To appear in *Proceedings of the Twentieth International Conference on Machine learning* (ICML-03, Washington, DC), 2003.

Utilizing Domain Knowledge in Neuroevolution

James Fan JFAN@CS.UTEXAS.EDU
Raymond Lau LAURK@CS.UTEXAS.EDU
Risto Miikkulainen RISTO@CS.UTEXAS.EDU
Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712

Abstract

We propose a method called Rule-based ESP (RESP) for utilizing prior knowledge in evolving Artificial Neural Networks (ANNs). First, KBANN-like techniques are used to transform a set of rules into an ANN, then the ANN is trained using the Enforced Subpopulations (ESP) neuroevolution method. Empirical results show that in the Prey Capture domain RESP can reach higher level of performance than ESP. The results also suggest that incremental learning is not necessary with RESP, and it is often easier to design a set of rules than an incremental evolution scheme. In addition, an experiment with some of the rules deleted suggests that RESP is robust even with an incomplete knowledge base. RESP therefore provides a robust methodology for scaling up neuroevolution to harder tasks by utilizing existing knowledge about the domain.

1. Introduction

Over the past decade there has been much interest in Hybrid Systems, where a large intelligent system is built from components with different architectures and designs, such as Artificial Neural Networks (ANN) and Rule-Base Systems (Wermter, 1997; Hudson et al., 1992). This way the strengths of the different approaches can be combined. Hybrid Neural Systems, i.e. those that include neural network components, have shown good promise and good results in numerous domains (McGarry et al., 1999). One successful such approach is KBANN (Towell & Shavlik, 1994), where a rule base is converted to an ANN which is then further trained with backpropagation.

In this paper, we will apply the hybrid systems idea to neuroevolution in a method called RESP (Rule-based ESP), based on the ESP (Enforced Subpopulations) neuroevolution system (Gomez & Miikkulainen, 1997). Neuroevolution in general and ESP in particular has been shown highly effective in several difficult reinforcement learning problems (Gomez & Miikkulainen, 1997; Gomez & Miikkulainen, 2002; Fogel, 2001; Nolfi et al., 1994; Cliff et al., 1993; Whitley et al., 1993), but it is often necessary to train the system through a series of incrementally more challenging tasks before the actual task can be solved (Gomez & Miikkulainen, 1997; Wieland, 1990). In RESP, we will solve this problem by first using a version of KBANN to convert domain knowledge into an ANN and then evolving the ANN further with ESP. By utilizing prior knowledge this way, we can show that:

1. domain knowledge allows RESP to achieve better performance than ESP;
2. incremental learning is not necessary for RESP;
3. it is often easier to come up with rules than to devise an incremental learning scheme;
4. RESP is robust and can perform well even with some rules omitted.

With RESP, neuroevolution can be applied to more complex domains than before, using the existing knowledge about the domain as a starting point.

2. Rule-based ESP (RESP)

RESP consists of a knowledge-transferring process and a training process. First, a modified version of KBANN, consisting of the following steps, is used to transfer the knowledge into an ANN:

