Automatic Methods for Continuous State Space Abstraction

Steven Loscalzo and **Robert Wright**
Air Force Research Laboratory Information Directorate
525 Brooks Rd.
Rome, NY, 13441
Steven.Loscalzo@rl.af.mil Robert.Wright@rl.af.mil

Abstract

Reinforcement learning algorithms are often tasked with learning an optimal control policy in a continuous state space. Since it is infeasible to learn the optimal action to take for every possible observation in a continuous state space, useful abstractions of the space must be constructed and subsequently learned on. Abstraction techniques that generalize the space into very few abstract states must take care to avoid creating an abstraction that prevents learning the optimal policy. Many commonly used abstractions, such as CMAC can take considerable effort to tune to ensure a learnable abstraction is created. In this work we propose three methods that derive state abstractions *automatically*, in part by making use of the dimensionality reduction capability of the RL-SANE algorithm. We show that abstractions derived from these automatic methods can allow a learning algorithm to converge to the optimal policy faster than with a fixed abstraction. Additionally, these techniques are able to break the space into very few abstract states, further facilitating rapid learning.

Introduction

Given a Markov Decision Problem (MDP) defined over a set of states $\mathcal{S}$ and actions $\mathcal{A}$, reinforcement learning (RL) algorithms seek to learn an optimal policy π^* which selects the appropriate action $a \in \mathcal{A}$ for each state $s \in \mathcal{S}$ to reach some specified goal state. Typical reinforcement algorithms learn π^* by repeatedly experiencing states leading to the goal state a number of times. In domains with a continuous set of states, the probability of repeatedly visiting *any* state approaches zero, preventing the learner from converging to π^*. State space aggregation or abstraction techniques must then be introduced to allow learning of an optimal policy by turning the continuous space $\mathcal{S}$ into a discrete space $\mathcal{S}'$.

State space abstraction techniques can be classified into five categories depending on the "coarseness" of the abstraction and what components of π^* in original state space are to be preserved in the abstraction (Li, Walsh, and Littman 2006). In that work, Li et al. proved that the optimal policy learned on several of the categories of abstractions (model-irrelevance, Q^π-irrelevance, and Q^*-irrelevance) resulted in an optimal policy in the original space. However, the two abstraction categories that produce the sharpest reduction in the size of the state space are not guaranteed to learn a policy that will converge to the optimal solution. This makes applying these powerful abstraction techniques (a^*-irrelevance and π^*-irrelevance) dangerous in general as they might prevent the learner from arriving at the optimal policy.

One of these classes of abstractions, the a^*-irrelevance abstraction, which group two base states together if they share the same optimal action is of particular interest. One of the most popular types of abstraction techniques, tiling, falls into this category. Examples of common tile based abstractions include CMAC (Sutton 1996), and U-tile distinction (Mccallum 1996; Uther and Veloso 1998). While tile based methods have been shown effective in a number of situations, there are serious drawbacks to using them effectively. Engineering a tiling is typically done by hand, and it can be very difficult to find an appropriate tiling for a given problem, or to correctly set the parameters in methods that build a tiling during the learning process. It has also been shown that tiling techniques cannot solve some standard benchmark problems (Gomez, Schmidhuber, and Miikkulainen 2006).

Here we propose and evaluate three *automatic* tiling methods that efficiently learn how to abstract a space but still allow a learner to converge to the optimal policy. These abstractions are applied to the one dimensional state space produced by the RL-SANE algorithm (Wright and Gemelli 2009). This allows us to focus on the partitioning methods without dealing with the dimensionality of the original state space. Each of these methods allows the abstract states, that are used by the learner, to be redrawn in an effort to get the states that share the same optimal action to fall into the same tile. This process results in smaller abstract state spaces which as a consequence greatly improves the speed of the learning.

In this paper we perform an empirical study comparing the three automatic tiling methods to the base RL-SANE algorithm using a fixed tiling. The results will show the methods proposed here allow the learning algorithm to significantly improve its rate of convergence. Additionally, we show that the automatic methods we propose here result in very few abstract states (tiles) being used in order to learn the test problems.

