In *Proceedings of the 1998 IEEE Conference on Evolutionary Computation (ICEC-98, Anchorage, AK).*
Piscataway, NJ: IEEE, 1998.

Hierarchical Evolution of Neural Networks*

David E. Moriarty† and Risto Miikkulainen
Department of Computer Sciences
The University of Texas
Austin, TX 78712
moriarty@isi.edu,risto@cs.utexas.edu

Abstract—
In most applications of neuro-evolution, each individual in the population represents a complete neural network. Recent work on the SANE system, however, has demonstrated that evolving individual neurons often produces a more efficient genetic search. This paper demonstrates that while SANE can solve easy tasks very quickly, it often stalls in larger problems. A hierarchical approach to neuro-evolution is presented that overcomes SANE's difficulties by integrating both a neuron-level exploratory search and a network-level exploitive search. In a robot arm manipulation task, the hierarchical approach outperforms both a neuron-based search and a network-based search.

I. Introduction

Artificial evolution is an effective method for forming neural networks in tasks where sparse reinforcement precludes normal supervised methods such as backpropagation. The evolutionary framework frees the implementor from generating training examples and provides a highly adaptive mechanism for dynamic environments. Recent work has shown evolved neuro-controllers effective in several unstable, dynamic control tasks [3], [6], [8], [13]. The bane of the evolutionary methods, however, has been the large number of fitness evaluations that must be performed to achieve a high level of performance.

Recently, we have developed a more efficient evolutionary approach called SANE (Symbiotic, Adaptive Neuro-Evolution) [6], which explicitly decomposes the evolutionary search for a complete solution into several parallel searches for partial solutions. In most approaches to neuro-evolution, each individual represents a complete neural network that is evaluated independently of other networks in the population [2], [5], [9], [13]. By treating each member as a separate full solution, the genetic algorithm focuses the search towards a single type of individual, which normally leads to convergence on a single solution [4].

In contrast, SANE's individuals do not represent complete solutions; they represent partial solutions. More specifically, each individual represents a single hidden neuron in a three layer neural network. Complete networks are built by decoding several individuals' chromosomes and neurons are co-evolved by evaluating how well they perform when combined with other neurons in the population. Since a single neuron cannot perform the whole task alone, effective networks must contain neurons that perform different functions. Inherent evolutionary pressures are therefore present to evolve several different types or *specializations* of individuals within the population. Thus, unlike the network-level evolution, the population does not converge to a single individual, but instead remains diverse throughout evolution.

Evolution at the neuron level provides two primary advantages. First, since the population is diverse, there will always be a rich collection of genetic material from which crossover operations can create new types of individuals. In contrast, a genetic search in a converged population is directed by the mutation operator and may progress very slowly. Second, evolution at the neuron level more accurately evaluates the fundamental building blocks of neural networks. In a network-level evolution, each neuron is implemented only with the other neurons encoded on the same chromosome. Consequently, a very good neuron may exist on a chromosome but be subsequently lost because the other neurons on the chromosome are poor. In SANE's neuron-level evolution, neurons are continually recombined with other neurons, to more accurately evaluate their contribution to the neural networks.

Empirically, we have found SANE's neuron-level evolution to perform very well in simple benchmarks. For example, in the well-known inverted pendulum problem, SANE found solutions much faster than temporal difference methods such as the Adaptive Heuristic Critic [1] and Q-learning [12], and other neuro-evolution systems such as GENITOR [13]. SANE is effective in these problems because solutions are plentiful and generally found in the first 10 generations [6]. Unfortunately, in more difficult problems that require high precision within the solution space, SANE is unable to capitalize on the top neuron combinations to improve the population in later generations. We have found that a converged network-level population operating through mutation eventually catches up to SANE and finds better solutions. Thus, an important research question is how to combine the early efficiency of a neuron-level search with the fine tuning of a network-level search.

This paper presents a hierarchical approach to neuro-evolution that incorporates the explorative nature of a neuron-based search with the exploitive feature of a network-based search. The approach maintains and simultaneously evolves two populations: a population of neurons and a population of network blueprints. In a sophisticated robot arm task, the hierarchical approach finds better solutions than a neuron-based search and is more efficient than

*This research was supported in part by the National Science Foundation under grant #IRI-9504317.
†Now at USC/ISI, 4676 Admiralty Way, Marina del Rey, CA 90292.

a network-based search. These results suggest that a hierarchical approach to neuro-evolution may be appropriate for solving large scale problems.