1. Create a set of rules that describe the domain knowledge. As mentioned in (Towell & Shavlik,

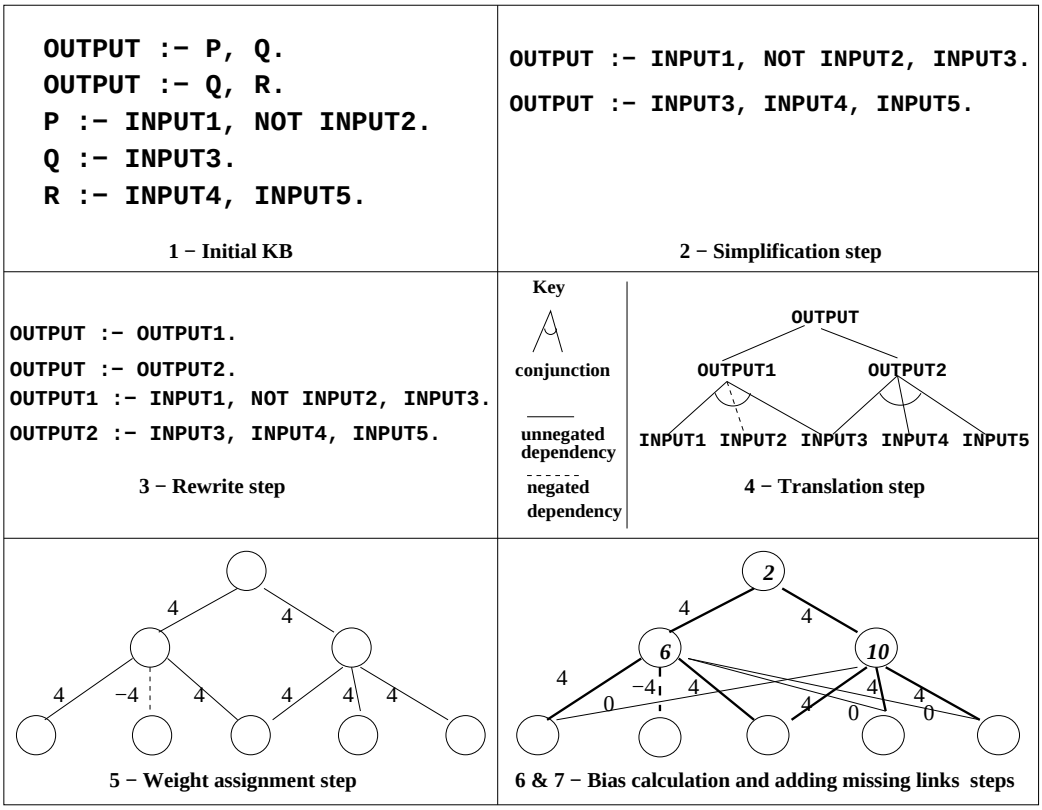

Figure 1. The rules-to-network algorithm on a sample KB. Panel *1* gives the initial KB in the form of 5 Horn Clauses. Panel *2* through *5* show the rules-to-network translation process, and panel *6 & 7* the final network after the translation. This process is similar to KBANN except step *2* is added to ensure that the resulting network has exactly 1 hidden layer so that it can be evolved with the standard ESP method.

1994), each rule should be a Horn Clause, and the rules must be propositional and acyclic. Panel *1* of figure 1 gives an example of such rule set (or Knowledge Base, KB).

2. Replace all intermediate states (states that are neither inputs nor outputs) with their antecedents. In RESP, this simplification step is added to KBANN so that the resulting network will have only one hidden layer, matching the standard ESP method. This is the only difference between KBANN and our knowledge-transferring process. Panel *2* in figure 1 shows the sample KB after this step. Note that intermediate states *P, Q, R* have been replaced.

3. If there is more than one rule for a consequent, then every rule for this consequent with more than one antecedent is rewritten as two rules. This rewrite step is necessary for step 6 (Towell & Shavlik, 1994). Panel *3* of figure 1 shows the sample KB after the rewrite.

4. Translate the rules into a tree structure where a parent node corresponds to a consequent and its child nodes to the antecedents in that rule. Panel *4* shows the tree structure converted from the dependencies in panel *3*.

5. Assign weights w to connections that correspond to nonnegated dependencies between parent nodes and child nodes, otherwise assign $-w$, where w can be any non-zero value. In this paper $w = 4$ is used. Because KBANN performance is not sensitive to this value other values will work as well (Towell & Shavlik, 1994). Panel *5* shows the resulting network structure.

6. Calculate the bias for every output node and every hidden node to implement the rule.

   For conjunctive rules, the bias is set to $(P - \frac{1}{2})w$, where P is the number of positive antecedents on a rule, and w is the link weight value used in step 5. For disjunctive rules, the bias is always set to $\frac{w}{2}$. The biases are shown inside of the nodes in italics in panel *6 & 7*.

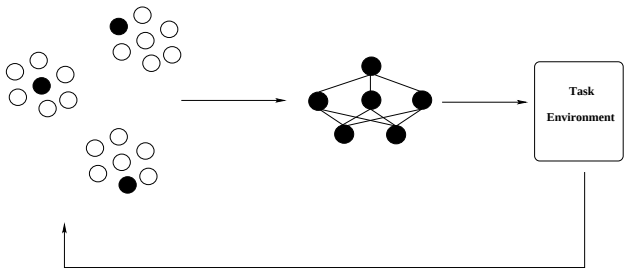

Figure 2. The ESP neuroevolution method. Each neuron in the network is evolved in a separate population, and networks are formed by randomly selecting neurons, one from each subpopulation. Each network is evaluated in the task and assigned a fitness. Each neuron's fitness is the average fitness of all networks in which it participated. ESP divides the search for a successful network into subtasks making it more robust and efficient than a search in the space of complete networks.

