# Learning Cases to Resolve Conflicts and Improve Group Behavior

Thomas Haynes and Sandip Sen
600 South College Ave.
Department of Mathematical & Computer Sciences
The University of Tulsa
Tulsa, OK 74104-3189 USA
e-mail: [haynes,sandip]@euler.mcs.utulsa.edu

**Abstract**

Groups of agents following fixed behavioral rules can be limited in performance and efficiency. Adaptability and flexibility are key components of intelligent behavior which allow agent groups to improve performance in a given domain using prior problem solving experience. We motivate the usefulness of individual learning by group members in the context of overall group behavior. In particular, we propose a framework in which individual group members learn cases from problem-solving experiences to improve their model of other group members. We use a testbed problem from the distributed AI literature to show that simultaneous learning by group members can lead to significant improvement in group performance and efficiency over agent groups following static behavioral rules.

**Running title:** Multiagent case based learning

# 1 Introduction

Research in multiagent systems have focused on the problem of coordinating groups of cooperative as well as self–interested agents (Bond and Gasser, 1988). Whereas achieving coordination in non–cooperative situations poses a more challenging problem in general, cooperative agent groups must also be responsive to environmental demands and the problem solving state of individual group members to achieve effective group performance. A critical problem in multiagent systems is that the optimal action for an individual agent from its local view of a problem–solving scenario might not be the optimal action for the entire group from the global view of the same problem–solving scenario (Ordeshook, 1995). Thus an agent can evaluate the utility of its actions at two levels: individual and group. The group level calculations require more information and impose greater cognitive load, whereas the individual level calculations may not always yield desirable results. If agents in a group are likely to interact, utility calculations from even the individual perspective requires reasoning about the possible actions of some or all of the group members. To reason accurately, each individual in a closely–coupled group should model the behavior of other group members, and use these models to derive expectations of the actions of other group members that can possibly infringe on its own plan of actions. This analysis holds irrespective of whether agents are cooperative, antagonistic, or indifferent to other agents.

If an agent's interactions with other agents is fairly infrequent and the environment is stationary, then a static set of behavioral rules may be sufficient in effectively fulfilling local goals. For a large number of practical and interesting scenarios, however, either agents interact with other agents of unknown composition or all possible agent interactions cannot be foreseen. Adaptation and learning are key mechanisms by which agents can modify their behavior on-line to maintain a viable performance profile in such scenarios (Sen, 1996; Weiß and Sen, 1996).

If agents have accurate and consistent models of other group members, reasoning can be expedient and help in achieving group coordination. But often, even in cooperative groups, an individual may not have an up to date model of fellow workers. This may be because of either different agent designers are not willing to share the internal details of their agent constitutions or agent behaviors change as the agent learns new abilities. In either case, a desirable characteristics of such agent groups is that these agents learn to adapt to each other over time. This is often witnessed in human groups where a collection of individuals go through a phase of adjustments before they can effectively perform as a team. The research question to focus on in this context is: what are the mechanisms available by which individual group members can, over time, better adjust to others and more effectively contribute to the group cause?

One approach for adaptation in a group can be for an agent to start with a very coarse or approximate model of other group members. For example, it can start with the default assumption that every one else is like itself, and modify this model based on experience. Since, in most realistic multiagent system, agents are likely to interact in unforeseen ways, a dynamic model of the group must be maintained by an individual. Problems of modeling another agent based on passive observation are many: discrepancy between the expected and actual capabilities, goals, relationships, etc. of the observed agent may lead to an inferred model which is inaccurate and misleading; different agents may perceive different views of the environment and hence the observing agent may not be able to correctly infer the motivations for a given action taken by another agent; actions can take different intervals of time and agents can be acting asynchronously. Even if agents are allowed to communicate, communication delays, improper use of language, different underlying assumptions, etc. can prevent agents from developing a shared common body of knowledge (Halpern and Moses, 1990).

Given the above assumption about the initial model of other agents, an adaptive agent can possibly use various different learning methods to incrementally improve its model of other group members. It would be preferable to use an incremental, rather than batch learning model, and for most multi-agent systems, an anytime learning algorithm that is computationally cheap in both the knowledge acquisition and application phases is desirable. Our goal in this research is to show that given some generic behavioral rules that are effective in achieving local goals in the absence of other agents, but are ineffective when they have to share resources with other group members, agents can adapt their behavior to achieve their goals in the presence of other agents. Some of the assumptions in our work are: agents are provided with a default set of behavioral rules to follow; repeated interaction with other group members allow agents to modify these behavioral rules in some but not in all cases; agents are motivated to achieve local goals but are cognizant of global goals; agents are autonomous; agent perceptions are accurate; agents do not communicate explicitly; all agents act and adapt concurrently.

