Mulitagent Reinforcement Learning in Stochastic Games with Continuous Action Spaces

Albert Xin Jiang
Department of Computer Science, University of British Columbia
jiang@cs.ubc.ca

April 27, 2004

Abstract

We investigate the learning problem in stochastic games with continuous action spaces. We focus on repeated normal form games, and discuss issues in modelling mixed strategies and adapting learning algorithms in finite-action games to the continuous-action domain. We applied variable resolution techniques to two simple multi-agent reinforcement learning algorithms PHC and MinimaxQ. Preliminary experiments shows that our variable resolution partitioning method is successful at identifying important regions of the action space while keeping the total number of partitions low.

1 Introduction

Stochastic games (SGs) has been used as a framework to study the *multiagent learning problem*, where an agent tries to learn a policy in the presence of other agents. Stochastic games are a natural extension of Markov decision processes (MDPs) to the multiagent scenario. Reinforcement learning (RL) [KLM96] has been successful at finding optimal policies in MDPs, and has been used as basis for learning in stochastic games.

Researchers has proposed quite a few learning algorithms for SGs, for example MinimaxQ [Lit94], PHC [BV01], IGA [SKM00] and HyperQ [Tes03]. These algorithms all assume that the set of actions available to agents is finite. In this paper, we investigate the problem of learning in SGs with continuous action spaces. In many multiagent domains such as Robot Soccer and some scenarios in economics, agents need to choose from a continuum of actions.

Compared with SGs with finite action sets, continuous action SGs are much more complex because agents have infinite number of choices. But in most of domains involving continuous actions, similar actions tend to produce similar effects. This allows us to use approximation techniques: actions that generate similar effects can be grouped together as one action. This also allows us to

generalize when learning: information we have learned about one action can be generalized to similar actions.

In this paper, we concentrate our attention to a special class of SGs: repeated normal form games. We discuss methods to model mixed strategies, and highlight issues in adapting RL algorithms for finite-action stochastic games to the continuous-action domain. We then adapted two simple RL algorithms (PHC and MinimaxQ) using one of our proposed variable-resolution techniques. In Section 2 we introduce the theory on normal form games of finite and continuous action spaces. Section 3 defines stochastic games and repeated games. Section 4 discusses issues on designing a reinforcement learning algorithm for repeated games, and describes our modified PHC and MinimaxQ algorithms. Section 5 describes our experiments and results. Section 6 discusses future work and gives concluding remarks.

2 Normal Form Games

Normal form games has been well studied in non-cooperative game theory. A normal form game is a tuple (n, A, U), where n is the number of agents, $A = A_1 \times \cdots \times A_n$ and A_i is the set of actions available to agent i, $U = U_1 \times \cdots \times U_n$ and $U_i : A \rightarrow \mathbb{R}$ is the payoff to agent i. Actions $a_i \in A_i$ are also called pure strategies. $a = (a_1 \ldots a_n) \in A$ is called a pure strategy profile. Let $a_{-i} \in A_{-i}$ denote the tuple of actions of agents other than i. Then we can write a as the tuple (a_i, a_{-i}). A two-person game is called *zero-sum* if $U_1(a) + U_2(a) = 0$ for all a.

If A_i are finite sets, a *mixed strategy* μ_i of player i assigns a probability to each pure strategies of agent i. A mixed strategy profile $\mu = (\mu_1 \ldots \mu_n)$ specifies a mixed strategy for each player. μ can also be written as (μ_i, μ_{-i}), where μ_{-i} denote the tuple of mixed strategies of agents other than i. Given a mixed strategy profile μ, the expected payoff for player i is

$$E_i(\mu) = \sum_{a=(a_1 \ldots a_n) \in A} U_i(a) \prod_{j=1}^{n} \mu_j(a_j) \tag{1}$$

where $\mu_j(a_j)$ denotes the probability assigned to a_j by j's mixed strategy.

A mixed strategy μ_i is a *best response* to μ_{-i} if $E_i(\mu_i, \mu_{-i}) \geq E_i(\mu_i', \mu_{-i}), \forall \mu_i'$. μ is a Nash equilibrium if each μ_i in μ is a best response to the other agents' mixed strategies. All finite games have at least one Nash equilibrium [Nas50].

