Evolving Neural Networks for Fractured Domains

Nate Kohl
Department of Computer Sciences
University of Texas at Austin
1 University Station C0500
Austin, TX 78712
nate@cs.utexas.edu

Risto Miikkulainen
Department of Computer Sciences
University of Texas at Austin
1 University Station C0500
Austin, TX 78712
risto@cs.utexas.edu

ABSTRACT

Evolution of neural networks, or neuroevolution, bas been successful on many low-level control problems such as pole balancing, vehicle control, and collision warning. However, high-level strategy problems that require the integration of multiple sub-behaviors have remained difficult for neuroevolution to solve. This paper proposes the hypothesis that such problems are difficult because they are fractured: the correct action varies discontinuously as the agent moves from state to state. This hypothesis is evaluated on several examples of fractured high-level reinforcement learning domains. Standard neuroevolution methods such as NEAT indeed have difficulty solving them. However, a modification of NEAT that uses radial basis function (RBF) nodes to make precise local mutations to network output is able to do much better. These results provide a better understanding of the different types of reinforcement learning problems and the limitations of current neuroevolution methods. Thus, they lay the groundwork for creating the next generation of neuroevolution algorithms that can learn strategic high-level behavior in fractured domains.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*connectionism and neural nets*

General Terms

Algorithms

Keywords

NEAT, neuroevolution, RBF networks

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

1. INTRODUCTION

The process of evolving neural networks using genetic algorithms (neuroevolution) is a promising new approach to solving reinforcement learning problems. While the traditional method of solving such problems involves the use of temporal difference methods to estimate a value function, neuroevolution instead relies on policy search to build a neural network that directly maps states to actions. This approach has proved to be useful on a wide variety of problems and is especially promising in challenging POMDP domains [8, 30].

The Neuroevolution of Augmenting Topologies (NEAT) method is one such neuroevolution algorithm that has been used successfully on many of reinforcement learning problems [15, 26, 29, 30, 31, 32]. However, despite its efficacy on a wide variety of low-level control domains (e.g. pole balancing, vehicle control, collision warning, character control in video games), other types of problems such as SAT, multiplexer, or high-level behavior selection have remained difficult for NEAT to solve. A better understanding of why NEAT works so well on some problems – but not others – can be very useful in designing the next generation of neuroevolution algorithms.

This paper presents the fractured domain hypothesis as a possible explanation for NEAT's poor performance on domains like those listed above. *Fractured domains* are defined as domains that have a highly discontinuous mapping between states and optimal actions. For example, as an agent moves from state to state in a fractured domain, the best action that the agent can take changes frequently and abruptly. In contrast, the optimal actions for a non-fractured domain change slowly and continuously. Many challenging supervised learning tasks are fractured, such as SAT, multiplexer, and the concentric spirals problem. Importantly for reinforcement learning, high-level decision tasks where an agent must choose between several sub-behaviors are often fractured as well. The fractured domain hypothesis posits that NEAT performs poorly on fractured domains because the neural networks that NEAT evolves have difficulty representing such abrupt decision boundaries.

This paper evaluates this hypothesis experimentally. NEAT is tested on several domains that possess a fractured decision space to varying degrees. NEAT proves to perform relatively poorly on these domains, and its performance is found to fall even lower as the amount of fracture is increased. The concept of an algorithm that can make local modifications to network output is introduced as a potential solution to neuroevolution in fractured domains. Drawing on radial basis

function literature from the supervised learning community, a variation of NEAT that uses RBF nodes designed to fit the local features of fractured domains is described, and found to perform much better on these fractured domains.

These results suggest that it is possible to evolve neural networks effectively for fractured domains. In essence, RBF-NEAT constructs networks made of local features that track fractured decision boundaries well. In the future, it may be possible to extend such a local modification approach to internal nodes of the network, thus making use of locality (i.e. modularity) in internal function as well as in the input space. Eventually, the approach may thus prove highly effective for evolving networks that learn strategic high-level behavior.

2. NEUROEVOLUTION OF AUGMENTING TOPOLOGIES (NEAT)

The Neuroevolution of Augmenting Topologies (NEAT) method [30] is designed to solve difficult reinforcement learning problems by automatically evolving network topology to fit the complexity of the problem. NEAT combines the usual search for the appropriate network weights with *complexification* of the network structure. It starts with simple networks and expands the search space only when beneficial, allowing it to find significantly more complex controllers than fixed-topology evolution. These properties make NEAT an attractive method for evolving neural networks in complex tasks. In this section, the NEAT method is briefly reviewed; see [30, 31] for more detailed descriptions.