The elimination of communication between agents further limits the use of existent mechanisms in multiagent systems literature that are used to achieve group coordination (Lesser, 1995). Though we value the wealth of information that can be shared and utilized by agents to improve group performance, our goal is to push the limits of group performance that can be achieved when agents are intelligently building and using models of others without any explicit help from them. The reason for our choice is to investigate how far we can go without using explicit communication. After having leveraged as much as possible from this mode of group problem solving, we believe we will be better equipped to add and exploit communication skills in agents.

We now introduce our proposed model of adaptation with which agents can use problem-solving experience to both update the model they have of other agents, and to some extent predict what others are going to do in a specific situation. This predictive abilities allow agents to choose actions that are less likely to be mutually conflicting or disruptive. We propose a learning framework in which agents learn cases to override default behavioral rules. When the actual outcome of the action of an agent using its behavioral rules is not consistent with the expected outcome based on the model the agent has of other agents, the agent recognizes that a conflict has occurred and that its behavior is not appropriate in that situation. For those situations, the agent learns exceptions to its behavioral rules that are likely to prevent future conflicts. Agents follow their behavioral rules except when a learned case suggests alternative actions. Through this process, the agents dynamically evolve a behavior that is suited for the group in which it is placed. The multiagent case–based learning (MCBL) algorithm thus utilizes exceptions to a default ruleset, which describes the behavior of an agent. These exceptions form a case library. The agent does not reason with these cases, as in CBR (Kolodner, 1988), but rather adapts an inaccurate individual model to improve performance. Though researchers have used CBR in multiagent systems (Sycara, 1987), little work has been done in learning cases in multiagent systems (Garland and Alterman, 1995; Prasad et al., 1995).

A typical multiagent situation in which case learning can be effectively used to adapt local behavior can be seen in the interactions of Adam and his cat Buster: Buster is diabetic, and must receive insulin shots every morning; he must also be given some food with his shot. Adam decides to administer the shot when he wakes up to go to work. He discovered that Buster would react to the sound of the alarm, and go to his food dish. As the alarm clock does not go off on weekends, the cat learned it has to wake up Adam to get its food. The latter is an exception to the routine behavior, and is learned when the cat's expectation of the alarm clock going off in the morning is not met.

The rest of this paper is organized as follows: Section 2 briefly case–based reasoning and defines multiagent case–based learning within multiagent systems. Section 3 discusses the predator–prey domain and why it is a suitable testbed for our research into MCBL. Section 4 describes the case windows considered for indexing into the case library. Section 5 presents our results from applying MCBL to the multiagent domain. Section 6 illustrates the applicability of MCBL in other multiagent domains. Section 7 discusses the implications of our findings in both the testbed domain and in multiagent systems in general.

# 2 Case–Based Learning

Case–based reasoning (CBR) (Golding and Rosenbloom, 1991; Hammond et al., 1990; Kolodner, 1993) is a reasoning process for information retrieval and modification of solutions to problems. A *case* is typically comprised of a representation of a state of a domain and a corresponding set of actions to take to lead from that state to another desired state. These actions could be either a plan, an algorithm, or a modification of another case's actions. A *case library* is a collection of cases. A CBR algorithm contains a module to determine if there is a case that matches the current state of a domain, and so then it is retrieved and used as is. If there is no such match, then cases that are similar to the current state are retrieved from the case library. The set of actions corresponding to the most relevant case is then adapted to fit the current situation. Cardie (Cardie, 1993) defined case–based learning (CBL) as a machine learning technique used to extend instance–based learning (IBL) (Aha et al., 1991). The IBL algorithm retrieves the nearest instance (for our purposes, an instance can be thought of a case) to a state, and performs the suggested actions. There is no case adaptation if the retrieved instance is not a direct match to the current state. With CBL, adaptation may be required.

We view cases as generalizations of sets of instances, and in the context of multiagent systems, we define MCBL as a learning system by which an agent can extend its default behavioral strategy to allow it to respond to exceptions to those rules. The adaptation lies in translating the general case to specific instances. In our framework, the cases in the MCBL system are used by agents to preferentially order their actions. In a single agent system, the state represents the environment, and in multiagent systems, it represents the environment and the agent's expectations of the actions of other agents. In the following we present our formalization of a CBL system tailored for use in multiagent systems.