For normal form games with continuous action space, $A_i \subseteq R^m$. We generally make the assumption that A_i be non-empty and compact[1]. A mixed strategy μ_i is then a probability distribution on A_i. Analogously we define expected payoff to be $E_i(\mu) = \int U_i(a) d(\prod_{j=1}^{n} \mu_j(a_j))$. Best response and Nash equilibrium are defined similarly.

The existence of Nash equilibrium in continuous action normal form games depends on the form of the payoff function $U_i(a)$. Glicksberg proved that if U_i

[1] compactness in R^m means that the set is closed and bounded

are continuous, there exists a mixed strategy Nash equilibrium [Gli52]. Certain classes of games with some discontinuities in payoff functions also have Nash equilibria. These equilibria can be approximated by the equilibria of a series of finite games [DM86, Sim87]. However, there exist games with simple piecewise-continuous payoffs that don't have equilibrium. Section 5.1.3 has a simple example of such games.

3 Stochastic Games and Repeated Normal Form Games

A stochastic game is a tuple (n, S, A, T, R), where n is the number of agents, S is a set of states, A_i is a set of actions available to i, T is a transition function $S \times A \times S \to [0, 1]$, and R_i is a reward function: $S \times A \to \mathbb{R}$. The goal of the game is to maximize the discounted future reward with discount factor γ.

SGs are combinations of normal form games and MDPs. In each state s, agents play a normal form game with payoff $R_i(s, a)$, and the transition probabilities for the next state is determined by their joint actions.

In general, a mixed strategy in a SG may depend on the entire history of the game. Formally, $\mu_i : (S \times A)^N \to D(A_i)$, where N is the set of natural numbers and $D(A_i)$ is the set of probability distributions on A_i. A *stationary* policy is a special kind of mixed strategy that only depends on the current state: $\rho_i : S \to D(A_i)$.

Just as in normal form games, we can define best response for i given μ_{-i} to be a mixed strategy that achieves maximum payoff against μ_{-i}, and Nash equilibrium to be a mixed strategy profile in which each player is playing her best response.

Repeated normal form games are a special class of SGs. In a repeated game, the same normal form game (called the *stage game*) is played repeatedly. This is equivalent to an SG with only one state and with the state transition function pointing to itself. A stationary policy profile that plays a Nash equilibrium strategy profile in the normal form game, is also a Nash equilibrium in the repeated game. However, there exist Nash equilibria of the repeated game that consist of non-stationary strategies.

4 Reinforcement Learning in Repeated Normal Form Games

4.1 Problem Statement

In this paper, we investigate the learning problem in repeated games with 1-dimensional continuous action space $A_i = [0, 1]$. We assume the payoffs R_i are bounded and piecewise-continuous[2]. In other words, the set of discontinuities

[2]This does not guarantee the existence of Nash equilibrium in the normal form game. See Section 5.1.3.

is of measure zero. This is generally a realistic assumption. We control an agent, in an environment of other agents. What we can observe depends on the particular problem. We should at least be able to observe all actions and our own payoff after each stage game.

The goal of the learning problem is to maximize our agent's payoff. Most previous approaches to multi-agent reinforcement learning also require the algorithm to converge to Nash equilibrium in self-play. We agree with [SPG03] that convergence to Nash equilibrium is not important in the multiagent learning problem on SGs. Although Nash equilibrium is a central concept in game theory, in the learning problem we do not assume agents to be perfectly rational. The other agents may be learning, or may be just playing some unknown mixed strategy. Our goal is to try to play the best response to other agents' mixed strategies.

A successful learning algorithm should achieve the following:

- It should be able to adapt to the other agents' changing strategies.
- It should be able to play a mixed strategy.
- If the other agents converge to a stationary policy μ_{-i}, our agent's strategy should converge to the best response to μ_{-i}. This is called Hannan consistency in game theory literature.

4.2 Overall Approach

A (stationary) mixed strategy is a probability distribution, which is a function on $[0, 1]$. This can be viewed as a vector of infinite dimensions. To make the storage and computation of mixed strategies possible, we have to model the space of probability distributions on $[0, 1]$ using a parameterized function $\mu_i(a_i|\theta_i)$, where θ_i is a vector of finite dimension. We observe that θ_i is similar to the mixed strategy probabilities in finite-action games: both are finite-dimensional vectors that determines a player's mixed strategy. This means that we can try to adapt existing RL algorithms for finite-action SGs that explicitly model mixed strategies, and use them to use θ_i instead. Some examples of such algorithms are MinimaxQ [Lit94], PHC [BV01], IGA [SKM00] and HyperQ [Tes03].