II. Hierarchical Neuro-Evolution

Our hierarchical approach incorporates ideas from a standard network-level evolutionary search and SANE's neuron-level search. Two populations are maintained and evolved: a population of neurons and a population of neural network blueprints. Each individual in the neuron population specifies a set of connections to be made within a neural network. Each individual in the network blueprint population specifies a set of neurons to include in a neural network. Conjunctively, the neuron evolution searches for effective partial networks, while the blueprint evolution searches for effective combinations of the partial networks.

The basic fitness evaluation algorithm for the hierarchical approach is given in the following pseudo-code, which is explained in the next two sections.

for each neuron n in population P_n
    $n.fitness \leftarrow 0$
    $n.participation \leftarrow 0$
for each blueprint b in population P_b
    $nn \leftarrow \mathbf{decode}(b)$
    $b.fitness \leftarrow \mathbf{task}(nn)$
    for each n in b
        $n.fitness \leftarrow n.fitness + b.fitness$
        $n.participation \leftarrow n.participation + 1$
for each neuron n in population P_n
    $n.fitness \leftarrow n.fitness$ / $n.participation$

A. The Neuron Population

The neuron-level evolution uses the same basic strategy as SANE [6] for evaluating and recombining good individual neurons. Neurons are evaluated based on the average performance of the networks in which they participate.

The population consists of a large collection of hidden neuron definitions for a three-layer feedforward network (figure 1). A neuron is represented by a series of connection definitions that describe the weighted connections routed through that neuron from the input layer and to the output layer. Each neuron has a fixed number of connections, but may allocate them arbitrarily among the units in the input and output layers. A connection definition consists of a label and weight pair. The label is an integer value that specifies a specific input or output unit, and the weight specifies the strength of the connection. Figure 1 gives three example hidden neuron definitions and the resulting neural network.

The neurons are evolved using a generational evolutionary algorithm that iterates over two phases: an evaluation phase and a reproduction phase. During the evaluation stage, neuron subpopulations of size ζ are selected for participation and combined to form a neural network. The participation selection method depends on the blueprint

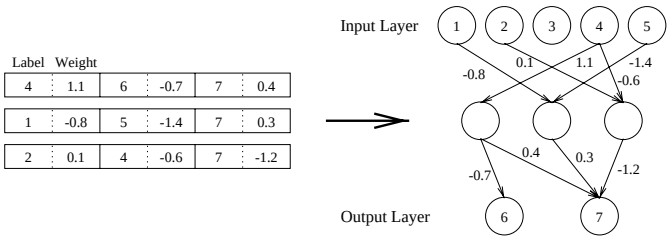

Fig. 1. A three-layer feedforward network is created from 3 neurons definitions. The neurons are shown on the left, and the corresponding network is shown on the right.

population and will be described in section II-B. The networks are evaluated in the task and the fitness is added to each participating neuron's fitness variable. At the end of the evaluation stage, each neuron's fitness is normalized by dividing the sum of its fitness scores by the number of networks in which it participated. The result is a measure of average fitness of the networks in which the neuron participated.

In the reproduction phase, genetic operators, such as selection by rank, one point crossover, and mutation, are used to obtain new neurons. For each neuron in the top 25% of the population (according to fitness rank), a mate is selected randomly among the top 25%. Each mating operation creates two offspring: a copy of one of the mates and a child created through one-point crossover. The second child created by crossover is discarded. Copying one of the parents reduces the effect of adverse neuron mutation on the blueprint-level evolution, as will be further explained in section II-B. The two offspring replace the worst-performing neurons in the population. Finally, Mutation at the rate of 1% per chromosome position is performed on the entire population as the last step in each generation.

Such an aggressive, elitist breeding strategy is not normally used in evolutionary applications, since it leads to quick convergence of the population. A neuron evolution, however, performs quite well with this aggressive selection strategy, since it contains strong evolutionary pressures against convergence.