NEAT is based on three key ideas. First, evolving network structure requires a flexible genetic encoding. Each genome in NEAT includes a list of connection genes, each of which refers to two node genes being connected. Each connection gene specifies the in-node, the out-node, the weight of the connection, whether or not the connection gene is expressed (an enable bit), and an innovation number, which allows finding corresponding genes during crossover. Mutation can change both connection weights and network structures. Connection weights are mutated in a manner similar to any NE system. Structural mutations, which allow complexity to increase, either add a new connection or a new node to the network. Through mutation, genomes of varying sizes are created, sometimes with completely different connections specified at the same positions.

Each unique gene in the population is assigned a unique innovation number, and the numbers are inherited during crossover. Innovation numbers allow NEAT to do crossover without the need for expensive topological analysis. Genomes of different organizations and sizes stay compatible throughout evolution, and the problem of matching different topologies [25] is essentially avoided.

Second, NEAT speciates the population so that individuals compete primarily within their own niches instead of with the population at large. This way, topological innovations are protected and have time to optimize their structure before they have to compete with other niches in the population. The reproduction mechanism for NEAT is explicit fitness sharing [7], where organisms in the same species must share the fitness of their niche, preventing any one species from taking over the population.

Third, unlike other systems that evolve network topologies and weights [10, 40], NEAT begins with a uniform population of simple networks with no hidden nodes. New structure is introduced incrementally as structural mutations occur, and the only structures that survive are those that are found to be useful through fitness evaluations. In this manner, NEAT searches through a minimal number of weight dimensions and finds the appropriate level of complexity for the problem.

This approach is highly effective: NEAT has outperformed other neuroevolution methods on complex control tasks like double pole balancing [30] and robotic strategy-learning [31]. However, it has turned out to be surprisingly difficult to get NEAT to perform well in fractured domains, as will be discussed next.

3. FRACTURED DOMAINS

The three principles described above allow NEAT to search quickly and efficiently through the space of possible network topologies to find the right neural network for the task at hand. However, the regular NEAT algorithm is limited to small, incremental changes in neural network structure. While such mutations are useful when building relatively small networks, tasks that require complicated or repeated internal structure are difficult for NEAT. Furthermore, any small mutations that NEAT makes to network structure have the potential to have a global impact on network output. If a task requires local adjustments to network output, NEAT's performance may suffer.

The domains to which NEAT has been applied so far can be solved by relatively small neural networks with continuous output. Therefore, these potential drawbacks may not have been an issue before. In fact, since NEAT starts with a minimal topology and uses small mutations to tweak network architecture, it may be unusually well suited to such domains. On the other hand, other domains may not be as amenable to this approach. For instance, it has been difficult for NEAT to solve problems like SAT, multiplexer, concentric spirals, and high-level decision tasks in general.

What makes these domains different from those on which NEAT and other neuroevolution algorithms have done so well? A possible explanation is that these domains share a common property: They possess a "fractured" decision space, loosely defined as a *space where adjacent states require radically different actions*. For a normal domain (such as the typical control domains, or the standard reinforcement learning benchmarks), the correct action for one state is similar to the correct action for neighboring states, perhaps varying smoothly and infrequently. In contrast, for a fractured domain, the optimal action that the agent should take changes repeatedly and discontinuously as the agent moves from state to state.

It seems reasonable that neuroevolution algorithms would have difficulties on such problems. In order for an agent to solve a fractured problem, it must be able to isolate individual areas of the decision space and associate different actions with each area. In a normal neural network (which has firing rate nodes with sigmoid activation functions) such a partitioning requires several components cooperating in multiple layers (Figure 1). Assembling enough components to isolate even a single fracture in the decision space is a difficult task, especially when the fitness function may only reward the network once all of the components are fully connected and working together. Furthermore, mutations to the structure of such a neural network could conceivably have a global effect on network output, disrupting any local effects that might already have evolved.

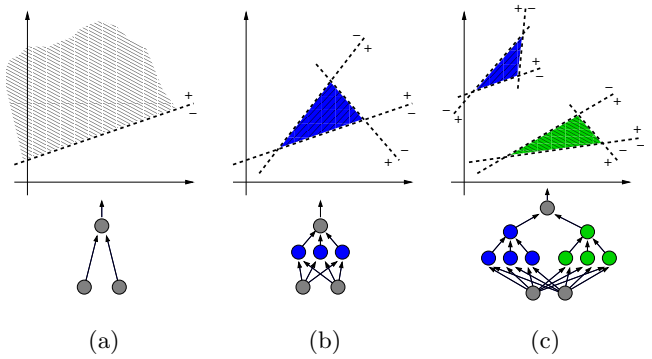

Figure 1: Examples of the amount of structure a neural network needs in order to isolate sections of a 2-d decision space. (a) One layer of weights splits the space in half. (b) The addition of three hidden nodes allows a triangular region to be isolated. (c) Isolating multiple triangular regions requires four nodes and 10 links per region. As the decision boundaries become increasingly fractured, it becomes difficult for evolution to discover such structures.