7. Add any missing links between any two adjacent layers of nodes in the tree, and set the initial weights of the new links to 0. Panel *6 & 7* of figure 1 shows the final ANN with the links added as thin lines. These links can be used by evolution to improve performance.

The knowledge transfer process results in a neural network that implements the functionality of the rule base. It is then further refined by evolving it in the actual task with ESP.

In standard neuroevolution where the population consists of complete neural networks, genetic operators such as crossover and mutation are applied to them in order to find a network that solves the task. In contrast, in ESP (Gomez & Miikkulainen, 1997; Gomez & Miikkulainen, 2002) each hidden neuron is evolved in a separate subpopulation, and full networks are formed by combining neurons from these subpopulations. The process consists of the following steps (figure 2):

1. Generate neuron subpopulations. Each subpopulation is made of 100 neurons, and each neuron is made of a list of floating point values denoting its input, output and bias weights. In standard ESP, the neurons in each subpopulation are created by randomly choosing weight values from a uniform distribution. In RESP, the neurons in each subpopulation are created from the initial KBANN network by randomly choosing weight values from a normal distribution, centered around the initial weights. A variance of 3 is used in experiments in this paper. This is the only difference in neuroevolution between ESP and RESP.

2. Form a full network by choosing one neuron from each subpopulation randomly.

3. Measure the fitness of the network by running it in the task, and update the average fitness of each neuron accordingly.

4. Iterate steps 2 and 3 until each neuron has been evaluated a sufficient number of times as part of a network.

5. Within each subpopulation, use one-point crossover between randomly-chosen neurons within the top 25% to generate new offsprings, and replace the worst 50% of the population with new neurons. An additional mutation step is applied with 40% probability on the new offsprings. For each mutated offspring a randomly generated value from a Cauchy distribution centered around 0 is added to a randomly selected value in the neuron.

6. Iterate steps 2 through 5 until a network with satisfactory performance has been found, or until a given number of generations has been reached.

Because ESP evolves neurons in separate subpopulations, it breaks the search for a successful network into subtasks. This strategy is highly efficient (Gomez & Miikkulainen, 1997), especially given the good initial starting point that KBANN provides.

3. Prey Capture

RESP was tested in the task of evolving a successful strategy for the predator in the Prey Capture task (Gomez & Miikkulainen, 1997; Miller & Cliff, 1994). To accomplish the task, a predator, moving through a simulated environment, must be able to apprehend a prey within a fixed number of timesteps. Scenarios of this kind are ubiquitous in natural ecosystems. They offer a relatively simple objective that requires complex sensory-motor coordination with respect to both the environment and the other agent. Variations of this task have been used to evaluate the effectiveness of various reinforcement learning methods, and it forms an appropriate domain for testing RESP as well.

3.1. Environment

The simulated environment consists of a square spatially and temporally discrete grid world (figure 3). The world has a size of 100×100, and it has a wall at the north and the west borders. The south and east

