Function Approximation Using Tile and Kanerva Coding For Multi-Agent Systems *

Cheng Wu
Northeastern University
360 Huntington Avenue
Boston, MA, U.S.A.
cwu@ece.neu.edu

Waleed Meleis
Northeastern University
360 Huntington Avenue
Boston, MA, U.S.A.
meleis@ece.neu.edu

ABSTRACT

Function approximation can improve the ability of a reinforcement learner to solve large-scale multi-agent problems. Tile coding and Kanerva coding are two classical methods for implementing function approximation, but these methods may give poor performance when applied to large-scale, high-dimensional instances. In this paper, we evaluate a collection of hard instances of the predator-prey pursuit problem, a classic multi-agent reinforcement learning problem, to compare these two methods and their optimization techniques. We first show that Kanerva coding gives better results than Tile coding when the dimension of the instances increases. Kanerva coding solves 37% of the hardest instances, while Tile coding solves only 17% of the instances. We then describe a feature optimization mechanism and show that it can increase the fraction of instances that are solved by both Tile coding and Kanerva coding. Kanerva coding with feature optimization solves 63% of the hardest instances, while Tile coding with feature optimization solves only 37% of the instances. Finally, we demonstrate that a fuzzy approach to function approximation can further increase the fraction of instances. We show that our fuzzy approach to Kanerva coding outperforms fuzzy Tile coding when feature optimization is applied. Fuzzy function approximation increases the average solution rate to 43% using Tile coding, and from 72% using Kanerva coding. We conclude that discrete and fuzzy Kanerva coding represent powerful function approximation techniques that can outperform discrete and fuzzy Tile coding on large-scale, high-dimensional multi-agent optimization problems.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

*(Produces the copyright information for AAMAS 2009). For use with aamas2009.CLS and aamas2009-extendedabs.cls

General Terms

Algorithms, Experimentation

Keywords

Reinforcement Learning, Function Approximation, Sparse Distributed Memory, Fuzzy Logic

1. INTRODUCTION

Reinforcement learning [12] is a useful machine learning strategy that has been used to solve a wide variety of hard optimization problems. Q-learning [14] has emerged as one of the most successful reinforcement learning strategies. The algorithm works by combining state space exploration and exploitation to learn the value of each state-action pair. Through repeated trials, the estimates of the values of each state-action pair can gradually converge to the true value, and these can be used to guide the agent to maximize its reward. Under certain limited conditions, Q-learning has been shown to always converge to an optimal policy.

A key limitation on the effectiveness of Q-learning is the size of the table needed to store the state-action values. The requirement that an estimated value be stored for every state-action pair limits the size and complexity of the learning problems that can be solved. The Q-learning table is typically large because of the high dimensionality of the state-action space, or because the state or action space is continuous. Function approximation [3], which stores an approximation of the entire table, is one way to solve this problem.

Many function approximation techniques exist, and they can be divided into two categories: feature-based and basis-function-based. Feature-based function approximation includes coarse coding [6], Tile coding [2] (also known as CMAC [14]). Coarse coding, for example, uses a collection of overlapping regions within the state-action space, each of which corresponds to a natural binary feature. Basis-function-based function approximation includes Radial Basis Function Networks (RBFNs) [10] and Kanerva coding [9]. For example, RBFNs use a series of radial basis functions that vary smoothly and are differentiable, not just natural features. There are guarantees on their effectiveness in some cases.

Tile coding is a special case of coarse coding. In Tile coding, *k tilings* are selected, each of which partitions the state-action space into tiles. The receptive field of each binary

feature corresponds to a tile, and a θ value is maintained for each tile. A state-action pair p is *adjacent* to a tile if the receptive field of the tile includes p. The Q-value of a state-action pair is equal to the summation of the θ values of all adjacent tiles. In binary Tile coding, used when the state-action space consists of discrete values, each tiling corresponds to a subset of the bit positions in the state-action space. Then each tile corresponds to an assignment of binary values to the selected bit positions.