The fractured domain hypothesis could explain why NEAT and other neuroevolution methods perform poorly on certain problems. Domains like SAT, concentric spirals and multiplexer both repeatedly require different network output after small changes in input, giving those domains a fractured quality. Many high-level decision tasks are fractured because they deal with high-level state and actions. For example, the state and actions for a low-level agent controlling a racecar through a curve change slightly and continuously as time passes, reflecting the relatively low-level nature of a racecar control task. Another agent in the same domain could be applied at a higher level to make strategic decisions about how aggressively to drive and when to go into pit stops. In contrast to the low-level driving agent, this high-level strategic agent needs to take a much broader view of the state and action space, aggregating many similar states into single states. This aggregation leads to a fractured decision space.

The fractured domain hypothesis offers a possible explanation why NEAT and other neuroevolution algorithms perform poorly on a number of supervised tasks as well as in high-level decision tasks. Understanding this problem is the first step; the next step is figuring out how to fix it.

4. APPROACH

In order to perform well in a fractured domain, a learning algorithm must be able to make local changes to its behavior. After the algorithm experiences a new state in the environment, it needs to associate a specific action for that state. Assuming the domain is fractured, it may not be useful to generalize from the actions of nearby states. Furthermore, any large-scale changes the algorithm may make could disrupt the individual actions tailored for other states. Therefore, the algorithm must be able to make local changes to isolate that particular state from its neighbors and associate the correct action with it.

One promising method for learning via local features is radial basis function (RBF) networks [14, 23, 24]. RBF networks originated in the supervised learning community, and are usually described as neural networks with a single hidden layer of basis-function nodes. Each of these nodes computes a function (usually a Gaussian) of the inputs, and the output of the network is a linear combination of all of the basis nodes. Training of RBF networks usually occurs in two stages: The locations and sizes of the basis functions are determined, and then the parameters that combine the basis functions are computed.

Since RBF networks are traditionally associated with supervised learning problems, most RBF-based algorithms are built to take advantage of labeled training data [14, 23, 24]. Several interesting hybrid algorithms have been proposed that use various flavors of genetic algorithms, usually to determine the number, size, and location of the basis functions [1, 3, 6, 9, 11, 12, 13, 22, 27, 37]. Most of these algorithms have not been proposed for use on reinforcement learning problems, where there is no labeled training data. However, the local processing in RBF networks has proved to be particularly useful on fractured problems like the concentric spirals classification task [6], suggesting that an RBF approach could be useful for fractured reinforcement learning domains as well.

Related results in value-function reinforcement learning support this idea. Even though value functions for fractured domains may look relatively smooth, values for adjacent states must differ slightly in order to generate fractured policies. When modeling these differences, value-function reinforcement learning methods frequently benefit from approximating value functions using highly local function approximators like tables, CMACs, or RBF networks [16, 19, 20, 33, 34]. For example, Stone et al. found that in the benchmark keepaway soccer domain, an RBF-based value function approximator significantly outperformed a normal neural network value function approximator [33]. Such results suggest that local behavioral adjustments could be useful for neuroevolution learning as well.

Learning classifier systems (LCS) are another interesting family of algorithms that use local processing to solve reinforcement learning problems. LCS algorithms approximate functions with a population of classifiers, each of which is responsible for a small part of the input space. A competitive contributory mechanism encourages classifiers to cover as much space as possible, removing redundant classifiers and increasing generalization. A number of LCS algorithms have been developed that vary both how the classifiers cover the input space and how they approximate local functions [4, 5, 17, 18, 38, 39]. The LCS literature provides an intriguing approach to local processing, but leaves unaddressed the question of how local processing might be best integrated into constructive neural network algorithms.

In the experiments reported in this paper, RBF nodes were incorporated into NEAT. The resulting RBF-NEAT algorithm is a relatively simple adaptation of NEAT. Instead of building networks using the normal sigmoid-based nodes, RBF-NEAT incrementally builds networks using radial-basis function nodes. Like NEAT, the algorithm starts with a minimal topology, in this case consisting of a single RBF node connected to both inputs and outputs. With probability $\epsilon = 0.05$ an "add RBF node" mutation occurs. Each RBF node is activated by an axis-parallel Gaussian with variable center and width. Whenever a new node is added, the weights of existing nodes in the network are frozen to focus the search process on the most recently added node. All

free parameters of the network, including RBF node parameters and link weights, are determined by a simple genetic algorithm similar to the one in NEAT [30].