B. The Network Blueprint Population

The purpose of the blueprint population is twofold: to organize the neurons into workable groups and to assign more trials to the better-performing neurons. In original SANE, networks are formed by randomly combining subpopulations of neurons. Here, the network blueprints specify which neurons should be connected together. As effective neuron combinations are found, they are preserved in the blueprint chromosomes and propagated to future generations.

Maintaining network blueprints produces more accurate neuron evaluations and concentrates the search on the best networks. Since neurons are connected systematically based on past performance, they are more consistently combined with other neurons that perform well together. Additionally, better-performing neurons gar-

ner more pointers from the blueprint population and thus participate in a greater number of networks. Biasing the neuron participation towards the historically better-performing neurons provides more accurate evaluations of the top neurons. The sacrifice, however, is that newer neurons may not receive enough trials to be accurately evaluated. In practice, allocating more trials to the top neurons produces a significant improvement over uniform neuron participation.

The primary advantage of evolving network blueprints, however, is the exploitation of the best networks found during evolution. Original SANE fosters no memory of the previous networks formed, and good neuron combinations found in one generation often never occur in subsequent generations. Such behavior causes the quick, explorative search to stall, because it cannot exploit the best neuron combinations. The hierarchical approach maintains the proficient collections of neurons in the blueprint chromosomes and ensures that the best networks are reconstructed. By *evolving* the blueprint population, the best neuron combinations are also recombined to form new, potentially better, collections of neurons. Hierarchical neuro-evolution thus provides a more exploitative search that can build upon the best networks found during evolution and focus the search in later generations.

Each individual in the blueprint population contains a series of neuron pointers. More specifically, a blueprint is an array, of size ζ, of address pointers to neuron structures. Figure 2 illustrates how the blueprint population is integrated with the neuron population. Initially, the blueprint pointers are randomly assigned to neurons in the neuron population. During the neuron evaluation stage, subpopulations of neurons are selected based on each blueprint's array of pointers. Thus unlike original SANE which forms networks by randomly selecting subpopulations of neurons, hierarchical neuro-evolution forms networks by following pointers in each blueprint.

Blueprints receive the fitness score of the neural network that they specify. After each blueprint has been evaluated, the blueprint population is ranked and crossover operations are applied to the top blueprints. The new offspring receive the same address pointers which the parent chromosomes contained. In other words, if a parent chromosome contains a pointer to a specific neuron, one of its offspring will point to that same neuron (barring mutation). The current genetic algorithm on the blueprint level is identical to the aggressive strategy used at the neuron level, however the similarity is not essential and a more-standard genetic algorithm could be used. Empirically, the aggressive strategy at the blueprint level coupled with the strong mutation strategy described below, has outperformed many of the more-standard genetic algorithms.

To avoid convergence problems at the blueprint level, a two-component mutation strategy is employed. First, one pointer in each offspring blueprint is randomly reassigned to another member of the neuron population. This strategy promotes participation of neurons other than the top neurons in subsequent networks. Thus, a neuron that does not participate in any networks can acquire a pointer and participate in the next generation. Since the mutation only occurs in the blueprint offspring, the neuron pointers in the top blueprints are always preserved.

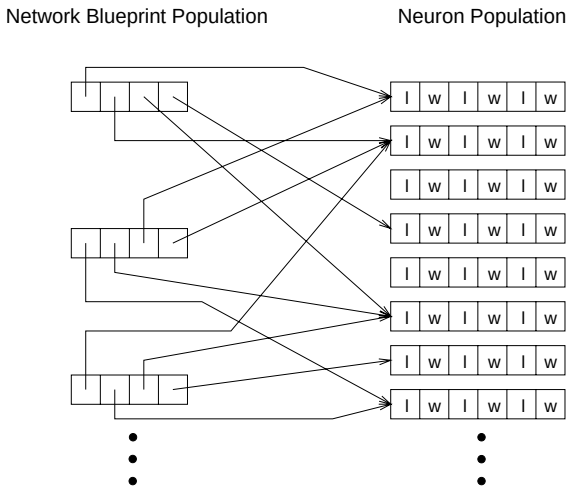

Fig. 2. An overview of the network blueprint population in relation to the neuron population. Each member of the neuron population specifies a series of connections (labels and weights) to be made within a neural network. Each member of the blueprint population specifies a series of pointers to specific neurons which are used to build a neural network.