4.3 Modelling Mixed Strategies

Mixed strategies are used in at least two aspects of an RL algorithm:

1. The RL algorithm may explicitly model other agents' mixed strategies, based on observations of their actions. An adequate mixed strategy model in this context has to support efficient incremental updates based on new data.
2. In each iteration of the normal form game, the RL algorithm has to compute our agent's mixed strategy, and choose an action by sampling from

this distribution. Thus our model of mixed strategy should be computationally easy to sample from.

Following are some possible ways to model mixed strategies. The best model may depend on the payoff function and the RL algorithm used.

1. **Rectangles**.
   This is a direct partitioning of domain $[0, 1]$ into intervals. Within each interval, the probability density is constant. The probability density function is shaped like combinations of rectangles. This essentially discretizes the continuous-action normal form game into a finite-action game. Sampling from the distributions is straightforward.

   There are several ways of partitioning $[0, 1]$. The simplest way is to partition it into k equal-sized intervals. θ_i would be a k-dimensional vector of probability densities in each interval. This is very easy to implement, and then most RL algorithms can be applied directly without any change.

   However, this partition scheme is very inefficient: this gives all parts of the action space equal resolution, while actually some parts of action space may require more resolution that others. We want the dimension of θ_i as small as possible, because higher dimensional vectors make the computation in the RL algorithms much more difficult. Therefore, we really want a partition that can have variable resolution.

   A possible solution is to maintain the partitions as leaves of a kd-tree. When we require more resolution at an interval, we bisect the interval and make the new intervals its left and right child. When we want less resolution in some areas, we can merge some intervals to their parents. This is similar to the approach used in Moore's Parti-Game algorithm [Moo94] for partitioning state space in a single-agent environment.

2. **Mixture of Gaussians**.
   If the payoff function is smooth, it makes sense to have mixed strategies that are smooth. Let $N(u, \sigma)$ represent a Normal distribution with mean u and standard deviation σ. We can represent a mixed strategy as a mixture of Gaussians: $\sum_c b_c N(u_c, \sigma_c)$. It is not easy to normalize the distribution in $[0, 1]$, but we can sample from it without normalization. The locations and widths (i.e. u_c and σ_c) of the Gaussians specify a soft partition of the domain. θ_i corresponds to the vector of coefficients b_c. When we want more resolution in a certain area, we can add a Gaussian to the mixture.

4.4 Adapting Reinforcement Learning Algorithms: General Issues

If we are using kd-trees or mixture of Gaussians to model mixed strategies, we will have to modify the existing RL algorithms to deal with these data structures.

Our RL algorithm also need to carry out an additional task: to decide which parts of the action space need more resolution. If we are modelling our opponents' strategy, an simple approach would be: increase the resolution at an

interval when the observed frequency distribution of actions within this interval is significantly non-uniform. If we are trying to compute our own mixed strategy, the criteria for increasing / decreasing resolution may depend on the details of the particular RL algorithm. A general heuristic could be: if by increasing the resolution, our computed best strategy is going to be significantly different from our strategy at current resolution, we should then increase the resolution. Similar heuristics can be built for deciding when and where to decrease resolution: if decreasing resolution is unlikely to change our best strategy and our expected payoff, then it is safe to do so.

We took two simple multi-agent RL algorithms, PHC and MinimaxQ, and modified them to use variable-resolution partitioning of action space with kd-trees. Sections 4.5 and 4.6 describes our modified algorithms.

4.5 Variable Resolution PHC

Policy Hill-Climbing (PHC) [BV01] is a simple algorithm based on single-agent Q-learning. Single-agent Q-learning learns a Q function $Q(s, a)$, which is the discounted future reward of playing action a at state s, then playing optimally afterwards.