RBF-NEAT is decidedly simple and does not include all of the sophisticated components that make up the regular NEAT algorithm, such as speciation, fitness sharing, and historical markings. It is meant to evaluate whether local processing nodes are useful in reinforcement learning domains. Moreover, because RBF-NEAT is simple, it is easy to understand it and draw conclusions from it. It thus serves as a starting point for studying how neuroevolution can be extended to domains with fractured decision spaces.

5. EXPERIMENTAL RESULTS

In order to understand the differences between regular NEAT and RBF-NEAT, both algorithms were evaluated on three different domains: **N-Points**, **3-Player Soccer**, and **Racing Strategy**. These domains were chosen for several reasons. First, they all represent domains on which NEAT has performed poorly. Second, these domains are fractured, demonstrating that there are several interesting problems that possess a fractured decision space. Third, and perhaps most importantly, the simplicity of these domains makes them easy to analyze. The purpose of these experiments is to demonstrate that the local processing approach works and that by using it, it may be possible to extend neuroevolution to fractured, high-level domains.

5.1 The N-Points Domain

The N-Points domain is one of the most straightforward examples of a fractured domain, representing a class of supervised problems (like SAT, multiplexer, and concentric spirals) that is difficult for NEAT. At its heart, it is a binary classification task cast as a reinforcement learning problem. The goal of a successful network is to positively classify N points in one-dimensional space, while negatively classifying all other points. The parameter N can be varied to change the difficulty of the task. For example, if $N = 3$ then three positive points are chosen at random within the interval $[0, 1]$ and six negative points (two for each of the N positive points) are chosen near the positive points. These nine points are fed into a candidate network one at a time, and the network should only fire its output when it recognizes one of the three positive points. The output should be zero for the six negative points.

The N-Points domain clearly possesses the fractured quality described above. As the network experiences different states, the correct output jumps between zero and one in a discontinuous manner. Another benefit of the N-Points domain is that it is possible to increase the amount of fracture by making N larger. This flexibility makes it possible to measure the performance of NEAT and RBF-NEAT as the algorithms attempt to solve increasingly fractured problems.

Figure 2 shows the performance of NEAT and RBF-NEAT on the N-Points domain, averaged over 100 runs. In order to observe the effect on performance as the amount of fracture increases, four versions of the N-Points domain were evaluated: $N = 1, 3, 5$ and 10. For the relatively lightly fractured $N = 1$ version, there is no significant difference between NEAT and RBF-NEAT. However, as the amount of fracture increases, NEAT's performance falls quickly. RBF-NEAT proves to be much more proficient at solving the highly fractured versions of this domain.

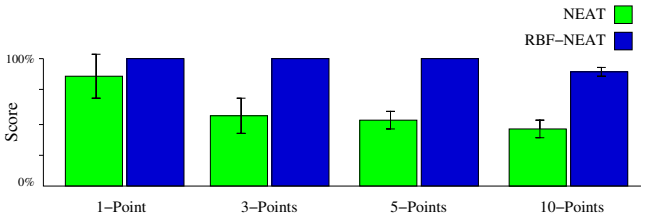

Figure 2: Performance of regular NEAT and RBF-NEAT on the N-Points domain for $N = 1, 3, 5$ and 10. The problem becomes more difficult as the amount of fracture increases, which causes NEAT's performance to suffer. RBF-NEAT proves to be significantly more successful ($p > 0.95$) at solving highly fractured domains.

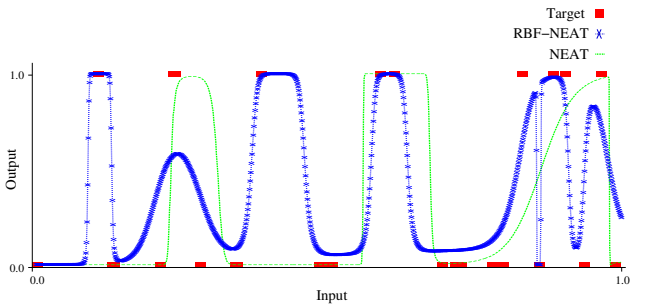

Figure 3: Example output of the best networks found by regular NEAT and RBF-NEAT for the 10-Points domain. The network found by RBF-NEAT properly classifies all of the points, dramatically outperforming the best network found by regular NEAT. This result shows how RBF-NEAT is able to approximate a fractured decision boundary.