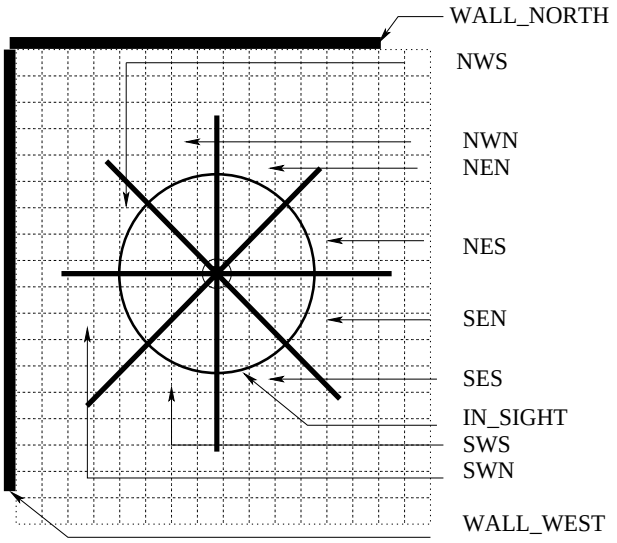

Figure 3. The Prey Capture environment and the predator's inputs. The best way for the predator to catch the prey is not to move straight towards the prey, but rather move into the southeast corner first, and then "drive" the prey into the northwest corner. The predator sees the prey in 8 sectors, far and near positions, and moves North, South, West, and East (N, S, W, E).

```
S :- S_CORNER
S :- S_BACKOFF
S_CORNER :- WALL_NORTH, SWN, NOT INSIGHT
S_CORNER :- WALL_NORTH, SWS, NOT INSIGHT
S_BACKOFF :- WALL_NORTH, NES, INSIGHT
S_BACKOFF :- WALL_NORTH, NEN, INSIGHT

W :- WALL_WEST, NWS

E :- E_CORNER
E :- E_BACKOFF
E_CORNER :- WALL_WEST, NES, NOT INSIGHT
E_CORNER :- WALL_WEST, NEN, NOT INSIGHT
E_CORNER :- WALL_WEST, SES, NOT INSIGHT
E_CORNER :- WALL_WEST, SEN, NOT INSIGHT
E_BACKOFF :- WALL_WEST, SWN, INSIGHT
E_BACKOFF :- WALL_WEST, SWS, INSIGHT


N :- N_CORNER
N :- N_BACKOFF
N_CORNER :- WALL_N, NWN
N_BACKOFF :- WALL_N, SES, INSIGHT
N BACKOFF :- WALL N, SEN, INSIGHT
```

Figure 4. The initial KB. The rules direct the predator to surround the prey and then to corner it instead of moving directly towards it.

borders are open. Both the predator and the prey occupy a single grid space, can sense the approximate direction of the other agent and can move in one of four directions (N, S, W, E) or stay still at each time step. If the predator or the prey reaches a wall, it cannot move beyond the wall; it can only move along it. The predator captures the prey when they both occupy the same grid. The prey has successfully evaded the predator when it reaches the non-walled border or when the time limit expires.

The prey moves probabilistically with a tendency to move away from the predator. Its behavior is described by the *move factor*, which is the probability that it will make a move at a given timestep, and by the *flee factor*, which is the probability that its move will be directed away from the predator. The flee factor applies when the prey is less than 30 steps away from the predator (i.e. inside the *IN_SIGHT* circle in figure 3); otherwise it moves randomly. In the hardest version of the task, these factors are both 1.0, and prey can be caught only about half of the time, depending on the initial position of the prey and the predator.

The predator's behavior is determined by a feed-forward ANN. The input layer in the network consists of 11 binary inputs that represent the approximate direction and distance of the prey and the location of the walls. These inputs are labeled *NES* (NorthEastSouth) through *SEN* (SouthEastNorth), *IN_SIGHT* (representing proximity), *WALL_NORTH* and *WALL_WEST* (representing whether a wall is nearby). The output layer consists of 5 nodes: *N, S, W, E,* and *NO_MOVE* (figure 3).

The fitness f of the predator is

$$f = \begin{cases} 10 \times (D_0 - D_a) & \text{if the prey was caught,} \\ D_0 - D_a & \text{otherwise,} \end{cases}$$

where D_0 is the initial distance between the prey and the predator, and D_a is the final distance between them. This function rewards predator that can get closer to the prey, and it gives an extra bonus for catching the prey in the end. The initial positions are randomly chosen.

3.2. Initial KB

The initial KB consists of rules described in figure 4. The inputs to the rules are *NES* through *SEN*, *IN_SIGHT*, *WALL_NORTH* and *WALL_WEST*. The outputs are *S*, *W*, *E* and *N*. The intermediate states, *S_CORNER*, *S_BACKOFF*, *E_CORNER*, *E_BACKOFF*, *N_CORNER* and *N_BACKOFF* specify whether the predator is in *corner* mode, i.e. between the prey and the open space attempting to drive the prey towards the corner, or in *backoff* mode, i.e. between the corner and the prey and in danger of letting