The rest of this paper is organized as follows: the next section describes related tile encoding methods and gives back-

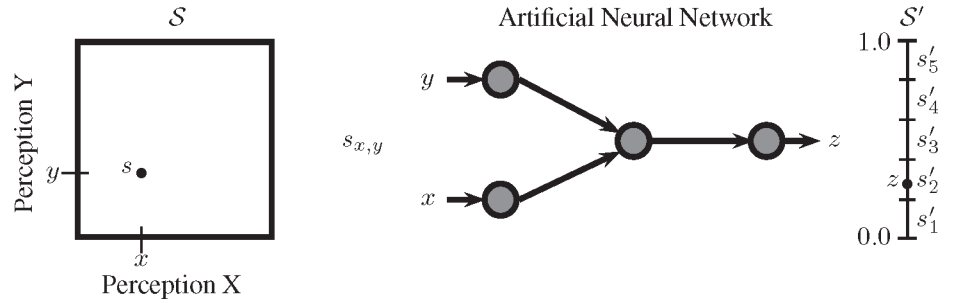

Figure 1: Overview of the RL-SANE algorithm transforming a ground state $s_{x,y}$ in a sample two dimensional state space $\mathcal{S}$ to the abstract state s'_2 in the one dimensional abstract state space $\mathcal{S}'$.

ground on the RL-SANE algorithm. The three abstraction techniques are described in depth in the Automatic Methods section. The methods are then applied to the two problems given in the Experimental Setup section, and the results of this study is given the Experimental Results section. Finally, the paper concludes with a summary of our contributions and possible future directions.

Background and Related Work

Here we give some necessary background on various tile encoding methods that have appeared in the literature, as our methods can be interpreted as tile encodings as well. We also provide details on the RL-SANE algorithm, the platform we use to reduce the dimensionality of the given learning problems and on which we examine our proposed methods.

Tile Encoding

Many tile encoding methods exist in the literature, and several popular mechanisms are variations on the Cerebellar Model Articulation Controller (CMAC) algorithm (Miller, Glanz, and Kraft 1990) which generalizes the learning of one state to a set of nearby states and based on how the human brain is thought to respond to stimuli (Albus 1971).

In tile encoding methods, the state space can be thought of as being broken apart into a number of tiles, and every time one state is observed, all the other states that belong to the same tile (or tiles, in the common event of overlapping layers of tiles) also experience the learning rewards. The size of the tiles controls the resolution of the abstraction. Smaller tiles result in a finer resolution, but require more states to be learned before the RL can find π^*. The location in the state space where the boundaries occur in the tiling can have a great impact on the ability of an algorithm to learn π^*. For example, suppose in order to optimally solve a problem, action a_1 must be taken from state s_1 and a_2 must be taken from s_2. If s_1 and s_2 happen to fall on the same tile (same abstract state) the learning algorithm will not be able to learn an optimal policy since only one action can be taken from the abstract state.

A fundamentally different approach to the tile encoding problem is taken in (Mccallum 1996) with the U Tree algorithm and later extensions to this work (Uther and Veloso 1998) to have it work in continuous domains. Here, the coarseness of the abstraction is not fixed *a priori* by setting a tiling, but rather, the space is viewed as a single tile and then repeatedly split in areas where it is determined a finer abstraction is needed. The algorithm decides to spilt a tile in two when each of the subtiles show a different distribution of observations than the whole tile, indicating that more information about the problem can be gained by splitting the tile. These approaches allow for automatic construction of a state space abstraction, however, they still suffer from a fundamentally arbitrary splitting mechanism.

When the U Tree algorithm finds that more resolution can differentiate observations in one area of the state space, the algorithm simply splits the tile in two at the center of the tile. Splitting tiles into two halves has a significant drawback, however. It is highly improbable that the actual decision point, where a change in action selection would result in a better policy, occurs in the exact center of a tile. As a consequence, many splits will need to take place before this area achieves the necessary resolution. This causes the abstraction to include and subsequently to learn many unnecessary tiles which unnecessarily slows learning. In contrast, the methods we propose allow the split locations to be positioned according to the needs of the problem and not in a pre-specified location.