Figure 3 shows the actual output of the best network found by each algorithm for the difficult $N = 10$ version of the problem. The network found by NEAT has managed to properly classify a few of the positive points, but its performance pales in comparison to that of RBF-NEAT, which has discovered a network that properly classifies every single point (to be considered correct, the output of the network must be ≤ 0.5 for negative points and > 0.5 for positive points).

RBF-NEAT thus outperforms the regular NEAT algorithm on this straightforward fractured domain. This result confirms the hypothesis that NEAT performs poorly on fractured domains and shows that RBF-NEAT is an initial step towards a more robust learning algorithm.

5.2 The 3-Player Soccer Domain

The 3-player soccer domain represents a class of high-level strategy problems that have previously proved difficult for NEAT to learn. Instead of directly controlling the low-level actions of an agent, a successful network must analyze the state of a soccer game and choose between several predefined "macro" behaviors. The correct behavior changes repeatedly as the network encounters different states, giving this domain a fractured quality. Perhaps the most important feature of the 3-player soccer domain is that it represents an interesting class of problems: it is a decision task where a

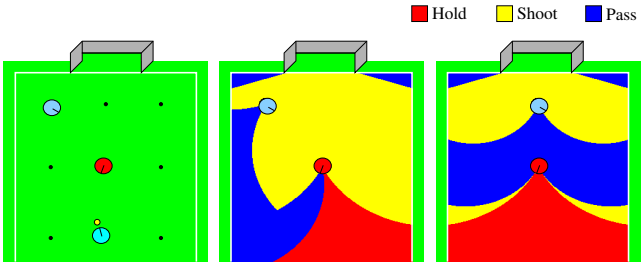

Figure 4: (a) A typical example of the 3-player soccer domain. The player (shown near the ball) must choose one of three high-level behaviors: hold, pass, or shoot on goal. The black dots indicate possible positions for the players. (b)-(c) Optimal actions for different player locations given two possible configurations of teammate and opponent. As the player moves, the optimal action changes frequently and abruptly. Thus this domain is an example of an interesting high-level learning problem that possesses a fractured decision space.

number of subroutines need to be integrated to achieve a cohesive high-level behavior.

Figure 4a shows a typical setup for the 3-player soccer domain. The input to a network consists of the player's position, the position of a single opponent, and the position of a teammate. The player starts with the ball, and must choose between holding the ball, passing to the teammate, or attempting a shot on the goal. In order to keep the task simple, the input was discretized into 27 unique states (three possible locations for each of the three players), and a single correct action was assigned to each state. A less-fractured version of the problem that contained nine states (three possible locations for both the opponent and teammate) was also evaluated.

Figure 5 shows the performance of NEAT and RBF-NEAT on the two versions of the 3-player soccer domain, averaged over 100 runs. Again, as the level of fracture increases, the performance of NEAT falls. In contrast, RBF-NEAT demonstrates that it can match the correct action with each state, easily outperforming regular NEAT on both versions of the problem.

The results from the 3-player soccer domain are important for two reasons: First, this domain provides an example of an interesting high-level learning problem that possesses a fractured decision space. Figures 4b and 4c illustrate the nature of this fracture by showing how the correct action changes depending on the player's position. The colors on the field – which show the optimal action for the player if it were at that location – can vary quite abruptly in certain places. For example, in Figure 4c, the best action when blocked by the opponent is to hold the ball. As the player moves around the opponent, it becomes possible to pass the ball to the teammate, who has a clear shot on the goal. However, there is a thin area where the opponent would be able to intercept a pass to the teammate, which makes the optimal action for that area to take a shot on the goal. Such rapid transitions between optimal actions are the hallmark of a fractured domain.

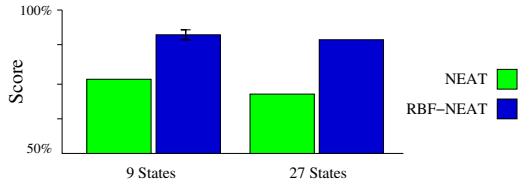

Figure 5: Performance of NEAT and RBF-NEAT on the 3-player soccer domain. RBF-NEAT is able to match each state with the correct action, easily outperforming regular NEAT on both versions of the problem ($p > 0.95$). This result demonstrates that the local processing approach works in high-level decision tasks that are fractured.