The second mutation component is a selective strategy designed to take advantage of the new structures created by the neuron evolution. Recall that a breeding neuron produces two offspring: a copy of itself and the result of a crossover operation with another breeding neuron. Each neuron offspring is thus similar to and potentially better than its parent neurons. The blueprint evolution can use this knowledge by occasionally replacing pointers to breeding neurons with pointers to offspring neurons. In the experiments described in this paper, pointers are switched from breeding neurons to one of their offspring with a 50% probability. Again, this mutation is only performed in the offspring blueprints, and the pointers in the top blueprints are preserved.

This selective mutation mechanism has two advantages. First, because pointers are reassigned to neuron offspring resulting from crossover, the blueprint evolution can explore new neuron structures. Second, because pointers are also reassigned to offspring that were formed by copying the parent, the blueprints become more resilient to adverse mutation in the neuron evolution. If pointers were not reassigned to copies, many blueprints would point to the same exact neuron, and any mutation to that neuron would affect every blueprint pointing to it. When pointers are occasionally reassigned to copies, however, such mutation is limited to only a few blueprints. The effect is similar to schema promotion in standard genetic algorithms. As the population evolves, highly fit schema (i.e. neurons) become more prevalent in the population, and mutation to one copy of the schema do not affect other copies in the population.

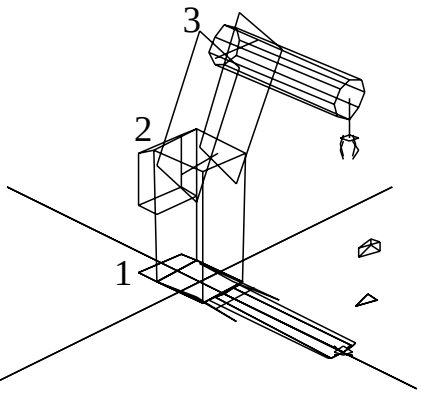

Fig. 3. The Simderella robot arm simulation of the OSCAR robot.

III. Evaluation: Manipulating a Robot Arm

Much of our research is currently directed at applications of neuro-evolution for robot control. Specifically, we are evolving neural networks to control a robot arm using visual input. Most neural network applications to this problem learn hand-eye coordination through supervised training methods which require examples of correct behavior. Unfortunately, the current approaches for generating training examples for robot arm control are very limited and ineffective in uncertain or obstacle-filled domains. Neuro-evolution, however, does not require input/output examples and can learn the intermediate joint rotations necessary for avoiding obstacles in uncertain environments. Thus, neuro-evolution provides a promising framework for the automatic development of robot arm controllers in realistic domains.

Three neuro-evolutionary approaches were used to develop controllers in this domain: a neuron-based search, a network-based search, and the hierarchical search. The particular robot arm simulator used in these experiments was the Simderella 2.0 package written by van der Smagt [10]. Simderella, shown in figure 3, is a simulation of the OSCAR 6 degree-of-freedom anthromoporphic arm.

The goal of the network controller is to maneuver the arm to a position within ten centimeters of a randomly-placed target object (A secondary network, not described in this paper, can learn the smaller, finite movements necessary to grasp the object [7]). The controller moves the arm by specifying joint rotations to the first three joints at each time step. The arm contains a camera, located in the end effector (hand), that provides the x, y, and z distances of the target object from the current end effector position. At each time step, the neural network controller receives its current joint positions and the target distances as it's input and generates the degrees to which each of its three controllable joints are to be rotated as its output.

A. Experimental Setup

To test the advantages of each approach, the three approaches (neuron-based, network-based, and hierarchical) were implemented in the Simderella simulator to evolve the hidden layer connections and weights of a neuro-control network. The neural network controller contained 6 input, 8 hidden, and 7 output units. The input units correspond to the x, y, and z relative distances returned by the hand camera and the current joint positions of the first three joints normalized between 0 and 1. The rotation of each joint was interpreted from two unique output units. The first unit specifies the direction of rotation (positive or negative) based on the sign of its total activation. The second output unit is a sigmoidal unit that specifies the amount of rotation, normalized between 0.0 and 5.0 by multiplying the sigmoid output by 5.0. Limiting each joint rotation to +/- 5 degrees forces the network to make several small joint rotations to reach the target. Thus a bad rotation in one time step can be more easily corrected in a subsequent time step. A final threshold output unit is included as an override unit that can prevent movement regardless of the activations of the other output units.

