2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology

Rough Sets-based Prototype Optimization in Kanerva-based Function Approximation

Cheng Wu
School of Urban Rail Transportation
Soochow University
Suzhou, China
e-mail: cwu@suda.edu.cn

Wei Li, Waleed Meleis
Department of Electrical and Computer Engineering
Northeastern University
Boston, U.S.A.
e-mail: li.wei@husky.neu.edu, meleis@ece.neu.edu

***Abstract*—Problems involving multi-agent systems can be complex and involve huge state-action spaces, making such problems difficult to solve. Function approximation schemes such as Kanerva coding with dynamic, frequency-based prototype selection can improve performance. However, selecting the number of prototypes is difficult and the approach often still gives poor performance. In this paper, we solve a collection of hard instances of the predator-prey pursuit problem and argue that poor performance is caused by inappropriate selection of the prototypes for Kanerva coding, including the number and allocation of these prototypes. We use rough sets theory to reformulate the selection of prototypes and their implementation in Kanerva coding. We introduce the equivalence class structure to explain how prototype collisions occur, use a reduct of the set of prototypes to eliminate unnecessary prototypes, and generate new prototypes to split the equivalence classes causing prototype collisions. The Rough Sets-based approach increases the fraction of predator-prey test instances solved by up to 24.5% over frequency-based Kanerva coding. We conclude that prototype optimization based on rough set theory can adaptively explore the optimal number of prototypes and greatly improve a Kanerva-based reinforcement learner's ability to solve large-scale multi-agent problems.**

***Keywords*-reinforcement learning; kanerva coding; rough sets; predator-prey pursuit problem**

I. Introduction

Reinforcement learning [1] enables learning from feedback received through interactions with an external environment with the objective of maximizing the cumulative reward. Q-learning, one of the most successful reinforcement learning strategies, works by combining state space exploration and exploitation to learn the value of each state-action pair. Through repeated trials, 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 [2].

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. Problems with large state or state-action spaces, such as multi-agent problems, can typically be difficult to solve.

Function approximation [3] can be used to store an approximation of this table. Kanerva coding [1], [4] is a function approximation technique that is particularly well-suited to problem domains with high dimensionality. In Kanerva coding, a set of prototypes, a subset of the state-action space, is used to represent the full state-action space. However, the performance of this approach is sensitive to the number of prototypes and prototype distribution [5].

To solve this feature optimization problem, we introduce rough sets-based prototype optimization that reformulates the process of selecting and modifying prototypes within Kanerva coding. This approach uses the structure of the equivalence classes induced by the indiscernibility relation on the prototypes to explain why prototype collisions occur. Our algorithm eliminates unnecessary prototypes by replacing an original prototype set with its reduct, and reduces prototype collisions by splitting the equivalence classes that contain two or more state-action pairs. We show that directly applying the theory of rough sets to adaptively prune the prototype set can improve the performance of the solver while reducing the number of prototypes needed. We show that the relative performance of traditional and frequency-based Kanerva coding can be understood by examining the structure of the equivalence classes induced by indiscernibility relations on the prototype sets. Finally, we show that the number of prototypes needed to achieve good performance is not related to the number of prototypes initially selected, and is a property of the problem itself. We will conclude that rough sets-based Kanerva-based function approximation can adaptively select an effective number of prototypes and greatly improve a Kanerva-based reinforcement learner's ability to solve large-scale multi-agent problems.

This paper is organized as follows. In Section II, we review related work. In Section III, we reformulate Kanerva coding using rough sets theory. In Section IV, we describe our rough set-based Kanerva-based function approximation and its experimental evaluation. In Section V, we discuss the effect of the number of initial prototypes on the performance of a reinforcement learner. We conclude the paper in Section VI.

978-1-4673-9618-9/15 $31.00 © 2015 IEEE
DOI 10.1109/WI-IAT.2015.179

II. Related Work

Most reinforcement learners use a tabular representation of value functions where the value of each state or each state-action pair is stored in a table. However, for many practical applications that have continuous state space, or very large and high-dimensional discrete state-action spaces, this approach is not feasible.

There are two explanations for this infeasibility. First, a tabular representation can only be used to solve tasks with a small number of states and actions. The difficulty derives both from the memory needed to store large tables, and the time and data needed to fill the tables accurately [6]. Second, most exact state-action pairs encountered may not have been previously encountered. Since there are often no state-action values that can be used to distinguish actions, the only way to learn in these problems is to generalize from previously encountered state-action pairs to pairs that have never been visited before. We must consider how to use a limited state-action subspace to approximate a large state-action space.

Function approximation has been widely used to solve reinforcement learning problems with large state-action spaces [7], [8], [9]. In general, function approximation defines an approximation method which interpolates the values of unvisited points in the search space using known values at neighboring points. Within a reinforcement learner, function approximation generalizes the function values of state-action pairs that have not been previously visited from known function values of neighboring state-action pairs.

A range of function approximation techniques has been studied in recent years. These techniques can be partitioned into three types: function approximation using natural features, function approximation using basis functions, and function approximation using Sparse Distribution Memory (SDM).

A. Function Approximation Using Natural Features

For each application domain, there are natural features that can describe a state. For example, in some pursuit problems in the grid world, we might have features for location, vision scale, memory size, communication mechanisms, etc. Choosing such natural features as the components of the feature vector is an important way to add prior knowledge to a function approximator.