The second reason that these results are significant is that while NEAT does not do well on this domain, the local processing of RBF-NEAT allows it to do quite well. Note, however, that the fractqures are partly due to a discretization in this domain. The next section provides an example of a more complicated continuous domain that also has a fractured decision space.

5.3 The Racing Strategy Domain

The racing strategy domain has a large, continuous state space similar to that of many real-world reinforcement learning tasks. Inspired by the IEEE CIG racing competition [21, 35], the goal is to control an agent that races a car against an opponent through a series of waypoints. Each waypoint may only be awarded to one of the two players, and after a fixed amount of time the player that has "collected" the most waypoints wins. In the past, NEAT has been successful at evolving agile drivers [28], but success in this driving task requires not only efficient low-level control but also a robust high-level strategy [35]. For example, one crucial decision that the agent must make is whether it should give up on the current waypoint – allowing the opponent to get it – and instead choose to position itself for a good run at the next waypoint.

The goal of the racing strategy domain implemented for this paper is to evolve a network that makes this decision on which waypoint the agent should focus. Given the state of the domain, the network predicts whether the agent or the opponent will be able to reach the current waypoint first. The input is the relative position of the waypoint from the perspective of each player. Velocity for each player is not modeled. The single output is interpreted as a binary signal describing which player should be able get to the waypoint first.

Generating the correct strategy for this domain is difficult because the position and orientation of both players and the location of the waypoint must all be considered when deciding if the agent will be able to beat the opponent to that waypoint. Furthermore, slight changes in the state of the domain can quickly change which player has the advantage. For example, consider the three scenarios shown in Figure 6. In scenario (a), the agent (marked with an $\times$) has the advantage over the opponent because the waypoint is directly in front of the agent. In scenario (b), the waypoint is the same distance from the agent, but the opponent actually has the advantage because it is difficult for players to reach

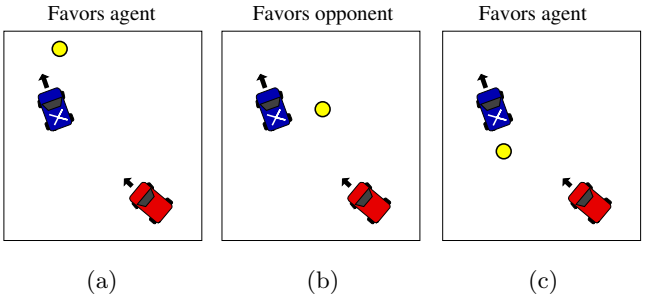

Figure 6: Three example scenarios from the racing strategy domain. In contrast with 3-player soccer, this domain is continuous but still fractured: relatively small differences in state cause the advantage to shift from one player to the other, giving the domain a fractured quality.

waypoints located to their sides. In scenario (c), the agent only has a small distance to travel in reverse, which gives it the advantage again. The discontinuities introduced by these small changes in state give the racing strategy domain a fractured quality.

Figure 7 shows the performance of NEAT and RBF-NEAT on the racing strategy domain, averaged over 100 runs. Again, RBF-NEAT proves to be better at predicting the outcome of the various training scenarios, outperforming the regular NEAT algorithm significantly. Thus the benefits of an algorithm with local processing are shown to extend to continuous high-level reinforcement learning domains that are fractured.

6. DISCUSSION AND FUTURE WORK

The results described in this paper define a class of problems called fractured domains and show that many interesting problems in the real world are fractured. The regular NEAT algorithm is found to perform poorly on these domains, whereas an alternative algorithm that is based on local processing does much better. These results serve as a promising starting point for both neuroevolution and neural network research in general and should eventually lead to evolving high-level behaviors in neural networks.

Fractured domains are defined above as problems where neighboring states require different actions. This definition is intuitive, but it could be useful to come up with a metric that quantitatively measures the amount of fracture of a given domain. Possibilities include metrics related to Kolmogorov complexity, Minimum Description Length, or the VC dimension of networks that work well on the domain [36, 2]. There is a great deal of related work concerning each of these concepts, and it may be possible to adapt that work to the proplems described in this proposal. A more rigorously defined metric would allow researchers to evaluate more accurately how well various learning algorithms respond to varying degrees of fracture.

It is also possible that the difficulties that NEAT has in dealing with fractured domains are symptoms of a more general problem. For example, it could be the case that while NEAT can evolve small networks efficiently, it might be unable to evolve large networks. Fractured domains, which typically require a large amount of network structure, may

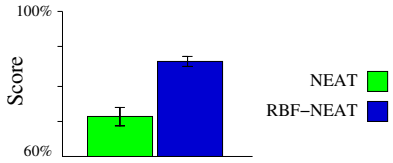

Figure 7: Performance of NEAT and RBF-NEAT on the racing strategy domain. While previous experiments have shown that NEAT is adept at evolving agile drivers, RBF-NEAT proves to be significantly better than NEAT ($p > 0.95$) at evolving high-level strategy.

thus be only one example of domains that are difficult to solve with NEAT. Exploring other such domains could result in further insights into how neuroevolution algorithms can be improved.

