Complexity-based speciation and genotype representation for neuroevolution

Alexander Hadjiivanov
School of Computer Science and Engineering
University of New South Wales
Sydney, NSW 2052, Australia
Email: a.hadjiivanov@student.unsw.edu.au

Alan Blair
School of Computer Science and Engineering
University of New South Wales
Sydney, NSW 2052, Australia
Email: blair@cse.unsw.edu.au

Abstract—This paper introduces a speciation principle for neuroevolution where evolving networks are grouped into species based on the number of hidden neurons, which is indicative of the complexity of the search space. This speciation principle is indivisibly coupled with a novel genotype representation which is characterised by zero genome redundancy, high resilience to bloat, explicit marking of recurrent connections, as well as an efficient and reproducible stack-based evaluation procedure for networks with arbitrary topology. Furthermore, the proposed speciation principle is employed in several techniques designed to promote and preserve diversity within species and in the ecosystem as a whole. The competitive performance of the proposed framework, named Cortex, is demonstrated through experiments. A highly customisable software platform which implements the concepts proposed in this study is also introduced in the hope that it will serve as a useful and reliable tool for experimentation in the field of neuroevolution.

I. INTRODUCTION

The process of evolving neural networks (neuroevolution; NE) has been the subject of extensive research for more than two decades. Although early research on NE focused on fixed-topology networks [19], [23], [31], where only the synaptic weights were evolved, the benefits of evolving the topology as well as the weights have also been recognised [2], [7], [11], [25], [27], [29]. However, the evolution of arbitrary network topologies introduces additional complexity in terms of genome representation, which is part of the reason why fixed-topology NE has not fallen completely out of favour [9]. This is related to the problem of *competing conventions* [2], [26], whereby the same network can be encoded by two or more different genomes, which reduces the effectiveness of the genome representation. Furthermore, allowing neural networks to evolve arbitrary topologies may render it difficult to match genomes encoding different topologies when performing crossover [27].

An issue of particular concern is that structural mutations in networks (the addition and deletion of neurons and connections) can initially degrade the performance of the network, even if the mutation is beneficial in the long run [29]. This problem can be mitigated by grouping networks into *species*, as a result of which networks only compete directly with other networks within their own species rather than competing against the entire ecosystem. While *niching* [18] as a form of speciation has been used to maintain diversity and prevent premature convergence to a suboptimal solution, speciation with such explicitly protective function was introduced more recently in the context of the NEAT framework [28]. In NEAT, speciation depends on the number of disjoint and excess genes [28, pp. 109–110] and the difference in mean synaptic weight. The importance of each of these three parameters can be adjusted by setting the corresponding parameter coefficients in the model. While the ability to adjust the granularity of speciation provides flexibility, choosing suboptimal values for the coefficients may compromise the performance of the model. Furthermore, although excess and disjoint genes indicate the addition and/or deletion of connections to the genome (and respectively to the network), this scheme does not directly take into account the addition or deletion of neurons, which is arguably the most disruptive form of mutation occurring during NE.

In the present study, the above problems are addressed by adopting a genotype representation and speciation framework which facilitates crossover by providing a simple way to match different network topologies. The genotype is very similar to the phenotype, and therefore the encoding is direct. This eliminates the need to decode the genotype into a network in order to evaluate its fitness, and allows the output to be computed in an efficient and reproducible way. Furthermore, the proposed representation is free of redundancies such as non-expressed genes, and thus becomes highly resilient to bloat. Finally, well-defined speciation based on the complexity of the search space emerges naturally, without the need for any adjustable parameters.

This paper is organised as follows. Section II provides a brief overview of existing research and the relevant problems addressed in this study. Section III introduces the proposed NE framework, and Section IV presents the results of experiments demonstrating some of its advantages. Finally, Section V concludes the paper with a discussion of possible directions for future research.

II. PREVIOUS WORK

As outlined in Section I, although evolving the topology as well as the weights of networks is sometimes problematic, it can be beneficial if designed and implemented properly. This section gives a brief overview of existing relevant research, with a focus on evolving network topology.

One of the early methods for evolving networks with arbitrary connectivity was the connectivity matrix scheme [15]. This method was used, for example, to evolve application-specific controllers with optimal size and structure [4]. Although evolving the connectivity proved to be more effective than fixed-topology methods, the number of neurons was essentially limited. One of the main advantages of evolving the network topology is that it eliminates the need to decide not only the connectivity, but also the number of hidden neurons. There is no rigorous method to decide *a priori* how many hidden neurons are necessary in order to obtain optimal performance on a given task, and the difficulty of finding the optimal number of hidden neurons increases proportionally to the complexity of the task. Note that this problem is not specific to NE and is unrelated to the method used for training the network.

One framework for evolving network topologies which allows for dynamic addition of new neurons is cellular encoding [11]. In this framework, networks are represented as graph grammars evolving through a process resembling cell division. In a formal comparison on the double pole balancing task [31], cellular encoding was able to evolve networks which generalised considerably better than ones with a fixed topology, while being much more compact (0–2 hidden neurons versus 10 for the fixed-topology network) [12]. This result demonstrated that a seemingly very difficult task might be solvable by a neural network controller whose structure is much simpler than anticipated by a human manually designing a controller for the same task. In this regard, it has also been demonstrated that neural networks with evolved topology are capable of generalising better than fixed-topology networks trained with backpropagation [5].

Evolving network topology by adding and deleting neurons is more involved than using a fixed number of neurons, and requires a suitable genome representation. The NeuroEvolution of Augmenting Topologies (NEAT) framework introduced a genome representation which proved to be highly successful in this respect [27], [28]. In NEAT, the genome is linear and grows gradually with the addition of new neurons and connections [28, Fig. 3]. When a new neuron is added, an existing connection is *disabled*, and two new connections to and from the new neuron are added. The rationale behind this scheme is that it introduces novel behaviour while minimising the impact on the network's fitness. As different genomes grow, they diverge and gradually become less and less compatible with each other. To address this issue, NEAT also introduced the concept of *speciation* with the explicit function of protecting

innovation. Specifically, genomes are compared in terms of the number of excess and disjoint genes and the difference in mean synaptic weight. If the discrepancies are sufficiently large, the genomes are placed in different species in order to prevent unfair competition between networks of different complexity. These ideas proved to be very successful and largely drove the surge in popularity of the NEAT framework, which has been extended with additional functionality [7], [25].