In function approximation using natural features, the θ-value of a feature indicates whether the feature is present. The θ-value is constant across the features' receptive region and falls sharply to zero at the boundary. These receptive regions may be overlapped. A large region may give a wide but coarse generalization while a small region may give a narrow but fine generalization. The advantage of function approximation using natural features is that the representation of the approximate function is simple and easy to understand. The natural features can be selected manually and their receptive regions can be adjusted based on the designer's intuition. A limitation of this function approximation technique is that it cannot handle continuous state-action spaces or state-action spaces with high dimensionality.

A typical function approximation technique using natural features is Tile Coding [10]. This approach, which is an extension of coarse coding [7], is also known as "Cerebellar Model Articulator Controller", or CMAC [10]. In Tile Coding, k *tilings* are selected, each of which partitions the state-action space into tiles. The receptive field of each 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 sum of the θ-values of all adjacent tiles. In binary Tile Coding, which is 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 and each tile corresponds to an assignment of binary values to the selected bit positions.

B. Function Approximation Using Basis Functions

For certain problems, a more accurate approximation is obtained if θ-values can vary continuously and represent the degree to which a feature is present. A *basis function* can be used to compute such continuously varying θ-values. Basis functions can be designed manually and the approximate value function is a function of these basis functions. In this case, function approximation uses basis functions to evaluate the presence of every feature, then linearly weights these values.

An advantage of function approximation with basis function is that the approximated function is continuous and flexible. However there are two limitations of this function approximation technique. The first is that selecting these basis functions parameters is difficult in general [1], [8], [9]. The second difficulty is that basis functions cannot handle state-action spaces with high dimensionality [5], [11]. The number of basis functions needed to approximate a state-action space can be exponential in the number of dimensions, causing the number of basis functions needed to be very large for a state-action space with high dimensionality.

A typical function approximation technique using basis function is Radial Base Function Networks (RBFNs) [1]. In an RBFN, a sequence of Gaussian curves is selected as the basis functions of the features. Each basis function ϕ_i for a feature i has a center c_i, and width σ_i. Given an arbitrary state-action pair s, the Q-value of the state-action pair with respect to the feature i is:

$$\phi_i(s) = e^{-\frac{||s-c_i||^2}{2\sigma^2}}.$$

The total Q-value of the state-action pair with respect to all features is the sum of the values of $\phi_i(s)$ across all features. RBFNs are the natural generalization of coarse coding with binary features to continuous features. A typical RBF feature unavoidably represents information about some, but not all, dimensions of the state-action space. This limits RBFNs from approximating large-scale, high-dimensional state-action spaces efficiently.

C. Function Approximation Using SDM

Function approximation using Sparse Distributed Memory (SDM) uses a class of features that can construct approximation functions without restricting the dimensionality of

the state-action space. The features are typically a set of state-action pairs chosen from the entire state-action space. In function approximation using SDM [4], each receptive region is defined using a distance threshold with respect to the location of the feature in the state-action space. The θ-value of a state-action pair with respect to a feature is constant within the feature's receptive region, and is zero outside of this region.

An advantage of function approximation using SDM is that its structure is particularly well-suited to problem domains with high dimensionality. Its computational complexity depends entirely on the number of prototypes which is not a function of the number of the dimensions of the state-action space.

A limitation of this technique is that more prototypes are needed to approximate state-action spaces for complex problem domains, and the efficiency of function approximation using SDM is sensitive to the number of the prototypes [5]. Even when enough prototypes are used, studies have shown that the performance of the reinforcement learner with SDM is often poor and unstable [1], [9], [12]. There is no known mechanism to guarantee the convergence of the algorithm.

D. Kanerva coding

Kanerva coding [13] is the implementation of SDM in function approximation for reinforcement learning. Here, a collection of *prototype state-action pairs* (*prototypes*) is selected, each of which corresponds to a binary feature. A state-action pair and a prototype are said to be *adjacent* if their bitwise representations differ by no more than a threshold number of bits. A state-action pair is represented as a collection of binary features, each of which equals 1 if and only if the corresponding prototype is adjacent. A value θ is maintained for each prototype, and an approximation of the value of a state-action pair is then the sum of the θ-values of the adjacent prototypes. The computational complexity of Kanerva coding depends entirely on the number of prototypes which is not a function of the number of dimensions of the state-action space. Kanerva coding can therefore greatly reduce the size of the value table that needs to be stored. Traditional Kanerva-based function approximation is based on Sutton's work [1].

A limitation of this technique is that more prototypes are needed to approximate state-action spaces for complex problem domains, and the efficiency of function approximation is sensitive to the number and allocation of the prototypes [5]. If the number of prototypes is very large relative to the number of state-action pairs, and the prototypes are appropriately allocated throughout the state-action space, each prototype will be adjacent to a small number of state-action pairs. In this case, the approximate state-action values will tend to be close to the true values, and the reinforcement learner will operate as usual. However if the number of prototypes is small, or if the prototypes themselves are not well chosen, the approximate values will not be similar to the true values and the reinforcement learner will give poor results. Therefore, adaptively choosing prototypes appropriate to a particular application is an important way to improve the performance of a reinforcement learner.

In our previous work [14], a prototype's *visit frequency* is defined as the number of visits to the prototype during a learning process. We found that the distribution of visit frequencies across all prototypes is often non-uniform over a converged learning process, i.e., most prototypes are either frequently visited, or rarely visited [15]. Prototypes that are rarely visited do not contribute to the solution of instances. Similarly, prototypes that are visited very frequently are likely to decrease the distinguishability of state-action pairs. Prototype optimization uses these visit frequencies to delete rarely-visited prototypes and split heavily-visited prototypes to generate new prototypes. In this way, prototype optimization can produce a set of prototypes whose visit frequencies are relatively uniform and that are appropriate to the particular application.