A related avenue for future work involves better understanding why the regular NEAT algorithm has problems with fractured domains. While the discussion in Section 3 offers a plausible explanation – that isolating sections of a fractured space is difficult to accomplish with ordinary neural networks – the reasons are not yet clear. The necessary structure may be hard to represent, to construct, or to evaluate. Understanding such results will in turn lead to better algorithms in the future.

RBF-NEAT is a first implementation of the concept of local processing nodes in a NEAT-like algorithm. Having shown that even a simple RBF-based algorithm can do better than NEAT on fractured domains, the question remains of how to best incorporate local processing into NEAT.

The first step is to examine how useful the advanced features of NEAT, like speciation and fitness sharing, are with RBF nodes. Standard RBF algorithms are known for their tendency to generate excessive numbers of RBF nodes. If networks evolved with RBF nodes are likely to be larger than regular NEAT networks, which might change the dynamics of the learning process.

It may also be possible to improve RBF-NEAT by implementing ideas from the supervised learning and reinforcement learning literature. As mentioned in Section 4, the supervised learning community has generated a large amount of research on RBF-related learning algorithms. While many of these algorithms rely on labeled training data, some of them may apply to reinforcement learning problems. For instance, an RBF network may be made to generalize better to new inputs by covering the same input area with fewer nodes. Another interesting approach would be to integrate the local processing and generalization mechanisms from the LCS literature with constructive neural network algorithms.

Finally, RBF-NEAT could be extended by allowing evolution to connect basis nodes to other hidden nodes, instead of only to inputs. The basis nodes could then gate and repeat any internal functions computed by the network. Such an organized repetition of internal function could allow a network to be built around several sub-behaviors, laying the groundwork for an algorithm that evolves modular neural networks. This approach could be very powerful in evolving high-level behavior, and constitutes a most interesting direction of future work.

7. CONCLUSION

Despite its success in the past, neuroevolution in general and NEAT in particular has surprising difficulty solving certain types of high-level reinforcement learning problems. This paper presents the hypothesis that this difficulty arises because these domains are fractured: The correct action varies discontinuously as the agent moves from state to state. Several examples of high-level reinforcement learning domains that possess such a fractured quality are presented, and NEAT is shown to perform rather poorly on these fractured domains. However, a modification of NEAT that uses radial basis function nodes to provide local modifications is able to do much better. These results provide a better understanding of the different types of reinforcement learning problems and the limitations of current neuroevolution methods. Thus, they lay the groundwork for creating the next generation of neuroevolution algorithms that can learn modular strategic high-level behavior.

8. REFERENCES

[1] P. J. Angeline. Evolving basis functions with dynamic receptive fields. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 5, pages 4109–4114, 1997.

[2] A. Barron, J. Rissanen, and B. Yu. The minimum description length principle in coding and modeling. *IEEE Trans. Information Theory*, 44(6):2743–2760, 1998.

[3] S. A. Billings and G. L. Zheng. Radial basis function network configuration using genetic algorithms. *Neural Networks*, 8:877–890, 1995.

[4] L. Bull and T. O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 905–911, 2002.

[5] M. V. Butz. Kernel-based, ellipsoidal conditions in the real-valued xcs classifier system. In *Proceedings of the 2005 conference on Genetic and Evolutionary Computation*, pages 1835–1842, 2005.

[6] N. Chaiyaratana and A. M. S. Zalzala. Evolving hybrid rbf-mlp networks using combinedgenetic/unsupervised/supervised learning. In *UKACC International Conference on Control*, volume 1, pages 330–335, 1998.

[7] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154, 1987.

[8] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Efficient non-linear control through neuroevolution. In *Proceedings of the European Conference on Machine Learning (ECML-06, Berlin)*, 2006.

[9] J. Gonzalez, I. Rojas, J. Ortega, H. Pomares, F. Fernandez, and A. Diaz. Multiobjective evolutionary optimization of the size, shape, and position parameters of radial basis function networks for function approximation. *IEEE Transactions on Neural Networks*, 14:1478–1495, 2003.

[10] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89. MIT Press, 1996.