**What do cases represent?** The behavioral rules that an agent has can be thought of as a function which maps the state ($s$) and the applicable action set ($A$) of an agent to a preference ordering of those actions:

$$BH(s, A) \Rightarrow A' \ = \ < a_{x_1} a_{x_2} \ldots a_{x_k} > .$$

The cases an agent learns allows it to modify this preference ordering:

$$CB(s, A') \Rightarrow A'' \ = \ < a'_{x_1} a'_{x_2} \ldots a'_{x_j} >, j \leq k.$$

A case need not fire every time the agent is to perform an action, i.e., $A''$ can be the same as $A'$. Cases can be positive or negative (Golding and Rosenbloom, 1991; Hammond et al., 1990). A positive case informs the agent what to do, i.e. it reorders the set of actions. A negative case can reorder the actions and/or delete actions from the set. The cases used in the system we are presenting in this paper are negative in the sense that they eliminate one or more of the most preferred actions as suggested by behavioral rules.

**When do agents learn cases?** An agent learns a case when its expectations are not met. If either the behavioral rules or a case predict that given a state $s_n$ and the application of an action $a_x$, the agent should expect to be in state $s'_n$, and the agent does not end up in that state, a case is learned by the corresponding agent. This case will then cause the action $a_x$ not to be considered the next time the agent is in state $s_n$. In multiagent systems, we expect cases will be learned primarily from unexpected interactions with other agents. Cases can be generalized by eliminating irrelevant features from the representation of the state. For example, if another agent is too far away to influence the state of an agent, $A_i$, then the expectations of its behavior should not be included by $A_i$ as it either indexes or creates a new case.

**What happens as models change?** If agent $A_i$ learns an exception to its default behavior based on an interaction with agent $A_j$ and agent $A_j$ does not modify its behavior, then $A_i$ does not have to check to see if that exception has to be modified at some later time. In a system where both agents are modifying their behavioral rules, $A_i$ must check to see if $A_j$ took the action corresponding to the case. If it has not, then $A_j$'s behavioral rules have changed, and $A_i$ must update its model of $A_j$.

# 3 The Pursuit Problem

We use a concrete problem from the distributed Artificial Intelligence (DAI) literature to illustrate our approach of MCBL in multiagent systems. The predator-prey, or pursuit, domain has been widely used in DAI research as a testbed for investigating cooperation and conflict resolution (Haynes and Sen, 1996; Haynes et al., 1995; Korf, 1992; Stephens and Merx, 1990). In this section, we provide an overview of the prior research into the domain and motivate why we utilize learning in aiding the predator agents in capturing the prey.

The earliest research allowed the agents to communicate their intentions to each other. Korf proved that simple greedy agents could always capture the prey without any form of communication (Korf, 1992). This effectively stopped any further research into this domain. In our earlier research, we have shown that some of the assumptions made by Korf facilitate capture and if these assumptions are made more "realistic", then either communication between the predators or more complex agents, employing memory, are needed to effectively capture the prey (Haynes et al., 1995). We disallow communication and investigate the adaptation of the individual to the group. Through on-line interaction, the agents learn coordination and cooperation.

## 3.1 History

The original version of the predator–prey pursuit problem was introduced by Benda, *et al.* (Benda et al., 1986) and consisted of four blue (predator) agents trying to capture a red (prey) agent by surrounding it from four directions on a grid–world. The movement of the prey agent was random. No two agents were allowed to occupy the same location. Agent movements were limited to either a horizontal or a vertical step per time unit. The goal of this problem was to show the effectiveness of nine organizational structures, with varying degrees of agent cooperation and control, on the efficiency with which the predator agents could capture the prey.

The approach undertaken by Gasser *et al.* (Gasser et al., 1989) postulated that the predators could occupy and maintain a *Lieb configuration* (each predator occupying a different quadrant, where a quadrant is defined by diagonals intersecting at the location of the prey) while homing in on the prey. This study, as well as the study by Singh (Singh, 1990) on using group intentions for agent coordination, lacks any experimental results that allow comparison with other work on this problem.