Studies have shown that such frequency-based prototype selection that adaptively chooses prototypes appropriate to a particular application is an important way to contribute prior knowledge and experience to a reinforcement learner [15], and that dynamic allocation of prototypes can refine the representation of state-action value functions and increase the efficiency of function approximation [16], [17]. A limitation of frequency-based prototype optimization is that the approach does not adaptively select the optimal number of prototypes. There is therefore a need for algorithms to adaptively select the number of prototypes for a particular application.

III. Rough Sets-Based Kanerva coding

Instead of using visit frequencies to select prototypes for frequency-based prototype optimization, we focus on the structure of equivalence classes induced by the set of prototypes which is a key indicator of the efficiency of function approximation.

In traditional Kanerva coding, a set of state-action pairs is selected from the state-action space as prototypes. We assume P is the set of prototypes, Λ is the set of all possible state-action pairs in the state-action space, and SA is the set of state-action pairs encountered by the solver. For Kanerva-based function approximation, $P \subseteq \Lambda$ and $SA \subseteq \Lambda$. Our goal is to represent a set of observed state-action pairs SA using a set of prototypes P.

A. Rough Sets

The theory of rough sets [18], [19] is a formal approximation theory that has been widely applied to pattern classification and data analysis. Rough sets describe the indiscernibility of a target set based on a given set of features. The indiscernibility relation partitions the objects in a target set into a collection of equivalence classes. The objects in the same equivalence class are indiscernible based on the given features. In this way, a target set is expressed using the equivalence classes induced by features. Rough sets theory also describes whether some features are more relevant to the equivalence class structure than other features. This approach provides a

potential way to judge whether irrelevant features exist and whether the number of features is optimal.

Given a prototype p and a state-action pair sa, the function $f_p(sa)$ represents the adjacency between p and sa; that is, if sa is adjacent to p, $f_p(sa)$ is equal to 1, otherwise it equals 0. The set of adjacency values for a state-action pair with respect to all prototypes is referred to as the state-action pair's *prototype vector*. Given a prototype set P, we define an *indiscernibility relation*, $IND(P)$, that contains all pairs of state-actions pairs that are indiscernible with respect to prototypes in P. That is,

$$IND(P) = \{(sa_1, sa_2) \in \Lambda^2 | \forall p \in P, f_p(sa_1) = f_p(sa_2)\},$$

where p is a prototype in P and sa_1 and sa_2 are two encountered state-action pairs, $sa_1 \in SA$, $sa_2 \in SA$.

The set of all state-action pairs with the same indiscernibility relation is defined as an *equivalence class*, and the ith such equivalence class is denoted E_i^P. The set of prototypes P therefore partitions the set SA into a collection of equivalence classes, denoted $\{E^P\}$. Note that if $n = |P|$, there are at most $min\{2^n, |SA|\}$ equivalence classes.

For example, assume ten state-action pairs, (sa_1, sa_2, $sa_3, ...,$ sa_{10}), are encountered by a solver, and we have six prototypes, (p_1, p_2, $p_3, ...,$ p_6). We attempt to express each state-action pair using prototypes. Table I shows a sample of the adjacencies between state-action pairs and prototypes. When the prototypes are considered, we can induce the following equivalence classes, (${E^P}_1$, ${E^P}_2$, ${E^P}_3$, ${E^P}_4$, ${E^P}_5$, ${E^P}_6$, ${E^P}_7$,) = ($\{sa_1\}$, $\{sa_2\}$, $\{sa_3\}$, $\{sa_4,\ sa_5\}$, $\{sa_9\}$, $\{sa_6\}$, $\{sa_7, sa_8, sa_{10}\}$).

The structure of equivalence classes induced by the prototype set has a significant effect on function approximation. Kanerva coding works best when each state-action pair has a unique prototype vector. That is, the ideal set of equivalence classes induced by the prototype set should each include no more than one state-action pair. If two or more state-action pairs are in a same equivalence class, these state-action pairs are indiscernible with respect to the prototypes, causing a prototype collision [20].

B. Prototype Reduct-based Prototype Deletion

Given a set of prototypes, one or more prototypes may not affect the structure of the induced equivalence classes, and therefore do not help discern state-action pairs. These prototypes can be replaced with prototypes that are more useful. To do this, we use a *reduct of prototype set*. A reduct is a subset of prototypes $\mathrm{R} \subseteq P$ such that (1) $\{E^{\mathrm{R}}\} = \{E^P\}$, that is, the equivalence classes induced by the reduced prototype set R are the same as the equivalence class structure induced by the original prototype set P; and (2) R is minimal, that is $\{E^{(\mathrm{R}-\{p\})}\} \neq \{E^P\}$ for any prototype $p \in \mathrm{R}$. Thus, no prototype can be removed from the reduced prototype set R without changing the equivalence classes E^P.

In the above example, the subset (p_2, p_4, p_5) is a reduct of the original prototype set P. This can be shown easily because (1) the equivalence classes induced by (p_2, p_4, p_5) are the same as the equivalence class structure induced by the original prototype set P; and (2) eliminating any of these prototypes alters the equivalence class structure induced by the original prototype set P. Replacing a set of prototypes with its reduct eliminates unnecessary prototypes. Note that the reduct of a prototype set is not necessarily unique, and there may be many subsets of prototypes which preserve the equivalence-class structure.