RL-SANE

The RL-SANE algorithm is a powerful reinforcement learning and state abstraction algorithm (Wright and Gemelli 2009). It combines a neuroevolution approach to constructing artificial neural networks (ANN) (Stanley and Miikkulainen 2002) with a fixed tiling over a one dimensional abstract state space to allow a learner to efficiently learn complex problems by learning the optimal action for each tile in this abstract space. An overview of this process is given in Figure 1.

For any dimensionality of input space $\mathcal{S}$ the artificial neural network layer of RL-SANE takes the input measured across m dimensions and reduces it to a single output value $z \in [0, 1]$ corresponding to a single abstract state $s' \in \mathcal{S}'$. This one dimensional output space can still represent infinitely many states, so a tiling is applied to it. The fixed tiling simply splits $\mathcal{S}'$ into a number of equal sized tiles with no consideration given to the exact position of where each

split occurs, meaning that the same problem that occurs with the fixed tiling methods can occur here as well.

The difference between a true tile encoding technique and the RL-SANE algorithm is the fact that if a given abstraction does not allow π^* to be found, the neuroevolutionary component can mutate the ANNs. Through the mutation of ANNs, RL-SANE can escape local maximum and derive state abstractions that preserve the discovery of optimal policies. The original RL-SANE algorithm included a user specified parameter β to determine the number of tiles to lay over $\mathcal{S}'$. Evidence displayed in (Wright and Gemelli 2009) shows that the algorithm's overall convergence is not very sensitive to β however, the rate of learning can be significantly impacted by a poor selection.

Automatic Methods

In this work we focus on three different types of automatic methods that are able to redraw the abstraction boundaries of $\mathcal{S}'$ online as the learner embedded in the RL-SANE algorithm is learning. Here we describe mutation, maximum density separation, and temporal relative extrema methods for automatic state abstraction.

Mutation

The mutation method of automatic state bound construction makes direct use of the neuroevolution process that is at the heart of the RL-SANE algorithm, and is the closest method to the fixed tiling out of the three. As mentioned above, the basic RL-SANE algorithm takes the number of abstract states to generalize to, β, as a parameter. The mutation method encodes β as another gene in the chromosome and allows it to be mutated during the evolution of each neural network. This allows the evolutionary process to automatically explore different state abstraction possibilities in an effort to find a new one that better groups similar states together based on the output of the neural network.

In this work, we experimented with two variations of this idea, the first allows the number of abstract states to increase or decrease by one per mutation, and the second allows the number of states to change by a random amount up to a user defined threshold in a single mutation. Whenever a mutation to the state bound occurs, the method redistributes the previously learned Q-values for each action in each abstract state into the new abstract states in proportion to the overlap between the old and new states. For example, if there are half as many new states as old ones, then each new state gets initial Q-values that are the averages from the two old states that the new one overlaps. This enables the learner to reuse values that it had previously learned while allowing more refined abstractions to come into existence and drive the learner to a better solution.

This method directly improves the situation of estimating the proper fixed number of states, achieving our goal of automatic state abstraction, however, there are still some drawbacks that need to be addressed. While the boundaries are redrawn according to the chance of a mutation to the neural network, they are still arbitrarily placed over the space. This means that each state in the abstraction all cover the same portion of the space. There is no intuition that implies that this is a good strategy in general. We would rather have the smaller states be introduced where finer resolution is needed, and broader states where a coarse abstraction would do.