The present study proposes a speciation scheme based on the complexity of the phenotype in terms of number of hidden neurons. This idea is inspired by the relatively recent discovery that several regions in the brain (notably, the dentate gyrus in the hippocampus) continue to produce *new* neurons throughout the life of an animal (at least in birds and mammals, including humans). Although at present the function of this sustained neurogenesis is not clear, there are indications that it might be associated with learning [10], the formation of temporal memories [1] and the complexity of tasks, the richness of the environment and/or general experience [6], [14]. In addition, it seems that the relation between neurogenesis and learning is bidirectional (in other words, learning stimulates neurogenesis and vice versa) [14]. Note that the proposed speciation method is not necessarily biologically accurate in the sense of speciation as seen in living creatures on Earth, which would fall in the realm of artificial life. Rather, it draws a parallel with the process of generation and removal of neurons in the brain in response to the complexity of the tasks being performed. The following section presents a complete NE framework named Cortex, which combines the abovementioned concept of speciation with a novel genotype representation, relevant genetic operations (crossover and mutation), as well as several techniques for promoting and preserving diversity which make use of speciation.

III. CORTEX NEUROEVOLUTION FRAMEWORK

A. Genotype representation

First, a note about terms and conventions is in order. The terminology used in this study is adopted in an attempt to avoid the ongoing controversy regarding the rigorous definition of terms such as genome, genotype and phenotype [17]. Specifically, **population** refers to a group of **individuals** of the **same species**, and the collection of all populations is referred to as **ecosystem**. In Cortex, the **genome** is common to all individuals belonging to a species. In fact, species are *defined* through their respective genomes, and it is up to the individuals to *express* the genome of their species by forming connections, thus creating the **genotype**. The genotype consists of three chromosomes representing outward, inward and recurrent connections. Based on this, we use the term *genotype* representation rather than *genome* representation to emphasise the fact that the genome is an abstract notion common to many individuals belonging to the same species, while the genotype encodes information about a particular individual.

The genotype representation in Cortex is inspired by the connectivity matrix representation [15] and the NEAT representation [28]. Like NEAT, it adopts a strict order for neurons: bias, input, output, hidden. However, the NEAT genome is linear, whereas the genotype in Cortex is two-dimensional. It is essentially composed of connectivity tables, which are reminiscent of connectivity matrices, with the exception that only existing connections are marked explicitly and recurrent connections are stored separately. An example of a genotype with the corresponding phenotype is presented in Fig. 1. There is no redundancy in the genotype since non-existent connections are not expressed. As a result, the genotype closely resembles the phenotype, similarly to the PDGP representation scheme [22]. Note that this representation minimises the competing conventions problem because as long as neurons have identical activation functions, they are indistinguishable from each other and cannot themselves participate in distinguishing network topologies.

Although the table of inward connections can be generated from the table of outward connections at runtime just before the network is evaluated, this will incur a time overhead proportional to the size of the outward connection table. Therefore, the inward connection table is presented as part of the genotype in order to illustrate the stack-based evaluation procedure in Fig. 6.

B. Speciation

The number of hidden neurons in a neural network determines the dimensionality of the search space that it can explore. Therefore, it is a direct indicator of the upper limit of the level of complexity of the task that the network is capable of learning or solving. Thus, the specific connectivity pattern of the phenotype determines what portion of this space is being explored. A fully connected network can potentially explore the entire available space, whereas a partially connected one explores only part of it. However, if a particular problem requires a minimal number of hidden neurons, a network exploring a smaller search space will never be able to find a solution. The immediate advantage of Cortex in this respect is that speciation is automatic, being based solely on the number of hidden neurons[1]. The granularity of speciation does not depend on any adjustable parameters.

It is noteworthy that the generic version of NEAT supports only the addition of neurons. This leads to the problem that the NEAT genome can evolve only networks of similar or higher complexity, without any means of *reducing* the complexity of the evolved topologies. Although an extension referred to as a *pruning phase* was introduced in order to allow NEAT to delete neurons and simplify topologies, the solution seems *ad hoc* because it is activated when the average complexity

---

[1]Technically, the *total* number of neurons, but in the current version of Cortex, only hidden neurons can be added or deleted. Therefore, the number of bias, input and output neurons remains constant for all individuals throughout their life.
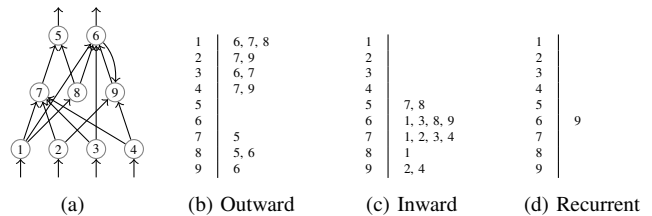

Fig. 1. (a) A phenotype with an unstructured topology. (b)–(d) Genotype of the phenotype in (a). The encoding is direct and without any redundancies. Inward and recurrent connections are stored separately from outward connections to facilitate the evaluation of the phenotype (Fig. 6).
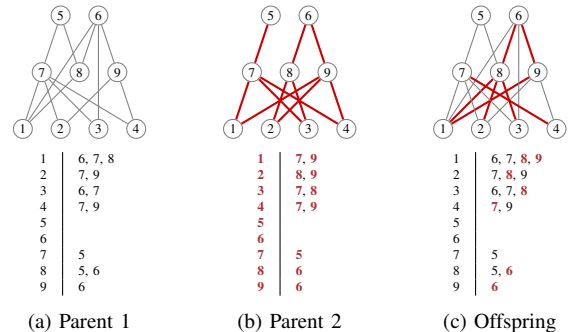

Fig. 2. Crossover operation in Cortex. (a,b) Parents participating in the crossover operation and (c) the resulting offspring. Although only the outward connectivity table is shown here, crossover for recurrent connections is performed in the same manner. The chromosome of inward connections is populated automatically from the one of outward connections and does not participate in crossover.

reaches a certain pre-set threshold. In contrast, the ability to delete neurons is built into Cortex, and thus by setting the respective rates of neuron addition and deletion, it is possible to evolve an ecosystem of networks whose complexity increases, decreases or 'spreads out' evenly in both directions.

