)
ANNs with Plasticity (MO-NEAT, Adaptive (ES) HyperNEAT, Adaptive HyperNEATv2, Seeded Adaptive HyperNEAT)	
Communication's optimization between a sensor node and a base station (Haggett and Chu, 2009)	Fitness Value (Risi and Stanley, 2010, 2012b, 2014)
Anomaly detection in time series data (Haggett and Chu, 2009)	Game's score (Gallego-Durán et al., 2013)
T-maze navigation (Risi and Stanley, 2010)	Number of generations (Risi and Stanley, 2010, 2014)
Maze navigation (Risi and Stanley, 2012b)	Computational time (Gallego-Durán et al., 2013)
The Open Racing Car Simulator (TORCS) (Gallego-Durán et al., 2013; Stanley et al., 2016)	Percentage of True Positives and False Positives (Haggett and Chu, 2009)
Line orientation task (Risi and Stanley, 2014)	Ratio of successful runs (Risi and Stanley, 2012b)
	Data specific metrics:
	-Percentage of transmitted packets (Haggett and Chu, 2009)
ANNs with Transfer Learning (TL-CPPN-NEAT, online NEAT, online rtNEAT, PLPS-NEAT-TL)	
Board game (Bahçeci and Miikkulainen, 2008)	Fitness value (Hardwick-Smith et al., 2017)
The Open Racing Car Simulator (TORCS) (Cardamone et al., 2010; Stanley et al., 2016)	Game's score (Bahçeci and Miikkulainen, 2008)
The Mario benchmark (Hardwick-Smith et al., 2017; Karakovskiy and Togelius, 2012)	Number of generations (Bahçeci and Miikkulainen, 2008)
	Size of networks (Hardwick-Smith et al., 2017)
	Data specific metrics:
	-Average lap time in a car simulator (Cardamone et al., 2010)

Table 10: Continued.

