Learning Coordination Strategies for Cooperative Multiagent Systems

by

Fenton Ho

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Systems Design Engineering

Waterloo, Ontario, Canada, 1997

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

This thesis proposes a new technique for learning multiagent coordination strategies that addresses the issues of convergence, complexity, credit assignment, and utility. Traditionally, strategies to control the behavior of multiple agents have been hand-coded to meet a designer's goals. This task is complex due to the interactions that can occur among agents. Recent work in this area has focused on how strategies can be learned. Yet, these systems suffer from a variety of problems that include lack of convergence or performance guarantees and from complexity concerns.

Following a formalization of the problem, a review of related works and a discussion of unresolved issues, a generic multiagent learning framework is presented. Then the basis of the proposed technique, probabilistic hill-climbing, is discussed and mapped into this framework. Implementation details are then described and experimental results on three different domains reported. Finally, an extension to reduce sample complexity is considered.

Acknowledgements

I would like to thank my supervisor, Professor Mohamed Kamel, for his patient support through all phases of this research. Thanks also goes to my examination committee: Professors Michael Huhns, Keith Hipel, Jan Huissoon, and Andrew Wong, for their criticisms and insights into this work.

My deepest heart-felt thanks go to those friends who kept me sane during these last four years of uncertainty and growth. Puiwing, Kai, Philip, Reda and Sharon, I'll cherish the times that we've had and look forward to spending my days in the "real world" with all of you. A special thanks goes to the Waterloo Chinese Christian Fellowship and my Sunday School "kids" at the Kitchener Waterloo Chinese Alliance Church. Thanks for letting me share my life with you and for sharing your lives with me.

I dedicate this work to my triune God. I confess that our relationship has been strained throughout these years but you've always been there to startle and confront me with who You are and who I am. I can't say that it's been always fun but You're a character.

Finally, I thank my parents for their sacrifice and their undying faith in me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

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

1. Complexity of the Domain: The complexity of the environment and/or the complexity of the tasks may render the design of a single monolithic system difficult.

2. Distributed nature of the Domain: Some domains, such as air traffic control [11], involve multiple autonomous entities.

3. Performance Requirements: Time constraints may require the use of more than one agent to meet objectives in domains that allow for parallel execution.

4. Fault Tolerance: Redundant capabilities in multiple agents would allow for goals to be achieved should a subset of the agents be damaged.

5. Leveraging Existing Components: Rather than building systems from scratch, intelligent agents can be used to coordinate the use of existing software [42].

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 the community acts in a coherent manner" [30]. The aims of the coordination process 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" [30]. Intuitively, coordination can be measured by the gain in system performance over naive collective behavior.

Some common approaches to coordination include organizational structures, the exchange of meta-level information, and multiagent planning.

- *Organizational Structures*

  An organizational structure is a method of defining the flow of information or control among a group of agents. The structure encodes the authority of each agent, its role and whom it can interact with. These can be used to achieve coordinated behavior. For instance, authority relations are means of fusing overlapping but possibly inconsistent data. Defined roles are methods of decomposing and assigning problems.

  Examples of systems that use organizational structures for coordination include DVMT [6] and those built on *social laws* [52]. In the latter approach, each agent adopts a set of social laws or conventions that dictate its behavior. One familiar example of a social law is coming to a halt at a four-way intersection. By convention, if two cars arrive simultaneously, then the one on the right goes first. This helps to avoid potential collisions.

In general, organizational structures are static, but work has been done to adapt the granularity of the agents assigned to each role [28] [29].

- *Metalevel Information Sharing*

  Coordination is achieved under metalevel information sharing by agents incorporating each other's goals, plans, and schedules into their own deliberations. Unlike the use of organizational structures, agents are free to change their roles and plans dynamically in response to the activities and anticipated activities of other agents. This style of coordination is exemplified by *partial global planning* (**PGP**) [7] and **CDS** [14]. In the latter approach, meta-reasoning is used to select the appropriate heuristics for coordination based upon the beliefs of the agents about expected interdependencies. Given the interaction probabilities, a decision theoretic framework can be used to formulate a "rational" solution to the coordination problem.

- *Multiagent Planning*

  In contrast to the use of organizational structures and metalevel information exchange, coordination in multiagent planning is achieved by following a complete, predefined plan. All the actions of the agents, as well as the synchronization points, are explicitly spelled out. As such, the agents have committed to a fixed course of action and are not free to change their behavior dynamically. Examples of multiagent planning systems include [3] and [32].

A commonality among the three approaches is that they are off-line and use extensive domain knowledge to derive the coordination strategies. Consider the task of designing collective behaviors for multiple behavior-based robots[37] [38], which is an instance of the organizational structure paradigm. Typically, the first step is to decide on a set of basic behaviors for each robot that will "span" the problem space. That is, this set of behaviors will allow the robots to solve the problems that they are

required to solve. The next step is to find a method of coordinating these individual behaviors across agents to arrive at the desired collective behavior. For instance, the collective flocking behavior is constructed by weighting the direction and velocity vectors from the local *safe-wander*, *disperse*, *aggregate*, and *home* behaviors in each robot. The weights are experimentally determined and are based on the dynamics and mechanics of the robots and the ranges of their sensors.

Tools have been developed to simplify the design process. A graphical, object-oriented approach has been implemented [38] to analyze the effects of different behavior configurations. Sequences of behaviors can be examined in the context of the possible preconditions that may have caused the behaviors to become active. Nevertheless, the task is complex and scaling is extremely difficult.

Moreover, two assumptions which underlie these techniques are the existence of a complete world model and the stationarity of this model. These conditions, particularly the first, are difficult to achieve in practice. Another issue is the computational complexity [18] of deciding the optimal coordination scheme given the nature of the interactions/dependencies between agents. The complexity of finding an optimal set of social laws for heterogeneous agents is [52] NP-hard. A similar statement is echoed by Parker [43]. Haynes suggests that "in most cases a coordination strategy is chosen if it is reasonably good" [24]. Parker concurs by stating the need for approximations [43].

1.1 Learning Coordination Strategies

Given the difficulties with hand-coding, model stationarity, completeness, and the cost of generating a good strategy, some researchers have developed systems that learn coordination strategies.

Learning can be viewed as the process of transforming the knowledge of an agent to achieve a particular goal [39]. Michalski's general learning process is shown in Figure 1.1.

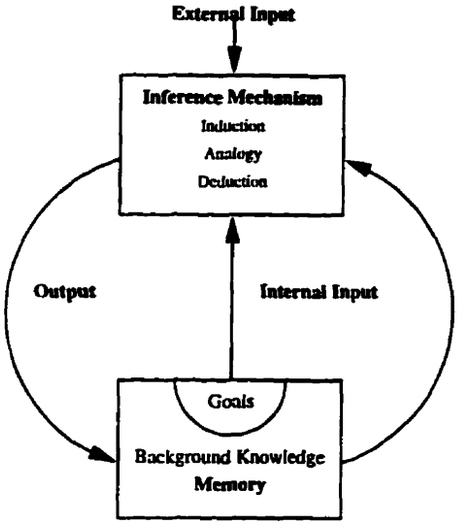

Figure 1.1: General Learning Process

The transformation procedure takes as its inputs training sample(s), the current knowledge of the agent, a set of potential transformations or inferencing methods and a set of goals. It then modifies the agent by either transforming and storing the sample(s) and/or by directly transforming the knowledge of the agent to meet the goals. For the multiagent coordination problem, the learning goal is to increase the coordination of the agents as measured by a global utility function. Since the goal is to increase performance and not to acquire new knowledge, the transformations will directly modify the agent's background knowledge. Within the learning community, this task is commonly known as a speed-up learning problem. The learning task can be formalized using the following definitions.

Definition 1: A problem solving strategy, Π_i, where i is the agent index, is a mapping from a problem context to an action. The form of the context and

action is dependent on the problem solver. For instance, it could be a fine-grained policy that maps a state to an action. At the other extreme, it could be a mapping from a problem description to a high-level solution method. The type of problem solver is not assumed.

Definition 2: A problem, $\mathcal{P}_l$, is an instance from $\mathcal{P}$, the stationary distribution of problems that will be encountered by the system. The stationarity assumption is required to make learning appropriate. If the system is constantly changing, then there is little point in learning. It can be argued that stationarity is rarely achieved due to issues such as system aging. Thus, stationarity in this context means that the distribution is evolving "slowly enough" for the results of learning to be useful. It is assumed that problems can be individually requested from an oracle.

Definition 3: A global utility function, $\mathcal{U}$, measures the performance of the entire multiagent system $(\Pi_1 \bigcup \Pi_2 \ldots \Pi_N)$, where N is the total number of agents, on $\mathcal{P}_l$. This function can take into account factors such as the cost of communications, the utilization of the agents, and the total number of steps required to solve a problem. This function does not have to be evaluated by a single overseeing agent. It can also be decomposed or distributed into local functions that reflect the contributions of an agent to global utility. The implication is that by maximizing these localized measures the global measure will also be maximized. A local as opposed to localized utility function is not required to meet this requirement. As a result, the agents may compete rather than cooperate.

Definition 4: A set of transformations, $\mathcal{T}_i$, is associated with each strategy Π_i.

Definition 5: A combined transformation, $\mathcal{C}_j$, j=1 $\ldots \mathcal{M}$, where $\mathcal{M}$ is the total number of combined transformations, is made up of individual transformations

on the part of each agent

$$\mathcal{C}_j \in \mathcal{T}_1 \times \mathcal{T}_2 \ldots \mathcal{T}_N$$

A key difference between multiagent and single agent systems is the assumption that there are interactions or interdependencies among agents. Otherwise, the domain can be modeled as a set of disjoint subdomains involving individual agents. Interdependencies can arise both "physically" and "mentally" [14]. In the first case, the agents could be sharing a resource, while in the second, they could be relying on each other's capabilities or knowledge. Mental interactions can also take the form of conflicts in desire, commitments and interests. Due to the interactions, transformations over **N** agents may only be beneficial if all **N** are simultaneously implemented.

Definition 6: A sample, $(\mathcal{C}_j, \Pi, \mathcal{P}_l, \mathcal{U})$, is the utility of $\Pi' = \mathcal{C}_j \otimes \Pi$ on problem $\mathcal{P}_l$ where $\otimes$ denotes the application of the combined transformation $\mathcal{C}_i$ on the complete strategy Π. It is assumed that samples are independent.

1.2 Statement of Research Problem

The goal of the research is to develop a method to solve the multiagent coordination learning problem as formalized in Figure 1.2. Using the general learning paradigm, training samples from $\mathcal{P}$ are used to guide the transformation of the $\Pi_i's$ such that the overall performance of the system is increased. The definition of "high" will be left up to the learning algorithm.

There is nothing particularly multiagent about the definition in Figure 1.2 except for the existence of multiple agents. Coordination is achieved through the use of a global utility function. If there are interdependencies in the domain, the goal of

Given $\mathcal{P}$, a stationary distribution of the problems that will be encountered by the system

$\mathcal{U}$, a global utility function over the solutions

$\Pi_1, \Pi_2, ...\Pi_n$ a problem solving strategy for each agent 1, ..., n

A set of potential transformations $\mathcal{T}_1, \mathcal{T}_2, \ldots \mathcal{T}_n$ on $\Pi_1, \Pi_2, \ldots \Pi_n$, respectively,

Find $\mathcal{C}^{\max}$ such that the expected utility of $\mathcal{C}^{\max} \otimes \Pi$ over $\mathcal{P}$ is "high".

Figure 1.2: Research Problem

optimality dictates that the agents work together. On the other hand, if the domain is decoupled and the agents are independent, then the definition would describe a collection of additive single-agent learners. Note that there are also no assumptions about the knowledge and/or capabilities of the agents. They could be either homogeneous or heterogeneous. There are also no assumptions about the form of the problem distribution.

Beyond the issue of optimality, there are also other criteria that can be used to evaluate the effectiveness of a particular learning algorithm. They include:

1. *Sample Complexity*

   One measure of how efficient a learning method is, is the number of training instances that it needs to learn a concept. This measure has different instantiations depending on the learning method. For instance, in reinforcement learning, it is a function of the number of trials required until convergence, and in genetic programming, it is a function of the population size, the number of fitness evaluations per generation, and the number of generations.

2. *Convergence*

   A basic requirement for all learning algorithms is that they converge. Recall

that, under the transformational model, learning is the transformation and addition of new knowledge or the transformation of existing knowledge. Theoretically, this means that a transformation can be applied and then reversed. As result the system can oscillate between a set of states and thus fail to halt.

3. *Scalability*

   An issue particular to multiagent learning systems is whether the approach scales when the number of agents is increased. Techniques that seem promising with a small number of agents might be computationally intractable or ineffective given a larger number of agents. Thus the use of scaling here includes both a cost as well as an efficacy dimension.

4. *Performance*

   The ultimate goal of any learning algorithm is to return a problem solver whose performance is optimized in some sense. This leads to the related questions of "what kinds of performance guarantees are afforded by the method?" and "what is the confidence that these guarantees will be achieved?"

To narrow the scope of the problem, the coordination approach will be restricted to an instance of the organizational structure paradigm. Each agent, i, is supplied with a problem solving strategy Π_i that it executes in a top down manner. This is done using a forward chainer where each agent executes the first action in the strategy that has its preconditions fulfilled. Implementation details can be found in Chapter 3. It is also assumed that no explicit communication occurs between agents. Agents take note of the presence of other agents by how they interact with the environment. Finally, it is assumed that the agents are cooperative and will defer to each other in order to fulfill the goal of increasing global utility.

1.3 Overview of Thesis

The remainder of the thesis is organized as follows.

Chapter 2 is a review of current research that is relevant to learning coordination strategies. A broad range of learning techniques is discussed along with their inherent limitations.

The bulk of the original work developed for this thesis is presented in Chapter 3. A new multiagent coordination learning scheme based on probabilistic hill-climbing is discussed. The work is situated within issues that underlie the generic problem of learning multiagent coordination strategies. In particular, the problem of multiagent credit assignment is considered.

Results of experiments, performed on three domains, are reported in Chapter 4. One of the tests assesses the performance of the approach in a controlled environment. The other domains directly compare the method to two existing techniques.

Chapter 5 discusses an extension to multiagent probabilistic hill-climbing that can decrease sample complexity. The underlying theory is presented and the technique is validated using a simple restaurant domain.

The final chapter highlights the contributions of this research and proposes further directions in which it can be extended.

Chapter 2

Related Works

The problem of learning multiagent coordination strategies is relatively new. Traditionally, coordination strategies have been developed off-line, but there has been growing interest in applying learning techniques as witnessed by recent workshops and symposia [48] [49]. This chapter will survey the current state of the art in learning multiagent coordination strategies.

The existing techniques are classified, based on the learning methods that they use, into either *explanation*-based, *reinforcement* based, *instance* based, *genetic programming* based, or *case*-based. A brief description of these learning methods is provided and then a few examples of systems that fall under each category are discussed.

2.1 Explanation-Based Learning

The task of explanation-based learning (**EBL**) is to operationalize, or convert to a more computationally efficient form, knowledge that is already known by the agents. Typically this is done by proving that part of an execution trace is an instance of a concept and then rewriting the concept using the proof such that it can be easily

classified. For instance, given an example of a failed planning trace and the concept of failure, first prove that the plan is a failure and then use the proof to quickly classify situations that could lead to failure and thereby avoid them. Another classic example is Winston's cup domain, where the task is to learn a useful conceptual definition for a cup. One high-level definition is that the object holds water and is graspable. However, this may be too abstract to allow for recognition. Given an example of a cup, the theory can be operationalized by rewriting it in terms of the observable characteristics of the example. A definition of a cup as any object that is convex and has a handle might be more more useful than the previous definition. The danger is that the new definition may be so specific that it misclassifies positive instances or it may still be too general to be efficiently computable. This leads to the issue of utility, which is discussed below.

EBL has been applied by Sugawara and Lesser [50] to learn plan modifications that avoid conflicts during coordination. The steps in deriving the modifications are as follows:

1. *Tracing the "mainstream".*

   Since **EBL** requires the entire problem solving trace, it must be pieced together from the contributions of each agent to the final solution. Note that the "mainstream" differs from normal **EBL** problem traces in that only those inferences that contributed to the solution are kept. This is just a matter of semantics, but it does effect the way errors are defined.

2. *Detection of LAPS.*

   LAPS are *learning analysis problems* or error contexts where agents have the potential for improving system performance. Some problems that can be detected are:

   - Long delays in understanding communicated information.

- Redundant activities in the "mainstream".
- Long delays in one of the "mainstream" activities.
- Redundant activities not in the "mainstream" that are costly (e.g., backtracking).
- Unused variables.

3. *LAP Analysis*

   The task of **LAP** analysis is to discover the causes of the problems and then to propose remedies. **LAP**s are caused by the lack of nonlocal knowledge and/or the lack of rules or efficient rules to interpret this knowledge.

4. *Propose Modifications*

   Given the causes from the previous step, modifications must be made to the agent's control knowledge to remove the errors. Typically this may require that multiple agents coordinate changes to achieve the desired effect. Individual agents may perform the following modifications:

   - Substitute another mid-level plan that achieves the same effect since the reasoning method is based on skeletal planning.
   - Postpone an action until relevant information is received from another agent or until the "correct" state is achieved.
   - Change the order that messages are sent.
   - Obtain information from another agent if it takes too long to compute it locally.

5. *Negotiation*

   If the proposed modifications cannot be accepted by all agents, then negotiation is required to reach a compromise. For example, agents can choose locally suboptimal solutions such that it may be possible to achieve a complete solution.

6. *Precondition Identification*

   The final stage of **EBL** is to identify when the new knowledge should be applied. In multiagent settings there is the added difficultly that components of the preconditions might not be local.

One problem with **EBL**-based approaches is the utility of the changes. Recent work [15] has indicated that supposedly good operationalizing transformations, measured over a single learning context, may lead to decreased performance over a set of queries. As a result, research in **EBL** has concentrated on measuring the utility of these changes [15] [21]. This work deviates from traditional **EBL** in that it requires a number of samples to quantify the utility.

The utility problem arises because the learner cannot decide between alternate operationalized theories. This is not to say that any of the alternatives is incorrect, but that some should be used before others. For instance, in a domain that uses paper cups, the definition that a cup is convex and made of paper is more useful than a cup being defined as convex and having a handle. Thus the "operationality" of a definition is highly dependent upon the problem distribution.

2.2 Instance-based Learning

Instance-based learning algorithms [1] (**IBL**) are supervised learning methods which can be traced to nearest neighbor pattern classifiers. Rather than using generalizations of instances to classify samples, **IBL** algorithms use selected instances as concept definitions for each class. **IBL** algorithms are comprised of the stored instances for each class, a similarity function, a classification function, and a concept description updater function [1]. A brief description of each function follows:

1. *Similarity Function*

   Given a labeled training instance, i, it computes the similarity between i and the instances in the concept descriptions.

2. *Classification Function*

   Given the results of the similarity function and the previous classification performance of the instances in the concept descriptions, it returns a classification for i.

3. *Concept Description Updater*

   Given i, the similarity value, the history of the classification results and the instances in the current concept definition, it decides how to modify the concept description.

Prasad et al [44] have used **IBL** to learn which one of 5 different coordination mechanisms should be applied in a given context. This research is situated within the partial global planning paradigm with the consequence that these mechanisms are high level. For instance, one of them decides at what level of detail results are communicated.

Learning involves testing the performance of each method on a set of trial problems. This builds up a set of triplets made up of a global situation vector, a coordination algorithm, and its performance within this context. A situation vector is formed by aggregating the local situations of each agent. This is individually done by each agent through integrating the communicated status of other agents with its own local perspective. The system performance measure is built up in a similar manner. Each agent stores its own set of triplets.

Once learning has terminated, the triplets are used to classify a new problem into one of the 5 methods. First the neighboring instances of the problem are determined using the similarity measure. Then a classification function based on the weighted previous performance of the neighbors is used to return one of the 5 methods.

Although the method learns context dependent multiagent coordination strategies, the learning takes place within a single agent framework. The multiagent component of the task is to aggregate the local situation information and performance measures.

2.3 Reinforcement Learning

Reinforcement learning is a learning approach where an agent discovers a mapping from situations to actions so as to maximize the value of a scalar reward or reinforcement signal. In contrast to other forms of learning, such as classification tasks, the agent is not told what the target actions should be, but instead must rely on trial and error to find which actions produce the highest reward. An example of a reinforcement learning problem is learning how to navigate a hallway to reach a certain location. For this task, the agent receives positive reinforcement only when it has reached the goal and negative reinforcement when it collides with any wall. Since the agent does not initially know the correct mapping, it may choose actions that result in collisions until it discovers the optimal policy. This combination of delayed reward and the use of exploration are two distinguishing characteristics of reinforcement learning.

- **Bucket Brigade with Shared Information**

  One approach to learning coordinated actions between agents is Weiß's ACE (**Action Estimation**) [56] and AGE (**Action Group Estimation**) [57] algorithms. Based on Holland's bucket brigade credit assignment model, each agent bids for the right to execute its actions. In the case of ACE these are independent actions on the part of each agent, while for AGE they are for a joint set of actions. Each bid for agent i's jth action is formulated as

$$B_i^j[\mathcal{S}] = \begin{cases} (\alpha + \beta) E_i^j[\mathcal{S}] & E_i^j[\mathcal{S}]\, \theta \\ 0 & \textit{otherwise} \end{cases}$$

where α is a small constant called the risk factor, β is a small random number called the noise term, θ is a constant called the estimate minimum, and $E_i^j[\mathcal{S}]$ is agent i's estimate of action j's utility in state $\mathcal{S}$. The risk term, α, represents the portion of E_i^j that agent i is willing to risk to execute action j. The noise term, β, reduces the risk of getting caught in a local minima.

After the agents have finished bidding, the agent with the highest bid is allowed to execute the action associated with this bid. All other agents then withdraw any of their bids with actions that are incompatible with the chosen action. The process repeats until no other action can be executed. A bucket brigade assignment scheme is then used to update the utility of the actions. At every stage, the agent that won the bid reduces the utility estimate of its winning action by its bid and passes this value to the previous winners.

The AGE algorithm is similar to ACE except that actions are grouped into compatible sets. For each set, the bids of each agent with an action in the set are summed. The set with the largest summed bid is then executed.

A drawback of the ACE algorithm is that it does not function in parallel. Agents must sequentially bid on the actions to be performed. This eliminates the potential performance improvements available with multiple agents, and does not take into consideration beneficial interactions between actions. AGE addresses this limitation by allowing for groups of operators to be considered. However, this raises the issue of computational complexity, since every combination of compatible actions must be considered as a group. The number of groups is bounded by the product of the number of applicable actions per agent in a given state. Moreover, there is the overhead of broadcasting the bids to all the agents and, once the bids have been received, of determining if the group is compatible. Dowell [5] has developed an addition to AGE that learns if a group of actions is compatible in a particular context. Only those that are

compatible are bid on. This may reduce the bidding overhead, but the added cost of learning the incompatible actions and filtering the groups must also be factored in. It is unclear if this method provides any benefits.