A limitation of this technique is that it cannot handle a state-action space with a large dimension. In general, the number of tilings needed to achieve good results grows exponentially with the number of dimensions in the state-action space [12].

Kanerva coding is a special case of RBFNs. It uses the architecture of sparse distributed memories [8] that can also reduce the amount of memory needed to store the state-action value table. This approach is particularly well-suited to problem domains with high dimensionality. A collection of k *prototype state-action pairs*, (prototypes) is selected, each of which again corresponds to a binary feature. A state-action pair s and a prototype p_i are said to be *adjacent* if their bit-wise representations differ by no more than a threshold number of bits. Normally we set the threshold as 1 bit. We define the *membership grade* $\mu_i(s)$ of s with respect to p_i to be equal 1 if s is adjacent to p_i, and equal to 0 otherwise.

$$\mu_i(s) = \begin{cases} 1 & \text{if } s \text{ is adjacent to } p_i, \\ 0 & \text{otherwise.} \end{cases}$$

A value $\theta(i)$ is maintained for the ith feature, and an approximation of the value of a state-action pair is then the sum of the θ values of the adjacent prototypes, that is $\sum_i \theta(i)\mu_i(s)$. In this way, Kanerva coding can greatly reduce the size of the value table that needs to be stored.

A tile in Tile coding represents information about only a subset of the dimensions of the state-action space because the computational complexity increases exponentially with the number of dimensions. In constrast, a prototype in Kanerva coding contains information about all dimensions of a state-action space. This allows the complexity of the approximate function to depend only on the number of prototypes, not the dimension of the state-action space.

Several researchers have investigated the use of Tile coding and Kanerva coding within reinforcement learners [11, 15]. However there have been few published attempts to compare Tile coding and Kanerva coding for multi-agent problems. Furthermore, optimizing features in Tile coding and Kanerva coding [11] has been shown to reduce the number of features needed to achieve a good solution rate. However no work has been done to directly compare the effect of this adaptive function approximation across Tile coding and Kanerva coding.

The paper is organized as follows. In Section 2, we describe our experimental design. In Section 3, we define feature collision and compare the efficiency of traditional Tile coding and Kanerva coding. In Section 4, we describe a feature optimization mechanism and compare the effect of feature optimization within Tile coding and Kanerva coding. In Section 5, we describe our fuzzy mechanism and demonstrate that fuzzy Kanerva coding outperforms fuzzy Tile coding when feature optimization is applied. We conclude the paper in Section 6.

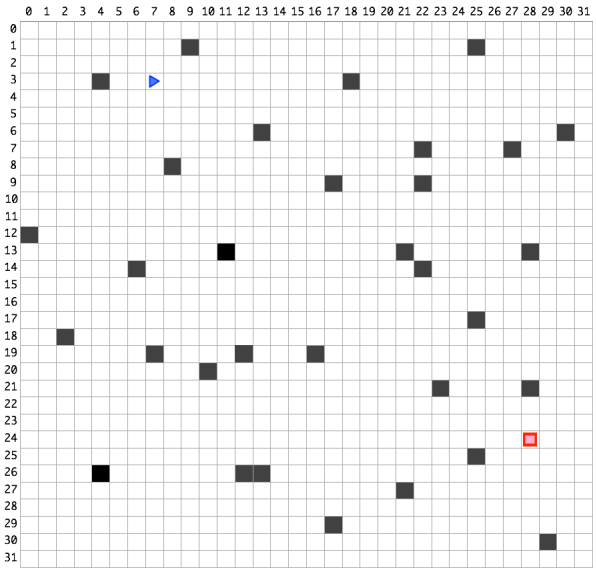

Figure 1: An example of our grid world with size of 32 x 32.

2. EXPERIMENTAL DESIGN

Multi-agent problems can be difficult to solve by traditional machine learning techniques because the state space is often very large. The predator-prey pursuit problem [4] is a classic example of such a multi-agent problem. Closed-form solutions to restricted versions of the problem have been found [1, 7], but most such problems remain open. Researchers have used approaches such as genetic algorithms [5] and reinforcement learning [13] to develop solutions.