Stephens and Merx (Stephens and Merx, 1989; Stephens and Merx, 1990) performed a series of experiments to demonstrate the relative effectiveness of three different control strategies. They defined the local control strategy where a predator broadcasts its position to other predators when it occupies a neighboring location to the prey. Other predator agents then concentrate on occupying the other locations neighboring the prey. In the distributed control strategy, the predators broadcast their positions at each step. The predators farther from the prey have priority in choosing their target location from the preys neighboring location. In the centralized–control strategy, a single predator directs the other predators into subregions of the *Lieb configuration*. Stephens and Merx experimented with thirty random initial positions of the predators and prey problem, and discovered that the centralized control mechanism resulted in capture in all configurations. The distributed control mechanism also worked well and was more robust. They also discovered the performance of the local control mechanism was considerably worse. In their research, the predator and prey agents took turns in making their moves. We believe this is not very realistic. A more realistic scenario is for all agents to choose their actions concurrently. This will introduce significant uncertainty and complexity into the problem.

The earlier research into the predator–prey domain involved explicit communication between the predator agents. Korf (Korf, 1992) claimed that such expensive communication was unnecessary, as simple greedy algorithms always lead to capture. In his research, Korf further claims that the orthogonal game, a discretization of the continuous world which allows only horizontal and vertical movements, is a poor approximation. In a diagonal version of the game, Korf developed several greedy solutions to problems where eight predators are allowed to move orthogonally as well as diagonally. In Korf's solutions, each agent chooses a step that brings it nearest to the predator. A *max norm* distance metric (maximum of $x$ and $y$ distance between two locations) is used by agents to chose their steps. The predator was captured in each of a thousand random configurations in these games. But the *max norm* metric does not produce stable captures in the orthogonal game; the predators circle the prey, allowing it to escape. Korf replaces the previously used randomly moving prey with a prey that chooses a move that places it at the maximum distance from the nearest predator. Any ties are broken randomly. He claims this addition to the prey movements makes the problem considerably more difficult.

## 3.2 Motivation for Learning Cases

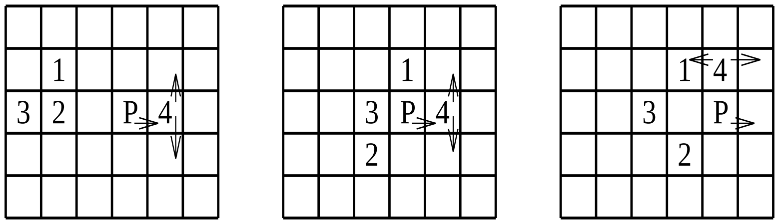
The goal is for four predator agents to capture a prey agent. In spite of its apparent simplicity, it has been shown that the domain provides for complex interactions between agents and no hand-coded coordination strategy is very effective (Haynes et al., 1995). Simple greedy strategies for the predators have long been postulated to efficiently capture the prey (Korf, 1992). The underlying assumption that the prey moves first, then the predators move in order simplifies the domain such that efficient capture is possible. Relaxing the assumption leads to a more natural model in which all agents move at once. This model has been shown to create deadlock situations for simple prey algorithms of moving in a straight line (Linear) or even not moving at all (Still) (Haynes et al., 1995)! Two possible solutions have been identified: allowing communication and adding state information. We investigate a learning system that utilizes past expectations to reduce deadlock situations.

The predator agents have to capture the prey agent by blocking its orthogonal movement. The game is typically played on a 30 by 30 grid world, which is toroidal (Stephens and Merx, 1990). The behavioral strategies of the predators use one of two distance metrics: *Manhattan distance* (MD) and *max norm* (MN). The MD metric is the sum of the differences of the $x$ and $y$ coordinates between two agents. The MN metric is the maximum of the differences of the $x$ and $y$ coordinates between two agents. Both algorithms examine the metrics from the set of possible moves, i.e., moving in one of the four orthogonal directions or staying still, and select a move corresponding to the minimal distance metric. All ties are randomly broken.

Korf (Korf, 1992) claims in his research that a discretization of the continuous world that allows only horizontal and vertical movements is a poor approximation. He calls this the orthogonal game. Korf developed several greedy solutions to problems where eight predators are allowed to move orthogonally as well as diagonally. He calls this the diagonal game. In Korf's solutions, each agent chooses a step that brings it nearest to the predator. The max norm distance metric is used by agents to chose their steps. The predator was captured in each of a thousand random configurations in these games. Korf does not however report the average number of steps until capture. But the max norm metric does not produce stable captures in the orthogonal game; the predators circle the prey, allowing it to escape. Korf replaces the previously used randomly moving prey with a prey that chooses a move that places it at the maximum distance from the nearest predator. Any ties are broken randomly. He claims this addition to the prey movements makes the problem considerably more difficult.