Algorithm 1 MDS (Maximum Density Separation)

required: number of bins for frequency distribution
output: new abstract state mapping
≪embedded within a reinforcement learning algorithm≫
get next state s' by following π^* from state s
if s' != fail state
  $s := s'$
  increment frequency distribution (s)
else if s' == fail state
  locate relative extrema in frequency distribution
  erase old partitions of $\mathcal{S}'$
  partition $\mathcal{S}'$ in the center of two relative maximums
    if a relative minimum occurred between them
end else if

Maximum Density Separation

The maximum density separation (MDS) approach takes a different tact in determining the tiling to be used. Unlike the mutation method, MDS can place the boundaries of a split anywhere in the state space and can add or remove as many abstract states at a time as the algorithm determines necessary. This method intuitively views dense clusters of observations as belonging to a single state, and abstracts the state space so that these dense clusters are located on separate tiles from one another. The split between tiles occurs at the farthest point between two dense regions of observations. This approach is principled by the idea that nearby states will prefer the same action, however the size of each of these groups may vary so we must use an adaptable partitioning solution.

An overview of the Maximum Density Separation method is given in Algorithm 1. For a single run of the problem in a given RL algorithm, this method records the frequency of observations across the state space until a failure or the goal state is reached. On a failure, the constructed frequency distribution is searched for relative extrema. A soft-thresholding approach is used to prevent small fluctuations in the distribution from leading to many spurious extrema. Once the relative extrema have been identified, a partition is placed in the space in the center of every two relative maximums, as long as a relative minimum occurred between them. The splits between abstract states are made in this fashion in accordance with the maximum margin principle (Mitchell 1997), which seeks to minimize the structural variance in a hypothesis. Positioning the splits as far as possible from the dense regions of observations minimizes the risk that, in the next run of the problem, new observations belonging to one dense region will spill into an adjacent state and mislead the learning there.This process is linear in the number of bins used to measure the frequency distribution, and in practice had only a negligible impact on the running time of each generation of the algorithm. After the new state abstraction has been set, the Q-values that were learned on the earlier state abstraction are transferred to the new ab-

straction in the same manner as described in the mutation method.

This method effectively overcomes two of the perceived limitations of the mutation method, abstract state partitions can be placed anywhere in the space and the number of states in the space does not directly depend on the previous number. The MDS method does introduce some other limitations, however. It could be the case that an area of dense observations is not really homogenous in terms of preferred action, but were coincidentally grouped together by the ANN. In this case, the abstract states might still become successful if the ANN adapts and separates these states into two different clusters in a later evolutionary stage. Another limitation of MDS is that it has no notion of the series of observations that led to the failure. These observations may easily be grouped with other observations with similar values but should be separated and allowed to pursue other actions as soon as possible. The MDS method makes no provision for this possibility and relies on the ANN to separate out the other states in a later generation. The next method addresses these shortcomings.

Temporal Relative Extrema

As the name implies, the Temporal Relative Extrema (TRE) method creates an abstraction by incorporating the order in which observations are generated and not just their values as in the MDS method. Like the MDS, TRE is capable of partitioning $\mathcal{S}'$ into as many abstract states as necessary, and can place partitions between states anywhere in the space. During preliminary studies, we noticed that the observations in $\mathcal{S}'$ frequently followed a periodic function, much like a sine curve. The TRE approach was created to encourage more exploration in the learner and break away from the periodic repetition of known states to get the learner to visit new and possibly more beneficial states.

The TRE method is summarized in Algorithm 2. All of the observations are recorded for a single run of the problem until a failure is encountered. At this point, the algorithm iterates through each stored observation and identifies relative maxima and minima as those places are associated with restarting the next period of observations. Each relative extrema is compared to the rest and if they are within a user defined threshold t away from each other then they are considered to be the same extrema and are merged together. For minima, the greatest (innermost) observation is stored after the merge, and for maxima, the smallest (again innermost) observation is stored. The state space $\mathcal{S}'$ is then partitioned at each of the final extrema locations and becomes the new abstraction for the next generation. The initial Q-values of the new abstraction are taken from the previous abstraction in the same manner as the mutation and MDS methods. The time complexity of the TRE method is linear in the number of observations for each run of the problem, and since the observations must be generated anyway, there is no noticeable effect on the overall running time of the learning algorithm.