- **Q-learning**

  Another reinforcement based approach to learning coordination strategies is Sen's [46] [47] use of independent Q-learners. Unlike Weiß's work, there is no communication between agents. Instead, agents treat each other as if they were part of the environment. Sen contends that some advantages of this approach include: robustness to communication delays and to failures of key agents, and tolerance to unreliable or misleading information. The use of independent learners can also reduce the computational complexity of the coordination learning task, since the examination of useful action groups is implicit in the joint behavior of the agents. Permissible groupings are not explicitly constructed. Moreover, the approach does not make any assumptions about the nature of the agents. They can be cooperative or antagonistic. The behavior of the system is totally dependent on the local utility functions.

  As in the standard reinforcement learning paradigm, Q-learning seeks to learn the utility of a given action **a** in state **s**. These utility values are called Q-values. For each agent i, in state **s**, the action **a** with the highest Q-value $Q(\mathbf{s}, \mathbf{a})$ is chosen for execution. The Q-values of the executed actions are then updated using the following rule:

$$Q(s,a) \longleftarrow (1-\beta)Q(s,a) + \beta(\mathcal{R} + \gamma \max_{a' \in A} Q(s',a'))$$

  where β is the learning rate and γ is the discount rate. The γ term allows delayed reinforcement values to be propagated to actions earlier in the chain.

  Multiagent Q-learning has been applied to a two-agent block-pushing problem [46] and to a four-agent robot navigation task [47]. In the first example, two

agents are assigned to push a block from an initial position to a goal position. Neither agent is aware of the capabilities or the existence of the other agent. The performance of the system was evaluated by measuring the number of trials required for it to "converge". Convergence is achieved when the block reaches its goal or if the number of steps exceeds a threshold.

The second task involved four agents navigating from one side of a 10 by 10 gridworld to opposite positions on the other side while avoiding collisions. Each agent was supplied with a local utility function that considered if an agent was involved in a collision. This domain will be examined in detail in Chapter 4.

A major weakness of the multiple, independent Q-learner approach is that it is not guaranteed to converge. One of the fundamental assumptions underlying Q-learning is that the world remains stationary. However, since the agents interact while modifying their behaviors, this stationarity requirement may be violated. Consider the following contrived example. Let there be three agents, each with only one state. They have been assigned three distinct symbols, one of which they emit during a given trial. The assignments are shown in Table 2.1. The utility function is global with the payoff received dependent upon the

Agent	*Symbols*
1	**A B C**
2	**D E F**
3	**G H I**

Table 2.1: Assigned Symbols

combined output of the agents and the current state of the world. The world has 3 equiprobable states and for each state there is only *one* combination of agent outputs which has positive utility. Possible world states and the required

agent outputs for positive reinforcement are shown in Table 2.2 Note that the outputs are listed in agent order with *?* referring to *don't care*. For instance, (**A D** ?) means that agent 1 and 2 must output symbols **A** and **D** respectively while it doesn't matter what agent 3 outputs. Given the standard formulation

World State	*Required Outputs*
1	(**A D** ?)
2	(? **E H**)
3	(**C** ? **I**)

Table 2.2: Positive Reinforcement Outputs

for Q-learning, the value of γ can be set to zero since there is only one state.

As formulated this problem will not converge to a steady solution using Q-learning. Instead the agents will oscillate between the three positive utility outputs. That is, the combined agent outputs will constantly shift between one of the outputs in Table 2.2. Convergence can be forced by halting learning after the agents have produced the same behavior for a pre-determined length of time. However setting this length is ad-hoc since there is no criteria for assessing the optimality of the solution.

On the other hand, the system will converge if one of the agents settles on a particular symbol. For instance if agent 1 settles on **A** then agent 2 will settle on **D** while agent 3 will "wander". This reflects the positive utility output for world state 1. In this case convergence means that the average utility will remain constant rather than the output being constant. Two of the agents will always emit the symbols corresponding to one of the high utility states while the third is free to vary.

- **Distributed Q-learning**

  DRLM (**Distributed Reinforcement Learning Model**) [22], is a Q-learning approach that starts from a different set of environmental assumptions than those found in [46] [47]. The two major differences are that the agents are allowed to communicate and that perceptual aliasing [58] is the problem.

  Actions within DRLM are divided into either *local* or *transfer* actions. As its name implies, *local* actions are those actions that are directly performed by the agent, whereas *transfer* actions transfer the control of a task to other agents.

  Agents communicate to obtain reinforcement and state information. Reinforcement is received when a *local* action is performed. An assumption of DRLM is that actions are individually rewarded or penalized. Thus an agent that performs a *transfer* action must make a request that it also receive the reinforcement signal when a *local* action is executed. Communication is also used to discern which agents transferred a task.

  Perceptual aliasing occurs when an agent is unable to distinguish between two world states using its direct perception. This can occur due to incomplete perception, where the agent's sensors; either physical or logical, are unable to sense the distinctions or where the differences cannot be directly sensed. In multiagent settings, the second case can arise when there are interdependencies between tasks such that the local information of each agent is insufficient to account for the state of the entire system. DRLM deals with these issues by including two stochastic models that handle incomplete perception and interdependencies respectively. A block diagram of the system is shown in Figure 2.1. The **HTM (Hidden Task Model)** maps the agents that transferred the task to a set of hidden tasks. This is done using a probabilistic model that associates an observable set of tasks to the cross-product of all potential tasks across agents. The one to many mapping uncovers tasks that cannot be directly

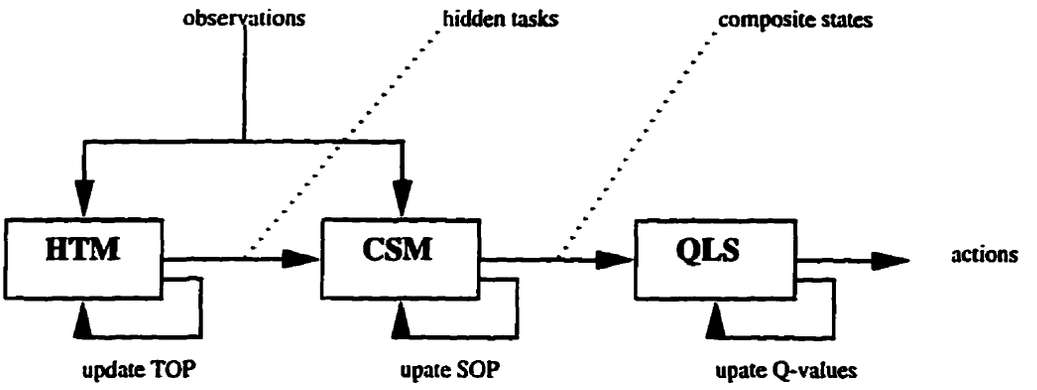

Figure 2.1: DRLM Block Diagram

observed. Next, the hidden tasks and observations from the world are used to construct a composite state that makes explicit the dependencies. Given the transfer agents, the composite state is estimated by first hypothesizing which state each agent is in and then by using the independence assumption to compute the joint probability of being in a composite state. Finally, it is this state that is used to index the Q-values.

DRLM contains more than one learning task. The first is standard Q-learning which maps state to action. The additional task is to update the probabilities in the task models. This is done using a variant of the Baum-Welch expectation maximization algorithm which is used to estimate the p.d.f of the distribution. The actual state that is passed onto the next stage of algorithm is biased towards the state with the highest probability. This is done using a stochastic selector based on the Boltzmann distribution. The size of these models, particularly the composite state model, can be very large since it is composed of the cross-product of the possible agent states.

- **Other Systems**

  As can be discerned by the number of reinforcement learning systems that

already have been discussed, a great many of the existing approaches fall under this paradigm. Other examples include [37] which uses local progress estimators in addition to goal functions, [13] which learns whether to be selfish or to follow the group, and [43] which uses windowed averages to learn how to prioritize between sets of behaviors rather than individual behaviors.

2.4 Genetic Programming

Genetic programming (**GP**) is a learning approach that uses a genetic or "evolutionary" paradigm. An initial population of individuals, in this case, programs, is subjected to a "fitness" or utility test. Those that perform well are selected to undergo "genetic operations" such as crossover, mutation, and simple reproduction.

Crossover involves exchanging "genetic material" between two programs. If both parents have high fitness scores, it is hoped that the resulting individual will inherit the good traits of its predecessors. Mutation is the random modification of the program. This provides a stochastic component to the learning process, thereby reducing the likelihood of it being trapped in a local minima. Finally, reproduction generates copies of an individual without modifying its program.

These operations are performed in cycles called "generations". During each cycle the performance of the individuals, or their "fitness" is measured. Those that pass are subjected to genetic operations and then promoted to the next generation. On the other hand, those that fail are discarded.

Haynes et al [23] [25] has used genetic programming to evolve a coordination strategy for the predator and prey pursuit game. In this task, four predators must coordinate to capture the prey within a toroidal grid world. The prey is captured when it is surrounded by four predators, since diagonal movement is not allowed. There is

no explicit communication between the predators and thus they cannot negotiate a capture strategy. Fitness is measured by a function that takes into account keeping close to the prey, surrounding the prey, and maintaining a position around the prey. These factors are necessary to provide feedback in the event that the prey is not captured.

Haynes' original experiments [23] evolved a single program that was used by all the agents. This assumes that all the agents are homogeneous and can also be viewed as a form of implicit communications. Later work focused on evolving different programs for each individual agent. Rather than evolving each one individually, a team approach was taken. This is a consequence of the credit assignment problem, which is the task of dividing the fitness results across agents. The agents act simultaneously and thus it is uncertain how each agent contributed to the fitness score. This problem will be discussed in detail in Chapter 3.

Earlier, AGE was criticized for the complexity of group-based methods. However the same criticism cannot be made of **GP**'s since there is a fixed population size. The potential combinatorics of examining the possible groups is reduced by stochastically generating a fixed number of combinations.

2.5 Case-based Learning

Case-based learning (**CBL**) [35] can be viewed as an extension of **IBL** that also allows for the adaptation of the instances. Consider an **IBL** problem where there is a solution method rather than a class label associated with each instance. When given a problem, the task becomes one of finding a matching case or stored instance and then returning the method. In addition to retrieval, **CBL**, allows for the method associated with the most similar case to be adapted to the current situation when an exact match is not found.

Haynes [24] has also applied **CBL** to the predator and prey domain. Unlike his previous work on genetic programming, cases are used to modify the behavior of a greedy strategy based on the *Manhattan Distance*(**MD**) between predator and prey [36]. The **MD** approach moves the predator so as to minimize this distance with all ties being randomly broken. Given a list of possible actions, the metric can be used to order them.

One problem with the **MD** algorithm is that it is susceptible to deadlock situations where the predators block each other from surrounding the prey [24]. To prevent this, cases are used by each agent to eliminate actions from the default **MD** ordering. A case, within this framework, is a mapping from a template situation to a rejection rule. If the current situation matches the template, then reject the action from the default strategy. This continues until an action from the strategy can be executed or until all the actions have been eliminated. In the latter case, Haynes' system executes the first action in the default strategy. Cases are learned when expectations are not met, such as when an action ends in a collision.

Haynes' use of **CBL** suffers from the "utility problem" in that the usefulness of the learned cases is uncertain. This stems from two factors: uncoordinated learning and uncontrolled learning. Learning is uncoordinated since each agent learns its own cases. This approach assumes that the other agents' strategies remain static, which is the same problem that plagues independent Q-learning. Again, the assumption is invalidated by the fact that the agents are allowed to learn simultaneously. Moreover, the learning is uncontrolled since a case is learned for each deviation from expectations. It is well known that the utility of generalizing an instance, in this case a template to a rejection rule pair, cannot be estimated from seeing a single instance [41] [15]. Thus, the utility of this technique beyond the tested domain is questionable.

2.6 Summary

The techniques that have been applied to learning multiagent coordination strategies are diverse and cover a large portion of the existing machine learning methods. Major observations and criticisms gleaned from analyzing these systems are summarized below:

1. **Considering Interactions**

   Two approaches to accounting for the interactions among transformations are grouping and the use of independent learners. The use of grouping can be combinatorial. In the case of AGE, if there are $\mathcal{N}$ agents with $|\mathcal{T}|$ transformations each, then the maximum number of groups is $|\mathcal{T}|^{\mathcal{N}}$ - 1. DRLM's construction of a composite state can also be viewed as a form of grouping. However, Haynes' use of grouping in the predator and prey domain does not theoretically suffer from this problem, since the groups are stochastically generated using a fixed population size. However, the size of the population will have a great impact on the solution.

   As discussed, grouping can occur at several levels. In the case of AGE and DRLM it is at the level of actions for a state, and in the case of Haynes it is at the level of an entire agent for the problem distribution. The latter's use of agent grouping begs the philosophical question "Is this multiagent learning?"

   The answer is not clear. **DAI** problems solvers have generally been characterized by their "relative autonomy and adaptability" [8]. This does not seem to be the case in agent grouping. Once the agents have been grouped, the problem appears to be a "single agent" learning task since the goal is to evaluate the utility of each atomic group. Transformations are applied to the group or in the case of genetic programming, crossover takes place between groups. This raises the question "Is group learning multiagent learning or an application of

learning techniques to multiagent domains?"

Arguably, the use of action grouping does not suffer from the same criticism since the agent can be seen as coordinating to ascertain the effects of a combined action. The agents are distinct and still distinguishable. The difference between the two types of grouping may be subtle but it calls attention to the need for a better definition of multiagent learning.

The use of multiple, independent learners does not suffer from the same combinatorial or philosophical problems as group learning but existing implementations are plagued by issues such as convergence.

2. **Multiagent Credit Assignment**

   An underlying problem behind systems with components that can act in parallel is how to allocate credit or blame to a component when global performance either increases or decreases. The existing approaches to this problem either equally allocate the credit, use a domain theory to identify the cause, or consider components only in groups. Since the task of learning is to identify a set of beneficial transformations, it is important to correctly identify the contributions. This issue will be discussed in detail in Chapter 3.

3. **Performance Guarantees**

   Many of the existing coordination strategy learning methods have vaguely defined notions of performance. In particular, **EBL** and **CBL** have inherent utility problems, while independent Q-learning is not even guaranteed to converge. Learning systems that can guarantee performance as good as, if not better than, the original system are desirable.

Chapter 3

Multiagent Probabilistic Hill-Climbing

This chapter describes the proposed system in detail. The first section is a general discussion of issues that must be resolved when implementing an approach that learns multiagent coordination strategies. Next, probabilistic hill-climbing is presented, followed by an examination of how it can be applied to multiple agents.

3.1 Learning Multiagent Coordination

The problem definition in Chapter 1 can be directly rewritten as the generic coordination learning algorithm shown in Figure 3.1. It is an on-line procedure that generates the potential transformations for each agent, combines them, and then assesses the combinations. The method then computes the best candidate from the set and, if the candidate passes a predefined criterion, the algorithm halts. Otherwise, the assessment process continues.

There are also variants on the approach, such as moving the transformation gen-

LEARN-COORDINATION($\Pi = \Pi_1, \Pi_2, \ldots, \Pi_N$)
  $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N\}$ = generate-transformations(Π) (1)
  loop while not *flag* do
    $\mathcal{P}_i$= retrieve-problem($\mathcal{P}$)
    $\mathcal{C} = \{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_M\}$= combine($\mathcal{T}$) (2)
    assess($\mathcal{C}$) (3)
    $\mathcal{J}$= compute-best($\mathcal{C}$) (4)
    flag= terminate?($\mathcal{J}$) (5)

Figure 3.1: Generic Coordination Learning Algorithm

eration inside the loop and performing batch learning by storing a fixed number of sample problems before assessment.

Placing the generation step within the loop allows for the dynamic inclusion of transformations. That is, they can be produced in response to the problems that have been encountered. Given that the system is required to solve problem **A**, only combined transformations $\mathcal{C}_4$ and $\mathcal{C}_7$ may be useful to the task. This can reduce the number of transformations and hence the cost of computing their utilities, but requires knowledge to discern which transformation may be effective for a given problem. The relationship between problem, transformation and effect might not be readily known, and thus a knowledge deficient solution is to generate all transformations so as not to miss a potentially useful one.

On-line procedures are preferred over batch approaches, unless the termination criteria is static or unless the problems can only be obtained in batches. Recall that the second condition has been ruled out by the assumption that problems are available individually. An example of a static criterion is to terminate after a fixed number of samples or generations in the case of genetic programming. On the other hand, a

dynamic criteria is based on the characteristics of the current best candidate. Thus, there is the potential for a dynamic scheme to halt before a static one. This requires a stronger termination criteria than just the number of samples. Note that a fixed sample method can also be used on-line.

The steps in Figure 3.1 have been numbered and will be discussed below:

1. *Computing Transformations*

   Given an initial multiagent problem solving strategy, Π, the first task is to compute a set of potential transformations. Two related decisions have to be made at this point. They are the "size" of the transformations as measured by the maximum syntactic distance that a transformed solver can be away from the initial one and what portion of these potential transformations should be included. In concert, these factors determine the nature of the learning algorithm and its performance limits.

   Consider the transformation of sequence {**B D A C**} into {**A B C D**}. This can be done by making a series of one-step moves, such as placing **A** in front of **B**, or a single two-step move which puts **A** in front of **B** and **C** in front of **D**. The second case can be regarded as a composition of two one-step moves. These examples correspond to transformations of size 1 and 2, respectively. Note that the set of size 2 transformations encompasses the set of size 1 transformations since ϕ, the null transformation, can be one of the composed steps.

   Now, let S_{max} be the maximum distance that a transformed solver can be from the initial solver. If the maximum size of the transformations is below S_{max}, then the algorithm has to be iterative if any notion of optimality is to be maintained, since not every solver configuration can be reached in one step. As such, `compute-best` can only be with respect to this iteration. However, iteration does not guarantee optimality since any distance-limited learning algorithm

has the potential to be trapped in a local minima. Another issue is the number of transformations to assess. For many domains, computational complexity rules out using the complete set of $\mathcal{S}_{max}$ transformations. Either a portion of the $\mathcal{S}_{max}$ set or an entire sub-$\mathcal{S}_{max}$ set is typically used. In the first case, a reduction in the "width" of the transformed solver space that is explored also contributes to the possibility of suboptimal performance. Combinations of both types of restrictions are also possible.

If an iterative version of the algorithm is used, then the LEARN-COORDINATION procedure is repeated until another, possibly related, termination criteria is met. Transformations for a subsequent iteration are generated with respect to the strategy that was promoted during the current iteration. More than one transformed solver can also be passed on.

2. *Computing Combined Transformations*

   Once the transformations have been computed, a method must be devised to determine if certain combinations interact beneficially. As discussed in Chapter 2, two such methods are grouping and the use of independent learners. In the algorithm skeleton, the `combine` step is placed within the main loop to allow for either case. Generating combinations "on the fly" would correspond to independent learning, while recalling a fixed set of combined transformations would correspond to grouping.

3. *Assessing the Transformations*

   Generally, the only way to determine the effect of a transformation is to actually use the modified strategy. In [20], a method is described to approximate the difference in cost between a transformed solver and its initial solver using only the latter's execution traces. However, the technique requires a complete cost model, which is usually not available.

4. *Computing the Best Transformation*

Upon termination, the algorithm must return the "best" combined transformation for implementation. The question is "what does best mean?" or equivalently, "which local transformations should be adopted?". The answer to both requires that the utilities of the component transformations be known. This leads to the multiagent credit assignment problem.

As briefly outlined in Chapter 2, the core of the issue is how to determine the individual contribution of a local transformation to the utility of the system as measured by a global function. The problem arises with the use of a single global utility measure, since it only returns one value. The simplest solution is to distribute the measure across agents. These functions need to return conditional utilities, since the usefulness of an individual transformation may be tied to the existence of other transformations due to interactions. Finding a good measure is difficult and is akin to developing a good search heuristic.

Currently, the only purported solution to the multiagent credit assignment problem, besides the use of domain knowledge [50], is the use of agent grouping in genetic programming [23] [25]. The central idea behind grouping [25] is that the transformations are evaluated only within the context of a set of transformations. Thus, it does not matter what the individual contributions are given that the group utility is known and that all transformations will be applied simultaneously. This would seem to sidestep the issue. However, this analysis is deceptive in that it does not look at the question from the standpoint of "what would happen if credit assignment is not performed?"

The answer depends on the type of learning algorithm. In particular, credit assignment only becomes an issue if the method uses simultaneous, mutually exclusive, irreversible transformations. By definition, the first case is always true of systems that learn multiagent coordination strategies. The remaining

two conditions are contingent on the learning paradigm. Consider the use of multiple, independent Q-learners [46], which is an example of a monolithic algorithm. This statement may be troubling, since the algorithm performs continual Q-value updates. However, these updates can be regarded as part of the task of finding the best transformation rather than as individual iterations. Note that the algorithm is only run once and, when and if it converges, a complete policy is returned.

Credit assignment is not explicitly performed in independent Q-learning and all agents are allocated the same reward. However, credit assignment is also not a problem in this scenario. Assume that the agents have converged and found an optimal policy. This implies that every transformation that has been implemented forms either an integral part of the policy or it does not hinder the policy. In this context, the only type of "error" that may have been committed is to adopt an useless and unnecessary transformation. This "error" has no effect on the performance of the system. Similarly, AGE does not suffer from this problem since it also learns an entire policy. Recall that grouping is used in AGE to determine interaction effects and not to solve the credit assignment problem. Thus, the "best" combined transformation in a monolithic learner is simply the combined transformation with the highest utility.

Now consider a similar scenario in a domain where the transformations are mutually exclusive and irreversible. In other words, the learning system is iterative and limits the size of the concept space. Hill-climbing is an example of such a system. Since useless is with respect to a step, the implementation of such a transformation may block the implementation of a beneficial transformation at a later step. Let there be two agents, $\mathcal{A}_1$ and $\mathcal{A}_2$. At time step 1, $\mathcal{A}_1$ can only apply transformation $\mathcal{T}_{11}$ while $\mathcal{A}_2$ can apply either $\mathcal{T}_{21}$ or $\mathcal{T}_{22}$. $\mathcal{T}_{11}$ negatively interacts with $\mathcal{T}_{22}$ but has no interactions with $\mathcal{T}_{21}$. In fact, the utility of $\mathcal{T}_{21}$ is

0. $\mathcal{T}_{11}$ also has the highest utility. As a result of aliasing under a single global utility function, both $\mathcal{T}_{11}$ and $\mathcal{T}_{21}$ are implemented. This can occur since the system can't tell the difference between candidates that have the same utility. $C^{\max}= \{\mathcal{T}_{11}\}$ and $C^{\max}= \{\mathcal{T}_{11}, \mathcal{T}_{21}\}$ are equivalent under a scheme that merely looks at utility. Now let the application of $\mathcal{T}_{11}$ make possible the application of $\mathcal{T}_{12}$. Furthermore, let $\mathcal{T}_{12}$ interact positively with $\mathcal{T}_{22}$. Due to the fact that $\mathcal{T}_{21}$ has been implemented, $\mathcal{T}_{22}$ is no longer available and thus the system halts in a suboptimal state. Note that merely considering the transformations in groups has no effect on the potential outcome. This does not imply that grouping is useless; only that the form used in [25] does not eliminate the credit assignment problem. As will be shown, comparing the utility of grouped transformations can be used to isolate the effects of their components.