The MN algorithm, as described by Korf (Korf, 1992), does not allow the predators to move to the cell occupied by the prey. (In his research, the prey moves first, followed by the predators in order. Thus conflicts are resolved between predators and prey by serialization.) Figure 1 illustrates a problem with this restriction. The cells to the *North* and *South* of **predator 4** are as equally distant from the **prey P** as the cell currently occupied by **predator 4**. Since all ties are non–deterministically broken, with each movement of the agents, there is a 66% probability that **predator 4** will allow the **prey P** to escape.

Assuming a Linear prey moving *East*, Figure 1 also illustrates the failure of the MN metric algorithms to capture a Linear prey. It is possible that a predator can manage to block the prey, but it is not very likely that it can keep the prey blocked long enough for a capture to take place. It is also possible that once captured, the prey may escape the MN metric algorithms. The MD metric algorithms do not suffer from this inability to make stable captures. They do however have a drawback which both the Linear and Still prey algorithms expose. We found that MD metric algorithms stop a Linear prey from advancing. Our original hypothesis was that the Linear prey moved in such a manner so as to always keep the predators "behind" it. Thus, the inability to capture it was due to not stopping its forward motion. We started keeping track of blocks, i.e., a situation in which a predator blocks the motion of the prey, and discovered that the MD metric algorithms were very good at blocking the Linear prey.




(a) (b) (c)

Figure 1: A possible sequence of movements in which a MN metric based predator tries to block the **prey P**. (a) **predator 4** manages to block **P**. Note that **4** is just as likely to stay still as move *North* or *South*. (b) **predators** **1** and **3** have moved into a capture position, and **predator 2** is about to do so. Note that **4** is just as likely to stay still as move *North* or *South*. (c) **predator 4** opts to move to the *North*, allowing the **prey P** to escape. Note that **4** is just as likely to stay still as move *East* or *West*.

The MD strategy is more successful than the MN in capturing a Linear prey (22% vs 0%) (Haynes et al., 1995). Despite the fact that it can often block the forward motion of the prey, its success is still very low. The MD metric algorithms are very susceptible to deadlock situations, such as in Figure 2. If, as in Figure 2(a), a predator manages to block a Linear prey, it is typical for the other predators to be strung out behind the prey. The basic nature of the algorithm ensures that positions orthogonal to the prey are closer than positions off the axis. Thus, as shown in Figures 2(b) and (c), the remaining predators manage to close in on the prey, with the exception being any agents who are blocked from further advancement by other agents. The greedy nature of this algorithm ensures that in situations similar to Figure 2(c), neither will **predator 2** yield to **predator 3** nor will **predator**

**3** go around **predator 2**. While the MN metric algorithms can perform either of these two actions, predator agents employing it are not able to keep the Linear prey from advancing. It is also evident that once the Linear prey has been blocked by a MD metric algorithm, the prey algorithm degenerates into the Still algorithm. This explains the surprising lack of captures for a prey which does not move.

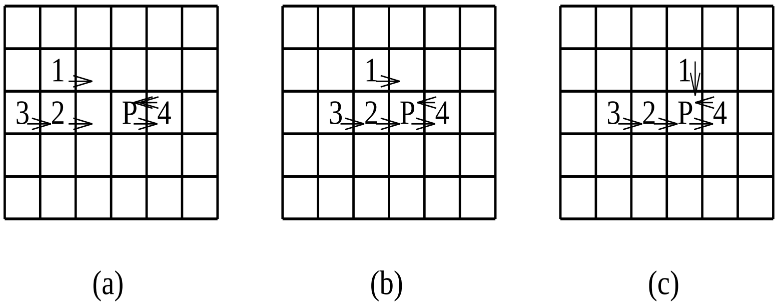



Figure 2: A possible scenario in which a MD metric based predator tries to block the **prey P**. (a) **predator 4** manages to block **P**. **predator**s **1**, **2**, and **3** move in for the capture. (b) **predator 2** has moved into a capture position. (c) **predator 1** has moved into a capture position. **predator 2** will not yield to **predator 3**. They are in deadlock, and the **prey P** will never be captured.