A general version of the predator-prey pursuit problem takes place on a rectangular grid with one or more predator agents and one or more prey agents. Each grid cell is either open or closed, and an agent can only occupy open cells. Each agent has an initial position. The problem is played in a sequence of time periods. In each time period, each agent can move to a neighboring open cell one horizontal or vertical or diagonal step from its current location, or it can remain in its current cell. All moves are assumed to occur simultaneously, and more than one predator agent may not occupy the same cell at the same time. Each predator agent can observe the location of each prey agent. If a predator agent is in the same cell as a prey agent at the end of a time period, then that target has been caught. Any predator agent may pursue and catch any prey agents. The predator agents' goal is to catch all the prey agents in the shortest time.

In our experiments, pursuit takes place on a rectangular grid with size of n x n and n is 16, 32 and 64. There are n closed blocks that are distributed randomly. Figure 1 give an example of our experimental grid world with size of 32 x 32. The agent receives a reward of 1 when it reaches the a

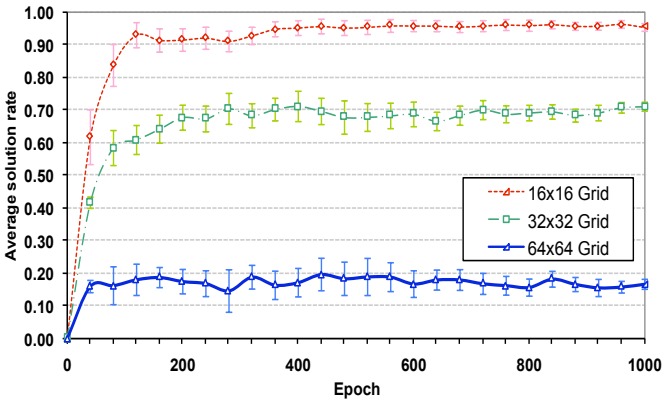

Figure 2: Average solution rate of traditional Tile coding.

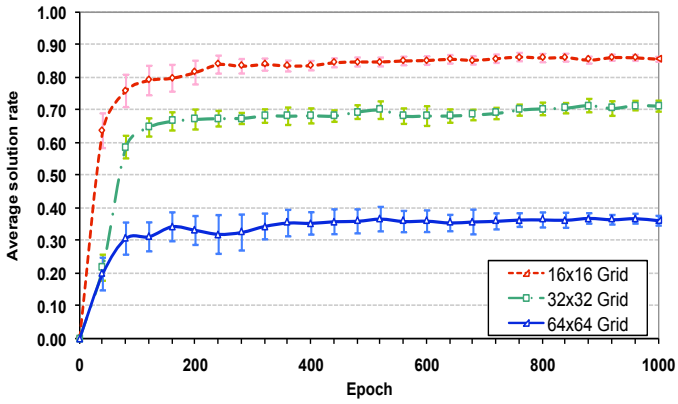

Figure 3: Average solution rate of traditional Kanerva coding.

fixed goal cell, and receives a reward of 0 in every other cell. In each epoch, we apply the learning algorithm with Tile coding or Kanerva coding to 40 random training instances followed by 40 random test instances. The exploration rate ϵ is set to 0.3, which we found experimentally to give the best results in our experiments. The initial learning rate α is set to 0.8, and it is decreased by a factor of 0.995 after each epoch. For every 40 epochs, we record the average fraction of test instances solved during those epochs. Each experiment is performed 5 times and we report the means and standard deviations of the recorded values. In our experiments, all runs were found to converge within 1000 epochs.

3. COMPARISON OF TRADITIONAL TILE CODING AND KANERVA CODING

We begin by comparing Q-learning with traditional Tile coding to traditional Kanerva-based function approximation as the dimension of the state-action space increases.