Since a prototype reduct maintains the equivalence class structure, prototype deletion can be conducted by replacing the set of prototypes with a reduct of the original prototype set. The following algorithm finds a reduct of the original prototype set. We consider each prototype in P one by one in random order. For prototype $p \in P$, if the set of equivalence classes $\{E^{P-\{p\}}\}$ induced by $P - \{p\}$ is not identical to the equivalence classes $\{E^P\}$ induced by P, that is, $\{E^{P-\{p\}}\} \neq \{E^P\}$, then p is in the reduct R of original prototype set P, $p \in R$, otherwise, p is not in the reduct R, $p \notin R$. We then delete p from the prototype set P and consider the next prototype. The final set R is a reduct of the original prototype set P.

Note that frequency-based prototype optimization can also eliminate unnecessary prototypes by deleting rarely-visited prototypes, but that approach cannot eliminate prototypes that are heavily-visited. For example, prototype p_6 in the above example is adjacent to all state-action pairs, but it can be safely removed since deleting the prototype does not change the structure of the equivalence classes.

C. Equivalence Classes-based Prototype Generation

Prototype collisions occur only when two state-action pairs are in a same equivalence class. If the number of equivalence classes that contain only one state-action pair increases, prototype collisions are less likely and the performance of a reinforcement learner based on Kanerva coding should increase. Prototype generation should therefore attempt to reduce the fraction of equivalence classes that contain two or more state-action pairs.

An equivalence class that contains two or more state-action pairs is likely to be split up if a new prototype equal to one of those state-action pairs is added. We implement prototype generation by adding new prototypes that can split equivalence classes containing two or more state-action pairs. For an

TABLE I
SAMPLE OF ADJACENCY BETWEEN STATE-ACTION PAIRS AND PROTOTYPES

	p_1	p_2	p_3	p_4	p_5	p_6
sa_1	0	0	0	0	1	1
sa_2	1	0	0	1	1	1
sa_3	1	1	0	0	0	1
sa_4	0	1	0	0	1	1
sa_5	0	1	0	0	1	1
sa_6	1	1	0	1	0	1
sa_7	0	0	0	0	0	1
sa_8	0	0	0	0	0	1
sa_9	0	1	1	1	1	1
sa_{10}	0	0	0	0	0	1

TABLE II
PSEUDO CODE FOR Q-LEARNING WITH ROUGH SETS-BASED KANERVA CODING

Initialization()
Select random prototypes $\vec{p}$
Initialize corresponding $\vec{\theta}$ values to 0

Q-with-Kanerva(s, $\vec{p}$, $\vec{\theta}$)
Repeat
  With probability $1 - \varepsilon$
    $a* = argmax_a Q(s, a)$ where $Q(s, a) = \sum \vec{\theta}$;
  else
    $a* =$ random action
  Take action $a*$, observe reward r, and get next state s'
  For all actions a under new state s'
    $Q(s', a) = \sum \vec{\theta}$;
  $\delta = r + \gamma * max_a Q(s', a) - Q(s, a)$
  $\Delta\vec{\theta} = \alpha_t * \delta$
  $\vec{\theta} = \vec{\theta} + \Delta\vec{\theta}$
Until s is terminal

Update-prototypes($\vec{p}$, $\vec{\theta}$)
$E(\vec{p}) =$ equivalence classes induced by $\vec{p}$
For all prototypes $p \in \vec{p}$
  $\hat{\vec{p}} = \vec{p} - p$
  $E(\hat{\vec{p}}) =$ equivalence classes induced by $\hat{\vec{p}}$
  If $\{E(\hat{\vec{p}}) = E(\vec{p})\}$
    $\vec{p} = \hat{\vec{p}}$
For each equivalence class $k \in E(\vec{p})$
  $n = size(k)$
  If $\{n > 1\}$
    For j = 1 to $\lceil log(n) \rceil$
      Randomly select a state-action pair sa
      $\vec{p} = \vec{p} \bigcup \{sa\}$

arbitrary equivalence class that contains $n > 1$ state-action pairs, we randomly select $\lceil log_2 n \rceil$ state-action pairs to be new prototypes. Note that $\lceil log_2 n \rceil$ is the minimal number of prototypes needed to distinguish all state-action pairs in an equivalence class that contains n elements. While this algorithm does not guarantee that each equivalence class will be split into new classes that contain exactly one state-action pair, in general this approach reduces the prototype collisions.

D. Rough sets-based Kanerva coding Algorithm

Table II describes our algorithm for implementing Q-learning with rough sets-based Kanerva coding. A vector $\vec{p}$ of prototypes is selected randomly, and a corresponding vector of theta values $\vec{\theta}$ is initialized to zero.

For each state s and each action a, the algorithm calculates an update to its expected discounted reward, $Q(s, a)$ as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha_t [r + \gamma \max_a Q(s', a) - Q(s, a)]$$

where r is an immediate reward, s' is the next state, α_t is the learning rate such that $0 \leq \alpha_t \leq 1$, and γ is the discount factor such that $0 \leq \gamma < 1$. An ϵ-greedy approach is used to select the next action to take. For some value of ϵ, $0 \leq \epsilon \leq 1$, a random action is selected with probability ϵ, and an action that gives the largest Q-value for state s is selected with probability $1 - \epsilon$.