Suppose our agent is at state s. We should choose action a that maximizes $Q(s, a)$. After we have played the action a, if we observe reward r and next state s', the Q function is updated as follows:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a')) \tag{2}$$

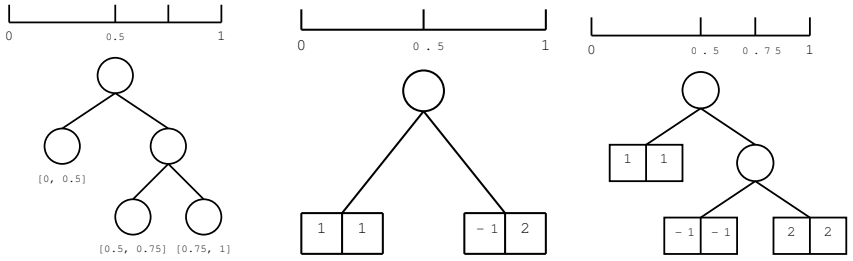
where $\alpha \in (0, 1]$ is a learning rate, and γ is the discount factor.

PHC is a simple extension of Q-learning to play mixed strategies. In addition to storing the Q values, the algorithm also maintains the current mixed strategy. After each Q function update as in Equation 2, the mixed strategy is improved by increasing the probability that it selects the action with the highest Q value. For finite-action SGs, PHC is guaranteed to converge to the best response if the other players are playing stationary strategies, given a suitable exploration policy.

Now we modify this algorithm to use the variable-resolution partitioning described in Section 4.3. In the 1-dimensional domain $[0, 1]$, a kd-tree is a binary tree whose leaves are non-overlapping intervals of $[0, 1]$. The leaves of the binary tree specify a partitioning of $[0, 1]$. See Figure 1(a). We can thus specify a mixed strategy by specifying the probability mass of each leaf.

In a repeated game, there is only one state, so $Q(s, a)$ becomes $Q(a)$. Since our action space is continuous, we define the Q values to be the expected future payoffs on intervals of the action space, rather than on particular actions. The Q values can be conveniently stored at the leaves of the binary tree.

Now we need to specify when to split an interval and when to merge two intervals. Recalling our heuristic from Section 4.4, and observing that in PHC our next mixed strategy is determined by the action that has the maximal Q value, we specify the following rule for splitting an interval: if by splitting an



(a) Top: a partitioning of the action space. Bottom: the corresponding tree.

(b) The right half of the right interval has a higher Q value than the current maximum

(c) After splitting the interval

Figure 1: Variable Resolution PHC

interval, one of the newly created sub-intervals would have a higher Q value than the current maximum Q value, we should then split that interval. We implement this using the following trick: instead of storing one Q value per leaf of the tree, we maintain two Q values, one for the left half of the interval and the other for the right half. Since we sample evenly within each interval when playing an action, the two halves of an interval would get played with roughly the same frequency. If the action falls in the left half, we update the Q value for the left half using Equation 2, and vice versa. We define an interval's Q value to be the average of the Q values from its left and right halves. This way, when one of the halves have a Q value higher than the current maximum, we split the interval by creating two child nodes under this node. A similar rule is set for merging two sibling leaf nodes: if both nodes have Q values that are at least a constant T from the maximum Q value, we merge the two nodes into their parent. Figure 1(b) and 1(c) show an example of splitting.

4.6 Variable Resolution MinimaxQ

MinimaxQ [Lit94] is an extension of Q-learning into two-person zero-sum games. It learns a Q function that depends on our agent's action a as well as our opponent's action b. The update rule for Q function is given by

$$Q(s,a,b) \leftarrow (1-\alpha)Q(s,a,b) + \alpha(r + \gamma V(s')) \tag{3}$$

$$V(s) \leftarrow \max_P \min_b \sum_a P(a)Q(s,a,b) \tag{4}$$

where P is our mixed strategy and $P(a)$ is the probability of playing action a. The algorithm then plays the mixed strategy P that achieves the max in the above equation. $V(s)$ and P can be computed using linear programming.

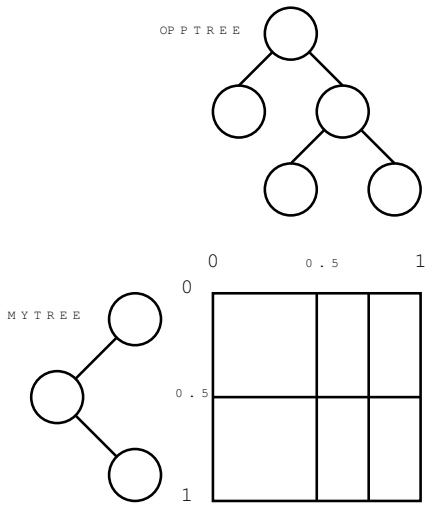


Figure 2: An Example of Data Structure for Variable Resolution MinimaxQ

When playing repeated games, MinimaxQ does not attempt to play a best response to its opponent's strategy, instead it plays the Nash equilibrium strategy of the normal form game defined by using the current Q function as the payoff function. It eventually converges to the stationary Nash equilibrium strategy of the repeated game. Thus MinimaxQ itself isn't a very satisfying learning algorithm according to our goal of maximizing payoffs, but we think it is a good stepping stone toward applying variable-resolution techniques to more sophisticated algorithms.