We implement Tile coding by representing the state-action pair as a binary vector. Each tiling corresponds to a 3-tuple of bit positions. For $n = 16$, the number of possible tilings is 364 which produces a total of 2912 tiles. We fix the number of tilings to 364 across all grid sizes. In this way we can evaluate the effect of a fixed tiling on the performance of Q-learning as the dimension of the state-action space increases.

In a similar way, we implement Kanerva coding by randomly selecting 2912 prototype states from the state-action space. The number of prototype states is fixed across all grid sizes.

The graphs in Figures 2 and 3 show the results obtained by traditional Tile coding and traditional Kanerva as the size of the grid varies from 16 to 64. For a grid size of 16, Tile coding solves 96% of the test instances while Kanerva coding solves only 86% of the test instances, after 1000 epochs. However, for a grid size of 64, Tile coding solves 17% of the test instances while Kanerva Coding solves 37% of the test instances, after 1000 epochs.

The results show that when the number of dimensions is small, traditional Tile coding outperforms traditional Kanerva coding. However, as the number of dimensions increases, Tile coding's performance degrades faster than the performance of Kanerva coding when the number of prototypes is fixed. We conclude that Kanerva coding performs better relative to Tile coding when the dimension of the state-action space is large.

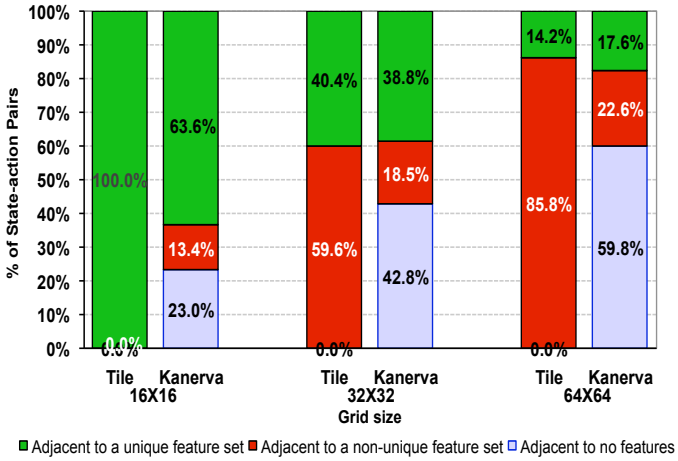

Figure 4: Distribution of the number of collisions per prototype using Tile and Kanerva coding in a sample run.

Function approximation works best when each state-action pair is adjacent to unique subset of features (that is, tiles in Tile coding, and prototypes in Kanerva coding). If features are not well distributed across the state-action space, many state-action pairs will either not be adjacent to any features, or be adjacent to identical sets of features. Such feature *collisions* reduce the quality of the results because the solver will not be able to distinguish distinct state-actions pairs, and the Q-values of such state-action pairs will be equal.

Figure 4 shows the fraction of state-action pairs that are adjacent to no features, adjacent to more than one feature, and adjacent to exactly one feature when traditional Tile

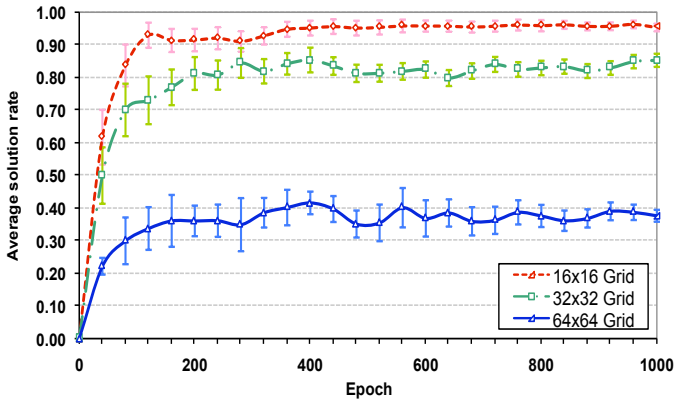

Figure 5: Average solution rate of Tile coding with feature optimization.