The heuristic of partitioning the state space based on where the learner begins to repeat observations for the same problem is quite distinct from the previous approaches mentioned here. In effect, this groups the heavily repeated observations into the same abstract state while allowing for the relative extrema to more easily find their own preferred actions, which can lead to an improved learning rate. If the extrema prefer the same action as the other heavily repeated observations then there is not much harm done by separating them, as their initial Q-values will be shared according to the previous abstraction anyway and should not hurt the overall learning rate. The similarity parameter t does not need to be significantly tuned. It suffices to set it small compared to the range of possible values for an observation. If t is very small (s.t. $|z_i - z_j| > t$ for nearly all observations $z_i, z_j \in \mathcal{S}'$ near relative extrema, with $i \neq j$), many abstract states will be created near the extrema. But, this has little real impact on the learning since they will generally share the same Q-values over successive generations.

Algorithm 2 TRE (Temporal Relative Extrema)

required: similarity threshold t for comparing extrema
output: new abstract state mapping
≪embedded within a reinforcement learning algorithm≫
get next state s' by following π^* from state s
if s' != fail state
  $s := s'$
  record s
else if s' == fail state
  for each relative extrema
    merge extrema if closer than t from an existing extrema
    otherwise store the extrema
  end for each
  erase old partitions of $\mathcal{S}'$
  partition $\mathcal{S}'$ on the inside each stored extrema
end else if

Experimental Setup

Here we compare the performance of the automatic state abstraction techniques proposed above, namely small mutation, large mutation, MDS, and TRE. In addition to measuring the techniques against one another, we also compare them to the base RL-SANE algorithm with a fixed abstraction over the input space, which has been shown to be a very capable learner (Wright and Gemelli 2009). The comparison examines the rates of convergence of each of the methods as well as the number of states that are used in their final abstractions on two benchmark RL problems, the mountain car and double pole balance. It is thought that the automatic abstractions will enable a faster convergence to the optimal policy by selecting a reasonable number of abstract states to learn on.

The mountain car problem (Boyan and Moore 1995) consists of a car trying to escape a valley. The car's engine is too weak to provide enough power to simply drive over the hill in front of it, and instead must build up momentum by driving partially up the hill behind it before moving forward towards the goal. To escape the learner must learn a policy of repeating a back-and-forth motion several times before building up the requisite power.

This problem is described by two perceptions: the positions of the car, and the velocity of the car. Time is dis-

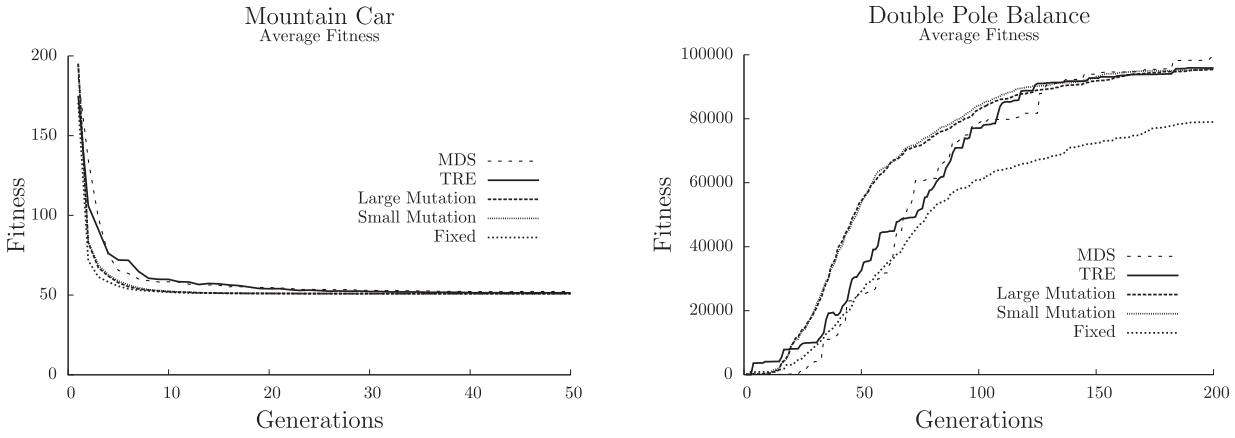

Figure 2: The performance of the four automatic abstraction methods and the Fixed abstraction on the Mountain Car and Double Pole Balance problems.

cretized into small intervals and the learner can choose one of two actions in each time step: drive forward or backward. The only reward that is assigned is -1 for each each action that is taken before the car reaches the goal of escaping the valley. Since RL algorithms seek to maximize the reward the optimal policy is the one that enables the car to escape the valley as quickly as possible.