The algorithm begins by initializing parameters, and repeatedly executes Q-learning with rough sets-based Kanerva coding. Prototypes are adaptively updated periodically. In each update period, the encountered state-action pairs are recorded. To update prototypes, the algorithm first determines the structure of the equivalence classes of the set of the encountered state-action pairs with respect to the original prototypes. Unnecessary prototypes are then deleted by replacing the original prototype set with its reduct. In order to split large equivalence classes, new prototypes are randomly selected from these equivalence classes. The optimized prototype set is constructed by adding newly generated prototypes to the reduct of original prototype set.

IV. EXPERIMENTAL EVALUATION

We evaluate our algorithms by solving a collection of instances in the predator-prey pursuit domain [21]. In our experiments, pursuit takes place on an $n \times n$ rectangular grid with open and closed blocks. The n closed blocks are distributed randomly. Each open cell in the grid represents a state that an agent may occupy. Two agents, a predator agent and a prey agent, are placed in two randomly selected starting cells. 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, vertical or diagonal step from its current location, or it can remain in its current cell. The predator agent receives a positive reward if it moves closer to the prey agent and a negative reward if it moves further from the prey agent. The prey agent moves randomly.

An epoch is a trial in which each learning algorithm is applied to 40 random training instances followed by 40 random test instances. To ensure that the space of initial conditions is uniformly evaluated, the learning algorithms apply one move to each training instance before considering the next training instance. The exploration rate ϵ is set to 0.3, which we found experimentally gives 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. This multiplicative decreasing learning rate is widely used, and under certain conditions these values guarantee the convergence of the learning process [1], [2].

A state-action pair consists of a sequence of bits that represent the 9 possible actions and the relative position of the prey agent relative to the predator agent along both axes. A prototype and a state-action pair are adjacent if their binary representations differ in no more than one bit position. The number of bits needed to represent a state-action pair is $a + 2 \times b$, where a is the number of bits needed to represent an action, 4 in our experiments, and b is the number of bits needed to represent the relative horizontal or vertical position of the prey agent relative to the predator agent within the grid, $\lceil log_2 n \rceil + 1$ in our experiments. The number of entries stored

Fig. 1. The fraction of test instances solved by traditional and adaptive Kanerva-based function approximation with 2000 prototypes.

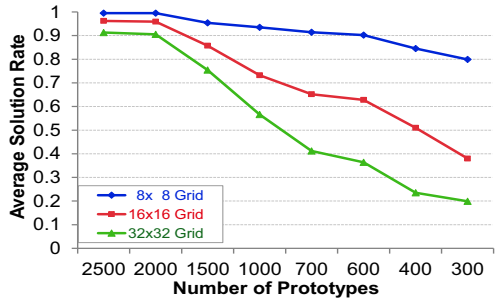


Fig. 2. The fraction of test instances solved using frequency-based prototype optimization as the number of prototypes decreases.

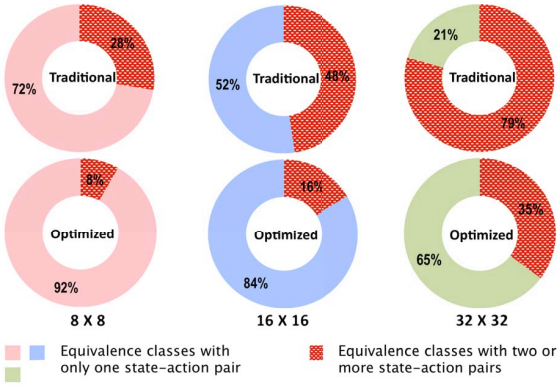


Fig. 3. Equivalence classes using traditional and optimized Kanerva-based function approximation with 2000 prototypes.

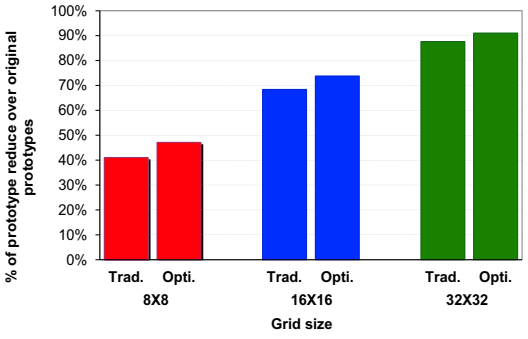


Fig. 4. Prototype Reduct using traditional and optimized Kanerva-based function approximation with 2000 prototypes.

in the table of Q values without using function approximation is $9\times(2n-1)^2$.

For every epoch, we record the average fraction of test instances solved during this epoch within $2n$ moves. This cutoff value was selected because it allowed us to distinguish the performance of our algorithms from one another. For every 20 epochs, we update the prototypes using frequency-based or rough sets-based prototype optimization. Each experiment is performed 5 times and we report the mean of the recorded values. Where error bars are shown, they indicate standard deviations. In our experiments, all runs were found to converge within 2000 epochs.

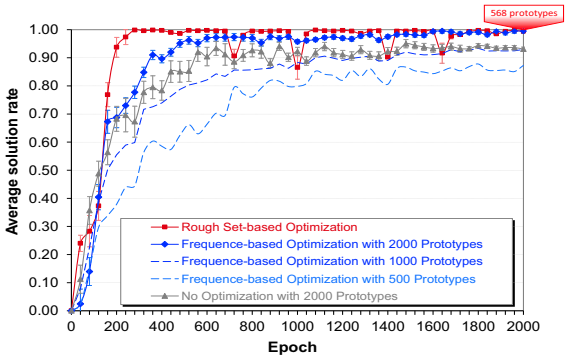
We first use the theory of rough sets to explain the relative performance of traditional and frequency-based Kanerva coding. Fig. 1 shows the average fraction of test instances solved after 2000 epochs by traditional and frequency-based Kanerva-based function approximation with 2000 prototypes, as the size of the grid varies from 8 to 32. The results indicate that frequency-based prototype optimization increases the fraction of test instances solved using dynamic prototype allocation.