C. Genetic operations

Cortex supports the usual genetic operations, such as crossover and different mutation types. Each type of operation is briefly outlined below together with some principles for promoting diversity within species and in the ecosystem as a whole.

1) Crossover: The genotype representation offers a very direct way to match the topologies of two networks because we know that the neurons are ordered in a particular way. An example of a crossover procedure is presented in Fig. 2. Mating between individuals from different species is also possible, with excess neurons and their corresponding connections inherited directly from the longer genotype where appropriate.

2) Mutation: The genotype representation requires that all neurons be numbered sequentially in the order of bias, input, output, hidden. This allows for straightforward addition and deletion of connections by simply populating the inward and outward tables when adding a feedforward connection or the recurrent table when adding a recurrent connection. It should
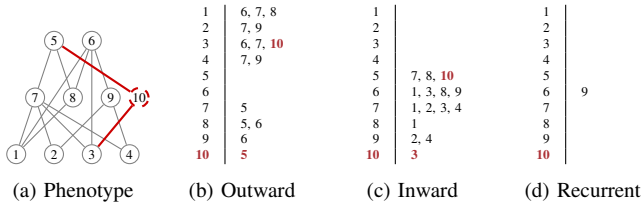
Fig. 3. (a) Adding a new hidden neuron to the phenotype. (b)–(d) The mutation is reflected in the genotype. The phenotype is placed in the species with four hidden neurons, which is created if it does not exist. Note that it is impossible to end up with a recurrent connection by accident.

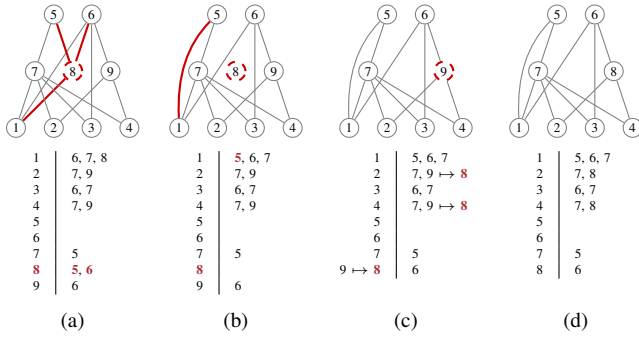| 1 | 6, 7, 8 | | 1 | **5**, 6, 7 | | 1 | 5, 6, 7 | | 1 | 5, 6, 7 |
2	7, 9		2	7, 9		2	7, 9 ↦ **8**		2	7, 8
3	6, 7		3	6, 7		3	6, 7		3	6, 7
4	7, 9		4	7, 9		4	7, 9 ↦ **8**		4	7, 8
5			5			5			5	
6			6			6			6	
7	5		7	5		7	5		7	5
8	**5, 6**		8			9 ↦ **8**	6		8	6
9	6		9	6						

| (a) | (b) | (c) | (d) |

Fig. 4. Deleting a neuron from a phenotype. Only the chromosome of outward connections is shown, but the same procedure is applied to the other two chromosomes. (a) First, a hidden neuron is selected at random. (b) All outward connections from the selected neuron are transferred to each source neuron if they do not exist already. At this stage, if a transferred connection would introduce a loop (a recurrent connection), it is not expressed. (c) Now the selected neuron does not have any connections and can be safely removed. However, this leaves the genotype in an inconsistent state since the neuron numbering must be consecutive. Therefore, all neurons whose ID is higher than the deleted neuron, as well as all connections to and from them, are shifted down by 1. (d) The new phenotype now has a consistent genotype and belongs to a different species.

be noted that before adding a feedforward connection, a check is performed to ensure that it would not form a loop.

Adding a hidden neuron is also straightforward. Similarly to NEAT, the new neuron is appended to the end of the genotype and given the next neuron ID (Fig. 3), after which it is connected randomly to an input and an output neuron. This avoids the overhead of having to check for accidental loops introduced by the new connections.

Deleting a neuron is somewhat more involved (Fig. 4). First, a hidden neuron is selected at random, and its connections are redistributed by connecting each of its inputs to each of its outputs while ensuring that this does not introduce loops in the graph. Subsequently, the neuron is deleted, and the IDs of all neurons with IDs greater than the deleted one are shifted down by 1 in order to maintain their consecutive numbering. Finally, the same procedure of shifting IDs down is performed for all outward, inward and recurrent connections.

It should be noted that adding or deleting a neuron results in the network being moved to the species with the corresponding number of hidden neurons. If such a species does not exist, it is created. In this way, search spaces of incrementally larger

size are being explored *simultaneously* by different populations of networks, which compete with each other indirectly at the species level. The purpose of this scheme is to converge on the optimal number of hidden neurons necessary for performing the task at hand.

Weights are mutated by selecting a random connection and changing its weight according to the following procedure. The first mutation for a given weight w is in a random direction (increase or decrease), where the new weight value is drawn from a normal distribution with a mean at the current value and a relatively large standard deviation σ_w (equivalent to about 5% of the allowable weight range). The rationale behind using a large initial value for σ_w is that in the early stages individuals are much more likely to be in a low-fitness part of the search space than close to an optimum, and therefore exploration should initially dominate exploitation. However, only the first mutation for each weight is in a random direction. After each mutation, the fitness of the individual is re-evaluated, and the difference from the old fitness is noted. If the fitness has improved, this means that the mutation has been beneficial, and the direction of weight update is maintained. Conversely, if the fitness has decreased, the mutation must have been unproductive, and the direction is reversed while at the same time reducing the magnitude of σ_w by a small fraction. This is a form of *meta-optimisation* which is a hybrid of two existing techniques for heuristic optimisation: pattern search (PS) and the Luus-Jaacola (LJ) method [16], [21]. In the PS method, the direction of weight update is maintained until the fitness stops increasing, at which point the direction is reversed and the sampling range is halved. The LJ method is an extension of the PS method in which *all* weights are updated simultaneously using a common sampling range (the standard deviation σ_w if the values are drawn from a normal distribution), which is decreased by a small fraction every time the fitness fails to improve.