For a concrete example, consider a much simplified scenario taken from a financial domain. Company **X** is losing money and thus it must make some changes to its business strategy. Unwisely, the firm is structured into two units that independently make their own decisions. The first is responsible for production and the other for sales and marketing. 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 have the option of spending more money on marketing A. There is sufficient capital to increase or start the production of A or B, respectively, and to promote A. In fact, the transformation associated with the increased marketing of A can be applied several times while still increasing or starting the production of A or B. There are insufficient resources to both increase the production of A and to start the production of B while simultaneously increasing the promotion of A.

The rationale behind producing more of A is that the increased economies of

scale may help to reduce costs and the rationale behind producing B is that there may be a market for it. Let C_1 be production deciding to build a factory for B while sales decides to promote A. After the application of the combined transformation it was found that promotion worked and demand for A was increased. As a result the inventory of A is sold and revenue increases. However, widget **B** fails to sell. Due to aliasing, production decides to build a new factory for B and sales decides to more aggressively market A. At this point, the only available transformation is for sales to again increase spending. Let the ad campaign be so successful that it exceeds production of A. Since the factory for B has already been built there are no additional resources to spend on A. This results in a sub-optimal solution.

Although the adoption of a useless transformation in a monolithic learning system has no effect, it can degrade the optimality of an iterative algorithm. However, aliasing is not the only problem in multi-step methods since blocking can also occur even if the utility of the transformation is positive. Thus a simple minimality criteria, requiring each component transformation to have positive utility, will be insufficient to resolve the issue.

One approach to the blocking problem is to try to minimize its occurrence. Recall that iterative algorithms can promote one or more of the "best" candidates from one iteration to the next. Thus one method of reducing the likelihood of a blockage is to advance a different combined transformation. Genetic programming takes this route by copying the best programs from one generation to the next. However, this does not guarantee that the problem will not occur and is also combinatorial. Another technique is to pass one candidate, as in hill-climbing, but then to minimize the number of component transformations that are implemented to those that are "necessary". The following definitions will be used to clarify this concept.

Definition 7: Joint Transformation

A joint transformation $\mathcal{J}$ is a set of transformations $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k \ni \mathcal{U}(\mathcal{J}) > \mathcal{U}(\mathcal{T}_1) + \mathcal{U}(\mathcal{T}_2) + \ldots + \mathcal{U}(\mathcal{T}_k)$. That is, there is an interaction between the effects of the transformations such that the joint utility is greater than the sum of the individual utilities. By definition, the set of individual transformations are also joint transformations.

Definition 8: Minimal Joint Transformation

A minimal joint transformation $\mathcal{J}^*$ is a joint transformation where there are no subsets of $\mathcal{J}^*$ that are joint transformations.

A necessary transformation is a member of a minimal joint transformation. In other words, if one element of the set is not implemented then the overall utility will decrease by more than will be explained by additivity unless the minimal joint transformation is a singleton. In other words, the best "atomic" transformation should be adopted. Additive transformations can be implemented in later steps if they have not been outperformed by new transformations enabled by the best $\mathcal{J}^*$.

5. *Terminating Learning*

A recurring problem in machine learning is deciding when to stop learning. This is an important issue since learning can be expensive. Effort that is used to explore a space of transformations can be spent solving problems. Moreover, transformations with negative utility can be harmful to an agent. Consider a robot that is learning how to disarm a bomb. Reducing the number of samples to examine would be highly beneficial.

The problem of termination is also related to utility. An agent should stop learning when it cannot improve its performance. This can occur when it has found the best transformation(s) or when it decides that none of the transformations

are effective. Note that none of the surveyed systems have a well-defined termination criteria. For instance, **GP** relies on a predefined number of generations, **RBL** uses the convergence behavior of the agents, and **CBL** does not have a termination criteria at all. These conditions are ad hoc and do not provide guarantees on performance.

If an agent is to use a utility based method of termination, then it must have a computable notion of best transformation. Ignoring the issue of credit assignment, the task of simply finding the combined transformation with the highest utility is difficult due to the fact that the performance of a transformation may be inconsistent across problems. In some cases, it may even have negative incremental utility. Secondly, the problems are assumed to follow a distribution. Thus the modified strategy may be performing well for a number of problems and then abruptly perform poorly as another portion of the distribution comes into play. Factoring in the topic of credit assignment adds another level of difficulty. The problem of inconsistent performance carries over to finding the minimal joint transformation. Positive interactions that are readily apparent in one context could be absent in another.

Another way of looking at both issues is from the vantage point of what the agent is learning. Abstractly, the problems can be described as trying to identify the concepts of high utility and interacting transformations. Given that the behavior of the system is problem and hence distribution dependent, a more meaningful definition for both would include the notion of expectation.

Expected utility can be defined in the usual statistical sense. A definition for expected interaction can built on top of this by requiring that the expected utility of an interacting transformation be greater than the expected utility of the sum of its components. If the problem distribution were known a priori then these values can be computed by assessing each transformation and then

by applying the probabilities. However the distribution is generally unknown. Referring back to the initial problem, the issue is: given the problems that have been encountered, how sure can the system be that it has found the expected minimal joint transformation with the highest expected utility. A hill-climbing approach can terminate a step under this condition.

Probably Approximately Correct (**PAC**) learning [54] is a learning paradigm that quantifies the "degree" that a concept has been learned. This allows statements to be made about the performance of an algorithm. Consider the task of learning what a cup is. To learn this concept exactly requires that an agent see every object that is a cup. However, for most learning problems, the world is not so accommodating as to enumerate all the possible cups in a sequence. The same type of cup may be seen many times before a new cup is introduced. The question is, "how many cups have to be seen before it can be said that the concept has been learned?"

Rather than exactly learning a concept, a **PAC** learner approximates it with error ϵ and probability 1- δ. Using the assumption that the instances that have been seen follow a fixed distribution, **PAC** learning guarantees that the approximation criteria are met by requiring the learner to see a sufficient number of samples. In the basic model, ϵ is the probability of misclassification. However, an alternate way of looking at the **PAC** problem is to consider the space of positive examples as a population [9]. The characteristic function for a concept can be viewed as a random variable whose mean is the mean of the population. **PAC** learning can then be considered as estimating the mean for the population within a confidence interval of $\pm\epsilon$ with confidence 1- δ. This observation allows for the incorporation of various statistical tools into the **PAC** framework. The **PAC** paradigm can be applied to the expected utility and interaction problems by requiring each concept to be learned with error ϵ and probability 1-δ.

3.2 Multiagent Probabilistic Hill-Climbing

A technique [26] [27] based on probabilistic hill-climbing has been implemented to address the four issues raised in the previous section. It is a two-stage hybrid approach founded on both individual and group learning. The first stage uses co-learning to assess the combined utility of the transformations. Each agent individually selects their "best" transformation from their local set while interacting with the other agents. Once this set is found, the method proceeds to the second stage where the transformations are collected into groups and evaluated. The joint minimality criterion is then applied to resolve the credit assignment problem. The following is a general discussion of probabilistic hill-climbing. It is followed by a discussion of how this technique can be applied to multiple agents.

3.2.1 Probabilistic Hill-Climbing

Probabilistic hill-climbing [21] [20] [15] is a **PAC** learning method that seeks to locally improve the average performance of a system until no incremental improvement can be made. The rationale for devising a satisficing solution is that it is too computationally expensive to derive an optimal strategy. For instance, a monolithic approach based on first learning the distribution $\mathcal{P}$ and then generating an optimal strategy is NP-hard [19]. The basic hill-climbing algorithm is shown in Figure 3.2. It resembles the generic procedure found in Figure 3.1.

A set of transformations on the current system, including the null transformation, ϕ, is proposed. These could have been generated by a simplistic mechanism or a sophisticated **EBL** domain theory. In each case, the average expected change in utility of this transformation is computed under some sampling scheme. Sampling could be performed round robin or be based on a function of the utility values.

```
HILL-CLIMB(S, ε, δ)
    i = 1
    T = generate-transforms(S)
    loop while not(termination-criteria(S)) do
        δ_i = distribute(δ,i)
        sample(T)
        if ∃t_i ∈ T ∋ climb-criteria(t_i, T, δ_i, ε) then
            S = t_i
            T = generate-transforms(S)
            i = i+1
        end if
    end loop
```

Figure 3.2: Hill-climbing Algorithm

When sufficient samples have been accumulated to indicate that the mean Δ-utility of the best transformation is better or within $\pm\epsilon$ of any other transformation with probability 1-δ_i then the system hill-climbs using this transformation.

Definition 9: ϵ-optimal Transformation

A transformation $\mathcal{T}_{\mathbf{max}}$ is ϵ-optimal when it has the highest expected utility in the set of $\mathcal{T}_i$'s or when all other $\mathcal{T}_i$ have expected utilities which are $\pm\epsilon$ of $\mathcal{T}_{\mathbf{max}}$ with probability 1-δ_m.

The ϵ parameter is a measure of *indifference*. That is, if the "best transformations" have average (Δ)-utilities within ϵ of each other, then the agent does not care which one is implemented. This process continues until no additional local modifications can be made or the "best" transformation is ϕ.

The confidence parameter δ is distributed over each hill-climbing step such that the sum of the probability of errors over all steps is less than or equal to δ. If the

number of hill-climbing steps is bounded, then the δ_i's can be computed by dividing δ by the number of steps. In cases where the number of steps is unknown, the δ_i's can be computed by multiplying δ by any series that sums to 1 at infinity; such as $\frac{\delta 6}{i^2\pi^2}$ [20]. This approach is guaranteed to return a system that is locally ϵ-optimal with probability 1-δ.

From the above discussion it can be seen that probabilistic hill-climbing address a number of issues in the generic coordination framework.

1. *Computing Transformations*

   By design, probabilistic hill-climbing is iterative. Thus there is a trade-off between optimality and complexity.

5. *Terminating Learning*

   As an instance of **PAC** learning, probabilistic hill-climbing has a well-defined termination condition.

Issues 2 and 3 will be examined below where the hill-climbing algorithm is extended to multiple agents.

3.2.2 Application of Probabilistic Hill-Climbing to Multiple Agents

The use of probabilistic hill-climbing requires the introduction of an additional definition:

Definition 10: ϵ-joint Transformation

A ϵ-joint transformation, $\epsilon - \mathcal{J}$, is a set of local transformations $\mathcal{T}_1, \mathcal{T}_2, \ldots \mathcal{T}_k$ such that:

1. $E(\mathcal{U}(\epsilon - \mathcal{J})) > E(\sum_{i=1}^{k} \mathcal{U}(\mathcal{T}_i))$ or
2. Another way of defining a ϵ-joint transformation is to define its opposite. In this case, a combined transformation $\mathcal{C}$ is not $\epsilon - \mathcal{J}$ if $|E(\mathcal{U}(\mathcal{C})) - E(\sum_{i=1}^{k} \mathcal{U}(\mathcal{T}_i))| \leq \epsilon$

with probability 1-δ_I. Related definitions for minimal ϵ-joint transformations, $\epsilon - \mathcal{J}^*$, and ϵ-optimal ϵ-joint transformations, ϵ-$\mathcal{J}^*_{\max}$ follow directly from 8, 9 and 10.

In practice, the definitions associated with 10 use Δ-utilities instead of actual utilities. This is in keeping with the use of hill-climbing where the task is to incrementally improve performance.

The most straightforward way to apply probabilistic hill-climbing to multiple agents is to use it to estimate the Δ-utility of grouped transformations. These groups would correspond to the cross-product of all the local transformations, including ϕ, at a given step. The ϵ-joint minimality criteria can be then applied to filter the groups with error probability δ_I. $\epsilon - \mathcal{J}^*_{\max}$ is then selected with error probability δ_m. If $\delta_i + \delta_m$ is less than δ then the best candidate is returned for implementation.

Another way of looking at the problem is to first select the ϵ-optimal group, $\mathcal{G}_1$, with error probability δ_{m1}. Then from out of $\mathcal{G}_1$, $\epsilon - \mathcal{J}^*_{\max}$ is *extracted* with error probability δ_{i1}. Again, if $\delta_{m1} + \delta_{i1}$ is less then δ then the system hill climbs. This variant is possible since $\mathcal{G}_1$ must contain $\epsilon - \mathcal{J}^*_{\max}$. The group with the largest utility must contain the atomic transformation with the largest utility.

Both approaches are based on standard grouping and still suffer from the same complexity concerns. Note that the grouping is performed on transformations and not on agents and thus the scheme is not as philosophically objectionable.

Between the two approaches, the second is arguably better in that it reduces the groups to one candidate. This is a weak argument, since there may not be any savings

in terms of sample complexity using this technique. However, the rationale for this variant becomes much stronger if the guarantee of ϵ-optimality is dropped and an approximation method used to find $\mathcal{G}_1$.

The proposed approach decomposes the learning task at each step into two stages. The first step individually computes the best transformation for each agent. In effect, this produces a set, $\mathcal{G}_2$, which is an estimate for $\mathcal{G}_1$. Next, $\epsilon - \mathcal{J}^*_{\mathbf{max}}$ is isolated from $\mathcal{G}_2$ using grouping.

During stage 1, each agent individually runs a copy of the probabilistic hill-climbing algorithm. Local transformations are evaluated without coordinating with the other agents. This scheme is identical to Sen's use of independent Q-learners, except that probabilistic hill-climbing is guaranteed to converge. An agent either decides that one of the transformations is useful or it defaults to its current strategy. When applied to the symbol emitting domain from Chapter 2, an earlier variant of the method halted in 50 out of 50 trials while returning one of the 3 equal positive utility solutions in 49 cases. The remaining trial returned a negative utility outcome.

Probabilistic hill-climbing is guaranteed to return the transformation with the highest expected $\epsilon - \Delta$-utility with probability 1-δ_m for each agent. However this does not guarantee that $\mathcal{G}_2$ is equal to $\mathcal{G}_1$. If this were true, then it is necessary for each agent to return an ϵ-optimal transformation, but this condition is not sufficient to guarantee the converse. Thus the local transformation method can be viewed as a heuristic for finding $\mathcal{G}_1$. It has been empirically proven to be effective in this task. Results will be discussed in Chapter 4.

The second stage of the procedure groups the members of the individual maximum transformations, samples the groups, computes the ϵ-joint transformations and then looks for the ϵ-joint transformation with the largest mean. Since the approach relies on two stages, δ_i is allocated across both stages as δ_{i1} and δ_{i2} respectively. Similarly, ϵ is also allocated as ϵ_1 and ϵ_2. The rationale for the first allocation is clear. In the

second case, ϵ is distributed to prevent the following scenario. Let the first stage approximate $\mathcal{G}_1$ exactly and return a $\mathcal{G}$ which is one ϵ away from the best group. During the second stage selection process, the system also returns a within ϵ solution. As a result the final group is 2ϵ away. To eliminate this problem, ϵ_1 and ϵ_2 must add up to ϵ.

The second stage algorithm is shown in Figure 3.3. Once $\mathcal{G}_2$ has been found, the **N**,

```
JOINT − TRANSFORMS(agents, ε, δ, N_0)
    J = compute-combinations(agents)
    i=1
    loop while not flag do
        simultaneous-extract(J)
        if (i > N_0) then
            [IT, i-error]= compute-independence(J)
            [MT, m-error]= find-max-trans(IT, ε)
            if ((i-error + m-error) < δ) then flag= TRUE
        i= i+1
    return(MT)
```

Figure 3.3: Joint Transformation Algorithm

N-1 to 1 combinations of the transformations are computed. This set represents the potential minimal joint transformations since it contains all the possible combinations of local transformations. There are still, $\binom{N}{1} + \binom{N}{2} + \cdots + \binom{N}{N} = 2^N - 1$, an exponential number of transformations but, **N** is only in the number of agents and not a function of the number of agents and the number of transformations. Consider the case of three agents with their individual best transformations being **A**, **B**, and **C** respectively. The seven potential joint transformations are **{A B C}**, **{A B}**, **{B C}**, **{A C}**, **{A}**, **{B}** and **{C}**. If a transformation is not chosen to be a member

of the joint transformation, then the corresponding agent adopts its default strategy.

The Δ-utilities of the potential joint transformations are assessed by applying them on the same problem instances. After an initial number, N_1, of problems have been solved; the potential joint transformations are tested for *interaction effects*. This is done by directly implementing the minimal joint transformation criteria in the `compute-independence` function.

The expected Δ-utility of a potential joint transformation is compared to the expected sum of the utilities of its individual components. Consider the combined transformation **A B C**. If all three component transformations interact, then the expected Δ-utility of **A B C**, $\mathrm{E}(\Delta - \mathcal{U}(\mathbf{A\ B\ C}))$, should be greater than of:

1. $\mathrm{E}(\Delta - \mathcal{U}(\mathbf{A\ B}\ \mathcal{D}_3) + \Delta - \mathcal{U}(\mathcal{D}_1\ \mathcal{D}_2\ \mathbf{C}))$
2. $\mathrm{E}(\Delta - \mathcal{U}(\mathbf{A}\ \mathcal{D}_2\ \mathbf{C}) + \Delta - \mathcal{U}(\mathcal{D}_1\ \mathbf{B}\ \mathcal{D}_3))$
3. $\mathrm{E}(\Delta - \mathcal{U}(\mathcal{D}_1\ \mathbf{B\ C}) + \Delta - \mathcal{U}(\mathbf{A}\ \mathcal{D}_2\ \mathcal{D}_3))$
4. $\mathrm{E}(\Delta - \mathcal{U}(\mathbf{A}) + \Delta - \mathcal{U}(\mathbf{B}) + \Delta - \mathcal{U}(\mathbf{C}) + \Delta - \mathcal{U}(\mathbf{D}))$

where $\mathcal{D}_i$ stands for the default strategy of agent i. When combined, each test corresponds to determining whether or not there is a positive interaction, the joint transformation condition, and whether this condition can be explained by a subset of the combined transformation, the minimality condition. For instance, in the first test, the expected Δ-utility of **{A B C}** is being compared to one of its second order subsets, **A B**. If the expected Δ-utility of **{A B C}** is greater than that of the expected utility of **{A B}** plus **C** then there is an interaction between **{A B C}**. On the other hand, if the expected Δ-utilities are the same then this implies that either the Δ-utilities in **{A B C}** are additive or that **{A B}** interacts. In either case, **{A B C}** should not be considered a minimal ϵ-joint transformation.

In general, there are $\binom{N}{N-2} + \binom{N}{N-3} + \ldots + \binom{N}{2} + 1$ tests where N is the number of agents or the maximum order of the potential joint transformation.

For each test, the probability that there is a positive interaction and the probability that the values are the same are computed. In some instances, it is easier to prove that the means of two distributions are within ϵ than to prove that one mean is larger than another [4]. This accounts for the second condition in Definition 10. Thus both probabilities are computed. In both cases, the probabilities across all tests must be integrated to reach a conclusion about a given potential joint transformation. This requires an error model, which is discussed in section 3.2.3. Given the combined probabilities of being larger or being the same, the hypothesis associated with the larger of the two, $\mathcal{P}_{max}$ is considered correct. Accordingly, the probability of error is 1-$\mathcal{P}_{max}$. Again, these error values have to be integrated across potential joint transformations. The integrated value is the probability, *i-error*, that the set of interacting joint transformations has been incorrectly identified. $\epsilon - \mathcal{J}^*_{\max}$ is found by sampling the current set of hypothesized joint transformations. The best interacting joint transformation along with its probability of error, *m-error*, is returned. When the sum of *i-error*, *m-error*, and the errors from stage 1 is less than δ_i, the system hill climbs using the best joint transformation. The remaining minimal joint transformations are then deleted to avoid the blocking problem. New transformations are generated and the process repeats until a predefined termination criteria, such as a bound on the number of hill climbs, or when the current best transformation is ϕ. The latter criteria is preferred since it is well founded. However, in some of the experiments, a climbing bound was used to reduce run time.

At this point, the remaining two issues in the generic multiagent learning algorithm can now be summarized.

2. *Computing Combined Transformations*

   The effect of combining transformations is ascertained by using independent learning. This trades off optimality for complexity. It is also an attempt to address the philosophical issue of autonomy in learning.

4. *Computing the Best Transformation*

   The minimal joint transformations are discovered using a method based on grouping. As in the case of AGE, grouping is performed at the level of transformations and not at the level of agents. Again, this is to address the issue of agent autonomy. By itself, grouping is useless. Although it can be used to assess the effect of a set of transformations, it cannot determine which subset of the group gave rise to the positive Δ-utility. This requires that the effects of a group be compared to those of its components to determine the minimal subgroup responsible. This, in turn, requires some criteria to determine the confidence of the utility estimates. A statistical technique is used to compute the error of estimation.

A rough comparison can be made between this method and group-based techniques by examining the number of expected utilities that have to be estimated. For instance, consider a domain with four agents, each with five transformations. Using grouping there would be $5^4 - 1 = 624$ means to be estimated. On the other hand, if the proposed approach is taken, there are 20 (4 x 5) utilities to be estimated in stage 1 and then 2^4-1 or 15 in stage 2 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 in the number of utilities. Moreover, the range in the number of samples required should be similar. In each case, the task is to estimate the parameters of a distribution to a prescribed confidence level.

3.2.3 Implementation Details

The suggested algorithm is a skeleton with many possible instantiations. Two questions have to be answered before an implementation can be developed. These are: what assumptions should be made when comparing the system error and what method should be used to sample the transformations.

Computing Errors

There are three places in the algorithm where errors are computed. In two of these cases, the current best transformation is compared to the remaining transformations to determine if it is the best and in the other, the error of missclassifying a joint transformation is calculated. All cases involve the comparison of distributions. The proposed system uses paired t-tests to determine if the mean of one distribution is strictly better than another or if they are within $\pm\epsilon$. More details can be found in Appendix A. Since a set of paired tests are used, the question is how to handle the results of these tests. That is, which error model should be employed? There are three possible models [16]:

1. *Worse Case.* Take the sum of all the errors.
2. *Independent Case.* Take the final error as $1 - \prod_{i=1}^{N}(1 - error_i)$.
3. *Best Case.* Take the largest error.

Consider the case of finding the best transformation. An error occurs when either: 1) the best transformation is properly estimated and another transformation is overestimated or 2) when the best transformation is underestimated and the other transformation properly estimated or 3) when the best is underestimated and the other is

overestimated. Thus, the errors can be dependent. Rather than potentially underestimating the error using either the independent or best case models, it was decided to overestimate the error using the worse case model. On the other hand, the misclassification error for an individual joint transformation was computed using the independent model. This is not totally correct, since the errors could be correlated if the subcomponents are additive. For instance, when testing {**A B C**} against {**A B**} and **A** + **B**, if $\mathcal{U}$(**A B**)= $\mathcal{U}$(**A**) + $\mathcal{U}$(**B**) then these tests may be correlated. This is mitigated by the fact that Definition 6 assumed that sampling is independent. To offset this, the worst case model was used across joint transformations.

