Machine Learning, 33, 155–177 (1998)
© 1998 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Learning Coordination Strategies for Cooperative Multiagent Systems

F. HO fho@watfast.uwaterloo.ca
M. KAMEL mkamel@watfast.uwaterloo.ca
Dept. of Systems Design Engineering, University of Waterloo, Waterloo, ON, Canada N2L 3G1

Editors: Michael Huhns and Gerhard Weiss

Abstract. A central issue in the design of cooperative multiagent systems is how to coordinate the behavior of the agents to meet the goals of the designer. Traditionally, this had been accomplished by hand-coding the coordination strategies. However, this task is complex due to the interactions that can take place among agents. Recent work in the area has focused on how strategies can be learned. Yet, many of these systems suffer from convergence, complexity and performance problems. This paper presents a new approach for learning multiagent coordination strategies that addresses these issues. The effectiveness of the technique is demonstrated using a synthetic domain and the predator and prey pursuit problem.

Keywords: learning, multiagent, coordination, cooperative, hill-climbing

1. Introduction

Cooperative distributed artificial intelligence (**DAI**) is concerned with how a group of intelligent agents can cooperate to jointly solve problems. Typically, AI systems have been distributed due to one or more of the following reasons:

1. The Complexity of the Domain. The complexity of the environment and/or the complexity of the tasks may make the design of a single agent difficult.

2. The Distributed Nature of the Domain. Some domains such as air traffic control or sensor fusion are inherently distributed.

3. Performance Gains. Time constraints may require the use of multiple agents in domains that allow for parallel problem solving.

4. Fault Tolerance. Redundancy in multiple agents would allow for goals to still be achieved should agents be damaged.

5. Integration of Existing Software Systems. Rather than building new applications from scratch, intelligent agents can be used to coordinate the use of existing programs (Newell & Steier).

One of the central issues in **DAI** is coordination. Jennings defines coordination as "the process by which an agent reasons about its local actions and the (anticipated) actions of others to try and ensure that the community acts in a coherent manner" (Jennings, 1996). The aims of coordination are to "ensure that all necessary portions of the overall problem are included in the activities of at least one agent, that agents interact in a manner which permits their activities to be developed and integrated into an overall solution, that team members act in a purposeful and consistent manner, and that all of these objectives are achievable within the available computational and resource limitations. (Jennings, 1996)"

Some current approaches to coordination include the use of organizational structures (Durfee, Lesser & Corkill, 1987, Tennenholtz, 1995) (Ishida, 1994), the exchange of metalevel information (Durfee, 1988), and multiagent planning (Cammarata, McArthur & Steeb, 1983, Kamel & Syed, 1994). These methods are all developed off-line and make extensive use of domain knowledge. As such, they assume the availability of complete and stationary world models that are hard to obtain in practice. Beyond the issue of models is the computational complexity (Grefenstette, 1992) of generating coordination strategies. The task of finding an optimal set of social laws for heterogeneous agents is NP-hard (Tennenholtz, 1995). Haynes suggests that "in most cases a coordination strategy is chosen if it is reasonably good" (Haynes & Sen, 1996).

In light of the problems with model completeness and the cost of generating good strategies, researchers have developed systems that learn coordination strategies (Sen, Sekaran & Hale, 1994, Haynes, et al., 1996, Parker, 1995, Weiß, 1993). However, these methods can suffer from difficulties that range from lack of convergence, to high computational complexity, and to potentially poor performance. This paper presents a new approach to learning coordination strategies, based on probabilistic hill-climbing, that makes a different set of trade-offs with respect to these criteria. The technique is guaranteed to converge, has lower complexity than existing techniques that converge, and also has a well-defined local performance measure.

The remainder of the paper is organized as follows. Section 2 defines the learning task and provides a survey and analysis of existing methods. Section 3 presents the multiagent probabilistic hill-climbing (MPHC) algorithm. First, an overview of the method is provided which is then followed by the algorithmic details. Experiments on two different domains are discussed in Section 4. The last section is a summary of the work and contains suggestions for future research.

2. Learning Coordination Strategies

Suppose there are *M* agents, each with a a simple reflex architecture that maps its current state, i.e. stimulus plus internal state, to an action. For each state, s_i, $i = 1 \ldots N$, let there be K_i potential actions. Also let there be a common payoff function, $\mathcal{U}$, that measures the utility of an action with respect to a given state. This payoff may be a function of an individual agent's state as well as the global state and may also be delayed. Thus the reward does not have to be the same for all agents. However, it is assumed that the payoff is positively correlated due to the cooperative nature of the domain. Using these definitions, an agent can be described as a mapping $\mathcal{M}$:

$$\mathcal{M} : s_i,\ i = 1 \ldots N \longrightarrow \mathcal{A}_j,\ j = 1 \ldots K_i$$

Now let the system be applied to a series of initial states that are randomly drawn from some fixed distribution, $\mathcal{P}$. Given the set of agent mappings, each initial state would induce a sequence of actions and new states with their associated payoffs. The goal of the system is to maximize the sum of the expected discounted payoffs for each agent over $\mathcal{P}$. Formally, the objective is to maximize:

$$E(\sum_{i=1}^{M}(\sum_{j=1}^{\infty}\gamma^{j-1}r_j))$$

where $0 \leq \gamma < 1$ is the discount factor, *r* is the payoff, and *E* is the expectation operator.

With this goal in mind, the learning task can be defined as finding the individual action maps, $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_M\}$, such that the sum of the expected discounted payoff is maximized. To situate the task in an algorithmic context, consider the skeleton of a generic single learner shown in Figure 1. Initially, an agent updates it's internal state by combining

```
(0) Algorithm LearnMAP()
(1)     repeat
(2)         state ← UPDATE-STATE(state, percept)
(3)         action ← CHOOSE-ACTION(state, map)
(4)         update-val ← GET-FEEDBACK()
(5)         UPDATE-MAP(update-val, map)
(6)     until TERMINATION-CONDITION(map)
(7)     return(map)
```

Figure 1. Skeletal Algorithm

it's current state with new stimuli. The latter may simply be input from sensors or it could also include messages from other agents. With the new state and it's action map, the agent chooses an action and executes it. Feedback is received and the map updated. When the termination criteria is reached, learning is halted.

The generic single agent skeleton is specialized for multiple agents by instantiating the CHOOSE-ACTION, UPDATE-MAP, and TERMINATION-CONDITION functions. A key requirement is that these functions be chosen such that the interactions between the individual action maps can be determined. Maximization of the sum of expected payoff necessitates the exploitation of any positive interactions. Since the domain is multiagent, it is assumed that there will be interactions and since it is also cooperative, it is assumed that something can be gained through cooperation.

Beneficial interactions can be detected using two approaches. The first is through the use of localized learning where other agents are treated as part of the environment. It is hoped that the agents will "co-learn" and adapt themselves to each other (Sen, Sekaran & Hale, 1994, Mataric, 1995, Sandholm & Crites, 1995). The second approach is to use "group" learning (Weiß, 1993, Haynes, et al., 1996), where agents coordinate their choice of actions in order to discern the joint effects. The following section highlights some previous work on learning multi-agent coordination strategies.

2.1. *Previous Work*

Existing multi-agent learning algorithms can be roughly classified into three types, those that use reinforcement learning individually (RLI), those that use reinforcement learning in groups (RLG) or those that use genetic programming (GP).

As it's name implies, agents using RLI (Sen, Sekaran & Hale, 1994, Mataric, 1995, Sandholm & Crites, 1995) run local copies of a reinforcement learning algorithm such as Q-learning. Typically, CHOOSE-ACTION is some stochastic action selector, UPDATE-MAP is the Q-value update rule, and TERMINATION-CONDITION is a function that measures the constancy of the agents' behaviors. An example of a stochastic selector is the Boltzmann exploration rule where the probability of executing an action $\mathcal{A}_j$ in state *i* is given by:

$$p(\mathcal{A}_j) = \frac{e^{Q(s_i,\mathcal{A}_j)/t}}{\sum_{l=1}^{K_i} e^{Q(s_i,\mathcal{A}_l)/t}}$$

where *Q* is the Q-value, or utility, of an action in a given state and where *t* is the temperature or annealing rate.