Each network contained 8 hidden units with 12 connections per unit. Connection definitions were decoded from the chromosomes as described in section II-A. In the neuron-based search, each individual consisted of a single hidden unit, while in the network-based search, an individual consisted of 8 hidden units concatenated on a single chromosome.

To focus the comparison on the different strategies of neuro-evolution, rather than the choice of parameter settings, several preliminary experiments were run to discover effective parameter values for each approach. With the network-based approach, a population size of 100 networks was found to be more effective than populations of 50 or 200. Keeping the number of network evaluations per generation constant across each approach, a neuron population size of 200 was used for the neuron-based approach and 800 for the hierarchical approach.

A neuron-based search requires a smaller population than our hierarchical approach because neurons are evaluated through random combinations. The population must be small enough to allow neurons to participate in several networks per generation. For example, randomly selecting 8 neurons for 100 networks in a 200 neuron population gives each neuron an expected network participation rate of 4 networks per generation. In the hierarchical approach, the neuron population is not as restricted since neuron participation is dictated by the network individuals. The hierarchical approach skews the participation rate towards the best neurons and leaves many neurons unevaluated in each generation. An unevaluated neuron is normally garbage, since no network individual uses it to build a network.

Not surprisingly, the aggressive genetic selection strategy described in section II-A performed poorly in the network-based search. Some good solutions were found quickly, however, the populations often converged to suboptimal solutions and essentially became stuck. A binary tournament selection strategy produced more consistent and better average results for the network-based search.

Each neural network evaluation began with random, but legal, joint positions and a random target position. A total of 450 target positions were created and separated into a

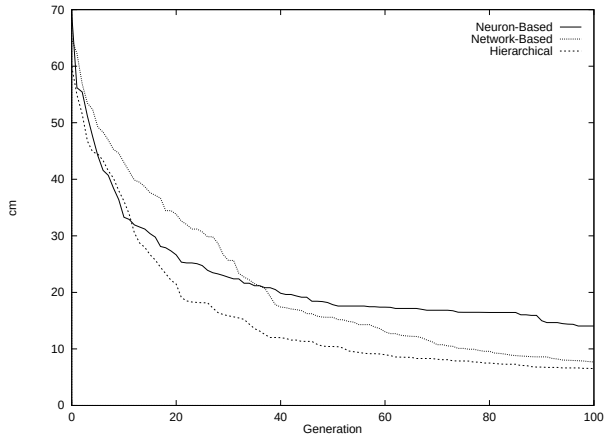

Fig. 4. The average distance from 50 randomly placed targets per generation for each level of evolution. The distances are averaged over 20 simulations.

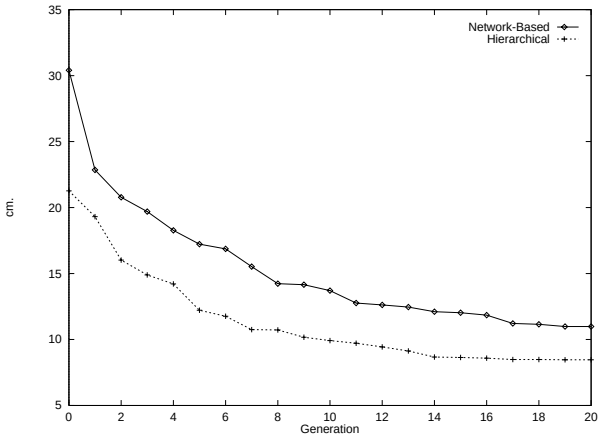

Fig. 5. Adaptation comparisons between the hierarchical and network-based searches

400 position training set and a 50 position test set. During each trial, a network was allowed to move the arm until one of the following conditions occurred: the network stopped the arm, the arm is in an illegal position, or the maximum number of moves (25) is exceeded.

The score for each trial was computed as the percentage of distance that the arm covered from its initial starting point to the target position. For example, if the arm started 120 cm from the target and its final position was 20 cm from the target, the network received a score of $(120-20)/120 = 0.83$. The percentage of distance covered, instead of the absolute final distance, provides a fairer comparison between a network that receives a close target and a network that receives a distant target.