Sampling Schemes

There are a variety of ways in which the transformations could be sampled. The simplest approach is to use a round-robin scheme and to consider each in turn. This is commonly known as simultaneous extraction [21].

In general, the optimal scheme would return the best transformation in the least number of samples. It is assumed that testing is expensive and thus should be minimized. The learning community has proposed various solutions to the problem [12] [31] [17] [4]. There are also related problems in statistics, such as finding the member of a population with the largest mean [53] [51] or finding the "best arm" in a bandit problem [45] [2]. The latter task involves an imaginary, multiarmed slot machine that returns a random award depending on the arm that was pulled. The goal is to maximize the long-run total reward using the results of previous trials.

Some of the techniques sequentially sample all the transformations until it has been proven that a particular transformation cannot be the best [53] [51], while others compute dominance or confidence intervals for the means of the transformations [12] [31] [17] [45]. The transformation that dominates or has the largest upper bound on

its confidence interval is sampled. Finally, the remaining approach [17] [4] weighs the cost of sampling a transformation against the benefits in error reduction and then picks the one with the best ratio.

While these techniques reduce sample complexity for single agent learners, the learning environment for multiagent systems is different. The difference is that the utilities of the transformations may not be independent, but instead depend on the transformations being tested by other agents. Thus, sequential methods will tend be useless, since they would "smear" the utility estimates for potentially good joint transformations over all the transformations. This has been empirically confirmed for certain types of distributions. See section 4.2 for results. Intuitively, a "good" multiagent sampling scheme would have each agent explore its space of transformations and then settle on a subset that has high utility when combined with those of the remaining agents. This would allow the agents to "converge" on a set of joint transformations.

The development of a new multiagent sampling scheme is beyond the scope of this thesis. In the proposed system, for all but one experiment, agents sample their transformations using the Z-heuristic [45] which was developed to solve the bandit problem. Initially, N_0 samples of each transformation are randomly taken. The mean and the variance of the mean of each transformation are then used as parameters to a set of normal distributions. A random number is generated from each distribution and the transformation associated with the largest number is sampled. This approach has characteristics of the intuitive scheme discussed above. Its effect will be discussed in Chapter 4.

Experiments in the navigation domain used interval estimation (IE) [12] [31] rather than the Z-heuristic. This is a historical consequence of it being one of the earliest experiments performed. After some discussion with the author of [12] and some experiments, it was decided that the sample complexity of the Z-heuristic was

better than that of IE and thus the remaining experiments used this scheme. The results of these initial trials will not be presented here.

Chapter 4

Experimental Validation

Three sets of experiments are performed to test the effectiveness of the technique. The first set involves a synthetic domain where the characteristics of the problems can be easily controlled to determine their influence. The second set are performed using Sen's [47] robot navigation domain. These tests stress the scalability of the first stage of the approach, since the domain can be viewed as involving a maximum of 400 agents. The final experiment applies multiagent probabilistic hill-climbing to the predator and prey pursuit domain. This is a well studied problem in **DAI** with published results for the **GP** learning paradigm. A simplified restaurant domain will also be presented in Chapter 5, but it will be used to demonstrate a sample complexity reduction method.

The next section discusses the coordination approach and the associated problem solving framework that is used throughout the rest of this thesis. Each domain is then considered in turn.

```
(make-operator
    :template   '(get-drink-seven ?x)
    :preconds   '((send-dorder ?x) (seven ?x)
                  (have-seven ?y))
    :addlist    '((have-drink ?x) (not-pay-d ?x))
    :dellist    '((send-dorder ?x) (seven ?x)
                  (have-seven ?y)))
```

Figure 4.1: Seven-up Operator

4.1 Problem Solvers

The coordination paradigm used in the research is an instance of an organizational structure approach. A coordination strategy is implicitly encoded in the behavior of the agents. The behavior of an agent is controlled by a policy which is executed depth first. Given a context, an agent will "commit" to execute the first action or operator in the policy that is applicable. This occurs in parallel using the same context.

The actions/operators range from simple state to action mappings in the case of the navigation and the predator and prey domains to STRIPS operators in the restaurant domain in Chapter 5. An example of a state action pair is {(10,10), *north*}. If an agent is in grid location (10,10) then it should go *north*. Figure 4.1 shows an example of a STRIPS operator from the restaurant domain. The symbols preceded by "?" are variables. The function of this operator is to obtain a glass of Seven-up for agent ?x. As in the usual STRIPS paradigm, it can be applied when its preconditions are met. The result is that the literals on the **addlist** are added to the world and the literals on the **dellist** are deleted from the world. In this case, if agent ?x orders a drink, wants a Seven-up and there exists a free glass

of Seven-up then this operator can be used. These two requirements correspond to the `(send-dorder ?x)`, `(seven ?x)` and the `(have-seven ?y)` literals in the pre-conditions, respectively. Once the operator has been applied then agent `?x` has a drink which has not been paid for, the order has been filled and the glass of Seven-up has been assigned. The first condition is added by the `(have-drink ?x)` and the `(not-pay-d ?x)` literals in the **addlist**. The fact that the order has been filled and the glass used-up is reflected in the three elements of the **dellist**.

In domains containing variables, bindings are resolved at execution time. Thus two agents can execute the same action using different objects. If more than one agent selects an operator with only one binding, then a simple agent priority scheme is used to resolve the conflict. Agents are assigned numerical id's. The agent with the lower id executes the operator while the remaining agents are idle. Bindings are assigned in the order that they are found. The strategies do not explicitly control bindings.

Since strategies are executed depth first, transformations correspond to reordering the elements of the strategy. This could entail moving one element or moving multiple elements. To reduce complexity, the implemented solvers only use one-step transformations. This scheme can contribute to the system being trapped in a local minima.

4.2 Synthetic Domain

In this domain, each one of four agents outputs a symbol from the distinct set that it has been assigned. The feedback received is a function of the world state and the combined set of outputs from all agents. As described above, strategies are ordered lists of symbols. An agent outputs the first symbol in its strategy. This problem is similar to the contrived example presented in Chapter 2. To determine

the performance of the method, the domain and the experiments were parameterized in the following ways:

1. *Changing the shape of the problem distribution*

   As its name implies, the problem distribution refers to the frequency of the problems that will be encountered by the system. For the symbol domain, this corresponds to the frequency of the states. If the utilities of the transformations are equal then the transformation that improves performance on the most common problem should be implemented. The use of equal utilities is a simplifying assumption. In general, the transformation that has the best expected utility should be selected.

   The experiments used a variant of the simplifying assumption, but exchanged what was fixed. Instead of varying the distribution and fixing the utility, the distributions were kept uniform and the utilities varied. This was done to simplify the setup of the experiments, since it is immediately apparent which transformations were ϵ-local of the best. Thus, problem distributions refer to the distribution of high utility transformations.

   The algorithm was tested on three different problem distributions to ascertain their effects on performance. In all three cases, the distributions required the use of 3rd order joint transformations. That is, utility values are based on the returned symbols from three different agents. For the system to return a within ϵ result, three out of the four agents must return the appropriate symbols.

   Histograms for the three cases are shown in Figures 4.2, 4.3 and 4.4 respectively. The y-axis of the histogram measures the number of triples that are within the respective multiples of ϵ from the best triple. For the lack of better labels, the distributions will be referred to as hard, center, and skewed, respectively. The hard distribution not only has the least amount of ϵ_1 entries but also

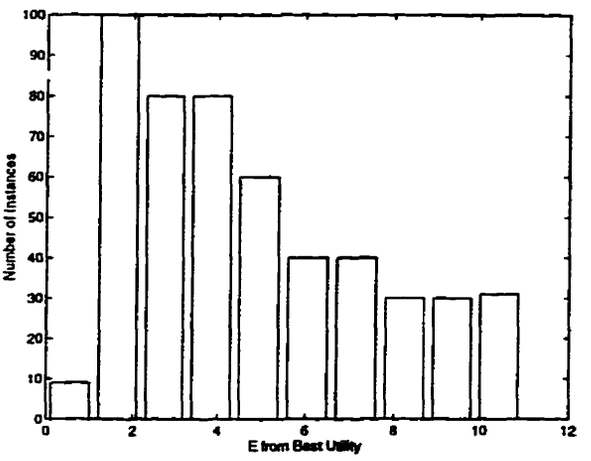

Figure 4.2: Hard Problem Distribution

has a skewed distribution of ϵ_2 outcomes. The remaining two have different proportions of ϵ_1 entries with the center distribution being, naturally, more centered and the skewed distribution being skewed to the right with a large number of ϵ_1 entries. In terms of percentages, the hard, center, and skewed distributions have 1.8%, 4.2% and 20% of within ϵ entries.

2. *Changing $\mathcal{N}_0$*

   The number of initial samples can be changed to assess the sensitivity of the algorithm to this parameter. Experiments used $\mathcal{N}_0$ values of 5, 10, and 15.

3. *Changing the allocation of δ*

   As discussed in Chapter 3, δ must be allocated across the stages, since the proposed approach is a multistage algorithm. Allocation can be controlled using a single parameter that defines the fraction of δ that can be used in stage 1 with the remainder being allotted to stage 2. Parameter values of .25, .5, and .75 were used in the experiments.

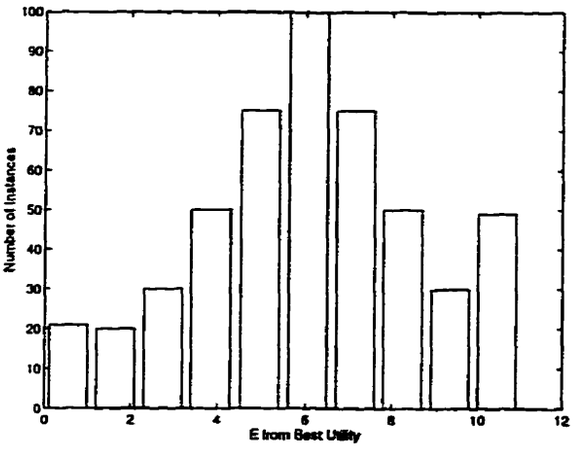

Figure 4.3: Center Problem Distribution

4. *Changing the sampling scheme*

   The effect that the sampling scheme has on the overall results can be determined by varying the scheme. In this case, the Z-heuristic was compared to trials using random and round robin (simultaneous extraction) sampling.

Throughout the tests, the number of agents, the number of symbols per agent, the ϵ, the δ and the $\mathcal{N}_1$ parameters were held constant at 4, 5, 0.4, 0.05, and 30, respectively. All initial symbol lists were ordered randomly. It was found that all trials used more than 30 samples during stage 2 and thus $\mathcal{N}_1$ was not varied. A cutoff of 5000 and 800 samples were placed in stage 1 and 2 of the trials, respectively. If the algorithm did not return an answer by this time, the failure was noted. Results are tallied by distributions in Tables 4.1, 4.2, and 4.3. These correspond to the hard, center and skewed problem distributions, respectively. The first column of each table represents the test conditions in order of δ allocation, $\mathcal{N}_0$, and sampling scheme. In the case of the last parameter, **Z** refers to the Z-heuristic, **R** to random sampling, and **S** to simultaneous extraction. Fifty trials were run for each combination and the following statistics stored:

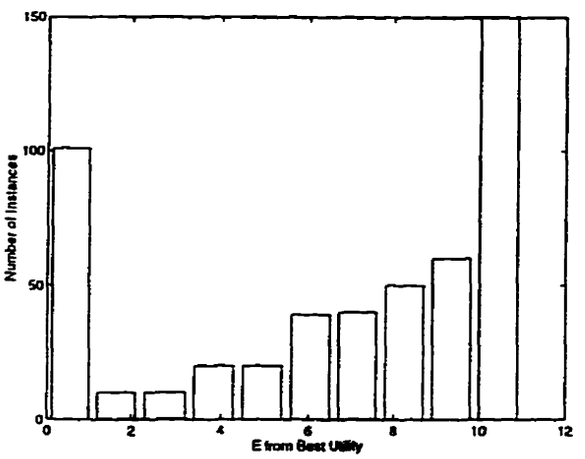

Figure 4.4: Skewed Problem Distribution

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

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

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

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

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 1 D.N.H entries that would have resulted in a stage 1 failure if the algorithm had returned the current best set as the final best transformations.

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 2 D.N.H entries that would have resulted in a stage 2

failure if the algorithm had returned the current best joint transformation as the actual one.

7. *Samples1*: The total number of stage 1 samples accumulated over all trials. Totals for stage 1 and 2 samples are kept distinct due to the different cutoffs.

8. *Samples2*: The total number of stage 2 samples accumulated over all trials. Note that each stage 2 sample is equivalent to 15 (2^4-1) stage 1 samples since each group is assessed using the sample.

The effects of the test parameters are linked and thus the result of modifying one individual parameter may be hard to isolate. The discussion will be broken down into three headings: 1) distribution/sampling scheme, 2) $\mathcal{N}_0$ and δ allocation, and 3) error rate.

1. *Changing the shape of the distribution/sampling scheme*:
   The consequences of changing the distribution and the sampling scheme are coupled. When the number of ϵ_1 entries is small, sampling schemes that do not converge on a set of transformations make many errors, as can be seen in Tables 4.1 and 4.2. As suggested, simultaneous extraction tends to "smear" the Δ-utility estimates for a joint transformation across the joint transformation. Random sampling has a similar effect. However, these methods can return a ϵ_1 solution in a smaller number of samples than the Z-heuristic if the number of ϵ_1 entries is large. Consider the results for the skewed distribution. The Z-heuristic performs poorly in terms of sample complexity, due to the large number of potential solutions. The learning algorithm must keep sampling until it has determined that either the best transformation is the best or that every other transformation is within $\pm\epsilon$ of it. Since many sets of transformations fall into the latter category, it will take correspondingly many samples.

Parameters	ϵ_0	ϵ_1	*F1*	*F2*	*DNH1*	*DNH2*	*Samples1*	*Samples2*
.75/5/Z	5	15	2	4	(24/1)	0	138626	2194
.75/10/Z	11	28	3	6	(1/0)	(1/0)	17502	6042
.75/15/Z	8	25	7	8	(2/2)	0	25030	3443
.50/5/Z	8	10	2	3	(27/5)	0	152989	1700
.50/10/Z	13	27	0	6)	(4/3)	0	29318	3405
.50/15/Z	8	22	5	7	(8/8)	0	54317	3796
.25/5/Z	4	4	1	4	(47/5)	0	195403	768
.25/10/Z	9	24	1	7	(9/2)	0	66952	3495
.25/15/Z	8	31	2	6	(3/3)	0	30793	2944
.75/10/R	1	2	46	1	0	0	29230	124
.50/10/R	0	4	45	1	0	0	38348	310
.25/10/R	1	6	42	1	0	0	43142	249
.75/10/S	2	12	31	5	0	0	46616	1592
.50/10/S	1	14	22	1	(12/8)	0	62634	2079
.25/10/S	1	14	22	1	(12/8)	0	62770	1930

Table 4.1: Results for the Hard Distribution

Parameters	ϵ_0	ϵ_1	*F1*	*F2*	*DNH1*	*DNH2*	*Samples1*	*Samples2*
.75/5/Z	3	24	0	2	(21/0)	0	121330	2210
.75/10/Z	6	37	1	6	0	0	7521	4120
.75/15/Z	3	39	2	6	0	0	7769	3538
.50/5/Z	4	9	0	3	(34/0)	0	176533	1210
.50/10/Z	4	36	5	2	(3/0)	0	37492	3158
.50/15/Z	7	38	3	2	0	0	8215	3587
.25/5/Z	2	10	0	1	(37/0)	0	191894	994
.25/10/Z	7	32	0	3	(8/1)	0	53815	3293
.25/15/Z	5	37	3	4	(1/1)	0	12909	2741
.75/10/R	1	12	34	3	0	0	69807	773
.50/10/R	1	12	35	2	0	0	78104	620
.25/10/R	1	13	33	3	0	0	86803	662
.75/10/S	2	23	22	3	0	0	44267	2899
.50/10/S	2	16	17	4	(11/2)	0	57296	2135
.25/10/S	2	16	17	4	(11/2)	0	57473	1974

Table 4.2: Results for the Center Distribution

Parameters	ϵ_0	ϵ_1	*F1*	*F2*	*DNH1*	*DNH2*	*Samples1*	*Samples2*
.75/5/Z	3	4	0	0	(43/0)	0	234065	197
.75/10/Z	5	15	0	0	(30/0)	0	191779	1319
.75/15/Z	10	23	0	1	(16/0)	0	133241	2249
.50/5/Z	4	1	0	1	(44/0)	0	241079	366
.50/10/Z	4	3	0	0	(43/0)	0	133241	2249
.50/15/Z	5	14	0	0	(31/0)	0	179980	899
.25/5/Z	0	1	0	0	(49/0)	0	246356	31
.25/10/Z	1	2	0	0	(47/0)	0	245000	93
.25/15/Z	2	4	0	1	(43/1)	0	223868	389
.75/10/R	0	30	15	4	(1/0)	0	85634	1246
.50/10/R	0	34	10	4	(2/0)	0	244679	1753
.25/10/R	0	32	9	5	(4/0)	0	56243	1206
.75/10/S	7	41	0	1	0	1	17772	6586
.50/10/S	3	43	0	2	(2/0)	0	16736	3289
.25/10/S	3	43	0	2	(2/0)	0	17491	3103

Table 4.3: Results for the Skewed Distribution

On the other hand, random and round-robin sampling do not suffer from this decision problem, since they have difficulty in finding ϵ_1 solutions. These schemes simply return the best set of transformations that the agents have found without exploration. If the number of ϵ_1 entries is high, then the probability of returning one of these results increases. This effect can be seen by comparing the data for the center and skewed distributions. The percentage of correct results increases with the proportion of ϵ_1 entries. It is the most dramatic in the case of simultaneous extraction. Random sampling has problems exploiting the increased solution density due to the "noise" inherent in the method.

2. *Changing $\mathcal{N}_0$ and δ allocation*:

   These results indicate that the approach is sensitive to the value of $\mathcal{N}_0$. Halving the value from 10 to 5 dramatically increases the total number of stage 1 D.N.H's. This can be explained by looking at the number of times that a given transformation has been sampled. Generally, the combination of a distribution with a small number of ϵ_1 entries and a low $\mathcal{N}_0$ causes a large disparity in the number of samples across transformations. That is, the transformations corresponding to an ϵ_0 or ϵ_1 entry will have many more samples than the remaining transformations. This is an artifact of the Z-heuristic sampling scheme. One problem is that it tends to spend most of its time sampling the best transformation. This is exacerbated by distributions where a few transformations are much better than the others.

   Confidence in the utility estimates and hence the error is a function of the number of samples. Given that most of the samples are concentrated on one transformation, the incremental error reduction per sample is low. Increasing $\mathcal{N}_0$ forces the procedure to sample each transformation at least a fixed number of times. Zeroing in on the best transformation works well for bandit problems and fits in with the idea of the system converging on the best transformations.

However, it may not optimally address the simultaneous goal of reducing sample complexity. Note that the Z-heuristic does implicitly factor in the uncertainty of the estimate. This can be seen in its use of the variance of the mean.

Increasing $\mathcal{N}_0$ to 15 had a marginal effect on the results. Thus, as long as $\mathcal{N}_0$ is not "too low", the algorithm will not use an inordinate amount of samples. However, note that the limiting case on increasing $\mathcal{N}_0$ is purely random sampling. As has been shown, random sampling does not perform as well as the Z-heuristic on sparse distributions.

The effect of changing the δ allocation can be seen in the sample complexity. As may be expected, as less of δ is allocated to the first stage, the number of stage 1 samples increases. Conversely, the number of stage 2 samples decreases as its share increases. Beyond this observation, it was hard to gauge the effect of modifying the weighting.

3. *Error Rate*:

It is hard to quantify the error rate of the method due to the sample cut-offs. Simply summing the ϵ_0 and ϵ_1 entries indicates that the best result was 10% in the case of .50/15/Z if the partial round-robin results for the skewed distribution are ignored. This is below the theoretical requirement of 5% for a system that does not use approximations. Recall that the approach does approximate $\mathcal{G}_1$ and thus this condition is invalid. It can be argued that, in general, taking the difference between the number of samples and the sum of ϵ_0 and ϵ_1 overestimates the error rate. For instance, an examination of the stage 1 D.N.H's indicates that most of them would have returned a within ϵ_1 solution if it had been allowed to halt. This may lead to more ϵ_1 final results. The validation of this hypothesis is pragmatically beyond the computational capabilities of the Sparcstation IPX used in these experiments.

As the results indicate, multiagent probabilistic hill-climbing is capable of returning ϵ-local joint transformations with a good success rate given parameters appropriate to the distribution. Thus, like other parameterized learning techniques, it may require several trials using different values to realize these outcomes.

4.3 Navigation Domain

The navigation problem involves four robots that have to maneuver from one side of a ten-by-ten gridworld to preassigned locations on the other side while avoiding collisions. A diagram of the gridworld is shown in Figure 4.5. An agent's policy is represented by an ordered list of actions, one per square. These actions are *north*, *south*, *east*, *west*, and *hold* as specified by Sen [47]. When an agent enters a square, it executes the first action in the list. This will result in the agent moving to a new square or remaining in the same one. The second case occurs during a collision or when the *hold* action is executed. If this occurs then the next action in the list is executed during the next time step. Consider the following strategy for square (10,10): {(10,10),(*north west hold south east*)}. Given that the current move has resulted in a collision, the next move would be to the *west*. This prevents deadlock, since the agents would normally execute the same action that resulted in the collision, since they are at the same grid location. *Hold* actions are useful for letting another agent pass to avoid a collision. However, an agent should move during the next step.

Each of these squares can be viewed as an agent since it has it own local policy and local transformations. Since the grid is ten-by-ten and there are four agents, there is the potential that the problem may involve up to 400 agents. This number is not normally realized since the agents do not tend to wander all over the grid. Thus, this experiment assesses the scalability of the technique.