The question that arises from these findings is how should the agents manage conflict resolution? A plausible answer can be found in the ways humans manage conflict resolution, using past history stored as typical cases (Kolodner, 1993). If **predator 1** senses that **predator 2** is in its *Northeast* cell, and it has determined to move *North*, then it realizes that if **predator 2** moves *West* there will be a conflict. **Predator 1** should then learn not to move *North* in the above situation, but rather to its next most preferred direction.

From one move to the next, the MD algorithm usually suffices in at least keeping a predator agent equidistant from the prey. Since the prey effectively moves 10% slower than the predators and the grid world is toroidal, the prey must occasionally move towards some predators in order to move away from others. Therefore the predators will eventually catch up with it. It is when the predators either get close to the prey, or bunched up on one of the orthogonal axes, that contention for desirable cells starts to come into play.

In this research, we examine multiagent case–based learning of potential conflicts. The default rule employed by predators is to move closer to the prey, unless an overriding case is present. If a case fires, the next best move is considered. This process continues until a move is found without a corresponding negative case. If all moves fire a negative case, then the best move according to the default behavior should be taken. If the suggested move, either by the default rule or a case firing, does not succeed, then a new case is learned.

For the majority of moves in the predator–prey domain, either the max norm or MD metric algorithms suffice in at least keeping the predator agents the same distance away from the prey. As discussed later, the prey effectively moves 10% slower than the predators, the grid world is toroidal and the prey must occasionally move towards some predators to move away from others. Therefore the predators will eventually catch up with it. Contention for desirable cells begins when the predators either get close to the prey or are bunched up on one of the orthogonal axes. What the predators need to learn is table manners. Under certain conditions, i.e. when two or more predator agents vie for a cell, the greedy nature of the above algorithms must be overridden. We could simply order the movements of the predators, allowing **predator 1** to always go first. But it might not always be the fastest way to capture the prey. No ordering is likely to be more economical than others under all circumstances.

Also, if we consider real world predator–prey situations, the artificial ordering cannot always be adhered to. Consider for example a combat engagement between fighter aircraft and a bomber. If there are only two fighters, the ordering rule suggests that **fighter 1** always moves before **fighter 2**. If they are in the situation depicted in Figure 3(a), then **fighter 1** cannot fire on the **bomber B**, because doing so will hit **fighter 2**. Clearly, **fighter 2** should first move *North* or *South*, allowing both it and the other fighter to have clear fire lanes. But under the proposed ordering of the movements, it cannot move in such a manner. So, the default rules is that **fighter 1** moves before **fighter 2**, with an exception if **fighter 2** is in front of **fighter 1**. The rule can be modified such that the agent in front gets to move first. However, if we add more fighters, then the situation in Figure 3(b) does not get handled very well. How do **fighter 2** and **fighter 4** decide who shall go first? What if they both move to the same cell *North* of **fighter 2**? These are the very problems we have been discussing with the MD metric algorithm.

What is needed is a dynamic learning mechanism to model the actions of

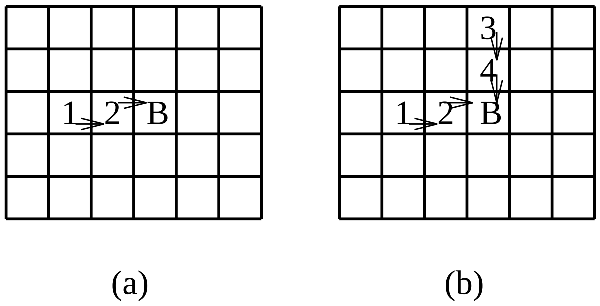



Figure 3: Conflicts in firing lanes for fighter planes strafing a bomber. (a) **fighter 1** is blocked from firing by **fighter 2**, and (b) Not only is **fighter 1** blocked, but so is **fighter 3** by **fighter 4**.

other agents. Until the potential for conflicts exist, agents can follow their default behaviors. It is only when a conflict occurs that an agent learns that another agent will act a certain way in a specific situation $S_j$. Thus agent $A_i$ learns not to employ its default rule in situation $S_j$; instead it considers its next best action. As these specific situations are encountered by an agent, it is actually forming a case–base library of conflicts to avoid. As an agent learns cases, it begins to model the actions of the group. Each agent starts with a rough model of the group, and improves it by incrementally refining the individual models of other agents in the group.