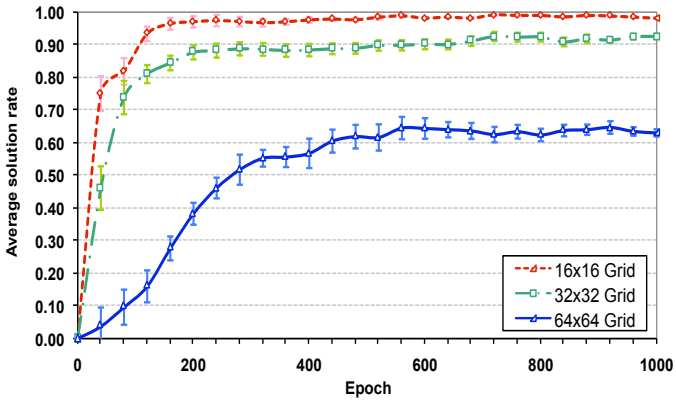

Figure 6: Average solution rate of Kanerva coding with feature optimization

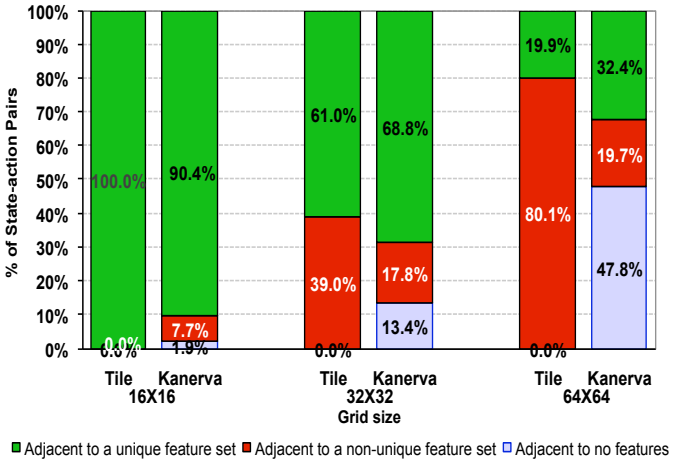

Figure 7: Distribution of the number of collisions per prototype using Tile and Kanerva coding with feature optimization in a sample run.

coding and Kanerva coding are applied to sample predator-prey instances of varying sizes.

For Tile coding, the fraction of state-action pairs that are adjacent to exactly one tile varies from 100% to 14% as the size of the grid increases, so up to 86% of the state-action pairs cause collisions. For Kanerva coding, the fraction of state-action pairs that are adjacent to exactly one tile varies from 64% to 18% as the size of the grid increases, so up to 82% of the state-action pairs cause collisions. Note that with Tile coding, every state-action pair is adjacent to at least one tile in every tiling. Therefore, there are no state-action pairs that are adjacent to no features.

These results show that when the dimension of the state-action space is small, Tile coding causes fewer collisions than Kanerva coding, which explains the good performance of Tile coding for these instances. Kanerva coding gives slightly fewer collisions when the dimension of the state-action space is large.

In both cases, the number of collisions increases sharply as the dimension of the state-action space increases, which explains the reduction in performance. It is therefore necessary to consider optimization mechanisms that adjust the distribution of features to reduce the number of collisions as the dimension of the state-action space increases.

4. EFFECT OF FEATURE OPTIMIZATION ON TILE AND KANERVA CODING

We say that a feature is *visited* during Q-learning if it is adjacent to the current state-action pair, and we optimize features based on these visit frequencies. Initial features are selected randomly. After a fixed number of iterations, features are updated using the following mechanisms.

Features that are rarely visited do not contribute to the solution of instances and indirectly cause collisions. We periodically delete each feature with probability equal to an exponential function of the visit frequency. I.e. the probability p_{del} of deleting a feature whose visit frequency is v is

$$p_{del} = \lambda e^{-\lambda v},$$

where $\lambda = 1$.