In Cortex, the PS and LJ methods are combined as follows. For a given network, only a single (randomly selected) weight is updated in each generation, and the direction of change is maintained in subsequent mutations of the same weight until the fitness fails to improve. However, instead of being halved at this point, σ_w is decreased gradually by being multiplied by a fixed coefficient $0 < d < 1$ (currently, the default value of d is 0.95, in accordance with [16]). In this way, it is possible to determine exactly what mutations are beneficial, without compromising the initial tendency towards exploration versus exploitation. With the original PS method, if the very first weight update is *not* beneficial, the sampling range would be instantly halved, potentially reducing the speed of convergence. This problem is largely avoided if the sampling range is decreased at a slower exponential rate.

3) Stagnancy: If the fitness of an individual fails to improve for a certain number of generations (denoted as g_{max} in Eq. 1), its fitness is gradually reduced by the stagnancy coefficient C_d as follows:
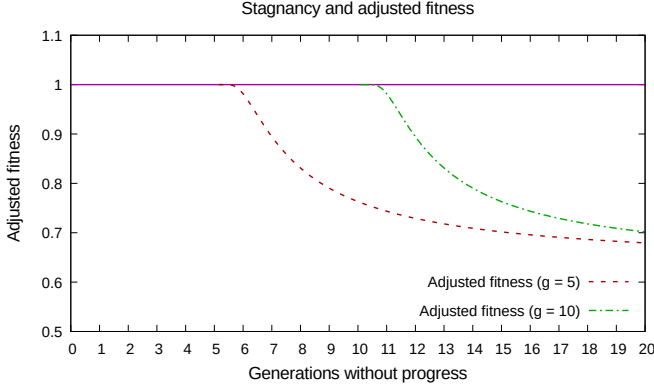
Fig. 5. The adjusted fitness of an individual is gradually decreased if it fails to make progress after a certain number of generations.

$$C_d = \begin{cases} 1, & g \le g_{max} \\ 1 - \exp\left(-\left(1 + \frac{1}{g}\right)^2\right), & g > g_{max} \end{cases}, \quad (1)$$

Here, g is the number of generations since the last fitness improvement. Essentially, the network is gradually penalised if it fails to make any progress after several generations (a stagnancy-based penalty scheme is also employed in NEAT). Using C_d, we calculate the adjusted fitness $\tilde{f}_i$ of each individual by normalising the true fitness f_i to the highest fitness f_{max} achieved in the entire ecosystem (Fig. 5):

$$\tilde{f}_i = \frac{C_d f_i}{f_{max}}. \quad (2)$$

In the operations involved in offspring generation, we utilise the adjusted fitness to rank individuals.

4) Offspring Generation: The crossover operator is applied separately to each species rather than to the ecosystem as a whole. In addition to individual fitness, each species is also assigned a fitness value based on the average fitness of the individuals belonging to it:

$$f_{s_i} = \frac{1}{n} \sum_{j=1}^{n} f_j, \quad (3)$$

where f_{s_i} is the fitness of species i, f_j is the fitness of the j-th individual of the same species, and n is the number of individuals belonging to that species. The adjusted fitness $\tilde{f}_{s_i}$ of species i is defined as the sum of the adjusted fitness values of all individuals belonging to that species:

$$\tilde{f}_{s_i} = \sum_{j} \tilde{f}_j \quad (4)$$

Similarly to the case of individuals, species are also ranked according to their adjusted fitness. Another metric which is re-calculated at each generation for each species is *diversity*. The diversity d_i of species i represents the variance of the fitness values of all individuals belonging to that species:

$$d_i = \frac{1}{n} \sum_{j=1}^{n} \left(\frac{f_j - f_{s_i}}{f_{max} - f_{min}} \right)^2, \quad (5)$$

where f_{max} and f_{min} are the maximal and minimal individual fitness values for species i, and n is the number of individuals (population size) for that species.

At each generation, the adjusted species fitness $\tilde{f}_{s_i}$ is employed to determine the number of offspring that each species is allowed produce, and the diversity is used to determine how many individuals from the species are allowed to reproduce, in other words, the number of parents. In this way, Cortex employs a double screening to ensure diversity. For example, on one hand, very fit species are given a large quota for offspring, but if the diversity within that species is low, the number of parents will also be low, and the actual number of offspring produced is unlikely to even reach the quota. Conversely, a species with low fitness but large diversity will be assigned a large parenting quota but a small offspring quota, and thus the resulting total number of offspring will be likewise small. In order to produce a large number of offspring, a species must be very fit *as well as* very diverse.

Note that the offspring quota q serves only as an upper limit and does not mean that q individuals are necessarily removed at each generation. Rather, q is distributed among the species according to the following formula:

$$C_q = \frac{q}{\sum_{i=1}^{n} \left(1 + \ln \tilde{f}_{s_i}\right)} \quad (6)$$

$$o_i = \left\lfloor C_q \ln \tilde{f}_{s_i} \right\rfloor \quad (7)$$

Here, $\tilde{f}_{s_i}$ is the adjusted species fitness as defined above, C_q is a quota distribution coefficient common to all species, and o_i is the number of offspring that species i is allowed to produce. This distribution scheme promotes diversity by ensuring that species with similar fitness will be given similar opportunities to reproduce while reducing the risk of a single species taking over the entire ecosystem. The parenting quota is determined in a similar manner, but using diversity instead of the adjusted fitness.

The ecosystem size in Cortex is fixed in order to prevent uncontrolled expansion, and the total number of individuals from all species cannot exceed the predefined limit. The global offspring and parenting quotas are taken as percentages of the maximal ecosystem size. Note that this percentage is the same for both the offspring and parenting quotas. At each generation, each species produces a number of offspring

depending on its diversity and fitness, and if the total number of individuals and offspring exceeds the maximal allowable size of the ecosystem, an elimination procedure is employed to bring the ecosystem size to within the limit. The elimination proceeds by selecting a random species according to the roulette wheel principle and removing the individual with the lowest fitness from it. This is repeated until the total number of individuals is equal to the maximal ecosystem size. The weight w_i for species i in the roulette wheel (a weighted distribution) is calculated as follows:

$$w_i = \frac{s_i p_i}{d_i}, \tag{8}$$

where s_i is the stagnancy of the species (the number of generations for which none of the individuals in that species made any progress in terms of fitness), p_i is the population size of the species, and d_i is the diversity of the species as defined above. Again, this metric is designed to promote diversity by increasing the probability of elimination for individuals from stagnant, overly populous and less diverse species while being gentle on progressive, small and diverse species. However, the stochastic nature of the roulette wheel also ensures that no species is immune to total extermination.