The double inverted pole balancing problem (Gomez and Miikkulainen 1999) is a very difficult RL benchmark problem. In this problem, the learner must balance two poles of different length and mass which are attached to a moving cart. The problem is further complicated by maintaining that the cart must stay within a certain small stretch of track. If the learner is able to prevent the poles from falling over after a specified amount of time then the problem is considered solved.

This is a higher dimensional problem than the mountain car problem, with six perceptions being given to the learner: the position of the cart, the velocity of the cart, the angle each beam makes with the cart, and the angular velocities of the beams. Once again, time is discretized into small intervals, and during any such interval the learner can choose to push the cart to the left or right or to leave it alone. In our experiment, the only signal the learner receives is a negative reward, -1, for dropping either pole or exceeding the bounds of the track. The goal is to balance the poles for 100000 time steps without dropping them or exceeding the bounds.

For all of our experiments we averaged 25 separate runs over the problem sets using different random seeds. The mountain car problem set consists of 100 preset initial start states, and for the double pole balance the problem set consists of 20 random initial start states. It should be noted that the mutation methods and the fixed RL-SANE all have a significant dependency on the initial number of abstract states, while the MDS and TRE methods do not. To account for this in the presented results the mutation methods and fixed RL-SANE values are the averages of each of the problem sets starting with $10, 20, \ldots, 100$ initial abstract states; in other words, the average of 250 problem set runs for both problems. For the fixed RL-SANE algorithm, these initial states did not change for the duration of the learning process, while the mutation methods are free to alter them over time. Prior experiments have shown that the fixed RL-SANE algorithm achieves the best learning rate with 50 abstract states in mountain car and 10 in double pole balance, both of which are included in the abstract state ranges that were tested on.

The RL-SANE algorithm was set to use a pool of 100 neural networks per generation, with a maximum of 200 generations of learning. We used the Sarsa(λ) learning algorithm with all learning and neuroevolution parameters set as in (Wright and Gemelli 2009). We allowed the large mutation method to alter the number of states by up to 5 per generation. For MDS the density of the observations in the state space was estimated using 1000 evenly spaced areas to collect observations. The exact value of this parameter is unimportant as long as it is significantly larger than the number of expected abstracted states in the solution. TRE considered two extrema in its results different if they were at least 0.1 apart (i.e. $t = 0.1$).

Experimental Results

Figure 2 shows the average fitness over each of the problem sets for the mountain car and double pole balance domains. The mountain car problem shows all five methods performing very similarly. All methods rapidly converge to a policy that takes on average approximately 50 time steps to navigate the car from the valley. Both of the more sophisticated methods, MDS and TRE, lag behind the top performers somewhat. This result indicates that this problem can be easily learned without complicated abstract state repartitioning. These results do serve to show that using automatically repartitioning of the abstract state space does not hurt the overall convergence of the learner on simple reinforcement learning problems even when the problem is simple enough that a fixed abstraction is sufficient.