Behavioral constancy is often measured by how many times, in a row, a prescribed goal is achieved. For instance, learning is halted in the block pushing domain (Sen, Sekaran & Hale, 1994) when the two agents have successfully pushed a block from start to end locations, ten times in a row.

The RLI approach is simple but it does not guarantee convergence to stable action maps. One assumption of algorithms like Q-learning is that the environment is stationary. Since the agents can individually change their maps, this assumption is violated.

RLG methods implement grouped learning. In (Weiß, 1993), CHOOSE-ACTION requires the agents to bid for the right to execute an action within the context of a group of actions. The group with the largest bid is selected for execution. This addresses the convergence problem since changes to an agent's action map are only in the context of groups and therefore must be coordinated with those of other agents. This eliminates the stochasticity introduced when agents unanimously make changes. The major difficulty of this method is high sample complexity due to the combinatorics of grouping (Dowell & Stephens, 1996). Given four agents, each with four states and two actions each, there would be $(4\text{x}2)^4$ = 4096 groups if all the different state to action mappings are considered. Thus there is an issue of scalability in the use of RLG's.

Existing genetic programming methods (Haynes, et al., 1996) apply this paradigm to groups of changes in the action maps. Sample complexity is reduced, as compared to RLG, since GP's implement a form of beam search. That is, only a subset of the potential groups is analyzed at the same time and from out of that group, a smaller subset is promoted to the next generation. By limiting the size of the subsets, complexity can be controlled. However, this incremental limited-width search may result in poor performance. This is an instance of the well known "multiagent credit assignment" problem. Consider one complete instantiation of the agent maps. As a group, let this combined set of maps have "high" expected payoff. The question is, which choice of state to action mappings made beneficial contributions and which, if any, were irrelevant? The simplest answer is to ignore the problem and say that one is only concerned with the entire group (Haynes, et al., 1996). As long as the group performs, it doesn't matter which choices made the contributions. This solution can cause problems in approaches that use beam search or hill-climbing (beam of width 1) since portions of the search space are discarded. An action assignment may be irrelevant but in committing to it, the system may have placed itself in a local maxima since choices may influence future decisions.

Consider a simplified scenario from a financial domain. Company X is losing money and thus it must make some changes to its business strategy. The firm is structured into two units that independently make their own decisions. The first is responsible for the production and the other for sales and marketing. In the current state, production has the option of either increasing the output of widget *A* or diversifying and building a factory to make widget *B*. Note that they currently do not sell all the *A*'s that they manufacture. Sales and marketing only has the option of spending more money on marketing *A*. There are insufficient resources to increase the production and the promotion of *A* while simultaneously starting the production of *B*. Let one of the groups be the promotion of *A* and the production of *B*. The advertisement campaign works and the inventory of *A* is sold, but widget *B* is a flop. From the point of view of profit, the combined map (group) that only promotes *A* is indistinguishable from the one that promotes *A* and builds *B*. However, the second map allows money to be spent on increasing the production of *A* to meet the new demand. Note that the problem has nothing to do with localized versus grouped methods. Each agent could have individually arrived at their own choice of actions. The problem occurs in incremental learning algorithms since decisions made in earlier steps can affect the optimality of those yet to made as they cannot be undone.

In summary, the existing approaches make different trade-offs with respect to each other. RLI's favor lower sample complexity and simplicity at the expense of convergence. RLG's sacrifice sample complexity for guaranteed convergence. Finally, GP's converge and have controls on complexity, but at the potential cost of performance.

There are also some commonalities between the three methods. One is that convergence is not well defined. Typically, the behavior of the agents is observed and if it is the same over some interval then the system is said to have converged. Another issue is the lack of quantification on performance. For instance, experiments may indicate that the algorithm works well on a particular domain but performance bounds, on another domain, may be hard to come by.

3. Multiagent Probabilistic Hill-climbing

The following sections present a multiagent learning technique based on probabilistic hill-climbing. First a general discussion of probabilistic hill-climbing and it's application to multiagent systems is provided. Next, the algorithmic details are discussed. Finally, the technique is theoretically evaluated in terms of the convergence, complexity and performance issues highlighted in Section 2.

3.1. Overview

A defining characteristic of probabilistic hill-climbing (PHC) algorithms (Greiner, 1996, Fong, 1995, Gratch, 1993) is that they make satisficing rather than optimal decisions. Without getting into details at this point, consider the simplest state to action mapping problem in the form of an agent with only one state. Let there exist a hypothesis selection (HS) algorithm that, given the parameters ϵ and δ, returns action A^* such that $\forall A_{j=1..k},\ A_j \neq A^*$, $\bar{\mathcal{U}}(A^*) > \bar{\mathcal{U}}(A_j)$ or $|\bar{\mathcal{U}}(A^*) - \bar{\mathcal{U}}(A_j)| \leq \epsilon$ with 100(1-δ) % confidence; where $\mathcal{U}$ is the payoff of action A_j in state *S*. In other words, A^* is ϵ-optimal in that every other action has an expected payoff which is either less than A^* or at most ϵ larger than A^*. The ϵ can be regarded as a measure of indifference. If the "best actions" have expected rewards that are within ϵ of each other, then the agent doesn't care which one is selected.

If a procedure exists for one state, it can be extended to handle multiple states by embedding it within a hill-climbing process. The choice of an action is regarded as a step. This requires that δ be allocated across the hill-climbing steps such that the sum over all steps is δ. If the number of steps is bounded then this can be done by dividing δ by the bound, else δ can be allotted by multiplying it by any series that sums to 1 at infinity (Greiner, 1996).

During the process, each state is tested for the existence of A^*, and if it exists, the action A_i^* and its $\bar{\mathcal{U}}(A_i^*)$ are stored. When all states have been tested, the map is modified to reflect the choice of the A^* with the highest expected payoff. Hill-climbing ends with the deletion of all statistics associated with the remaining states. The agent's state space has been altered and thus these values are no longer valid. This procedure continues until an action has been selected for each state. Given that an ϵ-optimal action must exist for each state, termination is guaranteed. A consequence of this approach is that the global optimality of the map is traded for local ϵ-optimality at each step.

Now consider a multiagent setting where each agent runs an individual copy of the PHC algorithm. Instead of hill-climbing at will, let the agents wait until they have all found their A^*'s. This can be accomplished using a simple messaging protocol.

Due to the use of hill-climbing, the system is susceptible to the multiagent credit assignment problem. There are *M* concurrent map changes to be made, but not all of these changes may be useful. The problem can be solved by noting that it stems from over-commitment. In the financial example, the increase in profits only required the marketing of widget *A* and not the production of widget *B*. A criteria that requires the system to minimize the number of action choices that are committed to would remove the unnecessary production of *B*. Intuitively, agents should only commit to the minimum number of choices needed to meet their goals. This would prevent irrelevant action assignments from blocking future assignments that may prove beneficial. Given that the hill-climbing task is equivalent to finding the "best step" at each point in the map space, the job is to define the "minimally best step". There are two cases of minimality to consider:

1. If the A_i^*'s do not beneficially interact then the "minimally best" step is the A_i^* with the largest $\hat{\mathcal{U}}$. Steps should be atomic. Thus, if the effects of the changes are simply additive, then the one with the biggest improvement should be implemented.

2. If subsets of the A_i^*'s beneficially interact then the "minimally best" step is the interacting subset with the largest $\hat{\mathcal{U}}$. Again, the ϵ-optimality criteria can be applied to compute the subset with the largest expected reward.