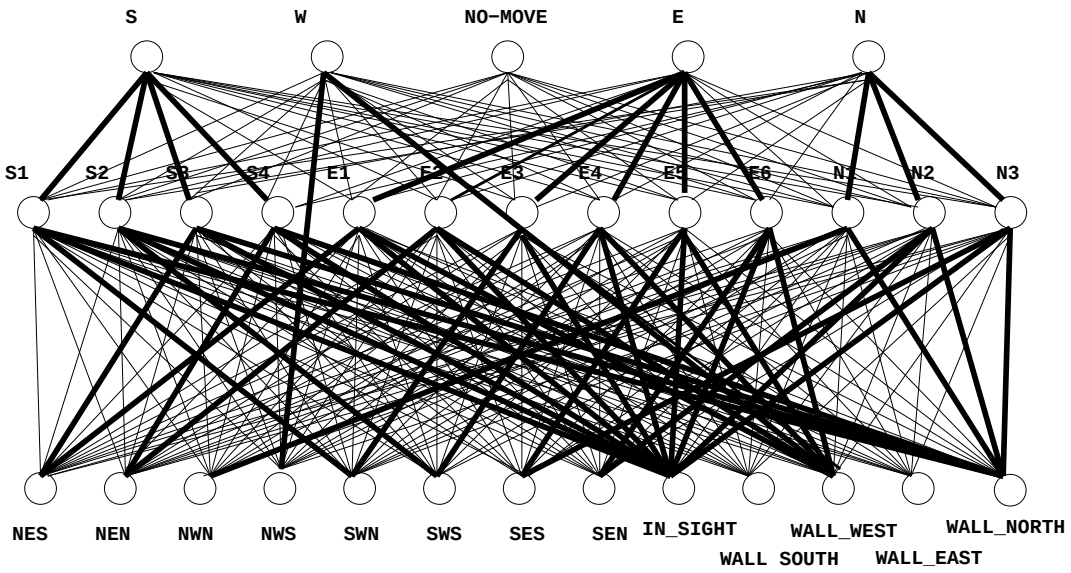

Figure 5. The ANN translated from the KB. The KB in figure 4 is transferred into an ANN using the seven-step process described in figure 1. In this process, the four ways of deducing S, the six for E, and the three for N become represented as separate hidden nodes in the network; their labels (S1,...N3) originate from the rewrite step (figure 4*c*). Subpopulations are then generated for each of the hidden node in this network, and evolved using ESP in the prey capture task.

the prey run away.

This KB is then transferred into a network as described in the previous section. Figure 5 shows the final network structure.

4. Prey Capture Performance

In this section, we present the evaluation of RESP on the Prey Capture task. Three different experiments were run. First, as a baseline we trained RESP directly on the full prey capture task. Second, to make learning easier we used an incremental learning scheme where the task was made gradually more difficult by changing the prey's behavior and the initial position of the predator. Third, to evaluate how robust RESP is with an incomplete rule base, we trained it multiple times in the full task, each time with more and more rules randomly deleted from the initial KB. In all these experiments, RESP was compared to three benchmarks:

1. ESP with four hidden units and random initial weights (ESP-BEST). Four hidden units were found experimentally to result in best ESP performance (ranging from 2 to 20 hidden units) in this task.
2. ESP with the same number of hidden units and connections as RESP but with initial weights randomly selected (ESP-SAMENET).
3. The initial RESP network, i.e. the network implementing the full initial knowledge base with no further learning (RULES-ONLY).

4.1. Direct Evolution

In direct evolution the challenge is to make good initial progress possible. If the task is too difficult, all individuals perform poorly and the GA gets trapped in an unfruitful region of the solution space. In RESP, the individuals should already perform relatively well based on the initial KB and the GA should be able to make good progress.

Figure 6 supports this hypothesis. While the fitness values of the two ESP systems stagnated below 200 (which means the networks rarely captured the prey at all) the fitness of RESP converged around 1350 (i.e. the prey was caught half the time). Thus, with direct evolution, RESP was several times better than the two ESP systems.