Using the aforementioned execution scheme, the transformations reordered the

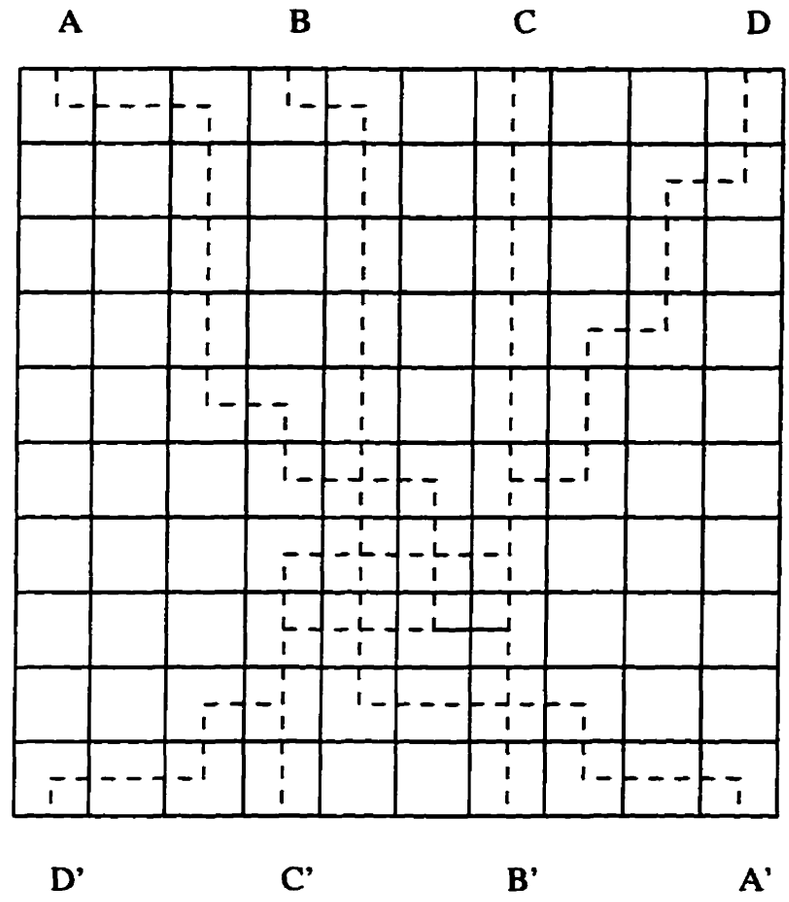

Figure 4.5: Robot Navigation Domain

action lists as in the case of the synthetic domain. Rather than simply permuting the actions, a "lock-step" approach was taken to compute the transformations. At any given hill-climbing step, the transformations only modified one specific position in the order. Initially this was the first position in the action list and was subsequently incremented upon hill-climbing.

Another issue is the reordering of the *hold* action since the utility of this transformation is dependent upon what comes after *hold*. The solution was to generate four different transformations for *hold*. Each one corresponded to one of the four remaining possible actions.

The reinforcement signal is a function of the position of the square being entered into with respect to the goal and also the occurrence, if any, of a collision. Thus, the utility function is local. If the new square is closer to the goal then a value of 1 is received. Conversely, if the new square is farther away then a value of -1 is returned. Otherwise, the agent obtains a value of zero. The cost of a collision depends on the status of the other agent involved. If it is stationary, that is, it is executing a *hold* action then the cost is -10; else it is -5. In the event that an agent tries to leave the grid, it receives a punishment of -10.

The utility function does not measure the number of steps in a solution as would appear in a global function. To remedy this, a payoff filter [40] was used to promote cooperation. The Δ-utility estimates involved a difference in local reinforcement values as well as the weighted Δ-utility of the best transformation in a subsequent grid location. Grid locations with strategies that move an agent closer to its goal should be entered to reduce the overall path length. The second term of the Δ-utility factors this in. This scheme corresponds to the delayed reinforcement used in Sen's paper. Similarly the discounting value or the weight, γ, was set to 0.8.

The aim of the experiment is to test the efficacy of co-learning during stage 1. A question is, "does the approach scale?" In this case, scaling does not refer to computational complexity, but rather to the ability of the technique to converge on beneficial transformations. This issue is important since errors at this point limit the effectivenss of the rest of the algorithm. As such, the second stage procedure was not used nor could it have been used under these circumstances. The cost of assessing $2^{400} - 1$ potential joint transformations is completely unrealistic. On the other hand, a complete grouping solution would require $8^{400} - 1$ means to be estimated. However, the domain does provide a mitigating factor in that it admits many solutions and thus the blocking problem may not occur. That is, the space of paths that an agent can take is large and thus if a local error is made, a workaround may be found. To

further reduce the chance of blockage, only one transformation was implemented each step. A consequence of this is that the co-learning mechanism must "rediscover" the remaining components of a beneficial transformation during subsequent iterations. This makes the test much more difficult.

For the experiment, N_0 was set at 5. Since the number of possible hill-climbs is bounded by the number of squares and the number of actions, a division scheme was also used to compute the $delta_i's$. Rather than dividing by 400, a dynamic scheme was used that enumerated all the grid locations that have been visited and used that number. As stated in Chapter 3, interval estimation was used as the sampling scheme. IE samples the transformation with the largest upper bound for the confidence interval on its mean. This approach is similar to the Z-heuristic but without the random element.

Using a common initial ordering of (*north*, *south*, *east*, *west*, *hold*) for all agents, the system returned the solution shown in Figure 4.5. The dotted lines indicate the paths taken by the agents. The corresponding agent coordinates, with respect to time, are shown in table 4.4. As can be seen, the solution is optimal with each agent arriving at its intending destination in the shortest number of moves while avoiding collisions. In comparison, Q-learning did not arrive at the optimal solution. Rather the paths contained small detours.

4.4 Predator and Prey Domain

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. Agents choose and execute their actions simultaneously without communicating their intent to other agents. This is in line with Korf's hypothesis that predators can locally

Step	*Agent 1*	*Agent 2*	*Agent 3*	*Agent 4*
1	1,1	4,1	7,1	10,1
2	2,1	5,1	7,2	10,2
3	3,1	5,2	7,3	9,2
4	3,2	5,3	7,4	9,3
5	3,3	5,4	7,5	9,4
6	3,4	5,5	7,6	8,4
7	3,5	5,6	7,7	8,5
8	4,5	5,7	7,8	8,6
9	4,6	5,8	6,8	7,6
10	5,6	5,9	5,8	7,7
11	6,6	6,9	4,8	6,7
12	6,7	7,9	4,9	5,7
13	6,8	7,10	4,10	4,7
14	7,8			4,8
15	7,9			4,9
16	8,9			3,9
17	8,10			3,10
18	9,10			2,10
19	10,10			1,10

Table 4.4: Agent Coordinates

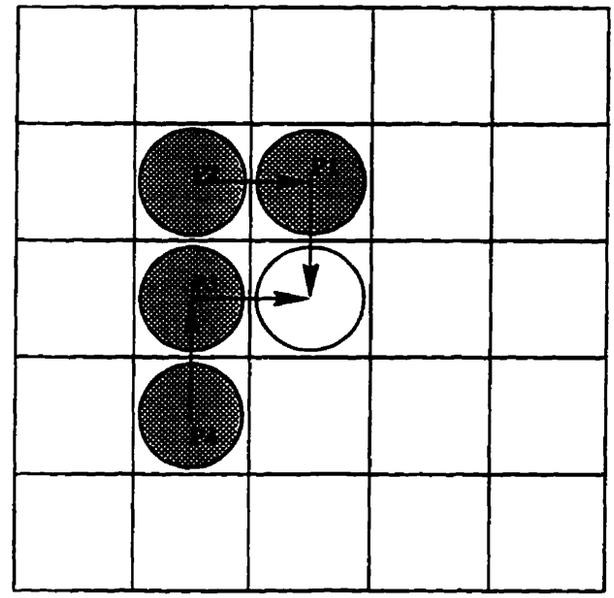

Figure 4.6: Configuration before pushing

choose optimal moves while still consistently capturing the prey [36].

At each time step, the prey has a 90% chance of moving. Thus the predators move faster than the prey. The prey has a simple strategy that moves it away from the closest predator. All ties are arbitrarily broken. Collisions occur when two agents try to occupy the same square. In this case, the agents remain in their previous positions. However, if a predator does not move then it can be pushed by another predator. A predator cannot push the prey. Consider the situation shown in Figure 4.6. The prey is white while the predators are grey. The prey has decided not to move. The arrows indicate the direction in which a predator wants to move. Predators 1 and 4 want to push the prey but they can't. Instead they are pushed by 2 and 4, respectively, resulting in the configuration shown in Figure 4.7.

Predators are controlled by a strategy that uses the relative position of the prey with respect to a given predator. Associated with each compass direction, (**N, S, E, W, NW, SW, NE, SE**), is an ordered list of four potential actions (*north*, *west*,

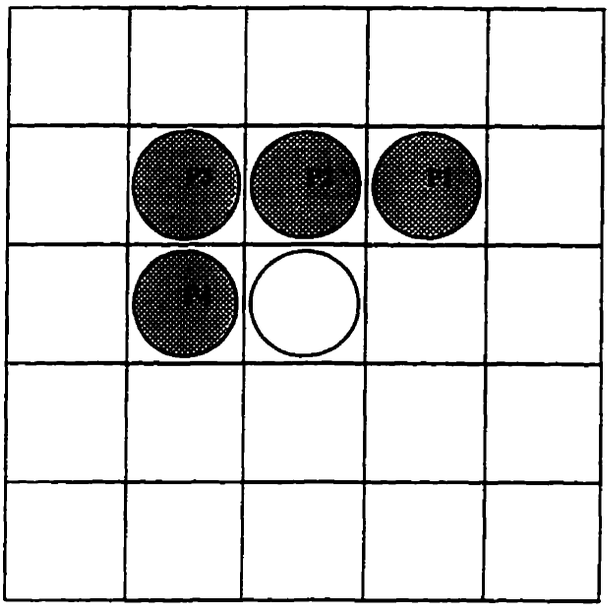

Figure 4.7: Configuration after pushing

south, *east*). As in the other domains, an agent executes the first action in the list given the relative direction of the prey. For example, one strategy element might be (N (*west*, *south*, *east*, *north*)). If the prey is to the north of the agent, then it will go west. Under this scheme, transformations correspond to reordering the lists for each agent for each one of the eight directions. This approach limits the number of potential hill-climbs to eight per agent. Thus, δ can be distributed over the iterations by simply dividing by eight.

Since the grid world is toroidal and there is no explicit representation for distances, a scheme was developed to map the relative position of the prey to one of the eight directions. Consider the situation shown in Figure 4.8. The light circle represents the prey while the darker one represents the predator. The question is, "Is the prey to the south of the predator or is it to the north?" The direction that minimized the distance between the predator and prey was the one returned to the predator. In this case, the prey was north of the predator. Although this representation is simple, it

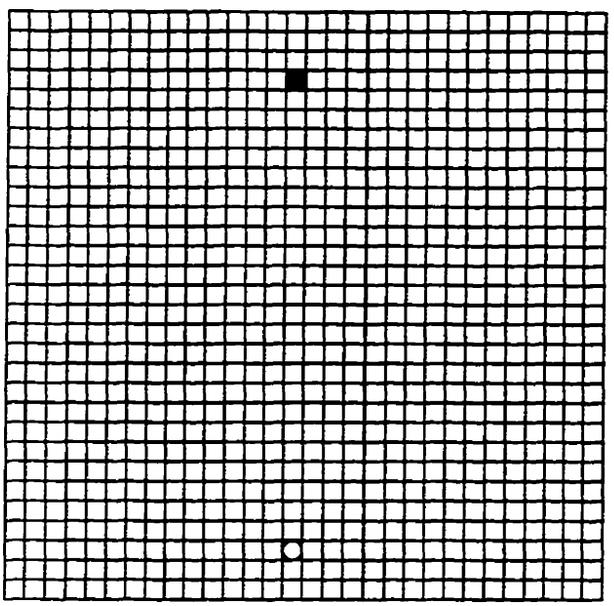

Figure 4.8: Relative Positions

is capable of encoding similar strategies to those found in [23] [25].

The reinforcement signal used was identical to that found in [25]. It included the following components:

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 transformations 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 number of moves in addition to any other reinforcement received. Clearly this favors transformations that aid in capture.

As in [25], the initial locations of the predators are random with the prey being

centered in the grid. The movement of the prey was synchronized with that of the agents. The maximum number of moves was set at 100. The trial ends when either this number is exceeded or if the prey is captured.

4.4.1 Comparisons to Genetic Programming

The approach was compared to Haynes' genetic programming results to ascertain its performance. Corresponding to the procedure in [25], experiments were performed using a maximum of 200 moves although learning only used 100 moves.

Three different trials were performed using multiagent probabilistic hill-climbing. The first two are based on an initial strategy that had every agent moving north regardless of the relative location of the prey. For experiment 1, the parameters were set as follows: $\epsilon_1 = 4$, $\epsilon_2 = 4$, $\mathcal{N}_0 = 70$, $\mathcal{N}_1 = 30$, and $\delta = .4$ which was evenly allocated between δ_{i1} and δ_{i2} over the eight possible hill-climbing steps. Experiment 2 had the same values except that $\mathcal{N}_0$ was lowered to 50. The third case used a completely random initial strategy. Its parameters were identical to the second trial. The runs will be denoted as MPHC1, MPHC2, and MPHC3, respectively. The number of captures and the average number of moves to capture over 100 trials are shown in table 4.5. The first two entries in the table are the **best** results for strategies generated by genetic programming. In the case of STGP [23], a single strategy was learned that was used by all the agents. Thus, there is an implicit form of communication between the agents, which are also assumed to be homogeneous. A1 [25] is an agent grouping approach to learning using the "TeamAll" crossover strategy in which there is a potential crossover point in every agent in every group. Given these characteristics, it is a prime candidate for the blocking problem.

Note that the results for STGP and A1 are not those reported in [25]. Rather new simulations were performed using the coordination strategies that were described.

Algorithm	*Captures*	*Avg. Moves*
STGP	26	79.9
A1	0	200
MPHC1	70	45.2
MPHC2	97	71.9
MPHC3	100	72

Table 4.5: Captures and Average number of Moves

This allowed the methods to be compared using the same test cases. In any event, the results were similar to those that were published. From figure 2 in [23], the capture rates for STGP seem to be in the 10% range while the value for A1 is close to 0 [25]. The average path length of 200 for A1 was included to indicate that it failed to capture.

It is also interesting to compare the strategies generated by genetic programming to those produced by hill-climbing. A visualization of the strategies is shown in figures 4.9 to 4.25. **P** indicates the prey and the arrows indicate the direction an agent 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.

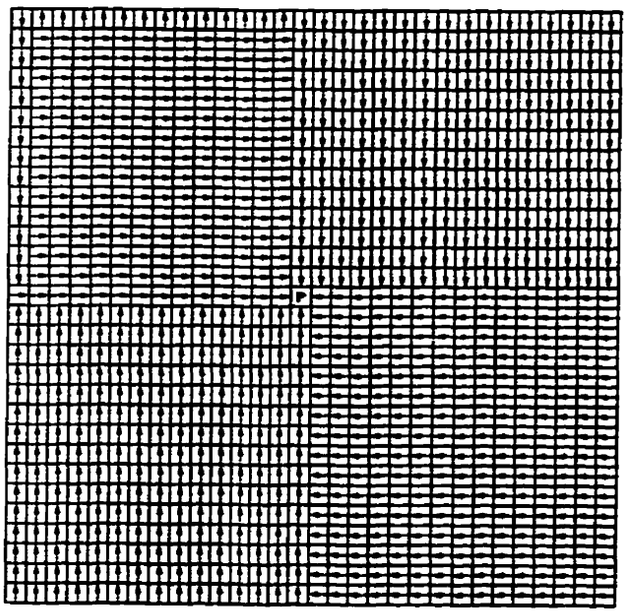

Figure 4.9: Strategy generated by STGP

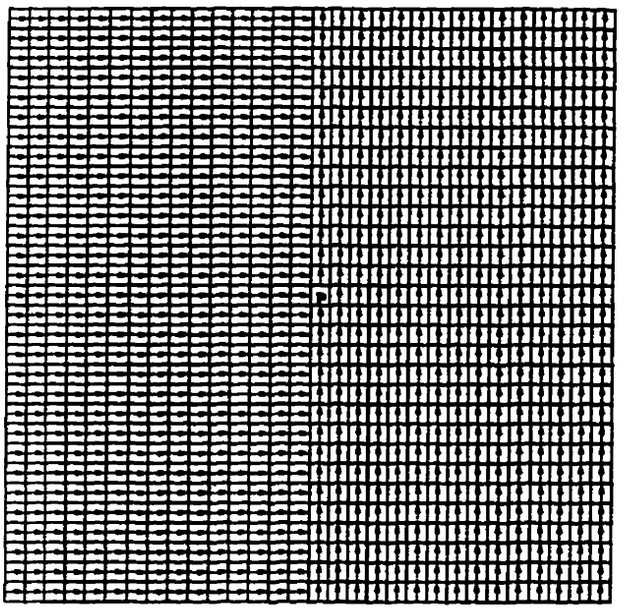

Figure 4.10: Strategy using A1 for Agent 1

The strategy produced by STGP moves the predator to an orthogonal position to the prey and then closes in. A1 produced a strategy which contains 2 orthogonal components that do not necessarily bring it closer to the prey.