The issue is to determine which, if any, of the A^*'s interact. This can be done by grouping the A^*'s. If there are beneficial interactions between all the elements of the group then the expected reward of the group should be greater than the sum of it's parts. This can be verified by comparing the payoff of a set of changes with the payoffs of it's subsets under the assumption that they are, at least, partially independent. That is, the payoffs are simply additive.

All combinations of groups of size 2 to M, where M is the number of agents, are generated since interactions can occur between any two or more of the A^*'s. The result is an exponential number of groups but the base of exponent is always 2 and not the number of states times the number of actions as in Section 2.1. Consider three agents and let their A^*'s be **A**, **B**, and **C**; respectively. If **A**, **B**, and **C** beneficially interact then $\hat{\mathcal{U}}$(**A**,**B**,**C**) should be greater than that of:

1. $\hat{\mathcal{U}}$(**A**, **B**, $\mathcal{D}_3$)+$\hat{\mathcal{U}}$($\mathcal{D}_1$,$\mathcal{D}_2$,**C**)
2. $\hat{\mathcal{U}}$(**A**,$\mathcal{D}_2$,**C**)+$\hat{\mathcal{U}}$($\mathcal{D}_1$,**B**,$\mathcal{D}_3$)
3. $\hat{\mathcal{U}}$($\mathcal{D}_1$,**B**,**C**)+$\hat{\mathcal{U}}$(**A**,$\mathcal{D}_2$,$\mathcal{D}_3$)
4. $\hat{\mathcal{U}}$(**A**)+$\hat{\mathcal{U}}$(**B**)+$\hat{\mathcal{U}}$(**C**)+$\hat{\mathcal{U}}$(**D**)

where $\mathcal{D}$ represents the choice of an action other than $\mathcal{A}^*$ for the state associated with $\mathcal{A}^*$. A complete mapping requires an action for every state and thus, if $\mathcal{A}^*$ is not chosen then some other action must be. The first three cases represent the scenario where two out of the three A^*'s interact while the third is independent. The last case represents the scenario where they are all independent.

Cooperation is required to compute the payoff of these (sub)sets. Each agent must know when and when not to use its A^*. Recall that initially the agents individually choose their actions until they have found their A^*'s. Once this been achieved they must coordinate their actions to discern the beneficial interactions. This requirement breaks learning into two stages. The first stage uses co-learning while the second uses grouped learning. Thus, MPHC uses a mixture of both paradigms.

3.2. Algorithmic Details

As discussed in Section 2, a learning algorithm is realized by instantiating the CHOOSE-ACTION, UPDATE-MAP, and TERMINATION-CONDITION functions. The key element of the MPHC algorithm is the HS procedure. Given a historical sequence of payoffs, the expected payoff, $\bar{\mathcal{U}}_j$, for each action can be approximated using its sample mean, $\hat{\mathcal{U}}_j$. The task is to find the action that satisfies the ϵ-optimality criteria. This can be done by formulating a series of pair-wise hypothesis tests that compare the mean of the action with the largest average value, $m = argmax_{j=1}^{k}(\hat{\mathcal{U}})$, with those of the remaining actions, $j \neq m$. If the probability: $\forall j \neq m, \hat{\mathcal{U}}_m < \hat{\mathcal{U}}_j - \epsilon$ is less than δ then *m* is chosen.

These calculations require assumptions to be made about the distribution of $\mathcal{U}$. The simplest case is to assume nothing and use distribution-free techniques such as Hoeffding's inequality (Hoeffding, 1963) to construct confidence intervals around the value of each $\hat{\mathcal{U}}$. However, these methods are sample hungry. MPHC takes a different tack and modifies the quantity that is being estimated. During initialization, each state is assigned a random default action. When an action is selected for execution by CHOOSE-ACTION, its payoff is computed. Then the payoff using the default mapping is evaluated. This is done by "replaying" the actions of the environment but changing the actions of the agent. The difference between the two values is stored as the payoff. Since differences are used, the central limit theorem (Kirkpatrick, 1974) can be invoked and the distribution of the payoffs considered to be normal after a number of trials. This allows T-tests to be used in the pair-wise hypothesis tests. The HS algorithm is shown in Figure 2 and is similar to the approach taken in (Chien, Gratch & Burl, 1995). The first step is to return the index, *m*, of

```
(0) **Algorithm** HS($\epsilon, \delta$)
(1) $m = argmax_{j=1}^{k}(\Delta - \hat{\mathcal{U}})$
(2) t-error=0
(3) $\forall j \neq m$
(4)     pwe= P-WITHIN-EPSILON($j,m,\epsilon$)
(5)     plt= P-GREATER-THAN($j,m$ )
(6)     pe= MAX(pwe,plt)
(7)     t-error=t-error+(1-pe)
(8) if (t-error $< \delta$) then
(9)     hill-climb=T
```

Figure 2. HS Algorithm

the action with the largest mean $\delta - \hat{\mathcal{U}}$. For every other action, *j*, the probability that it's mean lies within $\pm\epsilon$ of the mean of *m*, *pwe*, and the probability that it's mean is less than that of *m*, *plt*, are computed using T-tests. The larger of these two values is taken to be the actual case. That is, if *pwe* is larger than *plt* then action *j* is considered to have a utility that is within ϵ of *m*. One minus this value is the probability that *m* is not ϵ-optimal with respect to *j*. If the sum of these probabilities is less than δ then *m* is ϵ-optimal with confidence 100(1-δ)%.

Given an algorithm to determine $\mathcal{A}_i$, the next component is a method to test for interactions. Again, T-tests can be used to compute the probability that a group is greater than or within ϵ of its parts, under various independence assumptions. The TEST-INTERACTIONS algorithm is shown in Figure 3. Initially, all potential groupings of the $\mathcal{A}^*$ actions are generated. In the process, the $\mathcal{D}$'s are given the value of their random default assignment. Each element of the set of groupings is then tested for interactions. The probability that a group contains independent elements is determined by taking the product of a series of pairwise T-tests. This requires a number of samples, the parameter $\mathcal{N}_1$, be taken to ensure the validity of the normal distribution assumption. In the tests, the pairs correspond to comparing a group with it's subgroups as described in Section 3.1. For each case, the probability that their means are within ϵ of each other is computed. Another series of tests is then used to determine if the mean of the group is greater than its subgroups. Again, a product is used to combine these results which assumes that the errors are independent. The maximum of these two probabilities is taken to be the actual case as in the HS algorithm. If maximum value is the "greater than" case and the combined error is less than δ, the group is considered

```
(0) Algorithm TEST-INTERACTIONS($\mathcal{A}^*$,$\epsilon$,$\delta$,$\mathcal{N}_1$)
(1)  i-groups = $\phi$
(2)  groups = GENERATE-GROUPS($\mathcal{A}^*$)
(3)  if (num-samps $> \mathcal{N}_1$) then $\forall X,\ X \in$ groups
(4)       sub-groups = RETURN-SUB-GROUPS(X,groups)
(5)       pwe = $\prod^{sub-groups}$ GROUP-WITHIN-EPSILON(X,$\epsilon$)
(6)       pgt = $\prod^{sub-groups}$ GROUP-GREATER-THAN(X)
(7)       t-error = 1-MAX(pwe,pgt)
(8)       if (t-error = pgt and t-error $< \delta$) then
(9)            i-groups = i-groups $\bigcup$ X
(10) MAX-PAYOFF(i-groups)
```

Figure 3. TEST-INTERDEPENDENCE Algorithm

to be beneficially interacting. All interacting groups are collected and the one with largest expected payoff is returned.