D. Network Evaluation

As mentioned above, the genotype is used directly in the evaluation of the phenotype. Due to the ordered nature of the genotype, it is clear which neurons must be evaluated before an output is produced. The evaluation procedure is outlined in Fig. 6.

Although the network evaluation procedure is rarely mentioned explicitly in NE research, it is noteworthy that the evaluation process employed in some NE frameworks is incremental, meaning that the network becomes activated gradually over several time steps. However, this poses the problem that the network may never actually become stable [20]. For example, for many applications, NEAT requires that the network 'settle' on a value (or even a value within some small delta [30]) before it is considered to have produced an output. Clearly, this evaluation scheme does not always provide a reproducible output. In other words, when presented with the same input more than once, the network may produce different outputs, or it may never even settle on an output at all. This issue is addressed by the stack-based network evaluation procedure in Cortex, which ensures that the output of the network is reproducible even for networks with arbitrary topologies and recurrent connections. Specifically, the entire network is evaluated at every step. Once all outputs are evaluated, signals travelling via recurrent connections (if any) are propagated in preparation for the next evaluation step when the next external input arrives. This is the reason for storing recurrent connections in a separate chromosome.

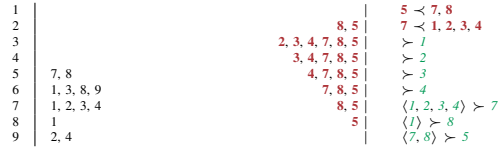(a) Inward connections   (b) Evaluation stack for output neuron 5.

Fig. 6. Stack-based phenotype evaluation procedure. (a) Inward connections for the network in Fig. 1. (b) Evaluation stack for output neuron 5 in (a). It is straightforward to derive the order in which neurons have to be evaluated before producing the final output. The evaluation proceeds backwards from the output neurons. If there are prerequisites (source neurons) which have not been evaluated, they are pushed onto the stack, and *their* respective prerequisites are evaluated. This is performed recursively until all input neurons are evaluated (input neurons have no prerequisites, indicated by the corresponding empty fields in the chromosome of inward connections). This procedure allows for efficient and reproducible evaluation of networks with arbitrary topology. After all output values are computed, activations are propagated along any recurrent connections in preparation for the next evaluation step.

IV. EXPERIMENTS

The advantages of the speciation principle and the overall performance of Cortex were evaluated experimentally by using a software platform written in C++ which implements the proposed framework[2].

A. XOR

As discussed above, certain tasks require networks to search a space of sufficient complexity in order to be able to find a solution reliably. In this regard, the proposed speciation principle allows the ecosystem to explore a range of search spaces of increasing complexity simultaneously. Since the speciation criterion is known *a priori*, the ecosystem can be initialised with more than one species, allowing it to explore spaces of different complexity *from the onset of evolution*.

A simple and clear way to test the effectiveness of this idea is the XOR task, which is not linearly separable and thus requires at least one hidden neuron. Ten experiments (referred to as 'evolving speciation' experiments below) were performed on the XOR task by using an ecosystem randomly initialised with 1–10 species (0–9 hidden neurons). Ten additional experiments were performed with the addition and deletion of neurons disabled ('fixed speciation' experiments) in order to test the impact of dynamic speciation on the speed of finding a solution and the success rate. Each experiment consisted of 100 runs, and a run was terminated if no solution was found within 100 generations. The ecosystem size limit was 150 networks, and the global offspring / parenting quota was 25% of this limit. At the onset of each simulation, the 150 networks were evenly distributed among the available species.

The probability weightings for adding and deleting a neuron were 30 and 5 units, respectively, in the evolving speciation experiments, and 0 in the case of fixed speciation experiments. Furthermore, the probability weighting of mutating a random

---

[2]The latest version of Cortex is available at https://github.com/trilobeat/cortex
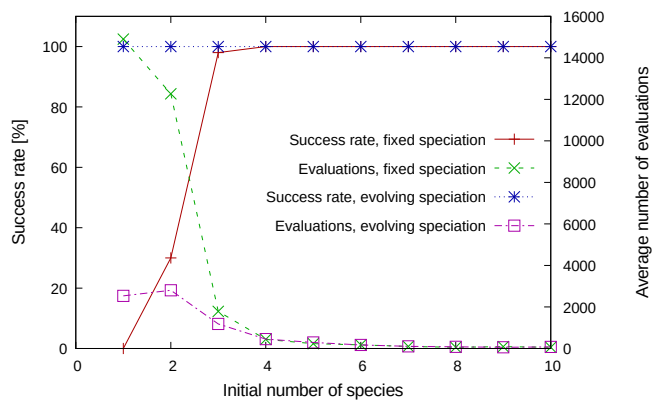
(a)
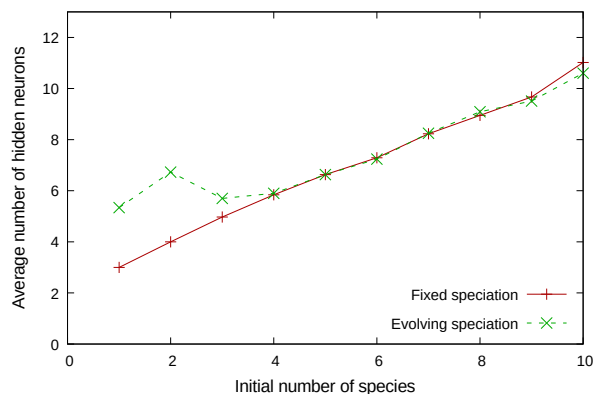
(b)

Fig. 7. (a) Results for the evolving and fixed speciation experiments on the XOR task. As expected, the ecosystem with one species (zero hidden neurons) in the fixed speciation experiments does not find a single solution in 100 runs. The situation is markedly improved with the addition of more species, and finding a solution is not a problem if there are four or more species. In contrast, a solution is always found in the evolving speciation experiments. Convergence is very rapid even if there is initially only a single species (average number of evaluations: 2535), but the same trend towards decreasing number of evaluations with increasing initial number of species is seen as in the fixed speciation experiments. (b) Average number of neurons in evolved solutions plotted against the number of species. The plots practically overlap when the ecosystem is initialised with four or more species, in tune with the data in (a).