Examining the fitness curves of the double pole balance problem shows several trends. The most obvious conclusion that can be drawn is that the automatic methods are all able

	Mountain Car	Double Pole Balance
MDS	3.36 ± 1.25	13.5 ± 5.93
TRE	63.64 ± 23.57	58.13 ± 13.48
Large	14.77 ± 8.6	24.51 ± 15.59
Small	14.82 ± 8.7	24.48 ± 15.43
Fixed	50	10

Table 1: Average number of final abstract states $\pm$ stdev and optimal number of states for the fixed abstraction.

to converge towards the optimal policy at a greater rate than the fixed RL-SANE algorithm. If the number of abstract states are tuned, the fixed RL-SANE method can find the optimal policy at a similar rate as the other algorithms. However, if a range of possible good parameters are used instead the algorithm does not do nearly as well. Whereas, the mutation methods, both small and large, are able to overcome the arbitrary initial parameters and efficiently repartition the abstract state space to allow the learner to quickly converge to the optimal policy. The MDS and TRE methods started out near the fixed method but rapidly improved due to the mutation methods. Towards the end of the reported generations the MDS method shows the best performance overall, validating the idea that allowing a more specialized partition of the state space can lead to improved convergence properties of the learner. TRE proves to be an able abstraction method as well, and the performance of that algorithm is noteworthy early on in the learning process. We can see that it experiences an almost immediate jump in fitness, which may be due to its heuristic which favors separating those observations which may be able to reach previously unexplored areas of the state space if they are able to follow actions that are not preferred by other nearby observations.

Table 1 contains the average final number of abstract states for each automatic abstraction method as well as optimal number of states for the fixed tiling. We can immediately see that the optimal number of states for the fixed RL-SANE algorithm is not the number of states that each of the automatic methods tend to; only TRE on mountain car and MDS on double pole balance are similar. TRE tends to break up the space into many more states than the other methods, while MDS leads the abstraction towards fewer states. This implies that there are relatively few clusters of observations in the abstract space, but there are many repetitive substructures in these clusters when the order of observations are considered. Both of the mutation methods converge to similar low numbers of states in the final abstractions, which explains why their fitness measures in Figure 2 are also very similar.

Conclusions and Future Directions

We have presented three types of automatic state abstraction techniques; mutation methods that make use of ANNs to abstract the space, Maximum Distance Separation which seeks to partition a space based on dense regions of observations, and Temporal Relative Extrema which builds abstract states by separating observations that lead to previously seen areas of the state space. Each of these methods has been shown to improve the learning rate as compared to using a fixed abstraction, and they make use of only a small number of states while doing so.

One future direction of this work is to experiment with the techniques on higher dimensional state spaces, instead of restricting them to the one dimensional abstract space at work in the RL-SANE algorithm. Another interesting possibility is to relax the current abstraction conceptualization and not require that a partition of the space be determined; instead only focus on those areas that need increased resolution.

References

Albus, J. S. 1971. A theory of cerebellar functions. *Mathematical Biosciences* 10:25–61.

Boyan, J. A., and Moore, A. W. 1995. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*, 369–376. MIT Press.

Gomez, F. J., and Miikkulainen, R. 1999. Solving non-markovian control tasks with neuroevolution. In *In Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1356–1361. Morgan Kaufmann.

Gomez, F.; Schmidhuber, J.; and Miikkulainen, R. 2006. Efficient non-linear control through neuroevolution. In *Proceedings of the European Conference on Machine Learning*, 654–662.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a unified theory of state abstraction for mdps. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 531–539.

Mccallum, A. K. 1996. *Reinforcement learning with selective perception and hidden state*. Ph.D. Dissertation, The University of Rochester. Supervisor-Ballard, Dana.

Miller, W.T., I.; Glanz, F.; and Kraft, L.G., I. 1990. Cmas: an associative neural network alternative to backpropagation. *Proceedings of the IEEE* 78(10):1561 –1567.

Mitchell, T. 1997. *Machine Learning*. McGraw Hill.

Stanley, K. O., and Miikkulainen, R. 2002. Efficient reinforcement learning through evolving neural network topologies. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, 569–577.

Sutton, R. 1996. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, volume 8, 1038–1044. MIT Press.

Uther, W. T. B., and Veloso, M. M. 1998. Tree based discretization for continuous state space reinforcement learning. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, 769–774. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Wright, R., and Gemelli, N. 2009. State aggregation for reinforcement learning using neuroevolution. In *ICAART 2009 - Proceedings of the International Conference on Agents and Artificial Intelligence*, 45–52.