3.2.1. UPDATE-MAP *Function* The HS and TEST-INTERACTIONS algorithms are both parts of the UPDATE-MAP function shown in Figure 4. This function implements a hill-climbing step as discussed in Section 3.1. The first thing to notice is that there are two

```
(0)  Algorithm UPDATE-MAP($\epsilon_1, \epsilon_2, \delta_1, \delta_2$,G)
(1)       $\forall S_i, i = 1 \ldots N$ : if $\exists A_i^*$=HS($\epsilon_1, \delta_1$) then
(2)            P = P $\bigcup A_i^*$
(3)       if P then
(4)            g = MAXIMUM-AVG-PAYOFF(P)
               ;;End Stage 1
               ;;G contains the $A^*$'s from the other agents
(5)            interacting = TEST-INTERACTIONS(g $\bigcup$ G,$\epsilon_2$,$\delta_2$)
(6)            if (g $\in$ interacting) then
(7)                 MODIFY-MAP(g)
(8)            DELETE-STATISTICS($S_{i=1 \ldots N}$)
```

Figure 4. UPDATE-MAP

ϵ's and δ's arguments. This is the result of using a two stage approach. The two δ's and ϵ's add up to the δ and ϵ that has been allocated to this step, but these values have been split across the stages.

In UPDATE-MAP, steps one to four correspond to stage one. Each agent runs HS on each of its states and collects the A_i^*'s into P. If P is not empty then the element with largest expected reward is chosen. Once all agents have found their $\mathcal{A}^*$, the algorithm moves onto stage two. As discussed, the agents must cooperate to determine the payoff of the groups. This requires a change to the CHOOSE-ACTION procedure such that the agents coordinate their choice of actions. The initial stage one instantiation of this algorithm will be discussed in Section 3.2.2. Step five returns the interacting group with the largest expected reward. If an agent's $\mathcal{A}^*$ is a member of this group then it's map is modified. The step ends with the deletion of all statistics since hill-climbing has been performed.

3.2.2. CHOOSE-ACTION *Function* An important component of any co-learning approach is the CHOOSE-ACTION function which has two roles. It must perform exploration to find the best action for a given state but it also must dwell on the current estimate for A_i^*. The second requirement allows the agents to determine if there are beneficial interactions between their current choices for A^*. Intuitively, joint sampling of their respective A^*'s allows the agents to measure the effect of the combination. The necessity of this is illustrated in Section 4.1 where some of the trials used a random strategy for CHOOSE-ACTION. Results for these cases were poor.

The constraints on CHOOSE-ACTION are similar to those for the k-arm bandit problem (Berry & Fristedt, 1985) where the task is to maximize the long-run total reward of sampling the arms of an imaginary multi-armed slot machine. On each iteration, and given the results of the previous trials, an arm is selected for sampling.

One approach to the problem is the Z-heuristic (Rivest & Yin, 1994). Given state *i*, a normal random number is generated for each action, A_j, that is parameterized by the mean and variance of $\mathcal{U}(A_j)$. The action with the largest random number is chosen for execution. This technique compliments the use of default actions and Δ-payoffs since the distribution of the $\mathcal{U}$'s can be assumed to be normal after a number of trials. Since a number of trials is required, the Z-heuristic is parameterized by $\mathcal{N}_0$, or the number of initial samples that must be seen before the strategy is invoked. At the outset, the actions are chosen at random.

3.3. TERMINATE-CONDITION *Predicate*

The use of MPHC results in a straightforward termination criteria since each state must be assigned one action. Thus learning can be haled when all agents have hill-climbed on each of their states. The problem is not as simple in practice since there may be domains with states that are rarely visited or that go unvisited. The set of problems may not entail these states or the states may be suboptimal. Non-termination is not a major issue for MPHC since the technique adds little overhead except when sampling actions in a non-terminate state. When an agent enters a state, it first checks to see if the state has been assigned an action. If so, the action is simply executed.

Termination can be enforced by adding an extra test to each state. Each agent keeps track of the number of trials that as passed. The probability that a state will be visited for less than or equal to ϵ_3 number of times can then computed using distribution free methods. If this value drops below an assigned δ then learning can be terminated on that state.

3.4. Theoretical Evaluation

MPHC has a number of advantages over the existing learning methods. As with many other approaches, an improvement in one criteria may require sacrifice in another. Thus the job is to select an appropriate set of trade-offs.

In terms of convergence, the approach is guaranteed to converge unlike RLI techniques since each state that is visited must have an ϵ-optimal action. This even holds for non-cooperative domains such as the case where there are two agents trying to learn different versions of the parity problem. Consider a situation where agent one has an odd parity payoff function while agent two has an even parity one. Each agent has a choice of returning a one or a zero. This results in four states, and if the payoff for even and odd parity is the same, the sum of payoffs for these states is constant. Thus the expected payoff for either action is also the same. Traditional RL techniques would flip back and forth between assignments as a positive reward was received. However, the ϵ indifference zone would come into play in MPHC and result in an action being assigned.

In terms of complexity, a rough comparison can be made between MPHC and non-genetic programming based group techniques by examining the number of $\hat{\mathcal{U}}$'s that need to be estimated. While the RLG approach does not explicitly estimate means, the nature of the problem is similar since finding the best group requires an overall measure of group performance. Consider a domain with one state and four agents, each with five actions apiece. If all groups are generated, there are 5^4-1=624 means to estimate. One the other hand, if MPHC is used, there are 20 (4 x 5) means to estimate in stage one and 2^4-1=15 in stage two, for a total of 35. These numbers cannot be directly compared since the number of samples required to perform the estimation may not be equal but there is a large difference. However, the range in number of samples required should be similar since the task is the same.

Complexity comparisons cannot be made between MPHC and RLI techniques since the latter methods are not guaranteed to converge. Comparisons are also inappropriate in the case of GP since their complexity is determined completely by how they are parameterized.

The theoretical performance of MPHC is hard to quantify. Decreased sample complexity compared to complete grouping is bought by using local hill-climbing. This invalidates any claims to global optimality. However, as compared to genetic programming, the proposed approach has the advantage that it accounts for the multiagent credit assignment problem. The actual ability of the technique to find useful coordination strategies must be empirically verified using test domains.

4. Experimental Validation

The effectiveness of multiagent probabilistic hill-climbing was tested using four different domains. Two of these, the symbol and predator and prey domains are highlighted below.

4.1. Symbol Domain

The symbol domain is a toy problem where a groups of agents learn which symbol to return from its assigned set. Payoffs are based on the particular combination of symbols with all agents receiving the same payoff. Consider four agents that have each been assigned five contiguous letters from the alphabet. These assignments are shown in Table 1. The payoff

Table 1. Agents and their Symbol Assignments

Agent	*Symbols*
1	{A B C D E}
2	{F G H I J}
3	{L M N O P}
4	{Q R S T U}

function maps triples of symbols to a scalar value. This mapping can be expressed in ϵ units given values for ϵ and the maximum reward.

Since the symbol assignments are unique, the construction of a particular triple requires the agents to cooperate. Beyond the need for cooperation, the payoff mapping also controls the difficulty of the learning task. A symbols is used in more than one triple and thus there is a probability that it will be part of some ϵ-optimal triples. If the number of ϵ-optimal triples is increased then the probability that any random symbol being a member of a ϵ-optimal triple also increases. This translates to a higher probability that the default actions will also form a ϵ-optimal triple.

The payoff distribution of the remaining non-optimal triples also affects the complexity of the task. A distribution that has a peak in the number of 2ϵ triples poses a greater degree of difficulty than one that is flat or one that has a peak at a higher multiple of ϵ. Triples that are closer to being ϵ-optimal are harder to distinguish than those that are further away.

The symbol domain is the simplest form of the learning problem found in Section 2. Each agent has only one state and a fixed number of actions, i.e. which symbol to return, that it can execute. The domain is also deterministic when the payoff function is fixed. This simplicity makes the domain useful for examining the performance of MPHC since any complicating details have been omitted. In particular, the intermediate results of the stages can be directly analyzed and the effectiveness of each stage determined.