Features that are visited frequently are also likely to cause many collisions. We reduce the number of collisions by splitting frequently visited features. A feature s_1 that has been visited the most times is selected, and a new feature s_2 is created based on s_1. For Tile coding, s_2 is constructed by randomly selecting and retaining two of the bit positions in s_1, and randomly selecting a new third bit position. For Kanerva coding, s_2 is constructed by inverting a fixed number of bits in s_1. In both cases, the feature s_1 remains unchanged.

The effect of feature optimization is shown in Figures 5 and 6 as the size of the grid varies from 16 x 16 to 64 x 64. The graph in Figure 5 shows the average solution rate obtained by Tile coding with feature optimization. Tile coding with

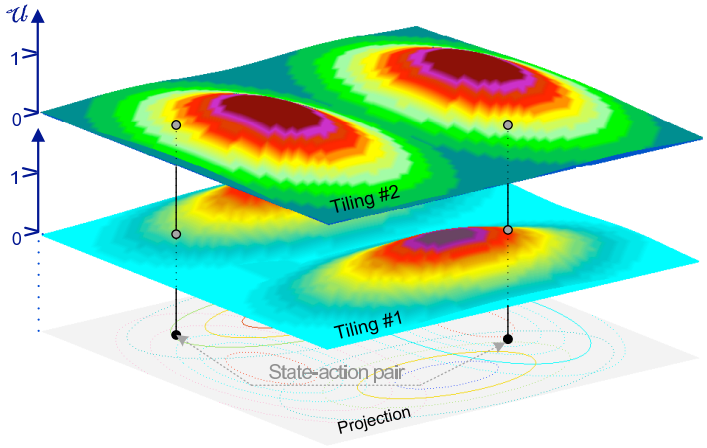

Figure 8: Sample membership functions for fuzzy Tile coding.

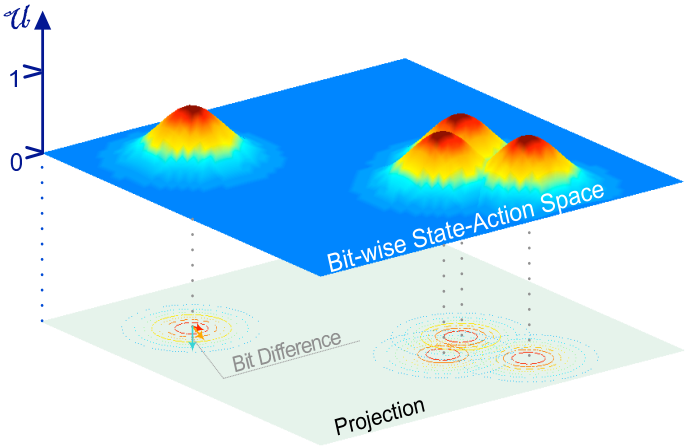

Figure 9: Sample membership functions for fuzzy Kanerva coding.

feature optimization solves an average of 85% of the test instances with a grid size of 32 x 32, and 37% with a grid size of 64 x 64. Kanerva coding with feature optimization solves an average of 92% of the test instances with a grid size of 32 x 32, and 63% with a grid size of 64 x 64. Note that feature optimization is not needed when using Tile coding with a grid size of 16 x 16 because all possible tilings are used.

These results show that feature optimization can significantly improve the average solution rate when using Tile coding or Kanerva coding. However, feature optimization causes a larger absolute improvement in the solution rate with Kanerva coding (26%) than with Tile coding (20%). We also observe that Tile coding gives a larger deviation in the solution rate than Kanerva coding. This shows that Tile coding with feature optimization is less stable than Kanerva coding with optimization.

Figure 7 shows the fraction of state-action pairs that are adjacent to no features, adjacent to more than one feature, and adjacent to exactly one feature when feature optimization with Tile coding and Kanerva coding are applied to sample predator-prey instances of varying sizes.