weight was 1000 units, and that for adding and deleting a connection was 50 and 5 units, respectively. These parameters are virtually identical to those used in the XOR experiments using NEAT in [28]. The elite size was 1, meaning that the champion network in each species was copied into the next generation unchanged. At each generation, offspring was produced through crossover with a 75% chance and through cloning combined with mutation with a 25% chance, and the remaining networks which were not part of the elite were mutated with the above parameters. The tanh activation function was used for all hidden and output neurons, whereby a positive output was interpreted as a 1 and a non-negative output was interpreted as a 0. The fitness function was calculated as $4 - \epsilon$, where ϵ is the sum of absolute errors for each of the four outputs. The results are shown in Fig. 7.

Clearly, the XOR task does not pose a significant problem if Cortex is allowed to speciate freely. In the evolving speciation experiments, it finds a solution reliably in 100% of the runs, even if it is initialised with a single species (no hidden neurons). Looking at the success rate in Fig. 7a, there is practically no difference in performance if the ecosystem is initialised with four or more species with both fixed and evolving speciation. Note that the average number of neurons for the two sets of experiments practically overlaps when the initial number of species is four or greater, suggesting that the proposed speciation procedure can lead to a significant improvement in performance (constant 100% success rate) with virtually zero overhead in terms of the *size* of the resulting solution.

One possible explanation for the low performance ($\sim 30\%$ success rate) of Cortex with fixed speciation in the case of two species (0 and 1 hidden neurons) is that the initial number of species determines the number of individuals standing a chance of solving the problem. Thus, with two initial species, only 50% of the individuals can solve the task, whereas with ten initial species 90% of all individuals can potentially solve it. In addition, any increase in the population of networks without hidden neurons further reduces the number of networks *with* hidden neurons due to the fixed size of the ecosystem. This 'dilution' illustrates the importance of diversity in terms of complexity, which can be ensured by allowing the ecosystem to spawn new species.

B. Double Pole Balancing

The XOR task was useful for illustrating the advantage of initialising an ecosystem with multiple species in cases where the solution depends on a minimal number of hidden neurons. However, the XOR task is too simple to provide a good estimate of the overall performance of the proposed framework. Therefore, experiments were conducted on the double pole balancing task with velocity information. This task can be successfully solved by a neural network controller without any hidden neurons for a certain set of conditions, and therefore it evaluates the performance of the proposed framework as a whole rather than emphasising the advantages of speciation as in the preceding experiment.

The equations of motion of the cart and the two poles are given in [31]. Experiments were conducted by taking into account all parameters, including friction coefficients. The fourth-order Runge-Kutta method was used for accurate calculation of the trajectories.

1) Fixed conditions: The first set of experiments were designed to verify whether Cortex could find a solution to the problem at all. The conditions used in the experiments are shown in Table I. In the simulations, all variables presented as input to the controller were scaled to the interval $[-1, 1]$.

TABLE I
CONDITIONS FOR DOUBLE POLE BALANCING EXPERIMENTS

	Unscaled	Scaled
Track length	$4.8\,m$	
Cart position relative to track centre	$\pm 2.4\,m$	± 1
Initial velocity of the cart	$0\,m/s$	0
Threshold angle	$\pm 36°$	± 1
Initial angle (long pole)	$\pm 6°$	± 0.167
Initial angle (short pole)	$0°$	0
Initial angular velocity (long and short poles)	$0\,rad/s$	0
Time step (RK method parameter)	$0.01\,s$	
Maximal force magnitude	$10\,N$	
Coefficient of friction (cart/track)	0.0005	
Coefficient of friction (pole/hinge)	0.000002	
Cart mass	$1\,kg$	
Long pole mass	$0.1\,kg$	
Short pole mass	$0.01\,kg$	
Long pole length	$1.0\,m$	
Short pole length	$0.1\,m$	

TABLE II
RESULTS FOR DOUBLE POLE BALANCING WITH VELOCITY

	Evaluations	Generations	Nets	Hidden neurons
SANE [8]	12600	63	200	–
ESP [8]	3800	19	200	–
NEAT [28]	3600	24	150	? (0–4)
Cortex	2547	17	150	0.27

The first experiment used conditions which were similar to those reported in other studies [8], [12], [28] in order to ensure that the comparison is performed on an equal footing. Specifically, the initial angle of the long pole was taken at random from the interval $[-6°, 6°]$, the initial velocity was $0\,m/s$, and the simulation started with the cart located at the centre of the track ($2.4\,m$ from either end of the track). The results are presented in Table II together with the results reported in [28] and [8]. It is indicative that the number of hidden neurons was very low, showing that on average only one in four solutions required a hidden neuron. The complexity of the solution is clearly low, in agreement with the conclusion in [12] that evolution can find surprisingly simple solutions to problems considered very difficult by humans. A human designing a controller for the double pole balancing task would therefore tend to overengineer the solution, as exemplified by the use of a fully connected, fully recurrent network with 10 hidden neurons in [31]. Potentially, the large number of hidden neurons could make the network prone to overfitting, thus actually *preventing* it from finding an optimal solution.