Example histograms for the distributions of triples are shown in Figures 5 and 6. The units along the x-axis are in terms of ϵ. For lack of better labels, the distributions are referred to as $\mathcal{D}$-1 and $\mathcal{D}$-2, respectively. Since each of the four agents has been assigned five symbols, there are a total of $\begin{pmatrix} 4 \\ 3 \end{pmatrix} 5^4 = 500$ different triples. Of these, 4.2% are within one ϵ of the best triple in $\mathcal{D}$-1. This number drops to 1.8% in the case of $\mathcal{D}$-2. Note that $\mathcal{D}$-2 also has a large number of within 2ϵ entries. As discussed, the skew makes the task of returning an ϵ triple even more difficult. Results of experiments on $\mathcal{D}$-1 and $\mathcal{D}$-2 are shown in Tables 2 and 3, respectively. During the trials, the value of $\mathcal{N}_0$ was varied to ascertain its effect. The remaining parameters, ϵ_1, ϵ_2, δ_1, δ_2, and $\mathcal{N}_1$ were held constant at 0.2, 0.2, 0.025, 0.025 and 30. In the case of $\mathcal{N}_1$, it was found that all trials used more than 30 samples during stage two and thus it was not necessary to determine its effect. All default actions

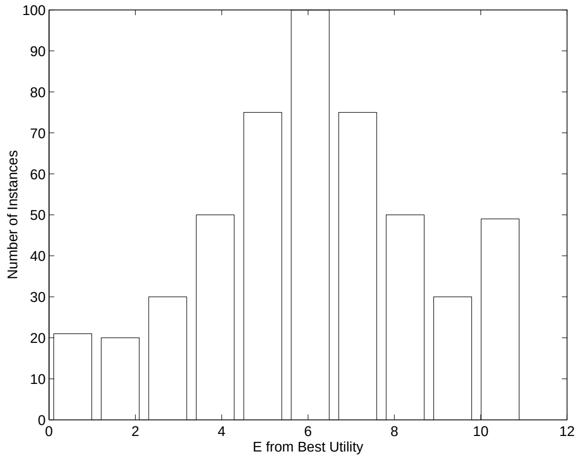


Figure 5. $\mathcal{D}$-1 Distribution

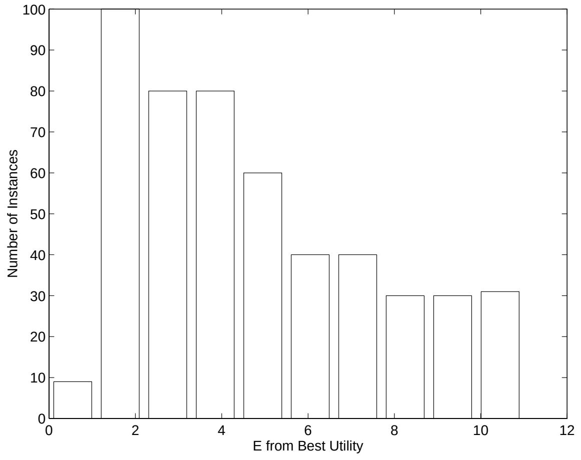


Figure 6. $\mathcal{D}$-2 Distribution

were assigned randomly. To reduce runtime, sample cutoffs of 5000 and 800 were placed on stage one and two, respectively. If the algorithm did not return an answer by this time, a failure was noted. For each distribution, experiments were also run using random and round-robin sampling schemes for CHOOSE-ACTION during stage one.

The first column of each table represents the test conditions in terms of $\mathcal{N}_0$ and the sampling method. Z stands for the Z-heuristic, while R and S denote random and round-robin sampling, respectively. Random and round-robin sampling used a $\mathcal{N}_0$ value of 10. Since neither scheme involved switching sampling methods, $\mathcal{N}_0$ was simply used to indicate when the hill-climbing criterion should be checked. Recall that the Z-heuristic starts out

Table 2. Results for $\mathcal{D}$-1

Parameters	ϵ_0	ϵ_1	*F1*	*F2*	*DNH1*	*DNH2*
5/Z	4	9	0	3	(34/0)	0
10/Z	4	36	5	2	(3/0)	0
15/Z	7	38	3	2	0	0
10/R	1	12	35	2	0	0
10/S	2	16	17	4	(11/2)	0

with random sampling and then changes over to focussed sampling. In the case of random and round-robin sampling, it always took much more than 10 samples to complete stage one. Thus, the arbitrary value of 10 has no effect on the performance of these approaches. The remainder of the table is organized as follows:

1. ϵ_0: The number of trials that found the best triple.

2. ϵ_1: The number of trials that found a triple that is within ϵ of the best.

3. *Failed Stage 1*: The number of trials that did not return an ϵ_0 or ϵ_1 triple during stage one.

4. *Failed Stage 2*: The number of trials that did not return either ϵ_0 or ϵ_1 triple during stage two.

5. *D.N.H Stage 1*: This item contains two entries. The first is the number of trials that reached the first stage sample limit of 5000 without halting. Entry two is the number of stage one D.N.H entries that would have resulted in a stage one failure if the algorithm had returned the current best set as the final best triple.

6. *D.N.H Stage 2*: This item contains two entries. The first is the number of trials that reached the second stage sample limit of 800 without halting. Entry two is the number of stage two D.N.H entries that would have resulted in a stage two failure if the algorithm had returned the current best triple as the actual one.

From the tables, it can be seen that MPHC is capable of returning ϵ-optimal results, with a good success rate, given parameters appropriate to the distribution. If the success rate is computed by summing the number of ϵ_0 and ϵ_1 entries, the best result is the case of 15/Z on the $\mathcal{D}$-1 distribution. However, its score of 90% fell short of the specified value of 95%. Recall that $\delta_1 + \delta_2$=0.05. This outcome underscores the heuristic nature of the approach. Yet, simply summing the ϵ_0 and ϵ_1 entries underestimates the success rate due to the *DNH* entries. For instance, an examination of the stage one *DNH*'s, especially for $\mathcal{D}$-1, indicates that most of them would have return an ϵ-optimal result if allowed to halt. This would lead to more ϵ-optimal final results in the case of the other parameterizations. The validation of

Table 3. Results for $\mathcal{D}$-2

Parameters	ϵ_0	ϵ_1	*F1*	*F2*	*DNH1*	*DNH2*
5/Z	8	10	2	3	(27/5)	0
10/Z	13	27	0	6	(4/3)	3
15/Z	8	22	5	7	(8/8)	0
10/R	0	4	45	1	0	0
10/S	1	14	22	1	12	0

this hypothesis was pragmatically beyond the computational capabilities of the Sparcstation IPX used in these experiments.

The large number of *DNH1* entries, particularly for small values of $\mathcal{N}_0$, is an artifact of the Z-heuristic. One problem is that it tends to spend most of its time sampling the best triple. This is exacerbated by distributions where a few triples are much better than the others. Confidence in the utility estimates, and hence error, is a function of the number of samples. Given that most of the samples are concentrated on one triple, the incremental error reduction per sample is low. Increasing $\mathcal{N}_0$ forces the procedure to sample each triple at least a fixed number of times.

4.2. Predator and Prey

The object of the predator and prey domain is for four predators to capture a prey within a 30 by 30 toroidal grid world. Diagonal moves are not allowed and capture occurs when the four agents are directly adjacent and orthogonal to the prey. At each time step, the prey has a 90% change of moving; thus, making the predators faster than the prey. The prey is controlled by a simple strategy that moves it away from the closest predator. All ties are arbitrarily broken. Predators have a choice of moving in one of the four orthogonal directions or staying put.

Collisions occur when two predators try to occupy the same square. In this case, the predators remain in their previous positions. However, if a predator does not move then it can be pushed by another predator. This can happen when a predator decides not to move or if its chosen move results in a collision. Predators cannot push the prey.