The effectiveness of Kanerva-based function approximation has a strong dependence on both the distribution and number of prototypes. Fig. 2 shows the average fraction of test instances solved using frequency-based prototype optimization as the number of prototypes varies from 300 to 2500. The results show that when the number of prototypes decreases, the fraction of test instances solved decreases from 99.5% to 79.9% in the 8x8 grid, from 96.2% to 38.8% in the 16x16 grid, and from 91.3% to 19.6% in the 32×32 grid. This indicates that the efficiency of Kanerva-based function approximation does decrease as the number of prototypes decreases.

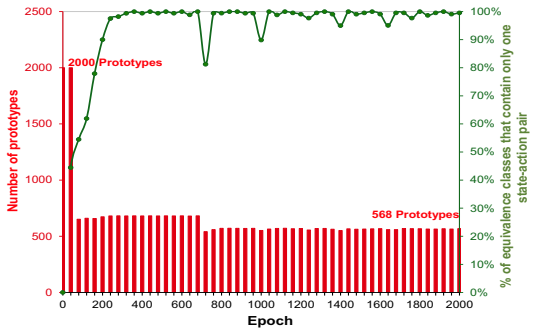
We now use the theory of rough sets to understand these results. We evaluate the structure of equivalence classes and the reduct of prototypes in traditional Kanerva coding and Kanerva coding with frequency-based prototype optimization. We apply traditional Kanerva and optimized Kanerva with 2000 prototypes to the same instances described above.

Fig. 3 shows the fraction of equivalence classes that contain two or more state-action pairs over all equivalence classes. These results in Fig. 1 and Fig. 3 show that as the fraction of equivalence classes that contain two or more state-action pairs increases, the performance of each algorithm decreases. For example, the average solution rate for the traditional algorithm decreases from 93.1% to 40.6% while the fraction of equivalence classes that contain two or more state-action pairs increases from 28% to 79% as the size of the grid increases. The average solution rate for the optimized algorithm decreases from 99.5% to 90.5% while the fraction of equivalence classes that contain two or more state-action pairs increases from 8% to 35% as the size of the grid increases.

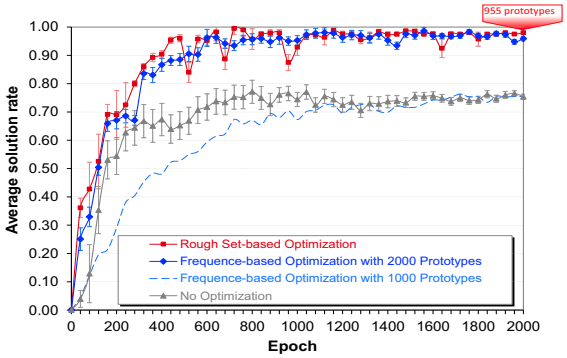
The results also demonstrate that improved performance of the optimized Kanerva algorithm over the traditional algorithm corresponds to a reduction in the fraction of equivalence classes that contain two or more state-action pairs. The op-



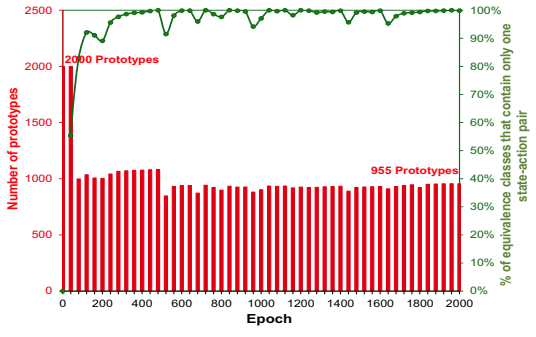
(a) Average solution rate for traditional Kanerva, frequency-based Kanerva and Rough sets-based Kanerva in 8x8 grid.



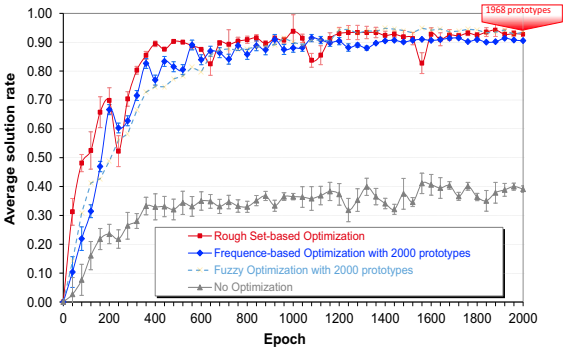
(d) Effect of Rough sets-based Kanerva on the number of prototypes and the fraction of equivalence classes in 8x8 grid that contain one state-action pair.