2) Generalisation Test: After solving the task for the restricted set of conditions presented above, Cortex was subjected to a more rigorous *generalisation* test to evaluate its performance over a wider range of conditions. It should be noted that previous studies [12], [28] have evaluated the generalisation performance by testing all combinations of 0.05, 0.25, 0.5, 0.75 and 0.95 of the maximal values of the cart position, cart velocity, long pole angle and long pole angular
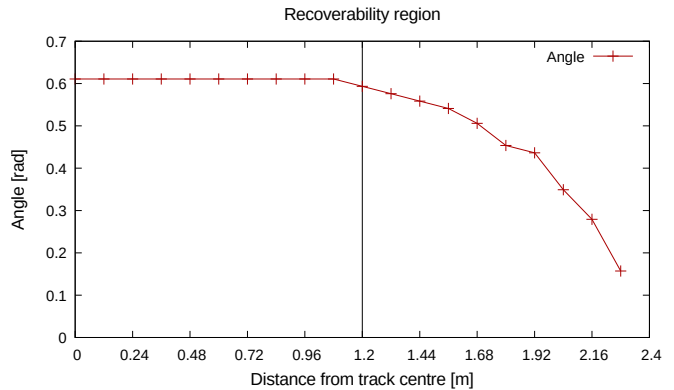

Fig. 8. Maximal angle from which the system can recover if the initial position of the cart is $2.4\,m$ from either end of the track. The initial velocity of the cart and the initial angular velocity of the long pole are both 0, and the maximal allowable force ($10\,N$) is applied constantly until the cart reaches the end of the track. The angles were tested from $1°$ to $36°$ in $1°$ increments. It is clear that the cart cannot push the long pole past the equilibrium point if the initial angle is $36°$ and the cart is more than $1.2\,m$ away from the centre of the track. This tendency becomes more pronounced as the initial position of the cart approaches the end of the track.

velocity, for a total of 625 ($=5^4$) experiments.

From a purely physical point of view, if the long pole is leaning at a large angle and the cart is located near the end of the track while travelling at high speed, it is highly unlikely that the poles can be recovered to a stable region from which they could be balanced for an appreciable amount of time. Performing such a test with deliberately impossible conditions was considered uninformative, and therefore a different approach to generalisation was adopted in this study. The approach was similar to that reported in [3], but with several extensions. First, a very simple experiment was designed with the aim to find the maximal angle from which the system could recover as a function of the initial position of the cart. In this experiment, the initial velocity of the cart and the initial angular velocity of the long pole were set to 0, and the maximal allowable force was applied in the same direction until the cart reached the end of the track. The initial position of the cart was varied from 0 to $2.4\,m$ in increments of $0.12\,m$, and the initial angle of the long pole was varied from $1°$ to $36°$ in increments of $1°$. Intuitively, the maximal angle from which the system can recover should gradually *decrease* as the initial position approaches the end of the track.

This intuition was confirmed by a simulation of this experiment (Fig. 8). Looking at the plot, it is immediately apparent that the system cannot physically recover to a long-term stable condition if the initial angle is $36°$ and the initial distance of the cart from the centre of the track is $1.2\,m$ or greater. In other words, under these conditions the long pole will never go past the equilibrium point, even if the cart is pushed *continuously* with the maximal allowable force all the way to the end of the track. Unsurprisingly, the situation worsens the closer the cart is to the end, which is alarming considering that this

experiment does not even include the effects of the initial velocity of the cart and initial angular velocity of the long pole. Therefore, it seems that the set of initial conditions should fall within the recoverability region in order to determine the generalisation performance of the controller in a meaningful way. Importance should also be given to the facts that the cart needs time to slow down to $0\,m/s$ *before* reaching the end of the track, and that the controller can never actually produce the maximal force (the `tanh` activation function approaches ± 1 asymptotically). Furthermore, in generalisation tests in previous studies, the controller was tested for only 1000 time steps ($10\,s$ of simulation time) for each of the 625 conditions described above. However, 1000 time steps seems insufficient to claim that the controller has evolved a working strategy.

In light of the above considerations, the generalisation test in this study checked whether the controller was able to balance the poles when the cart's initial position was -1.2 to $1.2\,m$ away from the centre of the track ($1.2\,m$ away from the centre in either direction) in increments of $0.3\,m$, and the initial angle of the long pole was $-15°$ to $15°$ in increments of $3°$. The limit of $15°$ was chosen because the cart should be given enough time to slow down to $0\,m/s$ before reaching the end of the track from any of the starting positions. These constraints were based on the results presented in Fig. 8. The remaining parameters were the same as those in Table I. The controller was deemed a solution if it could balance the poles for *each* combination of initial position and long pole angle for 200000 steps (more than $30\,min$ of simulation time). 50 experiments were performed, where an experiment was set to terminate after 1000 generations if no solution was found.

Cortex managed to generalise to all of the initial conditions in all 50 experiments with 27547 evaluations and 194 generations on average, which is commendable given the recoverability region in Fig. 8. The average solution had four hidden neurons. In comparison, NEAT required an average of 33184 evaluations to generalise to 286 of the 625 conditions [28]. Unfortunately, direct comparison with previous generalisation results is infeasible because the evaluation methods differ. In this regard, a more rigorous exploration of the recoverability region, including non-zero values for the initial speed and angular velocity for both poles, should provide an even more complete set of initial conditions for generalisation testing.

V. Conclusion

The approach to speciation and genotype representation proposed in this study, in combination with the accurate and reproducible network evaluation procedure, is shown to be effective in rapidly finding solutions to common benchmark problems, and allows Cortex to converge on a solution with reliability and speed competitive with existing platforms.

There are a number of points which remain to be addressed. First, the proposed encoding is *direct*, which means that there is a straightforward mapping of genotype to phenotype. As *indirect* encoding has been shown to evolve solutions faster than direct encoding for the same problems [7], [25], it would be instructive to extend Cortex to support indirect encoding in order to be able to compare it with existing indirect encoding NE schemes, such as (ES-)HyperNEAT. In this regard, Cortex also supports mutation of the neuron activation function. Although this idea has been considered in existing NE platforms, it is still a somewhat rare concept which is employed, for example, in indirect encodings such as HyperNEAT [13]. This mutation type introduces higher versatility and diversity in, for example, interactive NE applications, and its potential use in an indirect encoding version of Cortex will be explored in future studies.

Furthermore, synaptic plasticity allows networks to adapt to the environment at *runtime* by changing the weights in response to neuron activity. In the experiments above, the weights are changed only by mutations, after which the network is evaluated with fixed weights. It has been demonstrated that plasticity can play a positive role in allowing networks to adapt to their environment and to respond more smoothly to sudden changes thereof [24]. Therefore, the effects of synaptic plasticity on the proposed framework will also be explored, particularly with respect to highly dynamic tasks such as double pole balancing.

Lastly, preliminary experiments have indicated that the proposed genotype representation may also be beneficial for evolving the topology of spiking neural networks. This direction of research has been poorly explored, likely due to the difficulties involved in evaluating spiking neural networks of arbitrary topology. The Cortex software platform has been made available to promote research and development in this area.