Using the simple architecture from Section 2, predators are controlled by a strategy based on the relative direction of the prey with respect to a predator. Thus there is a state for each of the eight compass directions, (**N, S, E, W, NW, SW, NE, SE**). Associated with each state are five potential actions, (*north*, *west*, *south*, *east*, *hold*). Since the grid world is toroidal and there is no explicit representation for distances, the relative direction of the prey that minimizes the distance between prey and predator is the one returned. Although this representation is simple, it is capable of encoding similar strategies to those found in (Haynes, et al., 1996). Following (Haynes, et al., 1996), experiments used the following payoff scheme:

1. After each move, each predator receives a value of $\frac{1}{distance-to-prey}$. This encourages the predators to stay close to the prey.

2. At the end of the simulation, every predator that is orthogonally adjacent to the prey, receives a reward of the total number of moves allowed. This biases the mappings towards those that bring the predators next to the prey.

3. If the predators capture the prey then all predators are given a value of four times the maximum number of moves allowed in addition to any other reinforcement received. Clearly this favors mappings that aid in capture.

As in the case of (Haynes, et al., 1996), the initial locations of the predators were random with the prey being centered in the grid. The movement of the prey was synchronized with that of the predators. The maximum number of moves was set at 100. A trial ends when either this number is exceeded or if the prey is captured.

Experiments were performed using the same procedure as in (Haynes, et al., 1996) where learning used 100 moves while test runs allowed 200 moves for capture. Results for two different GP approaches and MPHC are shown in Table 4. In all cases, 100 trials were performed and the number of captures and the average number of steps to capture are reported. STGP (Haynes & Sen, 1995) and A1 (Haynes, et al., 1996) are both GP

Table 4. Predator and Prey Results

Algorithm	*Captures*	*Avg. Moves*
STGP	26	79.9
A1	0	200
MPHC-N	97	71.9
MPHC-R	100	72

approaches that differ in that STGP learns a single strategy that is used by all predators. Thus, there is an implicit form of communication between the predators and all predators are assumed to be homogeneous. A1 is the grouped methodology referred to in Section 2.1. Note that the results for the trials are not those reported in (Haynes, et al., 1996). Rather, new simulations were performed using the coordination strategies that were described. This allowed the methods to be compared using the same test cases.

The first MPHC trial started off with a default mapping that had all the predators go *north* regardless of the relative position of the prey. On the other hand, MPHC-R had a random default mapping. The MPHC trials were parameterized as follows: ϵ=4 (for both stages), $\mathcal{N}_0$=50, and δ=0.4. Since there are eight states and four agents, the total number of hill-climbing steps was bounded by 32. The resulting δ was evenly allocated over both stages.

MPHC performed better than STGP, both in terms of the number of captures, and the average number of steps to capture. This is surprising given the implicit communication between predators. The $A1$ strategy failed to capture the prey during all trials, reaching the cutoff limit of 200 moves. This is in line with results reported in (Haynes, et al., 1996) where the average number of captures was 0.115 out of 30 test cases over 26 trials. The average path length of 200 for $A1$ was included to indicate that it failed to capture. From the table, it can be seen that the average solution lengths for all strategies are similar.

In terms of sample complexity, MPHC-R required 170988 samples to terminate. The experimental setup for $STGP$ is unknown but $A1$ used a population size of 600 for 1000 generations (Haynes, et al., 1996). This implies 600000 samples. However, direct comparisons between these results are not very meaningful since the numbers for PHC are based on random default strategies while those for $A1$ are arbitrary.

It is also interesting to compare the strategies generated by genetic programming to those produced by probabilistic hill-climbing. A visualization of the strategies is shown in Figures 7 to 10. **P** indicates the prey and the arrows indicate the direction that a predator would move given its current relative position to the prey. For instance, in the $STGP$ strategy, if the prey is to the east of a predator then the predator will always move east. The perturbations at the edges of $STGP$ and $A1$ strategies are probably the result of the low level representation used in GP. They were reproduced in the runs.

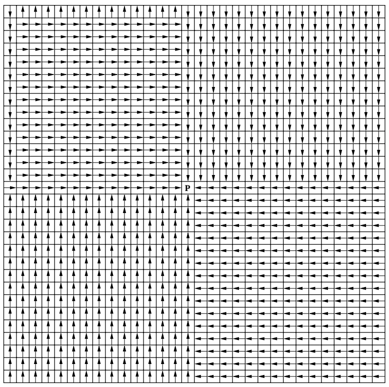
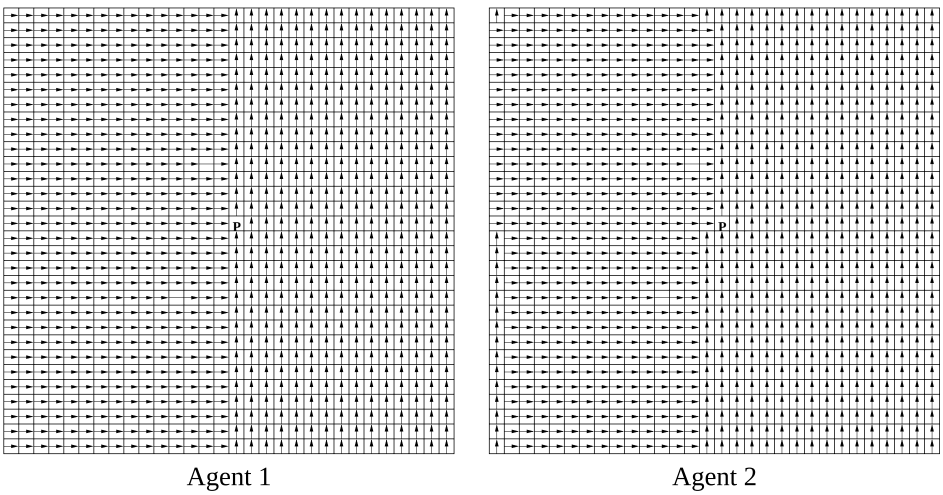


Figure 7. STGP Strategy

The strategies generated by $MPHC$-N are similar to those of $STGP$ except that they exhibit a higher degree of specialization. Predator 1 only approaches from the south. Predator 2 only approaches from the east. Predator 3 approaches from the north, south, and west; while predator 4 approaches again, only from the north. The best results are for $MPHC$-R, which also shows specialization, except for the case of predator 2. From Figure 10 it can be seen that this predator does not approach the prey at all. An examination of the pursuit paths suggests that predator 2 is "luring" the prey by staying on the diagonal before finally moving into position. This exploits the prey's weakness in that it moves away from the closest predator. Since Manhattan distances are computed, predators on diagonals seem farther away. The combination of strategies produce the pre-capture configuration is shown in Figure 11. Given that predators 1, 3, and 4 are all one unit away from the prey, it can only move south. However, predator 2 also wants to move into the same square and thus