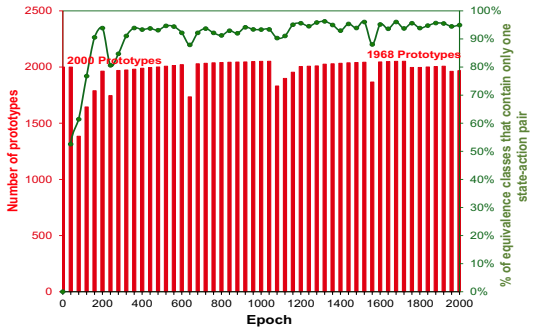
(b) Average solution rate for traditional Kanerva, frequency-based Kanerva and Rough sets-based Kanerva in 16x16 grid.



(e) Effect of Rough sets-based Kanerva on the number of prototypes and the fraction of equivalence classes in 16x16 grid that contain one state-action pair.



(c) Average solution rate for traditional Kanerva, frequency-based Kanerva, fuzzy Kanerva and Rough sets-based Kanerva in 32x32 grid.



(f) Effect of Rough sets-based Kanerva on the number of prototypes and the fraction of equivalence classes in 32x32 grid that contain one state-action pair.

Fig. 5. Average solution rates for different algorithms and effect of Rough sets-based Kanerva on number of prototypes and fraction of equivalence classes that contain one state-action pair

timized algorithm reduces the fraction of equivalence classes that contain two or more state-action pairs from 79% to 35% while the average solution rate for the optimized algorithm increases from 40.6% to 90.5% for a grid size of 32×32.

We evaluate the performance of rough sets-based Kanerva coding by using it to solve pursuit instances on grids of varying sizes. As a comparison, traditional Kanerva coding and frequency-based Kanerva coding with different number of prototypes are also implemented to solve the same instances. When rough sets-based Kanerva coding is implemented during a learning process, we also observe the change in the number of prototypes and the fraction of equivalence classes that contain only one state-action pair.

Fig. 4 shows the fraction of the original prototypes in the reduct. These results show that the structure of the equivalence classes can be maintained while using fewer prototypes. For example, the equivalence classes induced by 1821 prototypes for the optimized algorithm are the same as the equivalence

classes induced by 2000 prototypes for a grid size of 32.

Fig. 5(a)-5(c) show the average fraction of test instances solved when learning algorithms are applied to instances with varying grid sizes. The figures compare traditional Kanerva, frequency-based Kanerva, and rough sets-based Kanerva. The results show that the rough sets-based algorithm increases the fraction of test instances solved over the frequency-based Kanerva algorithm. After 2000 epochs, using the rough sets-based algorithm with 568 prototypes with the 8×8 grid increases the fraction of test instances solved over the frequency-based algorithm with 500 prototypes and 1000 prototypes from 87.3% and 92.9% to 99.3%. Fig. 5(b) shows that in the 16×16 grid, after 2000 epochs, using the rough sets-based algorithm with 955 prototypes increases the fraction of test instances solved over the frequency-based algorithm with 1000 prototypes and 2000 prototypes from 73.5% and 95.5% to 98.0%. Fig. 5(c) shows that in the 32×32 grid, after 2000 epochs, using the rough sets-based algorithm with 1968 prototypes increases the fraction of test instances solved over the frequency-based algorithm with 2000 prototypes from 90.5% to 92.7%.

The results in the grids of varying sizes indicate that rough sets-based Kanerva coding uses fewer prototypes and achieves higher performance. More importantly, rough sets-based Kanerva coding enables adaptive selection of the optimal number of prototypes for a particular application.

Fig. 5(d)-5(f) show the effect of our rough sets-based Kanerva coding on the number of prototypes and the corresponding change in the fraction of equivalence classes that contain only one state-action for a range of varying grid sizes. In Fig. 5(d)-5(f), the bars correspond to the axis on the left-hand side, and the continuous line corresponds to the axis on the right-hand side. Fig. 5(d) shows that the rough sets-based algorithm reduces the number of prototypes from 2000 prototypes to 568 prototypes and increases the fraction of equivalence classes with only one state-action pair from 44.3% to 99.5% when applied to instances of size 8×8. Fig. 5(e) shows that the rough sets-based algorithm reduces the number of prototypes from 2000 prototypes to 955 prototypes and increases the fraction of equivalence classes with only one state-action pair from 53.3% to 99.8% when applied to instances of size 16×16. Fig. 5(f) shows that the Rough sets-based algorithm reduces the number of prototypes from 2000 prototypes to 1968 prototypes and increases the fraction of equivalence classes with only one state-action pair from 52.5% to 94.9% when applied to instances of size 32×32.

These results also demonstrate that rough sets-based Kanerva coding can adaptively explore the optimal number of prototypes and dynamically allocate prototypes for the optimal structure of equivalence classes in a particular application.

Finally, we investigate the effect of rough sets-based Kanerva coding when varying the initial number of prototypes. We use our rough sets-based algorithm with 0, 250, 500, 1000, 1500 and 2000 initial prototypes to solve pursuit instances in the 16×16 grid. Fig. 6 shows the effect of our algorithm on the number of prototypes. The results show that the number

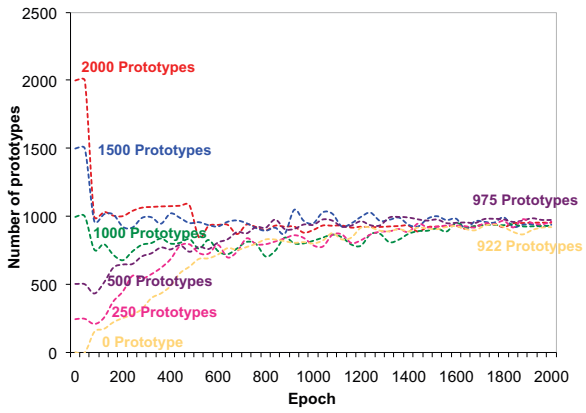


Fig. 6. Variation in the number of prototypes with different numbers of initial prototypes with rough sets-based Kanerva in a 16x16 grid.

of prototypes tends to converge to a fixed number in the range from 922 to 975 after 2000 epochs. The results demonstrate that our rough sets-based Kanerva coding has the ability to adaptively determine an effective number of prototypes during a learning process.