RESP was also able to improve the behavior of the initial KB. Recall that the initial RESP populations were created from the RULES-ONLY network by perturbing the neurons with a low level noise. As expected, the RULES-ONLY performed initially better than RESP. However, the performance of RESP improved dramatically over the course of evolution surpassing that of RULES-ONLY already after 8 generations. After 20 generations of evolution, RESP was

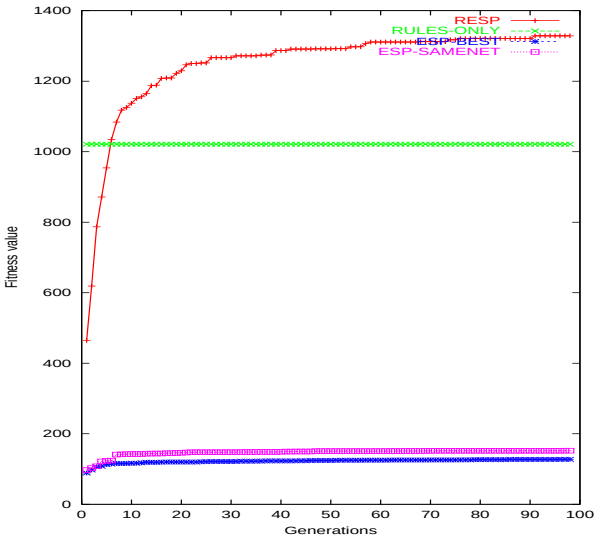

Figure 6. The learning curves of RESP, ESP, and RULES-ONLY in direct evolution. The predator starts from the northwest corner. The fitness values are averaged over ten runs. ESP-BEST and SAME-NET stagnated below 200; the networks rarely captured the prey at all. RULES-ONLY had a constant performance around 1050, which means it captured the prey around 40% of the time. After 20 generations of evolution, RESP significantly outperformed all the other approaches (based on a single-tailed t-test with 95% confidence). Its performance was still improving slightly at 100 generations when it reached a fitness of 1350, i.e. capturing the prey half of the trials. The initial KB provides an approximate solution that RESP can utilize to solve the task.

significantly better than all the other three schemes, and was still improving slightly after 100 generations. In other words, the initial KB contributed an approximate solution that RESP could then utilize as a starting point to solve the task, even though it couldn't solve it directly from a random starting point.

4.2. Incremental Learning

Next, to better understand the role of the initial KB, RESP was compared to ESP in an incremental learning setting. There were 16 increments with the difficulty determined by the prey's move and flee factors and the predator's initial location. For each increment, the location of the prey was randomly selected but the location of the predator was carefully chosen. After the fitness value exceeded a threshold, the current task was deemed solved, and evolution continued in a harder task.

In the simplest task, both the flee and move factors of the prey were set to 0.97 and the predator was ini-

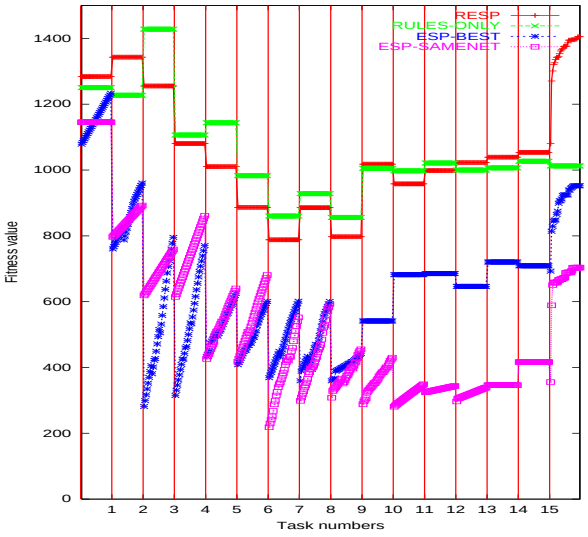

Figure 7. The performance of RULES-ONLY and the learning curves of RESP and ESP in incremental evolution. The x-axis represents generations of evolution through the different tasks, delineated by the vertical lines. Evolution continued in each task until the fitness exceeded a threshold of 250. Each learning curve was then stretched or compressed to a fixed width, and the curves of all 10 runs were averaged to obtain the curve shown in the figure. If a task was solved in very few generations, the stretching makes the learning curve appear flat. In such cases, the fitness often also exceeded the threshold by a wide margin. As a result of averaging, most of the curves appear elevated above the threshold level, even if they are not flat. In early tasks, RESP performs similarly to RULES-ONLY because it solves all these tasks in one or two generations. ESP-BEST and ESP-SAMENET in contrast learn a lot, almost catching up with the advantage given by the initial KB. However, the KB turns out to be a better foundation for the final task (where evolution was allowed to continue past the threshold), and RESP significantly outperformed all the other methods (single-tailed t-test with 95% confidence).

tially placed on the southeast corner. No matter where the prey is initially positioned, the predator can easily learn how to force the prey into the corner and capture it. In a more challenging task, the predator was placed closer to the northwest corner but still near one of the non-walled borders of the environment. The flee and move factors of the prey were also increased. In such a task the prey has a better chance of escaping to one of the unwalled borders. For the most challenging task, the predator was placed on the northwest corner and the flee and move factors of the prey were set to 1.0. In this case, the predator must learn to go around the prey and force it to flee in the direction of the walled corner, where it can be captured.