As before, we use binary trees to represent partitions of action spaces. We need one tree (MYTREE) for the partitioning of our action space and another tree (OPPTREE) for our opponent's action space. The Q function depends on both actions, so the Q values need to be stored in an 2-dimensional table. The leaves of MYTREE and OPPTREE store vertical and horizontal coordinates of the Q table, respectively. Figure 2 shows an example of MYTREE, OPPTREE and Q table.

We split a leaf node of MYTREE if after splitting, our new mixed strategy (which is the Nash equilibrium strategy for the game defined by Q) would be significantly different. Similarly, we split a leaf node of OPPTREE if our opponent's equilibrium strategy would be significantly different after splitting. We implement this by actually storing four Q values for each entry in the Q table: one for the top-left quarter, one for top-right, one for bottom-left and one for bottom-right. The actual splitting involves creating 2 children of the node in MYTREE (or OPPTREE) and adding a row (or column) to the Q table. Analogously we can set rules for when and where to merge two intervals in action space.

5 Experiments

To test our algorithms, we implemented a simple repeated game environment using C++. Section 5.1 describes the games whose rules we have implemented. Section 5.2 describes the types of players we have implemented. Section 5.3 gives some preliminary experimental results on our algorithms' performance.

5.1 Games

In this section we look at several games that we have included in our system. We think these games can serve as interesting testbeds for our learning algorithms. For all games in this section, $A_i = [0, 1]$ for all i.

5.1.1 Matching Pennies

This is a simple extension to the classical two-person zero-sum matching pennies game. The payoff $U_1(a_1, a_2)$ to agent 1 is given by[3]:

$$U_1(a_1, a_2) = \begin{cases} 1 & \text{if } (a_1 < 0.5 \text{ and } a_2 < 0.5) \text{ or } (a_1 > 0.5 \text{ and } a_2 > 0.5) \\ 0 & \text{if } a_1 = 0.5 \text{ or } a_2 = 0.5 \\ -1 & \text{otherwise} \end{cases} \tag{5}$$

Figure 3(a) gives a 2-D representation of U_1. This normal form game has infinite number of Nash equilibria: as long as the probabilities of choosing $a < 0.5$ and $a > 0.5$ are equal, it's a Nash equilibrium strategy profile.

5.1.2 Product Game

This is also an extension to the classical matching pennies game, but here the payoff function is smooth:

$$U_1(a_1, a_2) = (a_1 - 0.5)(a_2 - 0.5) \tag{6}$$

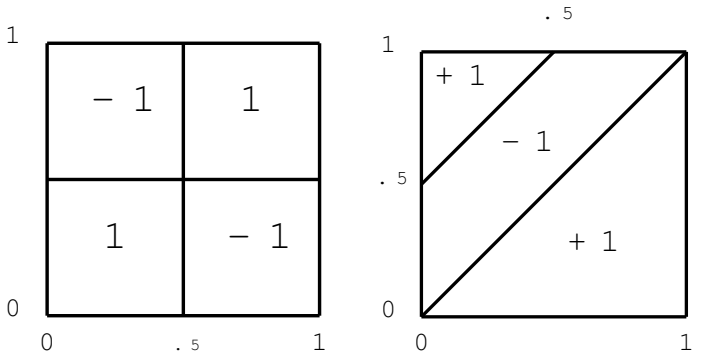
5.1.3 Sion and Wolfe's Game with Diagonal Discontinuities

A famous example from [SW57]: Consider a two-person zero-sum game with payoff

$$U_1(a_1, a_2) = \begin{cases} -1 & \text{if } a_1 < a_2 < a_1 + 0.5 \\ 0 & \text{if } a_1 = a_2 \text{ or } a_2 = a_1 + 0.5 \\ +1 & \text{otherwise} \end{cases} \tag{7}$$

Figure 3(b) gives a 2-D representation of the payoff U_1. This normal form game does not have a Nash equilibrium. This would be a very interesting test bed for reinforcement learning in repeated games.