Twenty simulations of each approach were run, each for 100 generations, which requires 10,000 network evaluations. The best network of each generation (according to fitness) was then tested on the 50 target test set.

B. Learning Speed Results

Figure 4 plots the average distance from the targets in the test set per generation. The distance refers to the best distance found at or before each generation during a single simulation. The average distance refers to the best distances averaged over 20 simulations. The graph effectively illustrates the problems of the neuron search. Good solutions are found early, but the population eventually stalls and cannot generate better solutions. The slower network search eventually surpasses the neuron search and continues to improve to the best solutions. The hierarchical search, however, achieves the same early efficiency of the neuron search and continues to generate better solutions throughout the evolution.

C. Adaptation Results

The second set of experiments tested the ability to adapt to changes in the domain. Populations were evolved as described above until a network was found that averaged less than 10 cm over the test set. The domain was then changed by removing the information of the position of the first joint. van der Smagt [11] showed that correct control decisions can be generated without knowledge of the first joint position; there is sufficient information from the hand camera and remaining joints to compute the position of the first joint. Thus, by evolving with the first joint information and then removing it, the population must adapt its network controllers to identify the first joint information from the other input units.

Figure 5 shows the adaptive performance of the network level and hierarchical approaches. The neuron-based search is not plotted since it failed to reliably find solutions averaging less than 10 cm. While the hierarchical search was able to adapt its control policy to reach the 10 cm target within 10 generations, the network-based search adapted much more slowly, requiring 24 generations on the average. Increasing the mutation rate did produce faster adaptation in the standard approach, but it hindered the original learning rate.

D. A Look Inside the Hierarchical Approach

As described earlier, one of the advantages of the hierarchical approach is that neuron participation is biased towards the top performing neurons. In other words, the best neurons will participate in more networks than the newer or poorer neurons. Figure 6 shows the typical rate of participation in the neuron population in the last generation. The data was collected from a single simulation, however, other simulations exhibited very similar behavior. Of the 800 opportunities for participation,[1] 40% were filled by neurons ranked in the top 6% of the population and 78% by neurons in the top 25%. Neurons ranked from 300 to 750 did not participate in any networks. Neurons ranked at the bottom of the population were evaluated but performed so poorly (e.g. moving the arm away from the target) that they were ranked below neurons which had no participations.

Table I shows the neurons of the top 20 network blueprints during the last generation of a single simulation. The neuron pointer numbers refer to the rank of that neu-

[1] 100 networks are formed per generation, each with 8 neurons.

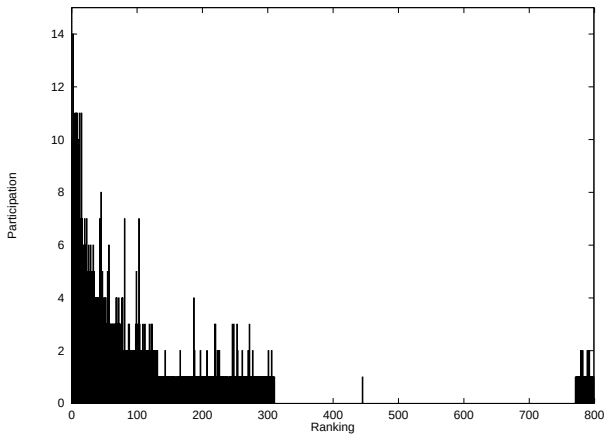

Fig. 6. The number of network participations for each neuron in the neuron population using the hierarchical approach.

Network Rank	Neuron Pointers							
1	121	134	0	13	2	135	38	17
2	12	133	34	41	136	137	29	9
3	132	138	0	13	46	18	67	17
4	40	75	72	28	27	106	61	187
5	40	82	74	115	33	39	10	23
6	76	4	144	16	30	1	140	11
7	42	24	37	6	7	8	64	3
8	32	48	0	13	2	18	38	17
9	58	147	139	84	148	36	146	3
10	129	88	72	145	87	128	130	22
11	52	45	29	16	30	1	5	11
12	56	142	34	49	85	20	10	141
13	76	4	21	66	7	14	99	113
14	150	24	37	6	153	83	64	3
15	42	24	151	73	152	104	5	79
16	59	60	52	28	149	1	51	22
17	81	156	78	95	100	49	160	158
18	70	26	55	6	103	57	43	9
19	56	35	155	6	154	161	159	157
20	105	62	8	90	2	1	35	3