## 3.3 Enhanced Behavioral Rules

In order to facilitate efficient capture, i.e., provide the agents with the best set of default rules, we enhanced the basic MD algorithm. This will also provide a better challenge for the learning algorithm to improve on. If we consider human agents playing a predator–prey game, we would see more sophisticated reasoning than simple greedy behavioral rules. When faced with two or more equally attractive actions, a human will spend extra computational effort to break the tie. Let us introduce some human agents: Alex, Bob, Cathy, and Debbie. Bob and Debbie have had a fight, and Bob wants to make up with her. He asks Alex what should he do. Alex replies that in similar situations he takes Cathy out to dinner. Bob decides that either Burger King or Denny's will do the trick (He is a college student, and hence broke most of the time). In trying to decide which of his two choices is better, he predicts how Debbie will react to both restaurants. Denny's is a step up from Burger King, and she will probably appreciate the more congenial atmosphere.

In the predator–prey domain, such a situation is shown in Figure 4(a). **Predator 1** has a dilemma: both of the cells denoted by $x$ and $y$ are 2 cells away from **Prey P**, using the MD metric. The sum of the distances between all the possible moves from $x$ and **Prey P** is 8 and the sum from $y$ to the **Prey P** is 10. Therefore using this algorithm, which we call the look ahead tie–breaker, **predator 1** should chose $x$ over $y$.

A second refinement comes from what happens if the look ahead tie–breaker yields equal distances for $x$ and $y$? Such a scenario is shown in Figure 4(b). Then **predator 1** should determine which of the two cells is less likely to be in contention with another agent. Predators do not mind contending for cells with the prey, but they do not want to waste a move bouncing off of another predator. By the least conflict tie–breaking algorithm, **predator 1** should pick $y$ over $x$ ($y$ has 0 contentions, while $x$ has 1).

Suppose that Bob and Debbie have had another fight, but this time Alex and Cathy also have fought. Furthermore, a new restaurant, the Kettle, has opened up in town. Since the Kettle is on par with Denny's, Bob is again faced with a need to break a tie. As he knows that Alex and Cathy have fought, he believes that Alex will be taking her out to make up with her. Bob does not want to end up at the same restaurant, as he and Debbie will have to join the other couple, which is hardly conducive to a romantic atmosphere. He decides to model Alex's behavior. Like Bob, Alex is a student and has never eaten at the Kettle. Since Cathy is placated by being taken to Denny's and Alex does not like changing his routine, then Alex will most likely take her there. Thus Bob decides to take Debbie to the Kettle. Notice that if Bob had not accounted for a change in the environment, then his case would have caused a conflict with his goal.

# 4 Case Representation and Indexing

The ideal case representation for the predator–prey domain is to store the entire board and to have each case inform all predators where to move. There are two problems with this setup: the number of cases is too large, and the agents do not act independently. Each agent could store this case library

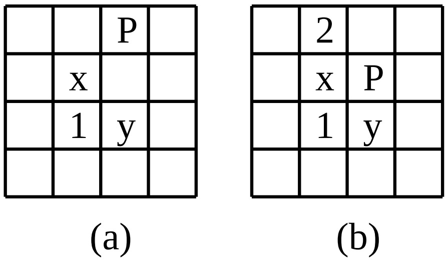



Figure 4: In both (a) and (b), the cells marked $x$ and $y$ are equal distant via the MD metric from the **Prey P** for **predator 1**. (a) $x$ is chosen because the sum of the possible moves from it to **Prey P** is less than the $y$'s sum of moves, and in (b) $y$ is chosen because while the look ahead is equal, there is a potential for conflict with **predator 2** at $x$.

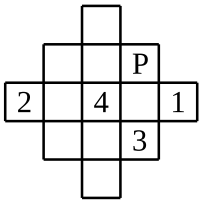



Figure 5: Representation of a simple case window for **predator 4**.

independently, but the problem size explodes. An effective "case window" for each predator is to represent all cells that could directly cause conflicts with any move taken by the agent, as shown in Figure 5. A drawback to this approach is that agents can be present in the case window, but not actually be part of the case. For example, **predator 2**'s location does not impact the desired move of *East* for **predator 4** in Figure 5. Another problem with this window is that **predator 2** could be just *North* of **predator 1**, and cause either **predator 1** or **prey P** to move differently than in the scenario presented. Finally, the search space is still too large.