[3]Since the game is zero-sum, $U_2(a_1, a_2) = -U_1(a_1, a_2)$.



(a) Matching Pennies (b) Sion and Wolfe's game

Figure 3: 2-D representation of Payoff functions. Player 1 chooses the horizontal coordinate, Player 2 chooses the vertical coordinate.

5.1.4 Location Game

This game has been studied in [Hot29], [DM86] and [Sim87]. Our version of the game on $[0, 1]$ is as follows: Customers are evenly distributed over $[0, 1]$. Firms choose locations from $[0, 1]$. Each customer purchases from the nearest firm. Assume there is no price competition. The players in this game are the firms, and they try to choose the location that maximize their market share. If more than one players choose the exact same location, the market for that location is evenly divided among the players that have chosen it.

For example, if there are two firms, and Firm 1 chooses 0.1 while Firm 2 chooses 0.5, then Firm 1's market share is from 0 to 0.3, and Firm 2's market share is from 0.3 to 1. Firm 1's payoff is then 0.3 and Firm 2's payoff is 0.7. Unlike the other three games, this game is not limited to two players. For our version of the game, mixed strategy Nash equilibria always exist.

5.2 Players

We have implemented our variable resolution versions of PHC and MinimaxQ as described in Sections 4.5 and 4.6. For comparison, we also implemented PHC and MinimaxQ with equal-interval partitions. The lp_solve library[4] is used to do linear programming computation in MinimaxQ. For these learning algorithms, the discount factor γ is set to 0.9, and learning rate α is set to 0.1. To make sure the MinimaxQ algorithms sufficiently explore the actions, we set their exploration probability to 0.1, i.e. 10% of the time they would play a random move from $[0, 1]$ instead of their computed mixed strategy.

We also implemented several non-learning players:

[4] Available at `http://groups.yahoo.com/group/lp_solve/`.

- Pure Strategy player just plays a stationary pure strategy.
- Random Player plays a uniform distribution in $[0, 1]$.
- Stationary Mixed Player plays a certain stationary mixed strategy given by the user. The user enters the mixed strategy by specifying the partition and probability densities within each partition.
- Periodic Player plays a sequence of actions periodically.

5.3 Results

We run several simulations of repeated games with our algorithms and some non-learning players. Following are some preliminary results.

5.3.1 PHC

The variable resolution and fixed resolution PHC algorithms did converge fairly quickly to best response against stationary players. The variable resolution PHC algorithm generally only need around 20 intervals. Figure 4 shows one simulation of the Product Game, with a pure strategy player that always plays action 0.4 as Player 1 and variable resolution PHC as player 2. Variable resolution PHC quickly converges to the action 1.0, which is the best response to the pure strategy 0.4. Variable resolution PHC performs equally well against stationary mixed strategy players.

Compared with fixed resolution PHC, variable resolution PHC can often get higher payoff against stationary players, while using fewer number of intervals. This is because the variable resolution algorithm tends to get finer and finer resolution near the best response, so the strategy it converges to is more "focused" than the strategy of the fixed resolution algorithm.

5.3.2 MinimaxQ

The MinimaxQ algorithm is not designed to play best response to other player's strategies, so we weren't surprised that our versions of the algorithm did not play the best responses to stationary opponents. The variable resolution and fixed resolution MinimaxQ algorithms generally converge to a Nash equilibrium of the zero-sum game. Variable resolution MinimaxQ generally only require less than 10 intervals per player.

6 Future Work and Conclusions

PHC and MinimaxQ are fairly simple algorithms that do not directly model opponents' mixed strategies. We would like to apply the variable resolution technique to stronger learning algorithms that explicitly model opponents' mixed strategies. It would also be interesting to explore the possibilities of using mixtures of Gaussians to model mixed strategies.