TABLE I

THE NEURONS IN THE TOP BLUEPRINTS IN THE LAST GENERATION OF A SIMULATION USING THE HIERARCHICAL SEARCH. THE TABLE SHOWS A VERY DIVERSE COLLECTION OF NEURONS AND SEVERAL EXAMPLES OF OFFSPRING NEURONS PRESENT IN THE BEST NETWORKS.

ron in the population after the fitness had been distributed. For example, the top blueprint included the 122nd ranked neuron, the 135th ranked neuron, the top ranked neuron, and so on. The first conclusion from the table is that the blueprint population is quite diverse. Blueprints 1 and 8 are the most similar, but they only share 5 out of the 8 neuron pointers. Thus the mutation strategy presented in section II-B appears to provide sufficient diversity, allowing the blueprint population to sample many different combinations of neuron pointers.

Table I also shows that many neurons ranked in the 100-150 range are included in the top networks. Interestingly, these neurons only participate in 1 or 2 networks per generation (figure 6). Closer inspection reveals that the majority of these neurons are new, offspring neurons generated during the previous generation. Thus, from the prevalent use of these neurons in the top blueprints, it can be concluded that the blueprint evolution is making effective use of the new neural structures created by the neuron evolution.

IV. Conclusion

Evolving neural networks at the neuron level offers important advantages over the more standard evolution of complete neural networks. Neuron evolution maintains population diversity and provides more accurate evaluation of the genetic building blocks, which produces a faster, more efficient genetic search. Unfortunately, neuron evolution alone, as in the SANE system, often cannot build upon the top neural networks in later generations and has difficulty pinpointing the best solutions. In contrast, the slower network-based search is more proficient at fine tuning and can more reliably reach the best solutions. Incorporating an outer-loop network blueprint evolution on top of SANE's neuron evolution combines the advantages of both approaches by focusing the search on the best neuron combinations. Demonstrated in a sophisticated robot arm manipulation task, Hierarchical SANE consistently reached the desired level of proficiency in half as many generations as a network-based search. Hierarchical neuro-evolution is an important extension of the SANE system, since it makes it possible to tackle more challenging tasks that require both timely and precise solutions.

References

[1] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846, 1983.

[2] Richard K. Belew, J. McInerney, and N. N. Schraudolph. Evolving networks: Using genetic algorithm with connectionist learning. In J. D. Farmer, C. Langton, S. Rasmussen, and C. Taylor, editors, *Artificial Life II*, Reading, MA, 1991. Addison-Wesley.

[3] Dave Cliff, Inman Harvey, and Phil Husbands. Explorations in evolutionary robotics. *Adaptive Behavior*, 2:73–110, 1993.

[4] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[5] John R. Koza and James P. Rice. Genetic generalization of both the weights and architecture for a neural network. In *International Joint Conference on Neural Networks*, volume 2, pages 397–404, New York, NY, 1991. IEEE.

[6] David E. Moriarty and Risto Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32, 1996.

[7] David E. Moriarty and Risto Miikkulainen. Evolving neurocontrollers for hand-eye coordination and obstacle avoidance in a robot arm. In *From Animals to Animats: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB-96)*, Cape Cod, MA, 1996.

[8] Stefano Nolfi, Dario Floreano, Orazio Miglino, and Francesco Mondada. How to evolve autonomous robots: Different approaches in evolutionary robotics. In *Artificial Life IV*, Cambridge, MA, 1994.

[9] Stefano Nolfi and D Parisi. Growing neural networks. In *Artificial Life III*, Reading, MA, 1992. Addison-Wesley.

[10] Patrick van der Smagt. Simderella: A robot simulator for neurocontroller design. *Neurocomputing*, 6(2), 1994.

[11] Patrick van der Smagt. *Visual Robot Arm Guidance using Neural Networks.* PhD thesis, The University of Amsterdam, Amsterdam, The Netherlands, 1995.

[12] C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, University of Cambridge, England, 1989.

[13] Darrell Whitley, Stephen Dominic, Rajarshi Das, and Charles W. Anderson. Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, 13:259–284, 1993.