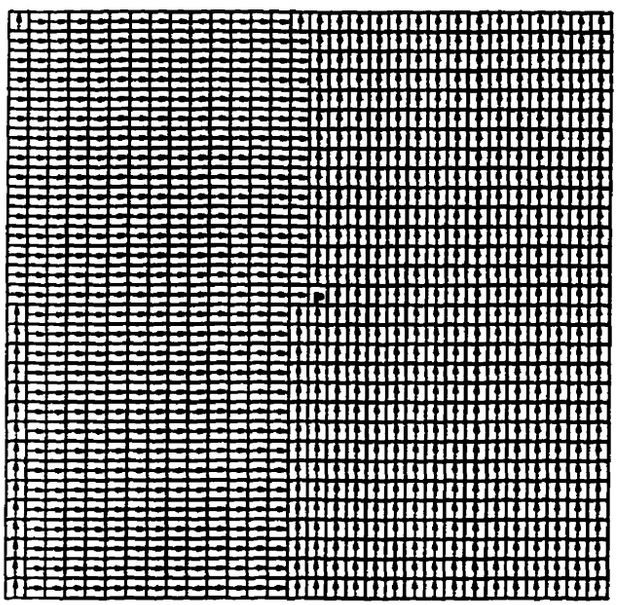

Figure 4.11: Strategy using A1 for Agent 2

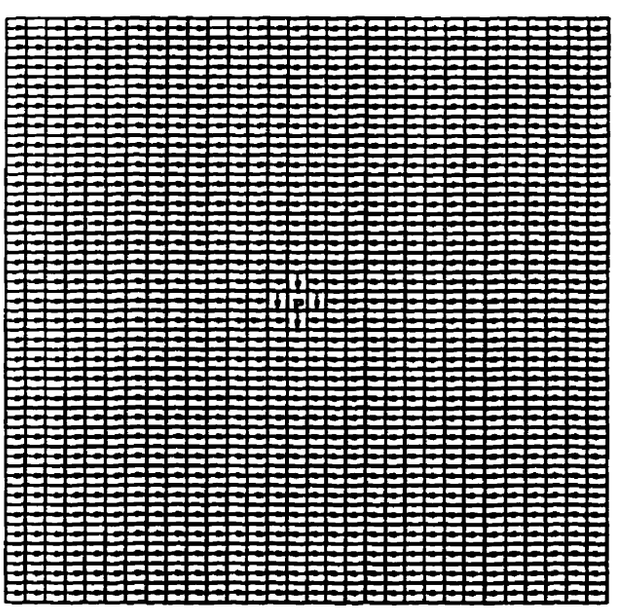

Figure 4.12: Strategy using A1 for Agent 3

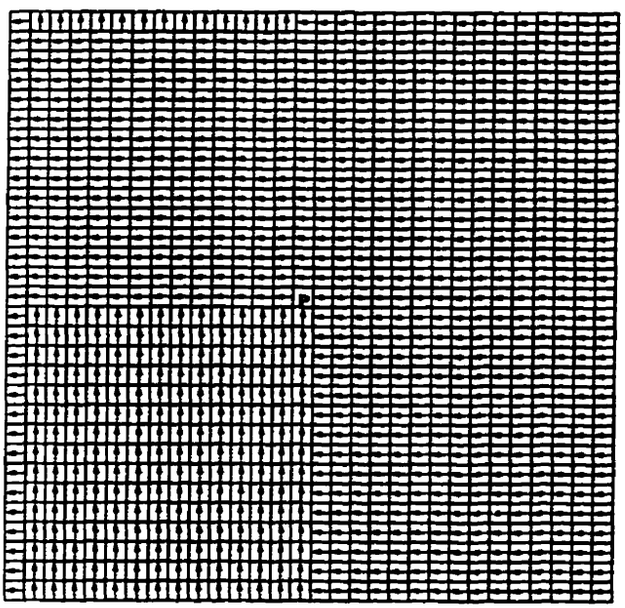

Figure 4.13: Strategy using A1 for Agent 4

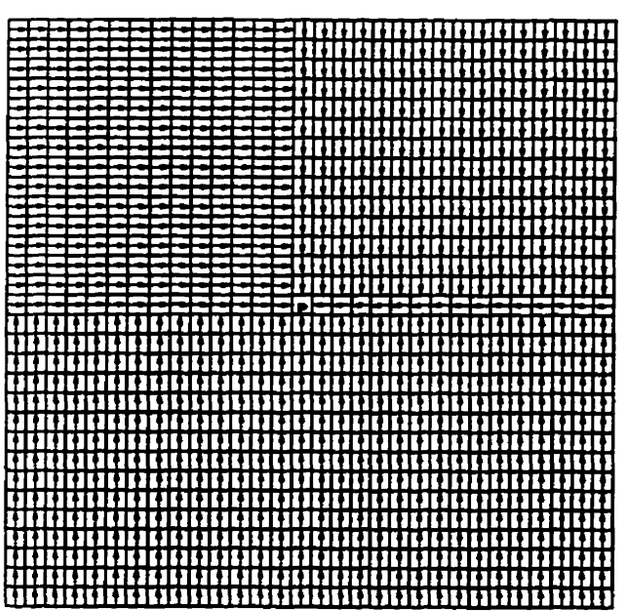

Figure 4.14: Strategy using MPHC1 for Agent 1

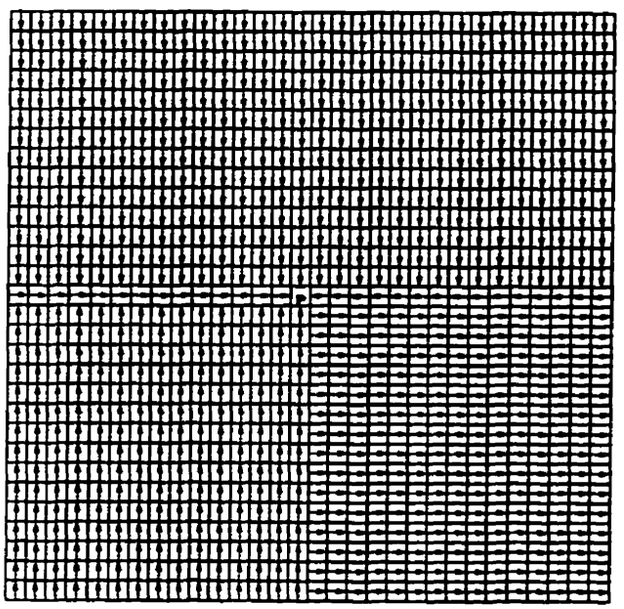

Figure 4.15: Strategy using MPHC1 for Agent 2

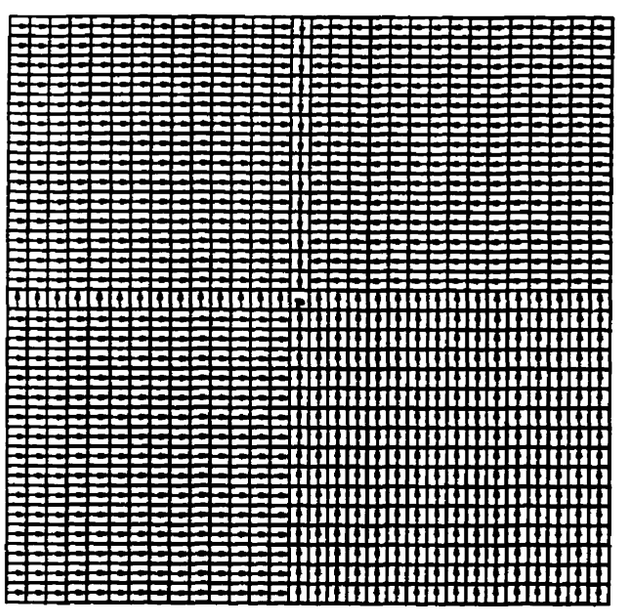

Figure 4.16: Strategy using MPHC1 for Agent 3

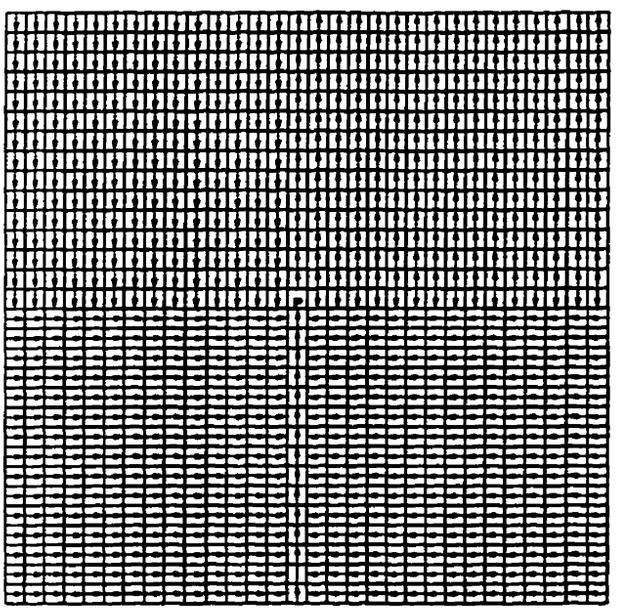

Figure 4.17: Strategy using MPHC1 for Agent 4

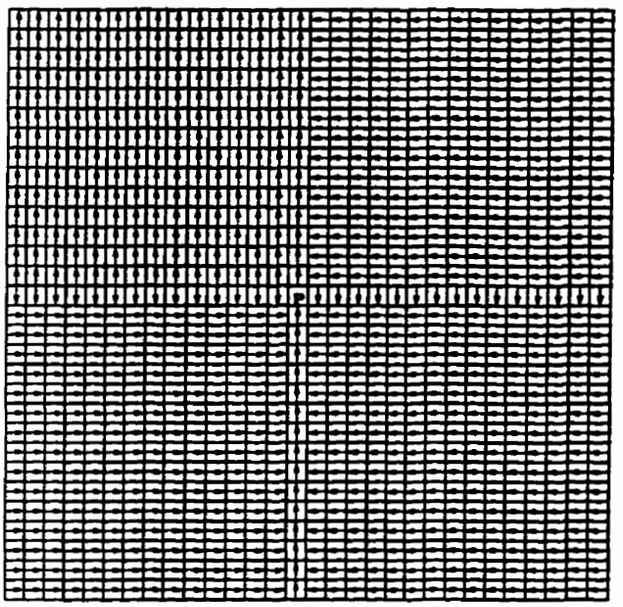

Figure 4.18: Strategy using MPHC2 for Agent 1

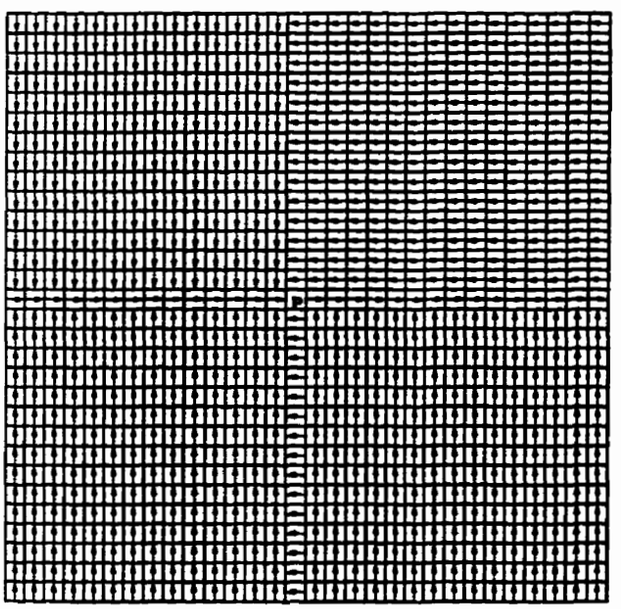

Figure 4.19: Strategy using MPHC2 for Agent 2

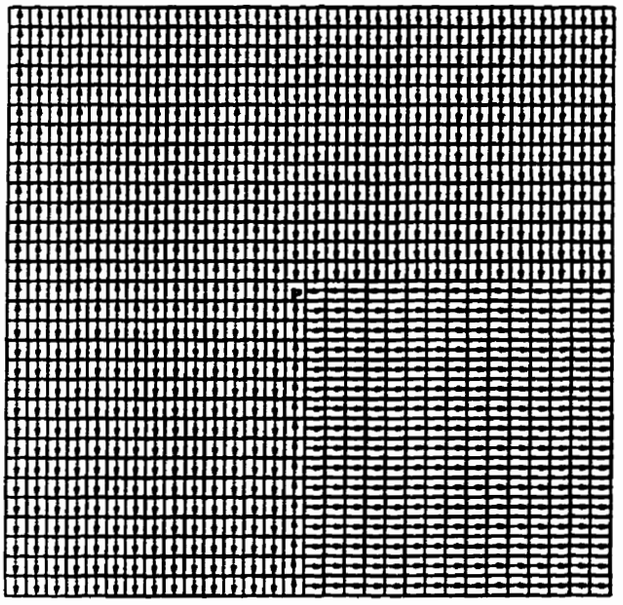

Figure 4.20: Strategy using MPHC2 for Agent 3

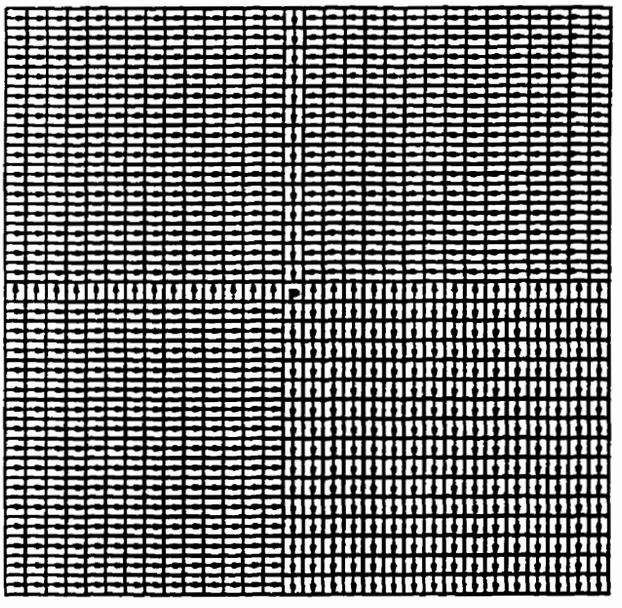

Figure 4.21: Strategy using MPHC2 for Agent 4

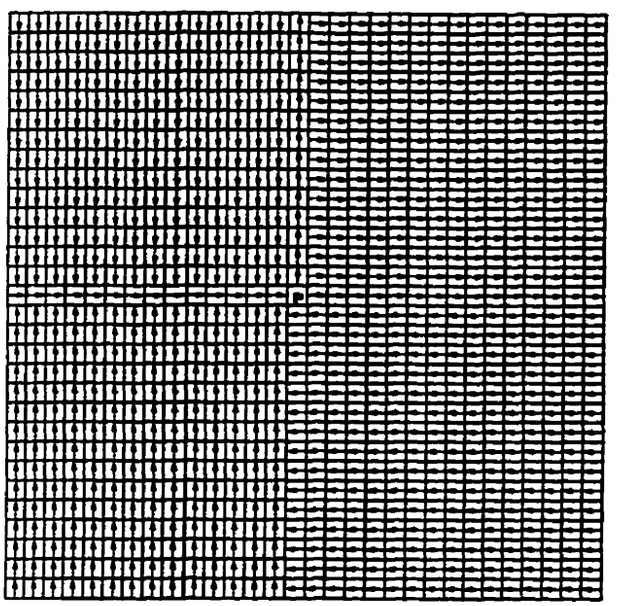

Figure 4.22: Strategy using MPHC3 for Agent 1

The strategies generated by MPHC1 are similar to those of STGP except that they exhibit a degree of specialization. This is particularly true for agent 4, which only approaches the prey from the southern direction. MPHC2 shows even greater differentiation in that agent 1 approaches from the south, agent 2 approaches from the east, agent 3 approaches from the north, south, and west, and agent 4 approaches from the north. The best results are for MPHC3, which again shows specialization, except for the case of agent 2. From figure 4.23 it can be seen that this agent does not approach the prey at all. One of the shorter pursuit paths for this strategy is shown in tables 4.6 and 4.7. The coordinates are not graphically displayed, since the paths overlap and make the resulting diagram confusing. It would seem that agent 2 is "luring" the prey by staying on the diagonal before finally moving into position. This exploits a weakness in the prey in that it moves away from the closest predator. Since Manhattan distances are computed, being on a diagonal would seem to leave an empty spot.

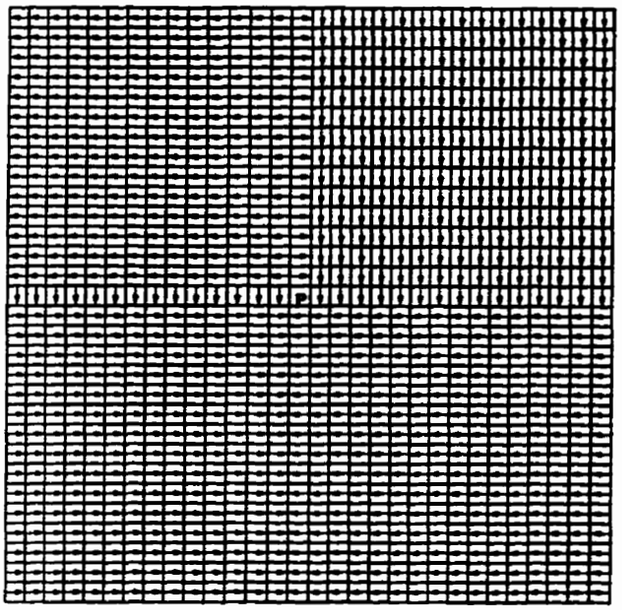

Figure 4.23: Strategy using MPHC3 for Agent 2

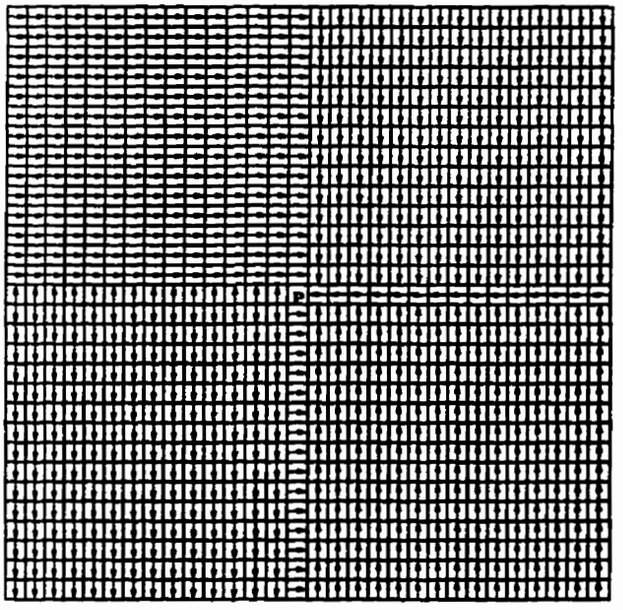

Figure 4.24: Strategy using MPHC3 for Agent 3

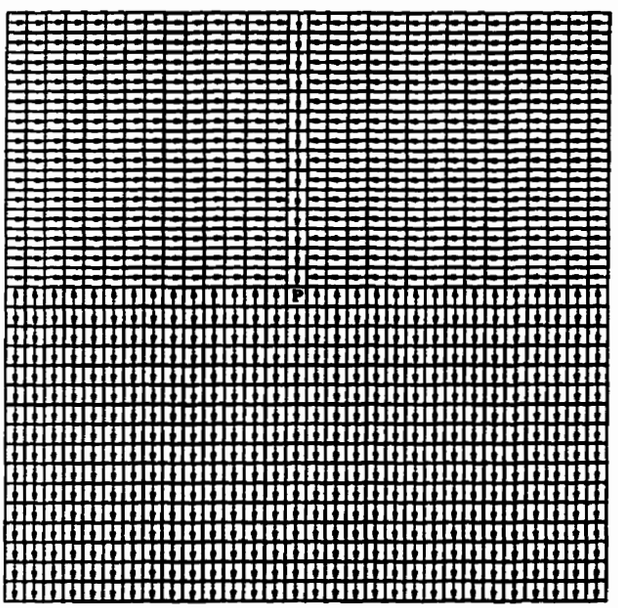

Figure 4.25: Strategy using MPHC3 for Agent 4

Step	*Prey*	*Pred 1*	*Pred 2*	*Pred 3*	*Pred 4*
1	(14 14)	(12 0)	(18 7)	(11 20)	(26 6)
2	(14 13)	(12 29)	(18 8)	(11 21)	(25 6)
3	(14 14)	(12 28)	(18 9)	(11 22)	(24 6)
4	(13 14)	(12 27)	(18 10)	(11 23)	(23 6)
5	(13 15)	(12 26)	(18 11)	(11 24)	(22 6)
6	(13 16)	(12 25)	(18 12)	(11 25)	(21 6)
7	(12 16)	(12 24)	(18 13)	(11 26)	(20 6)
8	(12 16)	(11 24)	(18 14)	(11 27)	(19 6)
9	(11 16)	(11 23)	(18 15)	(11 28)	(18 6)
10	(11 15)	(10 23)	(18 16)	(12 28)	(17 6)
11	(11 14)	(10 22)	(17 16)	(12 27)	(16 6)
12	(10 14)	(10 21)	(16 16)	(12 26)	(15 6)
13	(11 14)	(9 21)	(15 16)	(12 25)	(14 6)
14	(10 14)	(9 20)	(14 16)	(12 24)	(13 6)
15	(10 13)	(9 19)	(13 16)	(12 23)	(12 6)
16	(10 12)	(9 18)	(12 16)	(12 22)	(11 6)
17	(10 11)	(9 17)	(11 16)	(12 21)	(10 6)
18	(9 11)	(9 16)	(10 16)	(12 20)	(10 7)
19	(9 12)	(8 16)	(9 16)	(12 19)	(9 7)
20	(9 11)	(8 15)	(8 16)	(12 18)	(9 8)
21	(10 11)	(8 14)	(9 16)	(12 17)	(9 9)
22	(10 12)	(8 13)	(10 16)	(12 16)	(10 9)
23	(11 12)	(8 12)	(9 16)	(12 15)	(10 10)
24	(11 13)	(9 12)	(10 16)	(12 14)	(11 10)

Table 4.6: Example Pursuit Path

Step	*Prey*	*Pred 1*	*Pred 2*	*Pred 3*	*Pred 4*
25	(11 12)	(9 13)	(11 16)	(12 13)	(11 11)
26	(11 12)	(9 12)	(10 16)	(12 13)	(11 11)
27	(11 12)	(9 12)	(11 16)	(12 12)	(11 11)
28	(11 13)	(10 12)	(10 16)	(12 12)	(11 11)
29	(11 14)	(10 13)	(11 16)	(12 13)	(11 12)
30	(11 14)	(10 14)	(10 16)	(12 14)	(11 13)
31	(11 15)	(10 14)	(11 16)	(12 14)	(11 13)
32	(11 15)	(10 14)	(10 16)	(12 15)	(11 14)
33	(11 15)	(10 15)	(11 16)	(12 15)	(11 14)

Table 4.7: Example Pursuit Path: contd

The average solution length for all strategies are similar except for the case of MPHC1. It is conjectured that the value of $\mathcal{N}_0$ may have caused the algorithm to commit errors. The reduction of $\mathcal{N}_0$ in MPHC2 was an attempt to examine this hypothesis.

4.4.2 Analysis of Intermediate Results

Examining the intermediate steps between hill-climbs can uncover aspects of the system's behavior that cannot be seen from looking at the final results. For instance, is multiagent credit assignment working and is it really the problem as described in Chapter 3?

If credit assignment is a problem in this domain then, it might explain the poor performance of A1. Within probabilistic hill-climbing, the credit assignment problem can be seen by comparing the intermediate results of a sequence of climbs. For

instance, let the predators return transformations $\{\mathcal{T}_1$=**A**, $\mathcal{T}_2$=**B**, $\mathcal{T}_3$=**C**, $\mathcal{T}_4$=**D**$\}$, respectively, after stage 1. Stage 2 then returns $\{\mathcal{T}_1$=**A**,$\mathcal{T}_3$=**C**$\}$ as the joint transformation to be implemented. At a later point, stage 2 returns $\{\mathcal{T}_2$=**K**,$\mathcal{T}_4$=**L**$\}$ where **K** and **B** are mutually exclusive. A blocking problem would have occurred if $\mathcal{T}_2$= **B** had been implemented during the previous step.

During the experiments, the multiagent credit assignment problem was observed in several instances. For example, during the MPHC2 trial, agent 1 returned a transformation that changed the strategy for the southwest direction during stage 1. This change was filtered out by stage 2. Individually, the Δ-utility of this transformation was -8.47. Four climb steps later a transformation in the southwest direction was again returned in stage 1. This change was accepted. The transformation would not have been available if the initial "southwest" transformation had been implemented. Since the problem cropped up using the all north initial condition, it was decided to rerun the experiment without stage 2 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.

4.5 Overall Evaluation

When combined, the results hint at the efficacy of the proposed technique as compared to existing multiagent coordination learning methods. In particular, the findings for the robot navigation and the predator prey domain are promising. Beginning with a completely incorrect and a random strategy, multiagent probabilistic hill-climbing was able to learn a strategy that outperformed the best strategy produced by independent Q-learning and genetic programming. This does not imply that the technique is always better under every circumstance. Rather, if the basis of comparison is the "best" versus the "best" then the proposed method is "better" since the examples have been shown to be better without much parameter or initial state tuning.

One remaining issue is the correspondence between results for the synthetic domain and the simulated domains. The first class of experiments would seem to indicate that achieving ϵ-optimal performance is difficult without the appropriate parameters. However, it must be stressed that the problem distributions for this domain are difficult. Recall that the hard and center distributions had within ϵ percentages of 1.8% and 4.2%. In the case of the skewed distribution, the sample cutoffs may have prevented the algorithm from returning a solution. The simulated domains used as many samples as required to return a result.

This raises the question of direct sample complexity comparisons between the proposed technique and the existing ones. For the predator prey domain, MPHC3 required a total of 23433 stage 1 and 9837 stage 2 samples. Rewritten in terms of stage 1 samples, the total is 170988. The experimental setup for STGP was not revealed in [23], but A1 used a population size of 600 for 1000 generations [25]. The latter numbers imply 600000 samples. However, comparisons are not meaningful, since the samples are predicated on the random initial strategies in the case of MPHC and are arbitrary in the case of A1. Moreover, it can be argued that the representation used in A1 is more expressive and thus the algorithm should take longer. Thus, it may only make sense comparing these values for procedures using the same paradigm.

Chapter 5

An Extension to Probabilistic Hill-Climbing

One problem with hill-climbing on the minimal joint transformation is that this wastes samples, if the remaining joint transformations are "independent" from the one being implemented. Intuitively, independence implies that the remaining transformations will not block transformations that are made possible by the one just implemented. Conversely, it also implies that the recently adopted transformation does not affect these alternate transformations. The second case holds if the implemented and alternate transformations are additive since the joint transformation criterion indicates that they do not otherwise interact. Given independence, these transformations may eventually be adopted. The uncertainty arises from errors in the co-learning approximation of $\mathcal{G}_1$ and from the ϵ indifference zone. In the first case, co-learning may fail to return the same transformations due to error, while in the second, a different but within ϵ transformation is returned. Ignoring the issue of error, it doesn't matter which one of the within ϵ transformations is implemented since all meet the local optimality criterion. Thus, it makes sense to adopt the current "independent" trans-

formation rather than looking for another one. Yet, at the end of each hill-climbing step, the statistics for the unimplemented minimal joint transformations are deleted. This requires that the "independent" transformations be unnecessarily reevaluated.

Consider an example which is similar to the one used in the discussion of credit assignment. Let agent $\mathcal{A}_1$ be able to apply transformation $\mathcal{T}_{11}$ and agent $\mathcal{A}_2$ be able to apply transformations $\mathcal{T}_{21}$ and $\mathcal{T}_{22}$, which are mutually exclusive. As in the previous case, let $\mathcal{T}_{11}$ have the highest utility and let it "enable" $\mathcal{T}_{12}$. In this scenario, $\mathcal{T}_{12}$ does not positively interact with $\mathcal{T}_{22}$, but instead the utility of the transformations are ordered as follows: $\mathcal{U}(\mathcal{T}_{11}) > \mathcal{U}(\mathcal{T}_{12}) > \mathcal{U}(\mathcal{T}_{21}) > \mathcal{U}(\mathcal{T}_{22})$. Also, let the transformations of $\mathcal{A}_1$ be additive with those of $\mathcal{A}_2$. Thus, $\mathcal{T}_{21}$ can be regarded as independent of $\mathcal{T}_{11}$, since it does not block a transformation, which is enabled by the latter transformation and is also additive.

The ordering would cause the algorithm to implement $\{\mathcal{T}_{11}, \mathcal{T}_{12}, \mathcal{T}_{21}\}$ as a climb sequence. Using deletion would require that that the statistics for $\mathcal{T}_{21}$ be recomputed three times. If the independence of the transformations had been realized, then $\mathcal{T}_{21}$ could have been implemented during the first step.

One advantage of the agent grouping technique is that it implicitly exploits the case of independent transformation. The downside of this is that it does so indiscriminantly. The detection of independent transformations would reduce computation without incurring a reduction in utility in the event of a blocking problem or conflict.

5.1 Independent Transformations

The intuitive definition of the blocking problem can be formalized using the following definitions. ϵ's and δ's are omitted for notational clarity.

Definition 11: Enabler

> $\mathcal{T}_A$ *enables* $\mathcal{T}_B$, denoted by $\mathcal{T}_A \longrightarrow \mathcal{T}_B$, if the implementation of $\mathcal{T}_A$ at step i allows for the existence of $\mathcal{T}_B$ at step $i+1$.