Figure 7 summarizes the results of incremental evolution. As before, the x-axis represents the progress of evolution over time. Each area between a pair of vertical lines shows the learning curves for one task. The tasks are arranged left to right in the order of increasing difficulty. Because each task takes a different number of generations to learn with the different methods, the learning curves were stretched or compressed along the x-axis to make them equally wide for all methods. If a method solved a task in the first few generations, its learning curve appears flat in the figure. The threshold for success was set at 250, which was found experimentally to improve incremental performance the most. The threshold was usually achieved between 0 and 2 generations of evolution. Many of the learning curves appear higher than 250 because they are averages of 10 runs, and some of these 10 often had an early success with a very high fitness.

RULES-ONLY performed well above the threshold in simpler tasks, but its performance converged to around 1050 for the more challenging ones. RESP solved each of the increments in very few generations, before it had much change to evolve beyond its initial approximation of RULES-ONLY. As a result, its performance tracks that of RULES-ONLY until the last task, where its evolution was not cut short after it exceeded the threshold. Its final fitness was near 1400, which is not significantly better than its fitness in direct evolution. This result shows that the initial KB of RESP makes incremental evolution unnecessary in this task.

In contrast, the ESPs evolved a lot during the early tasks. In effect they were catching up with the advantage the initial KB gave to RESP and RULES-ONLY. Incremental evolution allowed both methods to learn the task to some degree, and ESP-BEST actually achieved the same level of performance as RULES-ONLY. In the end, however, incremental evolution did not put ESP in as good a position for learning the final task as RESP. ESP-BEST achieved a final fitness of 950, and ESP-SAMENET that of 700, both of which are significantly less than RESP's 1400. These results support the hypothesis that domain knowledge gives RESP an advantage over ESP in complex problems.

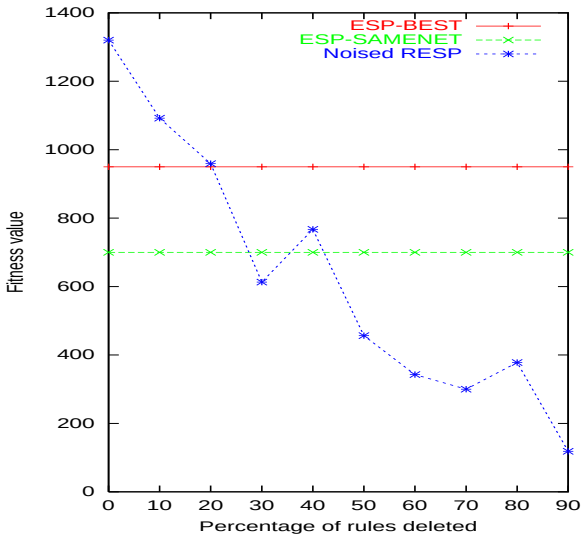

Figure 8. The fitness of RESP with rules deleted from the initial KB. The results for RESP were obtained with direct evolution, and those of the ESPs with incremental evolution. The plots represent averages of 10 runs. RESP achieved better fitness with up to 20% of the rules deleted, and comparable fitness with up to 40% of the rules deleted. The results show that RESP is robust even with an incomplete knowledge base.

Furthermore, it took us an order of magnitude longer to devise an effective incremental learning scheme than it took us to put together the initial KB used by RESP. This observation supports our claim that it is easier to come up with a set of initial behavior rules than a set of effective tasks for incremental learning.