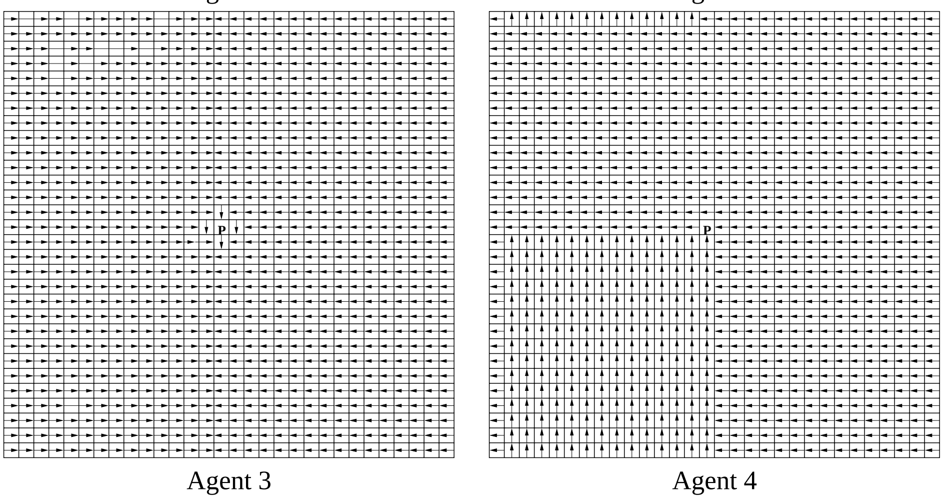


Figure 8. Strategies for A1

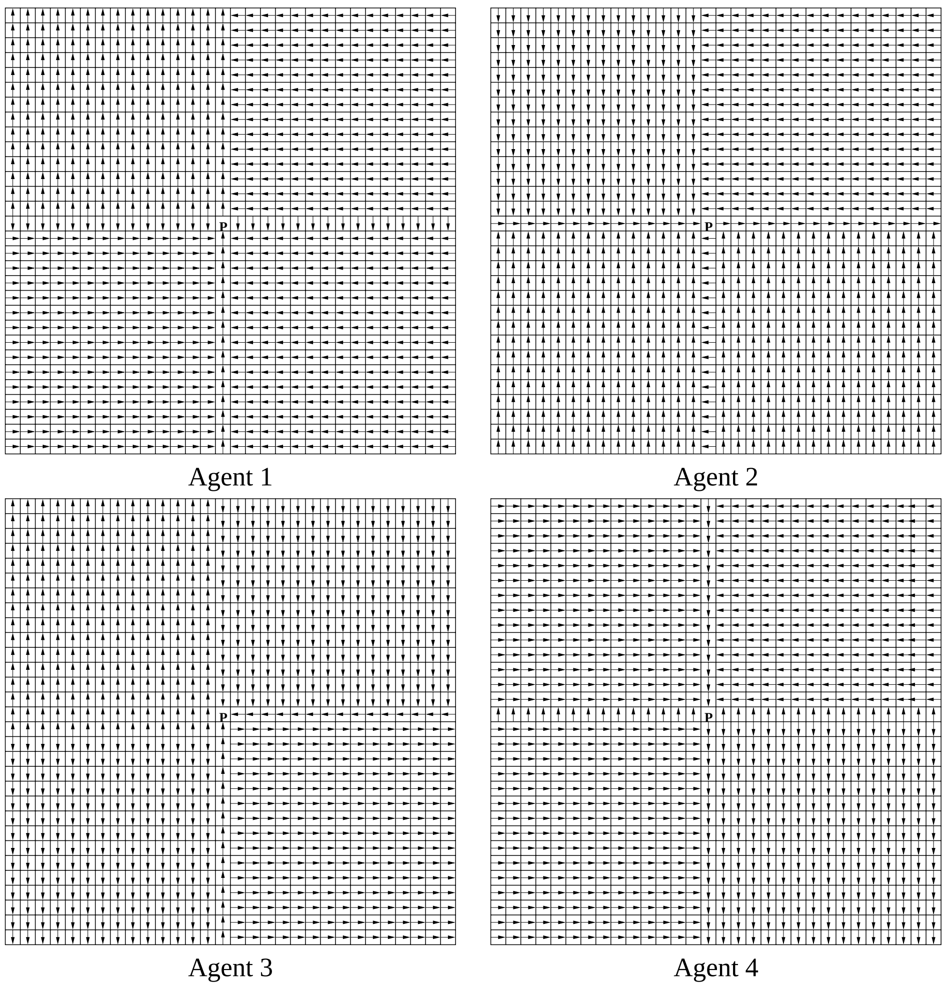


Figure 9. Strategies for MPHC-N

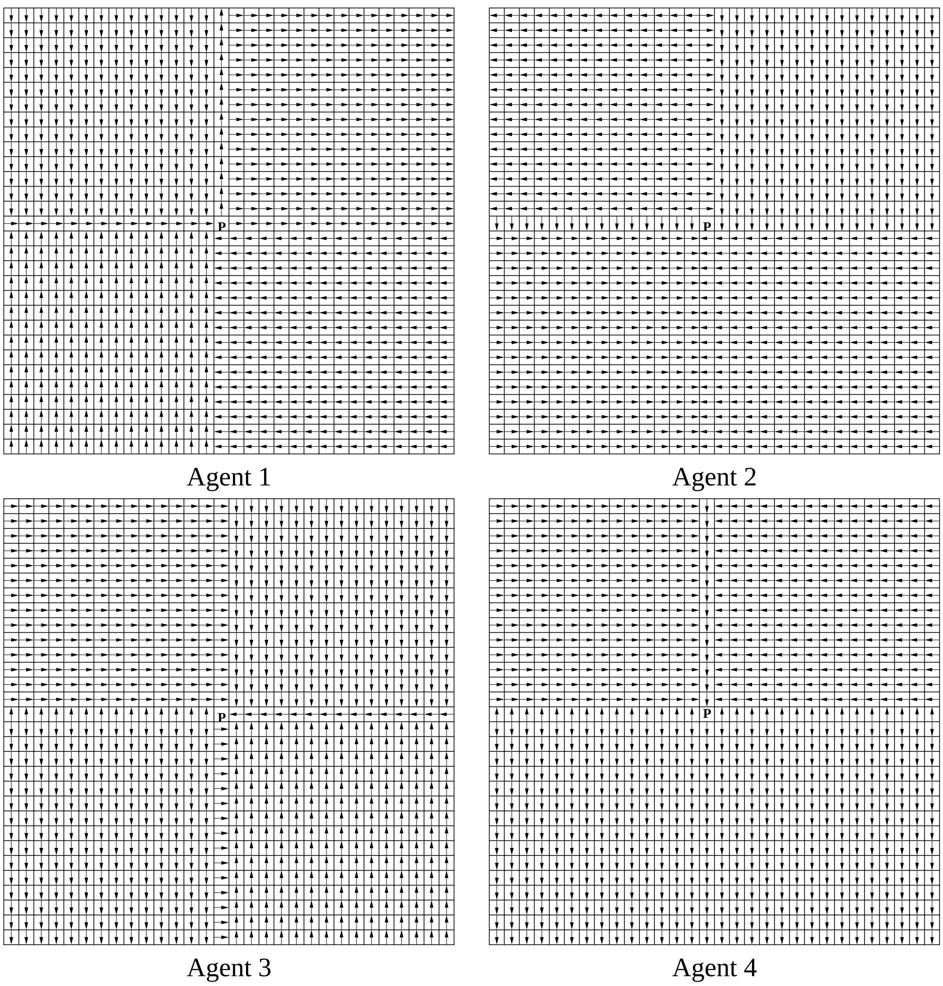


Figure 10. Strategies for MPHC-R

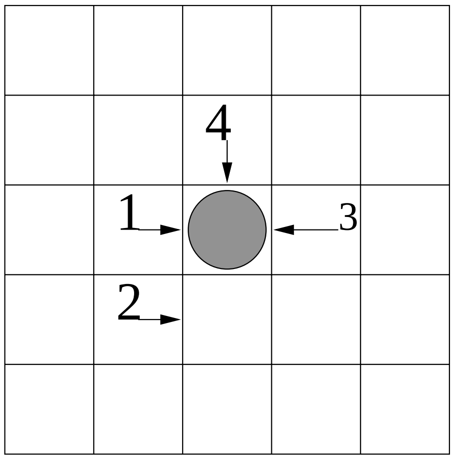


Figure 11. Before Capture

the agents stay in position. The deadlock is broken by the fact that the predators are faster than the prey. Eventually, the 10% probability that the prey will not moved is realized and predator 2 finally moves into position.