Datasets	Performance metrics
Large ANNs (Modular NEAT, HyperNEAT, HyperNEAT-LEO, Multiagent HyperNEAT, Adaptive HyperNEAT, ES-HyperNEAT(-LEO), Adaptive ES-HyperNEAT, Multiagent HyperNEATv2, Switch HybrID, NEATFields, SUPG-HyperNEAT, MSS-HyperNEAT, FNS-NEATFields, HyperNEAT-CCT, Seeded (Adaptive) HyperNEAT, MB-HyperNEAT, τ-HyperNEAT, Adaptive HyperNEATv2)	
Artificial board game (Reisinger et al., 2004)	Accuracy (Reisinger et al., 2004; Verbancsics and Stanley, 2011)
Visual Discrimination (Stanley et al., 2009)	Fitness Value (Risi and Stanley, 2010, 2012a, 2012b; Clune et al., 2011; Pugh and Stanley, 2013; Risi and Stanley, 2014; Schrum et al., 2016; Silva et al., 2017)
Retina classification problem (Verbancsics and Stanley, 2011; Huizinga et al., 2014; Risi and Stanley, 2012a; Kashtan and Alon, 2005)	Achieved fitness divided by the number of generations (Inden et al., 2013)
Multiagent predator-prey task (D'Ambrosio and Stanley, 2008)	Error Value (Stanley et al., 2009)
T-maze navigation (Risi and Stanley, 2010)	Game's score (Gallego-Durán et al., 2013)
Maze navigation (Risi and Stanley, 2012a, 2012b; Inden et al., 2013)	Number of generations (Reisinger et al., 2004; Risi and Stanley, 2010; Risi and Stanley, 2012a; D'Ambrosio et al., 2011; Risi and Stanley, 2014)
Multiagent patrolling task (D'Ambrosio et al., 2011)	Number of evaluations (Inden et al., 2012)
Target weights problem (Clune et al., 2011)	Computational Time (Gallego-Durán et al., 2013)
The Bit Mirroring Problem (Clune et al., 2011)	Ratio of successful runs (Verbancsics and Stanley, 2011; Risi and Stanley, 2012b; Risi and Stanley, 2012a; Inden et al., 2012; Pugh and Stanley, 2013)
Quadruped Controller (Clune et al., 2011)	Visual interpretation of crosslinks (Verbancsics and Stanley, 2011)
Finding the large square (Inden et al., 2012; Gauci and Stanley, 2007)	Success of a task (D'Ambrosio et al., 2011)
Multiagent maze exploration and group coordination task (Pugh and Stanley, 2013)	Networks' size (Risi and Stanley, 2012a; Morse et al., 2013)
Distinguishing orientations of shapes and textures (Inden et al., 2012)	Generalization ability (Inden et al., 2012)
Distinguishing orientations of area borders in gray scale images (Inden et al., 2012)	Data specific metric:
Evolving coordinated reaching movements for segmented arms (Inden et al., 2012)	-time remaining after the agents have captured all the preys (D'Ambrosio and Stanley, 2008)
Quadruped Robot Gait control (Morse et al., 2013; Inden et al., 2013; Silva et al., 2017)	-distance traveled by robot (Clune et al., 2011; Morse et al., 2013)
The Open Car Racing Simulator (Gallego-Durán et al., 2013; Stanley et al., 2016)	Generated gaits (Silva et al., 2017)
Distinction of 4 visual patterns (Inden et al., 2013)	Q-score (Huizinga et al., 2014; Newman, 2006)
Distinction of 2 visual patterns with variable position (Inden et al., 2013)	Modularity metrics:
Pole balancing (Inden et al., 2013)	-Modular decomposition (Huizinga et al., 2014)
5 independent XOR problems (Huizinga et al., 2014)	-number of crosslinks (Reisingeret al., 2004)
2 Hierachically nested XOR problems (Huizinga et al., 2014)	-Number of solved subproblems (Huizinga et al., 2014)
Line orientation task (Risi and Stanley, 2014)	Regularity metric (Huizinga et al., 2014)
Team/Lone patrol (Schrum et al., 2016)	
Dual Task (Risi and Stanley, 2012a; Schrum et al., 2016)	
Two rooms task (Schrum et al., 2016)	

Table 10: Continued.

Datasets	Performance metrics
ANNs with Memory Capacity (NEAT-CTRNN, NEAR, NEAT-LSTM, NEAT-LSTM-IM, τ-NEAT, τ-HyperNEAT)	
Pole balancing (Miguel et al., 2008; Chatzidimitriou and Mitkas, 2013) Sequence classification and recall in T-maze (Rawal and Miikkulainen, 2016) The Mackey–Glass time series (Chatzidimitriou and Mitkas, 2013; Caamaño et al., 2016) The multiple superimposed oscillator (MSO) (Chatzidimitriou and Mitkas, 2013) The Lorentz attractor time series (Chatzidimitriou and Mitkas, 2013) The mountain car task (Chatzidimitriou and Mitkas, 2013; Singh and Sutton, 1996) Server Job Scheduling (Chatzidimitriou and Mitkas, 2013) Monthly CO_2 concentrations (Caamaño et al., 2016; Thoning et al., 1989) Monthly number of international airline passengers (Caamaño et al., 2016; Box et al., 2015) Safe Crossing robot task (Caamaño et al., 2016) Mimicking motion robot task (Caamaño et al., 2016) Quadruped Gait control (Silva et al., 2017)	Fitness value (Silva et al., 2017) Accumulated reward (Chatzidimitriou and Mitkas, 2013) Error value (Chatzidimitriou and Mitkas, 2013; Caamaño et al., 2016) Number of fitness evaluations (Miguel et al., 2008; Chatzidimitriou and Mitkas, 2013) Generalization score (Miguel et al., 2008): Generalization metric: -Ratio of successful runs with different initial conditions (Miguel et al., 2008) Ratio of successful runs (Rawal and Miikkulainen, 2016) Size of networks (Miguel et al., 2008) Generated gaits (Silva et al., 2017)