V. Discussion

In Kanerva-based function approximation, the accuracy of the function approximation is sensitive to the number of prototypes. In general, more prototypes are needed to approximate the state-action space for more complex applications. On the other hand, the computational complexity of Kanerva coding also depends entirely on the number of prototypes, and larger sets of prototypes can more accurately approximate more complex spaces. Neither traditional Kanerva coding nor frequency-based Kanerva coding can adaptively select the number of prototypes. Therefore, the number of prototypes has a significant effect on the efficiency of traditional and frequency-based Kanerva coding. If the number of prototypes is too large relative to the number of state-action pairs, the implementation of Kanerva coding is unnecessarily time-consuming. If the number of prototypes is too small, even if the prototypes are well chosen, the approximate values will not be similar to the true values and the reinforcement learner will give poor results. Selecting the optimal number of prototypes is difficult for traditional and frequency-based Kanerva coding, and in most known applications of these algorithms the number of prototypes is selected manually [17].

VI. Conclusion

Kanerva coding can be used to improve the performance of function approximation within reinforcement learners. This approach often gives poor performance when applied to large-scale multi-agent systems. In general, it is difficult to determine the optimum number of prototypes. In this paper, we showed that rough sets can be used to understand and improve the performance of a reinforcement learning algorithm.

First, we showed that the relative performance of traditional and frequency-based Kanerva coding can be understood by examining the structure of the equivalence classes induced by indiscernibility relations on the prototype sets. Next, we developed a new rough sets-based approach to Kanerva coding that reformulates the set of prototypes and their implementation in Kanerva coding. This approach uses the structure of the equivalent classes induced by the indiscernibility relation on the prototypes to explain why prototype collisions occur. Our algorithm eliminates unnecessary prototypes by replacing an original prototype set with its reduct, and reduces prototype collisions by splitting the equivalence classes with two or more state-action pairs. We showed that directly applying the theory of rough sets to adaptively prune the prototype set can improve the performance of the solver while reducing the number of prototypes needed. Finally, we showed that the number of prototypes needed to achieve good performance is not related to the number of prototypes initially selected, and is a property of the problem itself.

We conclude that rough sets-based Kanerva-based function approximation can adaptively select an effective number of prototypes and greatly improve a Kanerva-based reinforcement learner's ability to solve large-scale multi-agent problems.

REFERENCES

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

[2] C. Watkins, "Learning from delayed rewards," *Ph.D thesis, Cambridge Univeristy, Cambridge, England*, 1989.

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

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

[5] P. W. Keller, S. Mannor, and D. Precup, "Automatic basis function construction for approximate dynamic programming and reinforcement learning," in *Proc. of Intl. Conf. on Machine Learning*, 2006.

[6] R. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding." in *Proc. of Conf. on Neural Information Processing Systems*, 1995.

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

[8] G. J. Gordon, "Stable function approximation in dynamic programming," in *Proc. of Intl. Conf. on Machine Learning*, 1995.

[9] 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.

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

[11] R. Munos and A. Moore, "Variable resolution discretization in optimal control." *Machine Learning*, 2002.

[12] K. Kostiadis and H. Hu, "Kabage-rl: kanerva-based generalisation and reinforcement learning for possession football," in *Proc. of Intl. Conf. on Intelligent Robots and Systems*, 2001.

[13] P. Kanerva, *Sparse distributed memory and related models.*, M. Hassoun, Ed., 1993.

[14] C. Wu and W. Meleis, "Fuzzy kanerva-based function approximation for reinforcement learning." in *Proc. Of 8th Intl. Conf. on Autonomous Agents and Multiagent Systens (AAMAS)*, 2009.

[15] ——, "Function approximations using tile and kanerva coding for multi-agent systems." in *Proc. Of Adaptive Learning Agents Workshop (ALA) in AAMAS*, 2009.

[16] ——, "Adaptive kanerva-based function approximation for multi-agent systems." in *Proc. of 7th Intl. Conf. on Autonomous Agents and Multi-agent Systems (AAMAS)*, 2008.

[17] M. Allen and P. Fritzsche, "Reinforcement learning with adaptive kanerva coding for xpilot game AI," in *Evolutionary Computation (CEC), 2011 IEEE Congress on*. IEEE, 2011, pp. 1521–1528.

[18] Z. Pawlak, S. K. M. Wong, and W. Ziarko, "Rough sets: Probabilistic versus deterministic approach." in *International Journal of Man-Machine Studies 29: 8195*, 1988.

[19] S. Greco, B. Matarazzo, and R. Slowinski, "Rough sets theory for multicriteria decision analysis," *European journal of operational research*, vol. 129, no. 1, pp. 1–47, 2001.

[20] C. Wu and W. Meleis, "Adaptive fuzzy function approximation for multi-agent reinforcement learning." in *Proceedings of IEEE/WIC/ACM Intl. Conf. on Intelligent Agent Technology (IAT), Milan, Italy*, 2009.

[21] 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.