Definition 12: Denier

> The opposite of an enabler is a denier. $\mathcal{T}_A$ is a denier of $\mathcal{T}_B$, $\mathcal{T}_A \not\longrightarrow \mathcal{T}_B$, if $\mathcal{T}_A$'s existence prevents $\mathcal{T}_B$ from being available. The relation is not necessarily symmetric. The form of enablers and deniers is dependent on the domain.

Definition 13: Conflict

> At a given climb step, i, $\mathcal{J}^*_{ji}$ *conflicts* with $\mathcal{J}^*_{\mathbf{maxi}}$ if $\exists\ \mathcal{T}_e \in \mathcal{J}^*_{\mathbf{maxi}}, \exists \mathcal{T}_d \in \mathcal{J}^*_{ji} \ni \mathcal{T}_e \longrightarrow \mathcal{T}_c, \mathcal{T}_c \in \mathcal{J}^*_{\mathbf{maxi+K}}, \mathcal{T}_d \not\longrightarrow \mathcal{T}_c$. That is, $\mathcal{T}^*_{ji}$ conflicts with $\mathcal{T}^*_{maxi}$ if there is an transformation in $\mathcal{T}^*_{ji}$ that denies a component of the best minimal joint transformation in some future step that was enabled by the best minimal joint transformation in the current step.

Any minimal joint transformation $\mathcal{J}^*_{ji}$ which does not conflict with $\mathcal{J}^*_{\mathbf{maxi}}$ can be implemented during the same step.

The difficulty with Definition 13 is that $\mathcal{J}^*_{\mathbf{maxi+K}}$ is generally not known ahead of time. Moreover, even if the minimal ϵ-optimal joint transformation is known, this does not imply that $\mathcal{T}_c$ is also known, since the denial relation may only hold in certain problem contexts. In other words, a few problems may have to solved after hill-climbing before the relation is noticed. These factors can be eliminated by relaxing the definition of a conflict.

Definition 14: Possible Conflict

> At a given climb step, i, $\mathcal{J}^*_{ji}$ *possibly conflicts* with $\mathcal{J}^*_{\mathbf{maxi}}$ if $\exists\ \mathcal{T}_e \in \mathcal{J}^*_{\mathbf{maxi}}, \exists \mathcal{T}_d \in \mathcal{J}^*_{ji} \ni \Diamond\mathcal{T}_e \longrightarrow \mathcal{T}_c, \Diamond\mathcal{T}_d \not\longrightarrow \mathcal{T}_c$, where $\Diamond$ denotes possibly.

Definition 15: Independent Transformation

At a given climb step, i, $\mathcal{J}^*_{ji}$ is *independent* of $\mathcal{J}^*_{\mathbf{max}i}$ if it does not possibly conflict with it.

Rather than dealing with *necessary* conflicts, Definition 14 side-steps the issue by looking for all *possible* conflicts. It ignores problem contexts and what the ϵ-local joint transformation will be in the next step. If a transformation has the potential for conflicting with the current best joint transformation then it is tagged as a possible conflict. Implementing only independent transformations is a conservative approach whose worst case performance is identical to that of promoting one joint transformation.

5.2 Independence in STRIPS Type Domains

Building a system that exploits independence requires that enablers, deniers, and possible conflicts be defined. As stated in the definitions, both are dependent on the problem solver. To illustrate the process, a exemplar was developed based on a STRIPS [10] operator representation.

Recall, from the previous chapter, that the STRIPS operators are used in forward-chaining mode and not as they typically are in goal regression. Given that an environment may not be totally controlled by the agents, the problem solving scheme can be viewed as a form of reactive planning. In hindsight, the sequences of operators executed by the agents can be seen as a plan. When viewed in this manner, the relations between operators can be regarded as one of *precondition establishment* [59]. The opposite of *precondition establishment* occurs when an operator deletes a precondition. The deleting operator is commonly known as a *clobberer*. Potential enablers and deniers can be reexpressed in terms of these two relations. The key is to notice

that under the reordering scheme, a transformation is tied to the operator that is being reordered. The implementation of a transformation implies that its associated operator is used in lieu of some other operator with the same preconditions. Thus the transformations can be analyzed by referring to their operators.

Definition 11a: Possible Enabler (under STRIPS)

$\mathcal{T}_A$ possibly *enables* $\mathcal{T}_B$, denoted by $\Diamond\mathcal{T}_A \longrightarrow \mathcal{T}_B$, if $\mathcal{T}_A$ is implemented at step i and either:

1. $op(\mathcal{T}_A)$ possibly *establishes preconds*$(op(\mathcal{T}_B))$ or
2. $\exists\, \mathcal{OP}_C \ni op(\mathcal{T}_A)$ possibly *establishes preconds*$(\mathcal{OP}_C) \wedge \mathcal{OP}_C$ possibly *establishes preconds*$(op(\mathcal{T}_B))$

The function *op* returns the operator that was reordered in a transformation while $\mathcal{OP}_C$ represent an operator from the domain.

Definition 12a: Possible Deniers (under STRIPS)

$\mathcal{T}_A$ is a possible denier of $\mathcal{T}_B$, $\mathcal{T}_A\Diamond \not\longrightarrow \mathcal{T}_B$, if $\mathcal{T}_A$ either:

1. $op(\mathcal{T}_A)$ possibly *clobbers preconds*$(op(\mathcal{T}_B))$ or
2. $\exists\mathcal{OP}_C \ni op(\mathcal{T}_A)$ possibly *establishes preconds*$(\mathcal{OP}_C) \wedge \mathcal{OP}_C$ possibly *clobbers preconds*$(op(\mathcal{T}_B))$

The second component of each definition uses an intervening operator and not a transformation. The operator in $\mathcal{T}_A$ may indirectly interact with the one in $\mathcal{T}_B$ through an existing operator in the strategy. Given that these relations are all problem context dependent and hence unknown, possible establishes and clobberers are identified by unifying add lists with preconditions and delete lists with preconditions, respectively, without referring to the strategies or problem distributions. Thus they can be computed off-line.

5.3 Generating Deletion Lists

Using the definitions, a set of potential deniers can be precomputed from the operators in the domain definition plus one additional operator. This operator arises from the observation that the last step in any plan can be viewed as one that does nothing except declare that the problem has been solved. Thus, the operator's preconditions are the literals in the goal state. Any operator that clobbers a literal added by another operator, which is not part of an explicit precondition, can be said to clobber a precondition for this implicit operator. The algorithm for computing the possibly deniers is shown in Figure 5.1. The first step is to compute the direct clobberers of

GENERATE-DELETION(*operators*)
  $\mathcal{C}$= compute-clobbers(*operators*)
  loop for **x** in *operators* do
    $\mathcal{E}_{\mathbf{x}}$= compute-enablers(**x**)
    $\mathcal{D}_{\mathbf{x}} = \mathcal{E}_{\mathbf{x}} \cap \mathcal{C}$
    $\mathcal{D}_{\mathbf{x}} = \mathcal{D}_{\mathbf{x}} \cup ancestors\mathcal{D}_{\mathbf{x}}$
  end loop

Figure 5.1: Selective Deletion Algorithm

each operator. Then, for each operator, a list of enabled operators/transformations are computed. Intersections between the enabling and clobbering lists are noted and the ancestors of each item are also collected. The second step corresponds to the second condition for being a denier. This process generates an associated list of operators for each operator.

After hill-climbing, the deletion operators for each component of $\mathcal{J}^*_{\mathbf{max}}$ are collected. If one of the alternate joint transformations has a component in this set then the transformation is removed. The remaining joint transformations are then avail-

able to be implemented if their effects are additive with $\mathcal{J}^*$)**max**. This step requires that the stage 2 algorithm be modified. The function **find-ind-max**, shown in Figure 5.2, is substituted for **find-max-trans** in the **joint-transforms** procedure.

FIND-IND-MAX($\mathcal{IT}$,*agents*, ϵ)
  [$\mathcal{MT}$, *m-error*]= find-max-trans($\mathcal{IT}$, ϵ)
  $\mathcal{NT}$= remove-dependent($\mathcal{IT}$, $\mathcal{MT}$)
  $\mathcal{NT}$= remove-non-additive($\mathcal{NT}$)
  if $\mathcal{NT}$
    (
      [$\mathcal{MMT}$, *mm-error*]= find-ind-max($\mathcal{NT}$, *agents*, ϵ)
      return [$\mathcal{MT} \cup \mathcal{MMT}$, *mm-error+m-error*]
)
    return [$\mathcal{MT}$, *m-error*]

Figure 5.2: Modifications for Stage 2

If there are any remaining joint transformations after an initial climb step then the best out of this set must be returned. Next the non-additive transformations are removed. The process recurses using the successive best joint transformations while removing elements from the set of alternate transformations. When the sum of errors from all these "best transformations" drops below δ minus *i-error*, then the system hill-climbs on all of them.

5.4 Application of Selective Deletion to a Restaurant Domain

The selective deletion technique was applied to a simplified restaurant domain. During each time step, there is a 20% chance that from one to three customers will enter

the restaurant. Each customer will then order a hamburger with a 60% chance that it will be topped with mustard and relish and the remaining 40% that it will be topped with lettuce and tomato. They will also order a drink with a 60% chance that it is a Coke and a 40% chance that it is a Seven-up. The task is to learn a coordination strategy involving four agents such that a cost measure is minimized.

At every time step, an agent can execute one of 22 actions depending upon the current state. Some of these operators are shown in Figure 5.3. A complete set of operators can be found in appendix B. Given the task description, the operators fall into one of four categories: making burgers, making toppings, getting drinks, and combining items. The first three classes of operator are similar. If there is an existing burger, topping, or drink, then it is returned, else a burger has to be ordered for cooking, a topped bun made, or a drink order filled. In the latter two cases there are distinct sets of operators, corresponding to the different toppings and drinks. For cooking burgers there are four operators. These do not correspond to different types of burgers, but rather to two different methods of cooking them. One method, `cook-burger1`, is less efficient than the other, `cook-burger2`, and is represented by a sequence of three operators which are required to cook a burger.

A few of the operators do not have preconditions, for instance **condiment1**, **condiment2**, **drink1**, **drink2** and **burger1**. In the first two cases, an agent pre-makes burger toppings without an order. Similarly, the next two cases pre-fill drinks. An agent pre-cooks burger patties without an order in the last case. The combination operators add the toppings to the burger, pool the filled orders, and manage payment for the meal.

Given a fixed simulation duration of 40 time steps, experiments were performed using a cost function based on the weighted sum of how long a customer had to wait for their food, how many orders were unfilled, how many orders were unpaid for at the end of the run, and how many excess burgers, toppings, and drinks remained.

The probabilistic hill-climbing trials used a "lock-step" transformation approach where only the operators at a specific position in each agent's strategy is modified. That is, during the first step, only the operator in the first position of the strategy was changed. Upon hill-climbing, the second position was modified and so on. Since not all agents implement one of their transformations at each step, the current positions across agents may not correspond. To reduce runtime, each agent was limited to two climbs. These constraints eliminated the possibility of arriving at an "optimal" solution, but were necessitated by computational resources.

5.4.1 Performance of a Random Initial Strategy

Each of the 4 agents was supplied with a strategy generated by randomly ordering the operators. The strategies are shown in Table 5.1. A typical execution trace using these strategies is shown in Table 5.2. In this case, only 20 out of the 40 steps is shown. The first column is the step number and the remaining columns, the actions taken by each agent. In some cases, the action is a list with `nil` as its first entry. This means that a conflict has occurred. Two or more agents have simultaneously selected an action for execution that involves a unique binding. Recall that the conflict resolution scheme simply picks the lower numbered agent for execution. The remaining agents do nothing and thus the `nil`. Step 2 is an example of a conflict. Step 1 has produced a burger that needs to be cooked. Since agent 1 has already began cooking the burger and in the process has used up the only stove, agent 2 does nothing during the step. As can be seen from the trace, the strategy is extremely inefficient. At the end of 20 steps, no one has received their order and there are many excess uncooked burgers and mustard/relish toppings. The average cost, over 50 runs, of using this strategy is 516.58.

5.4.2 Learning with Selective Deletion

Applying the generate deletion algorithm to the restaurant domain yields the lists shown in Figures 5.4 and 5.5. Operators that do not have a list have been removed for succinctness. Intuitively, some of the entries in the list make sense. For instance, consider the deletion items for the **burger1** operator, which includes **cook-burger1** and **cook-burger2**. The implementation of **burger1** would increase the number of burgers to be cooked and thus should be paired with a more efficient cooking method such as **cook-burger2**. However, the algorithm does not know this. It only knows that **burger1** establishes a precondition for **cook-burger1** and **cook-burger2**, and that there is a resource contention between these operators. Thus the system should hold off implementing a cooking operator until it has been determined which one has greater utility.

In other cases, the results do not make sense. The list for **drink1-no-d** is one example. Operator **get-drink-seven** interacts with **drink1-no-d** through the *send-dorder* precondition for **get-drink-coke**. However, these operators would never conflict due to the remaining preconditions, since one operator is for Coke and the other is for Seven-up. Again, the algorithm has no knowledge of this and being conservative flags this as a blocking problem. This situation occurs frequently throughout the list and reflects the absence of domain knowledge and the use of possible conflicts. In some domains, the use of possible rather than necessary conflicts may force the deletion lists to include every operator. ALPINE [34], which is a method for generating abstraction hierarchies for planning, suffers from the same problem, since it also uses possible interactions. The hierarchies frequently collapse since they are overconstrained. Note that, in the worst case, the learning algorithm would take as many samples as complete deletion.

Multiagent probabilistic hill-climbing with selective deletion was applied to the

random strategies using the following parameters: $\epsilon_1 = 5$, $\epsilon_2 = 5$, $\delta_1 = .15$, $\delta_2 = .15$, $\mathcal{N}_0 = 15$, $\mathcal{N}_1 = 30$. Recall that a climb limit of two steps per agent was set which corresponds to the total limit of eight. After a total of 6686 stage 1 and 1246 stage 2 samples, the algorithm halted. The execution trace for this learned strategy, on the same problem that produced Table 5.2, is shown in Table 5.3.

As highlighted in boldface, the learned strategy completely fills 3 orders within the 20 steps. On the same 50 problems as before, the learned strategy had an average cost of 279.2 for a savings of 46% over the initial strategy. However, it is not an optimal strategy in that the `cook-burger2` operator should be used to cook the burgers instead of `cook-burger1`. This example illustrates that the algorithm has become trapped in a local minimum.

Using complete deletion, the method consumed 10518 stage 1 and 1185 stage 2 samples. The strategies generated by the two methods were not identical. For instance they differed in two transformations. Complete deletion was also 5% more efficient but this difference can be accounted for by ϵ=10 within each step.

Selective deletion hill-climbed four times as compared to five for the case of complete deletion. During the first step, the former approach implemented one extra minimal joint transformation than complete deletion. This transformation was subsequently implemented by the second technique in the next hill-climbing step. The difference allowed selective deletion to examine an alternate transformation which reduced total sample complexity.

```
(make-operator
    :template  '(take-order ?x)
    :preconds  '((not-order ?x))
    :addlist   '((send-border ?x)(send-torder ?x)
               (send-dorder ?x))
    :dellist   '((not-order ?x)))

(make-operator
    :template  '(get-burger ?x)
    :preconds  '((send-border ?x)(have-order ?y))
    :addlist   '((have-burger ?x))
    :dellist   '((send-border ?x)
               (have-order ?y)))

(make-operator
    :template  '(get-drink-coke ?x)
    :preconds  '((send-dorder ?x)(coke ?x)
               (have-coke ?y))
    :addlist   '((have-drink ?x)(not-pay-d ?x))
    :dellist   '((send-dorder ?x)(coke ?x)
               (have-coke ?y)))

(make-operator
    :template  '(get-top-lt ?x)
    :preconds  '((send-torder ?x)(lettuce-tomato ?x)
               (have-lettuce-tomato ?y))
    :addlist   '((have-top ?x))
    :dellist   '((send-torder ?x)
               (lettuce-tomato ?x)
               (have-lettuce-tomato ?y)))

(make-operator
    :template  '(condiment1)
    :addlist   '((have-lettuce-tomato)))
```

Figure 5.3: Example Operators

Position	*Agent 1*	*Agent 2*	*Agent 3*	*Agent 4*
1	pay-for-food	get-burger	get-burger-no-b	get-top-lt
2	cook-burger1	get-drink-seven	get-burger	get-drink-coke
3	burger1	get-top-lt	cook-burger1b	condiment2
4	get-top-lt	pay-for-food	get-drink-coke	drink2-no-d
5	cook-burger2	drink1-no-d	get-top-lt	burger1
6	garnish-2-no-g	cook-burger1b	get-top-mr	cook-burger2
7	cook-burger1b	get-drink-coke	drink2	garnish-1-no-g
8	drink1	garnish-1-no-g	cook-burger1a	drink1
9	drink1-no-d	cook-burger1	pay-for-food	pay-for-food
10	get-drink-coke	take-order	cook-burger1	cook-burger1b
11	get-top-mr	condiment2	garnish2-no-g	drink2
12	drink2-no-d	drink2	condiment2	cook-burger1
13	drink2	combine	burger1	condiment1
14	get-burger	drink1	get-drink-seven	drink1-no-d
15	get-drink-seven	drink2-no-d	combine	combine
16	combine	garnish-2-no-g	drink2-no-d	get-burger
17	cook-burger1a	get-burger-no-b	garnish1-no-g	get-drink-seven
18	garnish-1-no-g	burger1	cook-burger2	get-burger-no-b
19	get-burger-no-b	cook-burger2	take-order	get-top-mr
20	take-order	cook-burger1a	drink1-no-d	cook-burger1a
21	condiment2	get-top-mr	condiment1	garnish2-no-g
22	condiment1	condiment1	drink1	take-order

Table 5.1: Initial Operator Order

Step	*Agent1*	*Agent2*	*Agent3*	*Agent4*
1	burger1	condiment2	drink2	condiment2
2	cook-burger1	(nil cook-burger1)	drink2	condiment2
3	burger1	take-order	drink2	condiment2
4	burger1	get-drink-seven	get-burger-no-b	condiment2
5	burger1	take-order	get-burger-no-b	condiment2
6	burger1	get-drink-seven	get-burger-no-b	condiment2
7	burger1	garnish-1-no-g	get-burger-no-b	condiment2
8	burger1	get-top-lt	get-burger-no-b	(nil get-top-lt)
9	burger1	take-order	get-burger-no-b	condiment2
10	burger1	get-drink-seven	get-burger-no-b	condiment2
11	burger1	garnish-1-no-g	get-burger-no-b	condiment2
12	burger1	get-top-lt	get-burger-no-b	(nil get-top-lt)
13	burger1	take-order	get-burger-no-b	condiment2
14	burger1	drink1-no-d	get-burger-no-b	condiment2
15	burger1	drink1-no-d	get-burger-no-b	get-drink-coke
16	burger1	garnish-1-no-g	get-burger-no-b	condiment2
17	burger1	get-top-lt	get-burger-no-b	(nil get-top-lt)
18	burger1	take-order	get-burger-no-b	condiment2
19	burger1	garnish-1-no-g	get-burger-no-b	condiment2
20	burger1	get-top-lt	get-burger-no-b	(nil get-top-lt)

Table 5.2: Typical Execution Trace: Original Strategies

```
((take-order ?x)
  ((get-top-mr ?x) (get-top-lt ?x) (get-drink-seven ?x) (get-drink-coke ?x)
    (get-burger ?x) (cook-burger2 ?x) (drink1-no-d ?x) (drink1)
    (drink2-no-d ?x) (drink2) (garnish-1-no-g ?x) (condiment1)
    (garnish-2-no-g ?x) (condiment2) (take-order ?x) (get-burger-no-b ?x)
    (burger1) (cook-burger1 ?x ?y) (cook-burger1a ?x ?y) (cook-burger1b ?x ?y)))

((garnish-1-no-g ?x)
  ((get-top-mr ?x) (take-order ?x) (garnish-2-no-g ?x) (condiment2)))

((garnish-2-no-g ?x)
  ((get-top-lt ?x) (take-order ?x) (garnish-1-no-g ?x) (condiment1)))

((drink1-no-d ?x)
  ((get-drink-seven ?x) (take-order ?x) (drink2-no-d ?x) (drink2)))

((drink2-no-d ?x)
  ((get-drink-coke ?x) (take-order ?x) (drink1-no-d ?x) (drink1)))

((get-burger-no-b ?x)
  ((cook-burger2 ?x) (take-order ?x) (get-burger-no-b ?x) (burger1)
    (cook-burger1 ?x ?y) (cook-burger1a ?x ?y) (cook-burger1b ?x ?y)))

((burger1)
  ((cook-burger2 ?x) (take-order ?x) (get-burger-no-b ?x) (burger1)
    (cook-burger1 ?x ?y) (cook-burger1a ?x ?y) (cook-burger1b ?x ?y)))

((cook-burger1 ?x ?y)
  ((cook-burger1 ?x ?y) (take-order ?x) (get-burger-no-b ?x) (burger1)))
```

Figure 5.4: Deletion Lists: Part 1

```
((cook-burger1a ?x ?y)
  ((cook-burger1 ?x ?y)(take-order ?x)(get-burger-no-b ?x)(burger1)))

((cook-burger1b ?x ?y)
  ((cook-burger1 ?x ?y)(take-order ?x)(get-burger-no-b ?x)(burger1)))

((condiment1)
  ((get-top-mr ?x) (take-order ?x) (garnish-2-no-g ?x) (condiment2)))

((condiment2)
  ((get-top-lt ?x) (take-order ?x) (garnish-1-no-g ?x) (condiment1)))

((drink1)
  ((get-drink-seven ?x) (take-order ?x) (drink2-no-d ?x) (drink2)))

((drink2)
  ((get-drink-coke ?x) (take-order ?x) (drink1-no-d ?x) (drink1))))
```

Figure 5.5: Deletion Lists: Part 2

Step	*Agent1*	*Agent2*	*Agent3*	*Agent4*
1	burger1	condiment2	drink2	condiment2
2	cook-burger1	(nil cook-burger1)	drink2	(nil cook-burger1)
3	burger1	take-order	cook-burger1a	condiment2
4	burger1	get-drink-seven	garnish-1-no-g	condiment2
5	burger1	get-top-lt	(nil garnish-1-no-g)	(nil get-top-lt)
6	burger1	cook-burger1b	get-burger-no-b	condiment2
7	cook-burger1	(nil cook-burger1)	get-burger-no-b	(nil cook-burger1)
8	burger1	get-burger	cook-burger1a	condiment2
9	burger1	cook-burger1b	(nil cook-burger1b)	combine
10	**pay-for-food**	cook-burger1	drink2	(nil cook-burger1)
11	burger1	take-order	cook-burger1a	condiment2
12	get-top-mr	get-burger	(nil get-burger-no-b)	condiment2
13	burger1	get-drink-seven	cook-burger1b	combine
14	**pay-for-food**	cook-burger1	drink2	(nil cook-burger1)
15	burger1	take-order	cook-burger1a	condiment2
16	burger1	get-burger	garnish-1-no-g	condiment2
17	burger1	get-drink-seven	garnish-1-no-g	get-top-lt
18	burger1	cook-burger1b	(nil cook-burger1b)	combine
19	**pay-for-food**	cook-burger1	drink2	(nil cook-burger1)
20	burger1	take-order	cook-burger1a	condiment2

Table 5.3: Typical Execution Trace: Learned Strategy (Selective Deletion)

Chapter 6

Conclusions and Future Research

The primary goal of the thesis was to develop a method to solve the problem of learning multiagent coordination strategies as outlined in section 1.2. That goal has been achieved with the introduction of a new learning approach based on probabilistic hill-climbing. The method has been implemented and evaluated using a variety of experiments.