4.3. Analysis of Intermediate Results

Examining the intermediate steps between hill-climbs uncovers aspects of the system's behavior that cannot be seen from looking at the final results. For instance, is the multiagent credit assignment a problem in this domain? If so, this may explain the poor performance of A1 since it uses grouped learning. Within probabilistic hill-climbing, the credit assignment problem can be seen by comparing the intermediate results of a sequence of climbs. Abusing notation, let the predators return the following A^*'s, $\{A_1^*$=**A**, A_2^*=**B**, A_3^*=**C**, A_4^*=**D**$\}$, after stage one. Here the indices represent the predator. Stage two then returns $\{\mathcal{A}_1^*$=**A**,$\mathcal{A}_3^*$=**C**$\}$ as the interacting subset with the largest expected payoff. At a later point, let stage two return the group, $\{A_2^*$=**K**,A_4^*=**L**$\}$. A multiagent credit assignment problem would have occured if predator 2 had committed to it's $\mathcal{A}^*$ during the previous step.

During the experiments, the multiagent credit assignment problem was observed in several instances. For example, during the MPHC-N trial, predator 1 returned an assignment that changed the strategy for the southwest direction during stage one. This change was "filtered out" by stage two. Individually, the Δ-utility of this assignment was -8.47. Four climb steps later, a change in the southwest direction was again returned in stage one. This change was accepted. The action assignment would not have been available if the initial "southwest" assignment had been implemented. Since the problem cropped up using the all north initial condition, it was decided to rerun the experiment without stage two while using the same δ_1 value. The resulting strategy had a capture rate of only 51% in an average capture time of 78.5 steps.

5. Summary and Conclusions

This paper has presented a novel solution to the problem of learning coordination strategies for multiagent systems. Compared to existing techniques, multiagent probabilistic hill-climbing has advantages in terms of convergence, complexity and in performance.

1. *Convergence*
   Unlike some of the previous approaches, multiagent probabilistic hill-climbing is guaranteed to converge and to return a solution. This is done using a well-founded rather than ad-hoc termination criteria.

2. *Complexity*
   Unlike completely grouped techniques that are also guaranteed to converge, MPHC uses a mixture of independent and group learning. This reduces sample complexity but at the potential cost of performance.

3. *Performance*
   Although MPHC sacrifices optimality for reduced complexity, it has been empirically shown to produce better results than genetic programming on the well-studied predator and prey domain. In the symbol domain, the approach was able to return ϵ-optimal results with a high success rate given the appropriate parameters.

In summary, MPHC combines features of both independent and grouped approaches to learning multiagent coordination strategies. The key is a probabilistic criteria that ensures the convergence of independent learning and that allows for the quantification of beneficial interactions in the joint action assignments.

Currently, work is being performed to determine the effect of adding an additional predator to the predator and prey domain [1] In particular, the ability of the approach to incrementally deal with the presence of the new agent, without resorting to complete re-learning, is being examined.

Notes

1. Suggested by Prof. M. Huhns, 1997.

References

Berry, D. & Fristedt, B. (1985). Bandit Problems: Sequential Allocation of Experiments. London: Chapman and Hall.

Cammarata, S., McArthur, D. & Steeb, R. (1983). Strategies of Cooperation in Distributed Problem Solving. *Proceedings of IJCAI-83* (pp. 767-770). Karlsruhe, FDR: William Kaufmann.

Chien, S., Gratch, J. & Burl, M. (1995). On the Efficient Allocation of Resources for Hypothesis Evaluation: A Statistical Approach. *IEEE Transactions on PAMI,* 17(7), 652-665.

Dowell, M. L. & Stephens, L. (1996). Mage: Additions to the AGE Algorithm for Learning in Multiagent Systems. *unpublished manuscript.*

Durfee, E., Lesser, V. & Corkill, D. (1987). Coherent Cooperation Among Communicating Problem Solvers. *IEEE Transactions on Computers,* 36, 1275-1291.

Durfee, E., H. (1988). *Coordination of Distributed Problem Solvers.* Boston: Kluwer Academic.

Fong, W. L. (1995). A Quantitative Study of Hypothesis Selection. *Proceedings of ML-95* (pp. 226-234) Morgan Kaufmann.

Findler, N. V. & Lo, R. (1986). An Examination of Distributed Planning in the World of Air Traffic Control. *Journal of Parallel and Distributed Computing*, 3, 411-431.

Gratch, J. (1993). Composer: A Decision-theoretic Approach to Adaptive Problem Solving. UIUCDCS-R-93-1806. University of Illinois at Urbana-Champaign.

Grefenstette, J. (1992). The evolution of strategies for multi-agent environments. *Adaptive Behavior*, 1, 65-90.

Greiner, R. (1996). PALO: a probabilistic hill-climbing algorithm. *Artificial Intelligence*, 84, 177-208.

Haynes, T. & Sen, S. (1995). Evolving behavioral strategies in predators and prey. In S. Sen (Ed), *Working notes of the Adaptation and Learning in Multiagent Systems Workshop, ICJAI-95*.

Haynes, T., Sen, S., Schoenefeld, D. & Wainwright, R. (1996). Evolving a Team. *AAAI Fall Symposium on Genetic Programming*.

Haynes, T. & Sen. S. (1996). Learning Cases to Resolve Conflicts and Improve Group Behavior. *Working Notes of the AAAI Agent Modeling Workshop*.

Hoeffding, W. (1963). Probability Inequalities for Sums of Bounded Random Variables. *American Statistical Association Journal*, 13-30.

Ishida, T. (1994). *Parallel, distributed, and multiagent production systems*. Berlin: Springer-Verlag.

Jennings, N. R. (1996). Coordination Techniques for Distributed Artificial Intelligence. In G. M. P. O'Hare & N. R. Jennings (Eds.),*Foundations of Distributed Artificial Intelligence*. New York, NY: John Wiley.

Kaelbling, L. P. (1993). *Learning in Embedded Systems*. Cambridge, MA.: MIT Press.

Kamel, M.& Syed, A. (1994). A multiagent task planning method for agents with disparate capabilities. *Journal of Advanced Manufacturing Technology*, 9, 408-420.

Kirkpatrick, E. (1974). *Introductory Statistics and Probability for Engineering, Science and Technology*. Englewood Cliffs, NJ: Prentice-Hall.

Mataric, M. (1995). Designing and Understanding Adaptive Group Behavior. *Adaptive Behavior*, 4, 51-80.

Newell, A. & Steier, D. (1993). Intelligent control of external software systems. *Artificial Intelligence in Engineering*, 8, 3-21.

Parker, L. (1995). L-ALLIANCE: A Mechanism for Adaptive Action Selection in Heterogeneous Multi-Robot Teams. ORNL/TM-13000, Oak Ridge National Labs.

Rivest, R. L. & Yin, Q. (1994). Simulation Results for a New Two-armed Bandit Heuristic. In S. J. Hanson, G. A. Drastal & R. L. Rivest (Eds.), *Computational Learning Theory and Natural Learning Systems Vol. 1: Constraints and Prospects*. Cambridge, MA.: MIT Press.

Sandholm, T. & Crites, R., H. (1995). Multiagent Reinforcement Learning in the Iterated Prisoner's Dilemma. *Biosystems: Special Issue on the Prisoner's Dilemma*.

Sen, S., Sekaran, M. & Hale, J. (1994). Learning to coordinate without sharing information. *Proceedings of AAAI-94* (pp. 426-431). Seattle, WA: AAAI Press.

Sen, S. & Sekaran, M. (1995). Multiagent coordination with learning classifier system. *Working Notes of Adaptation and Learning in Multiagent Systems Workshop, ICJAI-95*.

Tennenholtz, M. (1995). On computational social laws for dynamic non-homogeneous social structures. *J. Expt. Theor. Artif. Intell.*, 7, 379-390.

Weiß, G. (1993). Learning to Coordinate Actions in Multi-agent Systems. *Proceedings of IJCAI-93* (pp. 311-316). Chambéry France: Morgan Kaufmann.

Received February 5, 1997
Accepted April 23, 1998
Final Manuscript April 10, 1998