4.3. Robustness of RESP

How accurate does the initial KB have to be? To test the robustness of RESP against incomplete knowledge we randomly deleted rules from the initial KB, and observed how good networks RESP was able to evolve from that starting point. As can be seen in Figure 8, as more rules are deleted, the fitness of RESP tends to deteriorate approximately linearly. RESP was able to achieve a higher average fitness than ESP-BEST and ESP-SAMENET even with 20% of the rules deleted. With 40% of the rules deleted, RESP is still almost as good as these two methods. These results support our claim that RESP is robust and provides a significant

advantage even in domains where it may be difficult to come up with a complete set of rules.

5. Related Work and Future Plan

The application of RESP to the Prey Capture problem showed that initial knowledge can be useful in learning difficult tasks with neuroevolution. Because RESP is based on KBANN, the limitations of KBANN also apply to RESP, such as the problem of missing rules. As a possible solution, Opitz and Shavlik (1994) proposed a specific genetic algorithm to search for the best network topology for a problem. In the near future, we will explore alternative methods in dealing with this problem, such as evolving network topology (Stanley & Miikkulainen, 2002). Another important direction of future work is application of RESP to multi-agent domains such as Robotic Soccer. Success in such domains depends crucially on appropriately chosen incremental evolution, which could be surpassed with RESP.

6. Conclusion

In this paper, we described Rule-based ESP, a hybrid approach to reinforcement learning using a KBANN-like rules-to-ANN translation technique and the ESP neuroevolution method. RESP was evaluated in the Prey Capture domain. The results show that RESP can perform complex behaviors better than ESP, is less likely to need incremental learning, and is robust even with a significant amount of rules omitted from the initial knowledge base. Furthermore, because it is often easier to come up with an initial rule base than an effective incremental learning scheme, RESP will allow applying neuroevolution to more complex domains.

Acknowledgments

Special thanks to Chern Hang Yong for the source code of the ESP adapted for the Prey Capture domain and Faustino Gomez for suggestions on tuning ESP. This research was supported in part by Texas Higher Education Coordinating Board under grant ARP-003658-476-2001 and in part by NSF under grant IIS-0083776.

References

Cliff, D., Husbands, P., & Harvey, I. (1993). Evolving recurrent dynamical networks for robot control. *Proceedings of ANNGA93, International Conference on Artificial Neural Networks and Genetic Algorithms* (pp. 428 – 435). Innsbruck: Springer-Verlag.

Fogel, D. B. (2001). *Blondie24: Playing at the edge of AI.* San Francisco, CA: Morgan Kaufmann.

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

Gomez, F., & Miikkulainen, R. (2002). *Learning robust nonlinear control with neuroevolution* (Technical Report AI01-292). Department of Computer Sciences, The University of Texas at Austin.

Hudson, D., Banda, P., Cohen, M., & Blois, M. (1992). Medical diagnosis and treatment plans derived from a hybrid expert system. *Hybrid Architectures for Intelligent Systems* (pp. 330–244). CRC Press.

McGarry, K., Wermter, S., & MacIntyre, J. (1999). Hybrid neural systems: from simple coupling to fully integrated neural networks. *Neural Computing Surveys, 2*, 62–93.

Miller, G., & Cliff, D. (1994). *Co-evolution of pursuit and evasion I: Biological and game-theoretic foundations* (Technical Report CSRP311). School of Cognitive and Computing Sciences, University of Sussex, Brighton, UK.

Nolfi, S., Elman, J. L., & Parisi, D. (1994). Learning and evolution in neural networks. *Adaptive Behavior, 2*, 5–28.

Opitz, D. W., & Shavlik, J. W. (1994). Using genetic search to refine knowledge-based neural networks. *Machine Learning: Proceedings of the Eleventh International Conference* (pp. 208 – 216). New Brunswick, NJ: Morgan Kaufmann.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation, 10*, 990–127. In press.

Towell, G. G., & Shavlik, J. W. (1994). Knowledge-based artificial neural networks. *Artificial Intelligence, 70*, 119–165.

Wermter, S. (1997). *Hybrid approaches to neural network-based language processing* (Technical Report TR-97-030). UC-Berkeley, Berkeley, CA.

Whitley, D., Dominic, S., Das, R., & Anderson, C. W. (1993). Genetic reinforcement learning for neurocontrol problems. *Machine Learning, 13*, 259–284.

Wieland, A. P. (1990). Evolving controls for unstable systems. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski and G. E. Hinton (Eds.), *Connectionist models: Proceedings of the 1990 summer school*, 91–102. San Francisco, CA: Morgan Kaufmann.