6.1 Contributions

This thesis has raised many issues in learning multiagent coordination strategies. One method to examine the contributions of this research is to frame them within the context of each issue. The following is a discussion of each topic.

1. *Multiagent Credit Assignment*

   Of the existing iterative multiagent coordination learning techniques, the proposed method is the only one that deals with the credit assignment problem without the use of distributed utility functions or domain knowledge. This al-

lows the method to be applied to a broad range of domains where these two components may not be available.

2. *Convergence/Performance*

   Unlike some previous approaches, multiagent probabilistic hill-climbing is guaranteed to converge and to return a solution. This is done using a well-founded rather than an ad hoc termination criteria. Although, the approximation in stage 1 invalidates probabilistic hill-climbing's performance guarantees, the method has been shown to give better results than existing techniques on two different domains. Moreover, it is the only approach that uses a heuristic version of an optimality criteria.

3. *Scalability/Complexity*

   One trade-off that was made in the design of the algorithm was in the area of guaranteed performance versus computational scalability. Rather than using a grouped approach that would have provided performance guarantees, a heuristic procedure was used to reduce complexity. Using the navigation domain, the co-learning stage of the technique has also been shown to be scalable, not in a computational complexity sense, but by its ability to arrive at a useful strategy involving many agents. A selective deletion mechanism has also been devised to avoid unnecessary resampling after hill-climbing. It has been shown to reduce sample complexity.

In summary, multiagent probabilistic hill-climbing is an attempt at combining the best features of both an independent as well as a grouped approach to learning multi-agent coordination strategies. Independent learning is better in terms of complexity and captures the autonomy of individual agents. Grouped learning can be used to discriminate between useful and useless transformations. The key in combining these two methodologies is a probabilistic termination criteria that ensures the convergence

of independent learning and that allows for the quantification of interaction effects in the combined transformations.

6.2 Future Directions

Although there has been a flurry of activity, research in learning multiagent coordination strategies can still be regarded as in its infancy. The proposed probabilistic hill-climbing approach addresses a number of basic issues, but can be improved in a number of areas. Among these are:

1. *Sampling Strategies*:

   As discussed in section 4.2, one weakness of the Z-heuristic is that it may spend a long time sampling the best transformation rather than sampling other transformations, which may reduce the overall error. This may require an adaptive sampling scheme that takes into account the potential error reduction.

2. *Initial State*:

   A major determining factor in the overall performance of any hill-climbing approach is the initial state. This will directly influence the sample complexity, since the closer the initial state is to an optima, the fewer the number of samples required. Secondly, the initial state also influences the global optimality of the solution. If the starting conditions can be set such that the system avoids local minima, then the final utility of the solution will be increased. One ploy that may be effective is to perform off-line analysis using a uniform distribution assumption. That is, every problem is assumed to be as likely as another.

Appendix A

Statistical Tests

This section is a brief description of the statistical tests that were used throughout the system. It is divided into two sections. The first one discusses how the transformation with the largest expected utility was selected and the second discusses how the interacting joint transformations were determined. In both cases the data is assumed to be normally distributed. The central limit theorem [33] can be invoked here since the computations use differences in utility. If the variances are "reasonably homogeneous" then the theorem will hold. Normal plots have also been constructed from data and the assumption seems valid given the number of samples typically encountered.

A.1 Climbing Tests

Climbing tests are used in both stages of the credit assignment algorithm. In the first stage, they are used to individually compute the components of $\mathcal{G}_2$ while in the second stage they are used to select the best minimal ϵ-joint transformation. As briefly discussed in Chapter 3, a transformation is considered to be the best if its

mean is better than any other transformation with cumulative probability of 1-δ_i or if its mean is within $\pm\epsilon$ of every other transformation with the same probability. The algorithm computes both of these probabilities and considers the larger one . This scheme is the same one used in [4] except for the statistical tests. In [4] it was assumed that the variances are equal and thus the standard normal distribution was used. This assumption was not made and t-tests are used instead.

The following two equations compute the probability corresponding to the cases of being better and within $\pm\epsilon$ respectively.

$$t - cum_\nu\left(\frac{\bar{x}_{max} - \bar{y}_i}{\sqrt{\frac{s^2_{max}}{n_{max}} + \frac{s^2_2}{n_2}}}\right)$$

$$t - cum_\nu\left(\frac{\bar{x}_{max} - \bar{y}_i + \epsilon}{\sqrt{\frac{s^2_{max}}{n_{max}} + \frac{s^2_2}{n_2}}}\right) - t - cum_\nu\left(\frac{\bar{x}_{max} - \bar{y}_i - \epsilon}{\sqrt{\frac{s^2_{max}}{n_{max}} + \frac{s^2_2}{n_2}}}\right)$$

where ν, the degree of freedom, is equal to:

$$\nu = \frac{(s^2_{max}/n_1 + s^2_2/n_2)^2}{\frac{(s^2_{max}/n_{max})^2}{n_{max}-1} + \frac{(s^2_2/n_2)^2}{n_2-1}}$$

and where $t - cum_\nu$ denotes the t-cumulative distribution function.

The first equation is just the standard hypothesis test for comparing two means with unknown and not necessarily equal variances. Equation 2 computes the probability that the difference between the means lies between $\pm\epsilon$. It can be derived by taking the inverse to the standard confidence interval problem. That is, given confidence level γ, compute an interval such that the true mean lies within it with probability γ. In this case, the interval is given and the task is to find the probability.

Using the worst case error model, one minus the maximum of both values is summed for each paired comparison with the current best transformation. If this error falls below δ_i then the system hill-climbs.

A.2 Tests for Interactions

The goal of these tests is to discover if the mean of the joint transformation is significantly better than the sum of the means of its components. If so then there is a positive interaction between the components.

Since the utilities have been gathered using simultaneous extraction, it is possible to perform paired testing to eliminate some of the variability in the data. A new random variable, **d**, is computed for each of the tests to represent the difference between the utility of the potential joint transformation and the sum of the utilities of its components. For instance, one set of components for the joint transformation $\{\mathbf{A\ B\ C}\}$ is $\{\mathbf{A\ B}\ \mathcal{D}_3\}$ and $\{\mathcal{D}_1\ \mathcal{D}_2\ \mathbf{C}\}$. Thus **d** would assume the value of $\mathcal{U}(\mathbf{ABC}) - (\mathcal{U}(\mathbf{AB}) + \mathcal{U}(\mathbf{C}))$ for successive problem instances.

To compute whether a joint transformation has an interaction effect, the probability

$$\prod_{i=1}^{C} t - cum_{\nu}(\frac{\bar{\mathbf{d}}_\mathbf{i}}{s_{\mathbf{d_i}}/\sqrt{n}})$$

is determined, where C is the set of components and ν is 2n-2. This represents the probability that joint transformation has a utility which is greater than its parts.

Similarly, the probability that the joint transformation is within $\pm\epsilon$ of its components is computed by

$$\prod_{i}^{C}(t - cum_{\nu}(\frac{\bar{\mathbf{d}}_\mathbf{i} + \epsilon}{s_{\mathbf{d_i}}/\sqrt{n}}) - t - cum_{\nu}(\frac{\bar{\mathbf{d}}_\mathbf{i} - \epsilon}{s_{\mathbf{d_i}}/\sqrt{n}}))$$

As discussed in Chapter 3, the hypothesis associated with the larger of these two values is taken as the current hypothesis. One minus its probability is taken as the probability of error and these probabilities are summed across all potential joint transformations.

In some cases, the mean and variance of the difference is exactly zero. This happens when the components of the joint transformation do not interact. The utility of the joint transformation is simply the sum of the utilities of the individual components. As a result, the statistical tests that have been used up till now are inappropriate since the data behaves deterministically. This stems from the use of simulated rather than real data. There is no variability in the data and thus the difference is 0. In this case, the Hoeffding inequality

$$Pr[\bar{X} - \mu \geq t] \leq e^{-2nt^2}$$

is used to compute the probability that the means differ by more than ϵ. The only hypothesis in this case is that the means of the transformations are within ϵ and hence the combined transformation is not a joint transformation.

Appendix B

Restaurant Domain Operators

The complete set of 22 operators used in the restaurant domain is listed below.

```
(make-operator
    :template   '(pay-for-food ?x)
    :preconds   '((have-food ?x) (have-drink ?x)
                (not-pay-f ?x) (not-pay-d ?x))
    :addlist    '((paid ?x))
    :dellist    '((have-food ?x) (have-drink ?x)
                (not-pay-f ?x) (not-pay-d ?x)))

(make-operator
    :template   '(take-order ?x)
    :preconds   '((not-order ?x))
    :addlist    '((send-border ?x)(send-torder ?x)
                (send-dorder ?x))
    :dellist    '((not-order ?x)))
```

```
(make-operator
    :template  '(get-burger ?x)
    :preconds  '((send-border ?x) (have-order ?y))
    :addlist   '((have-burger ?x))
    :dellist   '((send-border ?x)
                (have-order ?y)))

(make-operator
    :template  '(get-drink-coke ?x)
    :preconds  '((send-dorder ?x) (coke ?x)
                (have-coke ?y))
    :addlist   '((have-drink ?x) (not-pay-d ?x))
    :dellist   '((send-dorder ?x) (coke ?x)
                (have-coke ?y)))

(make-operator
    :template  '(get-drink-seven ?x)
    :preconds  '((send-dorder ?x) (seven ?x)
                (have-seven ?y))
    :addlist   '((have-drink ?x) (not-pay-d ?x))
    :dellist   '((send-dorder ?x) (seven ?x)
                (have-seven ?y)))

(make-operator
    :template  '(get-top-lt ?x)
    :preconds  '((send-torder ?x) (lettuce-tomato ?x)
                (have-lettuce-tomato ?y))
    :addlist   '((have-top ?x))
    :dellist   '((send-torder ?x) (lettuce-tomato ?x)
                (have-lettuce-tomato ?y)))
```

```
(make-operator
    :template  '(garnish-1-no-g ?x)
    :preconds  '((send-torder ?x) (lettuce-tomato ?x))
    :addlist   '((have-lettuce-tomato)))

(make-operator
    :template  '(garnish-2-no-g ?x)
    :preconds  '((send-torder ?x) (mustard-relish ?x))
    :addlist   '((have-mustard-relish)))

(make-operator
    :template  '(drink1-no-d ?x)
    :preconds  '((send-dorder ?x) (coke ?x))
               :addlist '((have-coke)))

(make-operator
    :template  '(drink2-no-d ?x)
    :preconds  '((send-dorder ?x)
               (seven ?x)
    :addlist   '((have-seven)))

(make-operator
    :template  '(get-top-mr ?x)
    :preconds  '((send-torder ?x) (mustard-relish ?x)
               (have-mustard-relish ?y))
    :addlist   '((have-top ?x))
    :dellist   '((send-torder ?x) (mustard-relish ?x)
               (have-mustard-relish ?y)))
```

```
(make-operator
    :template  '(combine ?x)
    :preconds  '((have-top ?x)(have-burger ?x))
    :addlist   '((have-food ?x)(not-pay-f ?x))
    :dellist   '((have-top ?x)(have-burger ?x)))

(make-operator
    :template  '(get-burger-no-b ?x)
    :preconds  '((send-border ?x))
    :addlist   '((burger-order)))

(make-operator
    :template  '(burger1)
    :addlist   '((burger-order)))

(make-operator
    :template  '(cook-burger1 ?x ?y)
    :preconds  '((burger-order ?x) (free-stove ?y))
    :addlist   '((have-order1a ?x ?y))
    :dellist   '((burger-order ?x)
                 (free-stove ?y)))

(make-operator
    :template  '(cook-burger1a ?x ?y)
    :preconds  '((have-order1a ?x ?y))
    :addlist   '((have-order1b ?x ?y))
    :dellist   '((have-order1a ?x ?y)))
```

```
(make-operator
    :template  '(cook-burger1b ?x ?y)
    :preconds  '((have-order1b ?x ?y))
    :addlist   '((have-order ?x) (free-stove ?y))
    :dellist   '((have-order1b ?x ?y)))

(make-operator
    :template  '(cook-burger2 ?x)
    :preconds  '((burger-order ?x) (free-stove ?y))
    :addlist   '((have-order ?x))
    :dellist   '((burger-order ?x)))

(make-operator
    :template  '(condiment1)
    :addlist   '((have-lettuce-tomato)))

(make-operator
    :template  '(condiment2)
    :addlist   '((have-mustard-relish)))

(make-operator
    :template  '(drink1)
    :addlist   '((have-coke)))

(make-operator
    :template  '(drink2)
    :addlist   '((have-seven)))
```

Bibliography

[1] Aha, D., Kibler, D., Albert, M., "Instance-based Learning Algorithms", *Machine Learning*, 6 (1), pp. 37-66, 1991.

[2] Berry, D., Fristedt, B., *Bandit Problems: Sequential Allocation of Experiments*, Chapman and Hall, London, 1985.

[3] Cammarata, S., McArthur, D., Steeb, R., "Strategies of Cooperation in Distributed Problem Solving", *Proceedings of IJCAI-83*, pp. 767-770, 1983.

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

[5] Dowell, M., L., Stephens, L., M., "Mage: Additions to the AGE Algorithm for Learning in Multiagent Systems", unpublished manuscript, 1996.

[6] Durfee, E., Lesser, V., Corkhill, D., "Coherent Cooperation Among Communicating Problem Solvers", *IEEE Transactions on Computers*, 36, pp. 1275-1291, 1987.

[7] Durfee, E., *Coordination of distributed problem solvers*, Kluwer, Boston, 1988.

[8] Durfee, E., H., Lesser, V., R., Corkill, D., D., "Cooperative Distributed Problem Solving" in *The Handbook of Artificial Intelligence: Volume 4*, ed. by A. Barr, P., R., Cohen, E., A., Feigenbaum, Addison-Wesley, pp. 85-147, 1989.

[9] Etzioni, O., "Hypothesis Filtering: A Practical Approach to Reliable Learning", *Proceedings of ML-88*, pp. 416-429, 1988.

[10] Fikes, R., Nilsson, N., "Strips: a new approach to the application of theorem proving to problem solving", *Artificial Intelligence*, 2, pp. 189-208, 1972.

[11] Findler, N., V., Lo, R., "An Examination of Distributed Planning in the World of Air Traffic Control", *Journal of Parallel and Distributed Computing*, 3, pp. 411-431, 1986.

[12] Fong, W., L., "A Quantitative Study of Hypothesis Selection", *Proceedings of ML-95*, pp. 226-234, 1995.

[13] Fukuda, T., Iritani, G., Ueyama, T., Arai, F., "Optimization of Group Behavior on Cellular Robotic System in Dynamic Environment", *Proceedings of ICRA-94*, pp. 1027-1032, 1994.

[14] Ghenniwa, H.,, Kamel, M., "Coordination in Cooperative Distributed Systems: A Rational, Intelligent Agent Model", submitted to *Artificial Intelligence*, 1997.

[15] Gratch, J., DeJong, G., "A Hybrid Approach to Guaranteed Effective Control Strategies", *Proceedings of ML-91*, pp. 509-513, 1991.

[16] Gratch, J., "Composer: A Decision-theoretic Approach to Adaptive Problem Solving", UIUCDCS-R-93-1806, *University of Illinois at Urbana-Champaign*, 1993.

[17] Gratch, J., Chien, S., DeJong, G., "Improving Learning Performance Through Rational Resource Allocation", *Proceedings of AAAI-94*, pp. 576-581, 1994.

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

[19] Greiner, R., "Finding the optimal derivation strategy in a redundant knowledge base", *Artificial Intelligence*, 50(1), pp. 95-115, 1991.

[20] Greiner, R., Jurišica, I., "A Statistical Approach to Solving the EBL Utility Problem", *Proceedings of AAAI-92*, pp. 241-248, 1992.

[21] Greiner, R., "PALO Algorithms", unpublished report, 1993.

[22] Gu, P., Maddox, A., B., "A Framework for Distributed Reinforcement Learning", *Working notes of the Adaptation and Learning in Multiagent Systems Workshop*, IJCAI-95, pp. 26-31, 1995.

[23] Haynes, T., Sen, S., "Evolving behavioral strategies in predators and prey", *Working notes of the Adaptation and Learning in Multiagent Systems Workshop*, IJCAI-95, pp. 32-37, 1995.

[24] Haynes, T., Sen. S., "Learning Cases to Resolve Conflicts and Improve Group Behavior", *Working Notes of the AAAI Agent Modelling Workshop*, 1996.

[25] Haynes, T., Sen, S., Schoenefeld, D., Wainwright, R., "Evolving a Team", *AAAI Fall Symposium on Genetic Programming*, 1996.

[26] Ho, F., Kamel, M., "Learning Multiagent Coordination using Probabilistic Hill-climbing", *Proceedings of Robolearn-96*, pp. 38-44, 1996.

[27] Ho, F., Kamel, M., "Learning to Coordinate Multiple Robots", submitted to *ICRA-97*, 1996.

[28] Ishida, T., Yokoo, M., Gasser, L., "An Organizational Approach to Adaptive Production Systmes", *Proceedings of AAAI-90*, pp. 52-58, 1990.

[29] Ishida, T., *Parallel, distributed, and multiagent production systems*, Springer-Verlag, Berlin, 1994.

[30] Jennings, N. R., "Coordination Techniques for Distributed Artificial Intelligence" in *Foundations of Distributed Artificial Intelligence*, ed. by G. M. P. O'Hare and N. R. Jennings, Wiley, pp. 187-210, 1996.

[31] Kaelbling, L., P., *Learning in Embedded Systems*, MIT Press, 1993.

[32] Kamel, M., Syed, A., "A multiagent task planning method for agents with disparate capabilities", *Journal of Advanced Manufacturing Technology*, 9, pp. 408-420, 1994.

[33] Kirkpatrick, E., *Introductory Statistics and Probability for Engineering, Science and Technology*, Prentice-Hall, Englewood Cliffs, 1974.

[34] Knoblock, C., "Automatically generating abstractions for problem solving", CMU-CS-91-120, *Carnegie-Mellon University*, 1991.

[35] Kolodner, J., L., *Case-based Reasoning*, Morgan Kaufmann, San Mateo, 1993.

[36] Korf, R., E., "A simple solution to pursuit games", *Working Papers of the 11th International Workshop on Distributed Artificial Intelligence*, pp. 183-194, 1992.

[37] Mataric, M., "Designing and Understanding Adaptive Group Behavior", *Adaptive Behavior*, 4(1), pp. 51-80, 1995.

[38] MacKenzie, D., Cameron, J., Arkin, R., "Specification and Execution of Multiagent Missions", GIT-COGSCI-95/02, *Georgia Tech*, 1995.

[39] Michalski, R., "Inferential Theory of Learning as a Conceptual Basis for Multistrategy Learning", *Machine Learning*, 11, pp. 111-151, 1993.

[40] Mikami, S., Kakazu Y., Fogarty, T., C., "Co-operative Reinforcement Learning By Payoff Filter", *Proceedings of ECML-95*, pp. 319-322, 1995.

[41] Minton, S., *Learning search control knowledge: an explanation-based approach*, Kluwer, Boston, 1988.

[42] Newell, A., Steier, D., "Intelligent control of external software systems", *Artificial Intelligence in Engineering*, 8, pp. 3-21, 1993.

[43] Parker, L., "L-ALLIANCE: A Mechanism for Adaptive Action Selection in Heterogeneous Multi-Robot Teams", ORNL/TM-13000, *Oak Ridge National Labs.*, 1995.

[44] Prasad, M., V., Lesser, V., R., "Learning Situation-specific Coordination in Generalized Partial Global Planning", *AAAI 1996 Spring Symposium on Adaptation and co-learning in Multiagent Systems*, 1996.

[45] Rivest, R., L., Yin, Q., "Simulation Results for a New Two-armed Bandit Heuristic", in *Computational Learning Theory and Natural Learning Systems Vol. 1: Constraints and Prospects*, ed. by S., J., Hanson, G., A., Drastal, R., L., Rivest, MIT Press, Cambridge, pp. 477-486, 1994.

[46] Sen, S., Sekaran, M., Hale, J., "Learning to coordinate without sharing information" *Proceedings fo AAAI-94*, pp. 426-431, 1994.

[47] Sen, S., Sekaran, M., "Multiagent coordination with learning classifier system", in *Working Notes of Adaptation and Learning in Multiagent Systems Workshop*, IJCAI-95, pp. 84-89, 1995.

[48] *Working Notes of Adaptation and Learning in Multiagent Systems Workshop*, ed. by S., Sen, IJCAI-95, 1995.

[49] *Working Notes of Adaption, Co-evolution and Learning in Multiagent Systems Symposia*, ed. by S., Sen, AAAI Spring Symposium Series, 1996.

[50] Sugawara, T., Lesser, V., "On-Line Learning of Coordination Plans", **COINS-TR-93-27**, *University of Massachusetts*, 1993.

[51] Swanepoel, J., W., H., Geertsema, J., C., "Sequential Procedures with Elimination for Selecting the Best of k Normal Populations", *South African Statistics Journal*, 10, pp. 9-30, 1976.

[52] Tennenholtz, M., "On computational social laws for dynamic non-homogeneous social structures", *J. Expt. Theor. Artif. Intell.*, 7, pp. 379-390, 1995.

[53] Turnbull, Weiss., "A class of sequential procedures for k-sample problems concerning normal means with unknown unequal variances", in *Design of Experiments: Ranking*

and Selection: Essay in Honor of Robert E. Bechhofer, ed. by. T., J., Santer and A., C., Tahmhane, Marcel Decker, 1984.

[54] Valiant, L., "A theory of the learnable", *Communications of the ACM*, 27(1), 1984.

[55] Watkins, C.,J., *Learning from Delayed Rewards*, PhD thesis, Kings College, Cambridge, 1989.

[56] Weiß, G., "Action Selection and Learning in Multi-Agent Environments", *Proceedings of SAB-92*, pp. 502-510, 1992.

[57] Weiß, G., "Learning to Coordinate Actions in Multi-agent Systems", *Proceedings of IJCAI-93*, pp. 311-316, 1993.

[58] Whitehead, S., Ballard, D., "Learning to perceive and act by trial and error", *Machine Learning*, 7, pp. 45-93, 1991.

[59] Yang, Q., "A theory of conflict resolution in planning", *Artificial Intelligence*, 58, pp. 361-392, 1992.