References

[1] Aimone, J. B., Wiles, J. & Gage, F. H. "Potential role for adult neurogenesis in the encoding of time in new memories". *Nature neuroscience* 9.6 (2006), pp. 723–727.

[2] Angeline, P. J., Saunders, G. M. & Pollack, J. B. "An Evolutionary Algorithm That Constructs Recurrent Neural Networks". *IEEE Transactions on Neural Networks* 5 (1993), pp. 54–65.

[3] Blair, A. "Incremental evolution of HERCL programs for robust control". *Proceedings of the 2014 conference on genetic and evolutionary computation companion*. ACM. 2014, pp. 27–28.

[4] Dasgupta, D. & Mcgregor, D. "Designing Application-Specific Neural Networks Using the Structured Genetic Algorithm". *Proceedings of the International Conference on Combinations of Genetic Algorithms and Neural Networks*. 1992, pp. 87–96.

[5] Dodd, N. "Optimisation of network structure using genetic techniques". *International Joint Conference on Neural Networks*. IEEE. 1990, pp. 965–970.

[6] Gage, F. H. "Neurogenesis in the Adult Brain". *Journal of Neuroscience* 22 (2002), pp. 612–613.

[7] Gauci, J. & Stanley, K. O. "Autonomous Evolution of Topographic Regularities in Artificial Neural Networks". *Neural Computation* 22.7 (July 2010), pp. 1860–1898.

[8] Gomez, F. J. & Miikkulainen, R. "Solving non-Markovian control tasks with neuroevolution". *IJCAI*. Vol. 99. 1999, pp. 1356–1361.

[9] Gomez, F., Schmidhuber, J. & Miikkulainen, R. "Accelerated Neural Evolution through Cooperatively Co-evolved Synapses". *Journal of Machine Learning Research* 9 (2009), pp. 937–965.

[10] Gould, E., Beylin, A., Tanapat, P., Reeves, A. & Shors, T. J. "Learning enhances adult neurogenesis in the hippocampal formation". *Nature neuroscience* 2.3 (1999), pp. 260–265.

[11] Gruau, F. "Cellular encoding as a graph grammar". *IEE Colloquium on Grammatical Inference: Theory, Applications and Alternatives*. Ed. by Lucas, S. 92. IET, 1993, pp. 17/1–10.

[12] Gruau, F., Whitley, D. & Pyeatt, L. "A comparison between cellular encoding and direct encoding for genetic neural networks". *Proceedings of the 1st annual conference on genetic programming*. MIT Press. 1996, pp. 81–89.

[13] Huizinga, J., Clune, J. & Mouret, J.-B. "Evolving neural networks that are both modular and regular: Hyper-NEAT plus the connection cost technique." *GECCO*. Ed. by Arnold, D. V. ACM, 2014, pp. 697–704.

[14] Kempermann, G. "Why New Neurons? Possible Functions for Adult Hippocampal Neurogenesis". *Journal of neuroscience* 22.3 (2002), pp. 635–638.

[15] Kitano, H. "Designing Neural Networks Using Genetic Algorithms with Graph Generation System". *Complex Systems* 4 (1990), pp. 461–476.

[16] Luus, R. & Jaakola, T. H. I. "Optimization by direct search and systematic reduction of the size of search region". *AIChE Journal* 19.4 (1973), pp. 760–766.

[17] Mahner, M. & Kary, M. "What exactly are genomes, genotypes and phenotypes? And what about phenomes?" *Journal of Theoretical Biology* 186.1 (1997), pp. 55–63.

[18] Miller, B. L. & Shaw, M. J. "Genetic algorithms with dynamic niche sharing for multimodal function optimization". *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE. 1996, pp. 786–791.

[19] Montana, D. J. & Davis, L. "Training Feedforward Neural Networks Using Genetic Algorithms". *Proceedings of the 11th International Joint Conference on Artificial Intelligence*. San Francisco, CA: Morgan Kaufmann, 1989, pp. 762–767.

[20] Montana, D. J., Wyk, E. V., Brinn, M., Montana, J. & Milligan, S. "Evolution of internal dynamics for neural network nodes." *Evolutionary Intelligence* 1.4 (2009), pp. 233–251.

[21] Pedersen, M. E. H. "Tuning & simplifying heuristical optimization". PhD thesis. University of Southampton, 2010.

[22] Pujol, J. C. F. & Poli, R. "Evolving the Topology and the Weights of Neural Networks Using a Dual Representation". *Applied Intelligence Journal* 8.1 (1998), pp. 73–84.

[23] Richards, N., Moriarty, D. & Miikkulainen, R. "Evolving Neural Networks to Play Go". *Applied Intelligence* 8 (1998), pp. 85–96.

[24] Risi, S. & Stanley, K. O. "A unified approach to evolving plasticity and neural geometry". *IJCNN*. IEEE, 2012, pp. 1–8.

[25] Risi, S. & Stanley, K. O. "An Enhanced Hypercube-Based Encoding for Evolving the Placement, Density, and Connectivity of Neurons". *Artificial Life* 18.4 (2012), pp. 331–363.

[26] Schaffer, J., Whitley, D. & Eshelman, L. "Combinations of genetic algorithms and neural networks: a survey of the state of the art". *International Workshop on Combinations of Genetic Algorithms and Neural Networks*. IEEE, 1992, pp. 1–37.

[27] Stanley, K. O. "Efficient evolution of neural networks through complexification". PhD thesis. Department of Computer Sciences, University of Texas, 2004.

[28] Stanley, K. O. & Miikkulainen, R. "Evolving Neural Networks through Augmenting Topologies". *Evolutionary Computation* 10.2 (2002), pp. 99–127.

[29] Stanley, K. O. & Mikkulainen, R. "Competitive Coevolution Through Evolutionary Complexification". *Journal of Artificial Intelligence Research* 21 (2004), pp. 63–100.

[30] *The NeuroEvolution of Augmenting Topologies (NEAT) Users Page*. URL: https://www.cs.ucf.edu/~kstanley/neat.html (Accessed on Mar. 23, 2016).

[31] Wieland, A. P. "Evolving neural network controllers for unstable systems". *International Joint Conference on Neural Networks*. Vol. 2. IEEE. 1991, pp. 667–673.