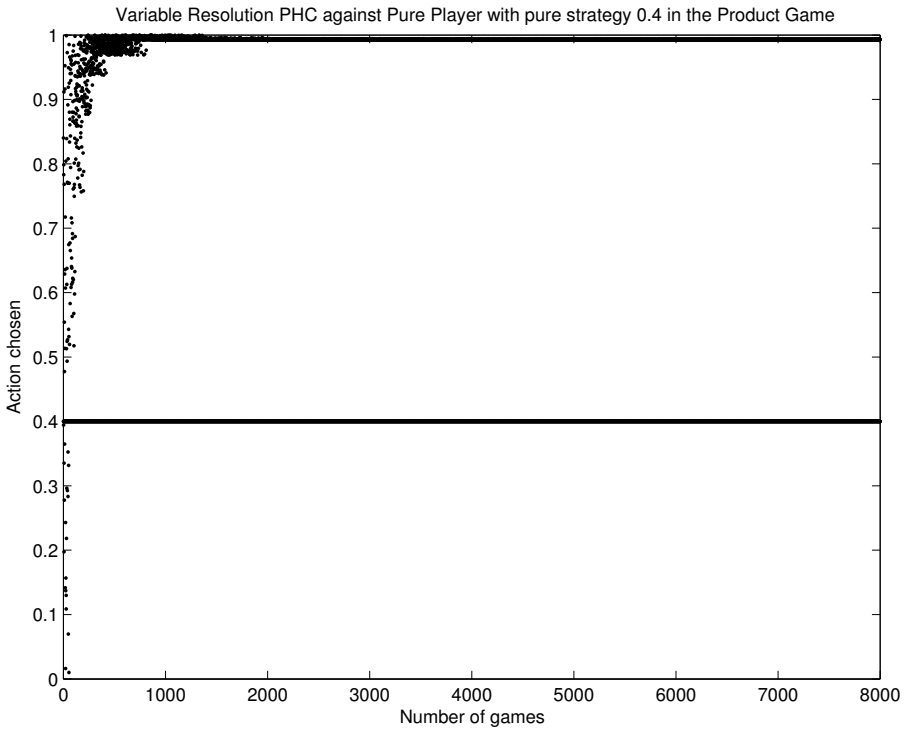


Figure 4: Simulation result of variable resolution PHC against pure strategy 0.4

We would also like to generalize our algorithms to general stochastic games. Current multi-agent RL algorithms like PHC and MinimaxQ can deal with stochastic games with finite number of states. But in many domains with continuous action space, the state space is also continuous. There has been some research in single-agent RL (surveyed in [KLM96]) that try to solve continuous state-space MDPs using variable resolution techniques. One example is the variable-resolution partitioning method Parti-Game [Moo94].

In this paper we investigated the problem of reinforcement learning in repeated games with continuous action spaces. This is an interesting unexplored area of research. We highlighted the issues in designing a learning algorithm for this problem. We then took two simple multi-agent RL algorithms and modified them to use variable-resolution partitioning of action spaces. Our experiments show that the variable-resolution technique is successful at identifying important regions of the action space, while keeping the total number of partitions low. We hope to apply this technique to more sophisticated problems.

References

[BV01] M. Bowling and M. Veloso. Rational and convergent learning in stochastic games. In *IJCAI 01*, 2001.

[DM86] P. Dasgupta and E. Maskin. The existence of equilibrium in discontinuous economic games, i: Theory. *The Review of Economic Studies*, 53(1):1–26, 1986.

[Gli52] I.L. Glicksberg. A further generalization of the kakutani fixed point theorem with application to nash equilibrium points. In *Proceedings of the American Mathematical Society*, 1952.

[Hot29] H. Hotelling. The stability of competition. *Economic Journal*, 39:41–57, 1929.

[KLM96] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research 4*, 1996.

[Lit94] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 157–163, 1994.

[Moo94] A. W. Moore. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. In *Advances in Neural Information Processing Systems 6*, pages 711–718, 1994.

[Nas50] J.F. Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences, U.S.A.*, 1950.

[Sim87] L. Simon. Games with discontinuous payoffs. *The Review of Economic Studies*, 54(4):569–597, 1987.

[SKM00] S. Singh, M. Kearns, and Y. Mansour. Nash convergence of gradient dynamics in general-sum games. In *UAI 00*, 2000.

[SPG03] Y. Shoham, R. Powers, and T. Grenager. Multi-agent reinforcement learning: a critical survey. Unpublished survey. `http://robotics.stanford.edu/~shoham/`, 2003.

[SW57] M. Sion and P. Wolfe. On a game without a value. *Contributions to the Theory of Games, III*, 1957.

[Tes03] G. Tesauro. Extending q-learning to general adaptive multiagent systems. In *NIPS 03*, 2003.