For Tile coding, the fraction of state-action pairs that are adjacent to exactly one tile varies from 100% to 20% as the size of the grid increases, so up to 80% of the state-action pairs cause collisions. For Kanerva coding, the fraction of state-action pairs that are adjacent to exactly one tile varies from 90% to 32% as the size of the grid increases, so up to 68% of the state-action pairs cause collisions.

These results show that when the dimension of the state-action space is large, feature optimization causes a larger absolute reduction in collisions using Kanerva coding (15%) than using Tile coding (6%). However, in both cases the number of collisions again increases sharply as the dimension of the state-action space increases.

5. COMPARISON OF FUZZY TILE AND KANERVA CODING

Feature receptive fields with crisp boundaries can cause frequent collisions. A more flexible approach to computing state-action values is to use receptive fields with fuzzy boundaries. For Tile coding, we allow a state-action pair to update θ values of all tiles within a tiling, instead of a single tile. For Kanerva coding, a state-action pair updates θ values of all prototypes, instead of just adjacent prototypes.

Figures 8 and 9 give an abstract description of the distribution of a state-action pair's fuzzy membership grade with respect to each feature. Figure 8 illustrates the fuzzy mapping of a state-action pair to tiles within two tilings. Similarly, Figure 9 illustrates the fuzzy membership function for four prototypes.

Instead of being binary values, membership grades vary continuously between 0 and 1 across features. Such fuzzy membership grades are larger for adjacent features and smaller for more distant features. Since feature collisions occur only when two state-action pairs have identical membership vectors, collisions are less likely.

In the fuzzy approach to Tile coding, the membership grade is defined as follows. Given a state-action pair s, the ith tile of the jth tiling $t_{i,j}$, and a constant variance σ, the membership grade of s with respect to the $t_{i,j}$ is

$$\mu_{i,j} = exp\left(-\frac{||s - t_{i,j}||^2}{2\sigma^2}\right),$$

where $||s - t_{i,j}||$ represents the bit difference between s and $t_{i,j}$. Note that the membership grade of a tile with respect to an identical state-action pair is 1.

In the fuzzy approach to Kanerva coding, the membership grade is defined as follows. Given a state-action pair s, the ith prototype p_i, and a constant variance σ, the membership grade of s with respect to p_i is

$$\mu_i = exp\left(-\frac{||s - p_i||^2}{2\sigma^2}\right),$$

where $||s - p_i||$ represents the bit difference between s and p_i. Note that the membership grade of a prototype with respect

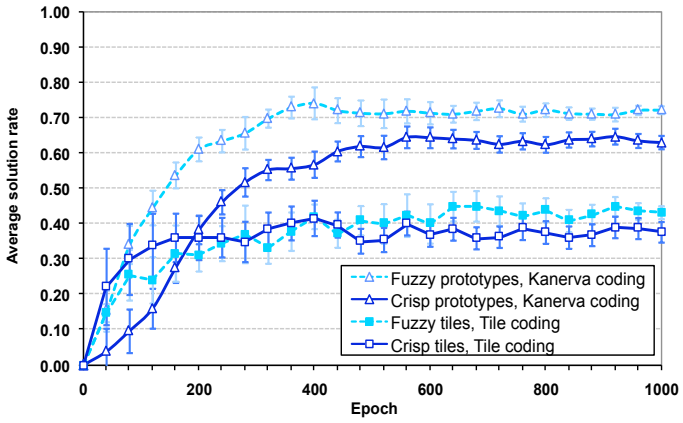

Figure 10: Average solution rate of fuzzy Tile and Kanerva coding with feature optimization.

to an identical state-action pair is 1, and the membership grade of a state-action pair and a completely different prototype approaches 0.

As with traditional Kanerva coding, a value $\theta(i)$ is maintained for the ith feature and an approximation of the value of a state-action pair is computed in the same way as before. The effect of an update to a prototype's θ-value is now a continuous function of the bit difference between the state-action pair and the prototype, and an update can have a large effect on immediately adjacent prototypes, and a smaller effect on more distant prototypes.