[11] A. Guillen, H. Pomares, J. Gonzalez, I. Rojas, L. J. Herrera, and A. Prieto. *Parallel Multi-objective Memetic RBFNNs Design and Feature Selection for Function Approximation Problems*, volume 4507/2007. 2007.

[12] A. Guillen, I. Rojas, J. Gonzalez, H. Pomares, L. J. Herrera, and B. Paechter. *Improving the Performance of Multi-objective Genetic Algorithm for Function Approximation Through Parallel Islands Specialisation*, volume 4304/2006. 2006.

[13] L. Guo, D.-S. Huang, and W. Zhao. Combining genetic optimisation with hybrid learning algorithm for radial basis function neural networks. *Electronics Letters*, 39:1600–1601, 2003.

[14] H. Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19:201–227, 2001.

[15] N. Kohl, K. Stanley, R. Miikkulainen, M. Samples, and R. Sherony. Evolving a real-world vehicle warning system. In *Proceedings of the Genetic and Evolutionary Computation Conference 2006*, pages 1681–1688, July 2006.

[16] R. Kretchmar and C. Anderson. Comparison of cmacs and radial basis functions for local function approximators in reinforcement learning. In *Proceedings of the International Conference on Neural Networks*, 1997.

[17] P. Lanzi, D. Loiacono, S. Wilson, and D. Goldberg. Classifier prediction based on tile coding. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1497–1504, 2006.

[18] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg. Xcs with computed prediction for the learning of boolean functions. In *Proceedings of the IEEE Congress on Evolutionary Computation Conference*, 2005.

[19] J. Li and T. Duckett. Q-learning with a growing rbf network for behavior learning in mobile robotics. In *Proceedings of the Sixth IASTED International Conference on Robotics and Applications*, 2005.

[20] J. Li, T. Martinez-Maron, A. Lilienthal, and T. Duckett. Q-ran: A constructive reinforcement learning approach for robot behavior learning. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robot and System*, 2006.

[21] S. Lucas and J. Togelius. Point-to-point car racing: an initial study of evolution versus temporal difference learning. In *IEEE Symposium on Computational Intelligence and Games*, pages 260–267, 2007.

[22] E. Maillard and D. Gueriot. Rbf neural network, basis functions and genetic algorithm. In *International Conference on Neural Networks*, volume 4, 1997.

[23] J. Moody and C. J. Darken. Fast learning in networks of locally tuned processing units. *Neural Computation*, 1:281–294, 1989.

[24] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3:246–257, 1991.

[25] N. J. Radcliffe. Genetic set recombination and its application to neural network topology optimization.

Neural Computing and Applications, 1(1):67–90, 1993.

[26] J. Reisinger, E. Bahceci, I. Karpov, and R. Miikkulainen. Coevolving strategies for general game playing. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2007.

[27] H. Sarimveis, A. Alexandridis, S. Mazarakis, and G. Bafas. A new algorithm for developing dynamic radial basis function neural network models based on genetic algorithms. *Computers and Chemical Engineering*, 28:209–217, 2004.

[28] K. Stanley, N. Kohl, R. Sherony, and R. Miikkulainen. Neuroevolution of an automobile crash warning system. In *Proceedings of the Genetic and Evolutionary Computation Conference 2005*, pages 1977–1984, 2005.

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

[30] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2), 2002.

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

[32] K. O. Stanley and R. Miikkulainen. Evolving a roving eye for go. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2004.

[33] P. Stone, G. Kuhlmann, M. E. Taylor, and Y. Liu. Keepaway soccer: From machine learning testbed to benchmark. In I. Noda et al., editors, *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020, pages 93–105. Springer Verlag, Berlin, 2006.

[34] M. Taylor, S. Whiteson, and P. Stone. Comparing evolutionary and temporal difference methods for reinforcement learning. In *Genetic and Evolutionary Computation Conference*, pages 1321–28, July 2006.

[35] J. Togelius, S. Lucas, H. D. Thang, J. Garibaldi, T. Nakashima, C. H. Tan, I. Elhanany, S. Berant, P. Hingston, R. M. MacCallum, A. Gowrisankar, P. Burrow, and T. Haferlach. The 2007 IEEE CEC simulated car racing competition. Unpublished manuscript.

[36] V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.

[37] B. Whitehead and T. Choate. Cooperative-competitive genetic evolution of radial basis functioncenters and widths for time series prediction. *IEEE Transactions on Neural Networks*, 7:869–880, 1996.

[38] S. W. Wilson. Classifiers that approximate functions. *Natural Computing*, 1:211–234, 2002.

[39] S. W. Wilson. Classifier conditions using gene expression programming. Technical Report 2008001, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 2008.

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