The effect of fuzzy function approximation with a grid size of 64x64 is shown in the graph in Figure 10. The graph shows the average solution rate obtained by fuzzy and crisp function approximation using Tile coding and Kanerva coding with feature optimization. Fuzzy function approximation increases the average solution rate from 37% to 43% using Tile coding, and from 63% to 72% using Kanerva coding. Fuzzy function approximation improves the performance of both approaches, and the absolute improvement in the average solution rate is larger using Kanerva coding than using Tile coding.

6. CONCLUSION

In this paper we have evaluated the performance of several function approximation techniques by applying them to large-scale, high-dimensional instances of predator-prey pursuit instances. We first showed that traditional Tile coding and Kanerva coding give poor performance. Kanerva coding solved 37% of the hardest instances, while Tile coding solved only 17% of the instances, demonstrating that Kanerva coding gives better results than Tile coding when the dimension of the instances is large.

We then showed that this poor performance is a result of the inefficient distribution of features which causes frequent collisions. We described a feature optimization mechanism and showed that it can increase the fraction of instances that are solved by both Tile coding and Kanerva coding. We showed that feature optimization reduced collisions from 86% to 80% using Tile coding, and from 82% to 68% using Kanerva coding. As a result, feature optimization increased the fraction of the hardest instances solved from 17% to 37% using Tile coding, and from 37% to 63% using Kanerva coding. We observed that for the hardest instances, feature optimization caused a larger absolute reduction in collisions with Kanerva coding compared to Tile coding (15% vs. 6%), and a larger absolute improvement in the solution rate (26% vs. 20%).

Finally, we showed that a fuzzy approach to function approximation can further reduce the number of collisions. We showed that our fuzzy approach to Kanerva coding outperforms fuzzy Tile coding when feature optimization is applied. Fuzzy function approximation increases the average solution rate from 37% to 43% using Tile coding, and from 63% to 72% using Kanerva coding.

We conclude that for large-scale, high-dimensional multi-agent optimization problems, discrete and fuzzy Kanerva coding represent powerful function approximation techniques that can outperform discrete and fuzzy Tile coding.

7. REFERENCES

[1] M. Adler, H. Racke, N. Sivadasan, C. Sohler, and B. Vocking. Randomized pursuit-evasion in graphs. In *Proc. of the Intl. Colloq. on Automata, Languages and Programming*, 2002.

[2] J. Albus. *Brains, Behaviour, and Robotics.* McGraw-Hill, 1981.

[3] L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proc. of the 12th Intl. Conf. on Machine Learning.* Morgan Kaufmann, 1995.

[4] M. Benda, V. Jagannathan, and R. Rodhiawalla. On optimal cooperation of knowledge sources. *Technical Report, Boeing Computer Services*, 1985.

[5] T. Haynes and S. Sen. The evolution of multiagent coordination strategies. *Adaptive Behavior*, 1997.

[6] G. Hinton. Distributed representations. *Technical Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh*, 1984.

[7] V. Isler, S. Kannan, and S. Khanna. Randomized pursuit-evasion with local visibility. *SIAM Journal on Discrete Mathematics*, 20(1):26–41, 2006.

[8] P. Kanerva. *Sparse Distributed Memory.* MIT Press, 1988.

[9] K. Kostiadis and H. Hu. Kabage-rl: kanerva-based generalisation and reinforcement learning for possession football. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2001.

[10] T. Poggio and F. Girosi. Networkds for approximation and learning. In *Proc. IEEE, vol78, no. 9, pp. 1481-1497*, 1990.

[11] B. Ratitch and D. Precup. Sparse distributed memories for on-line value-based reinforcement learning. In *Proc. of the European Conf. on Machine Learning*, 2004.

[12] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* Bradford Books, 1998.

[13] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative learning. In M. N. Huhns and M. P. Singh, editors, *Readings in Agents*, pages

487–494. Morgan Kaufmann, CA, 1997.

[14] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1989.

[15] C. Wu and W. Meleis. Adaptive kanerva-based function approximation for multi-agent systems. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.