Unless the entire world is used as a case, any narrowing of the case window is going to suffer from the above points of the "effective" case window presented above. This is symptomatic of an "open" and "noisy" domain: the same case can represent several actual configurations of the domain being modeled. The price to pay to have a clean case base is in the size of the search space. If we accept that the case windows are going to map to more

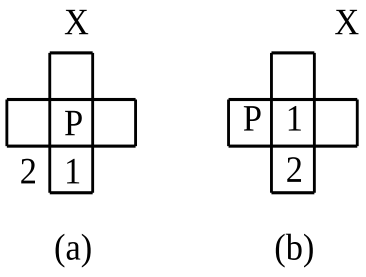



Figure 6: Representation of cases for **predator 1** in the example: (a) Moving (note **predator 2** is outside the case window), and (b) Staying still. The X denotes which of the three areas the prey is occupying.

than one physical situation, then clearly the issue is how to make the search space manageable. The case window in Figure 5 encompasses the potential conflicts when the agent moves in any of the allowable directions. If we limit the case window to simply represent the potential conflicts that can occur "after" the agent selects a move based on the default rules or learned case, then we can utilize the case windows shown in Figure 6. The case window in Figure 6(a) is rotationally symmetric for the directions the agent can choose, and the case window in Figure 6(b) is applied for when the agent remains stationary.

Our cases are negative in the sense they tell the agents what not to do. (A positive case would tell the agent what to do in a certain situation (Golding and Rosenbloom, 1991).) A crucial piece of information in deciding local action is where does the agent believe the other agents are going to move? This is modeled by storing the orientation of the prey's position with respect to the desired direction of movement of the agent. Specifically, we store whether the prey lies on the agent's line of advance or if it is to the left or right of the line. In the case window of Figure 9, the prey's relation to the line of advance is marked with a 'X'.

An agent has to combine its behavioral rules and learned cases to choose its actions. This is shown algorithmically in Figure 7. When an agent prepares to move, it orders its possible actions by the default rules (the MD distance metric with the additional tie–breaking mechanisms). It then iterates down the ranked list, and checks to see if a negative case advises against that move. To index a case, the agent first determines whether the possible action is for movement or staying still, which determines the four cells whose

1 *Preferentially order actions by behavioral rules.*
2 *Choose the first action which does not cause a negative case to fire, i.e., one containing a conflict.*
3 *If the state corresponding to the selected action is not reached, then the agent must learn a case.*

Figure 7: Algorithm for selecting actions based on negative cases.

contents it will examine for potential conflicts. The contents can be summed to form an unique integer index in a base number system reflecting the range of contents. The first possible action which does not have a negative case is chosen as the move for that turn.

We used case windows as shown in Figure 6 while learning to capture the Still prey. The results were promising, but we found that the predators would falsely learn cases, which hindered the efficient capture of the prey. Such an erroneous scenario is shown in Figure 8. The problem lies in **predator 1**'s interpretation of the scenario. It tried to move to the location of **prey P** and failed. It does not learn in situations involving **prey P**. Now **predator 2** pushes **predator 1**, and **predator 1** should learn to yield to **predator 2** if it is again faced with the situation shown in Figure 8(a). The case it learns is the stationary case window shown in Figure 6(b). This case informs **predator 1** that if its current best action is to stay still, then it should consider the next best action. However, this case will never fire because **predator 1**'s best action is to move into **prey P**'s cell, and since the predators do not learn cases against the prey, there is no case that will cause it to consider its remaining actions. Thus the exact same conflict will occur, and **predator 1** will re-learn the same case without ever applying it.

The case that should be learned is that when **predator 1** is in the situation depicted in Figure 8(a), then it should consider its next best action. However, as can be seen in Figure 6(a), such a situation can not be learned with the current case windows. In order to capture the necessary information, we decided to merge the two case windows of Figure 6 into one, which is shown in Figure 9. Now agents can store cases in which there is a potential for conflict for both the resource it desires and the resource it currently possess. The learned case is shown in Figure 9. If the same situation, as in

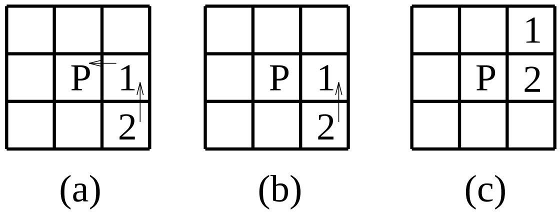



Figure 8: **predator 1** learns the wrong case. (a) **predator 1**'s best action is to move towards **prey P**. (b) **predator 1** bounces back from the **prey P**. It has now stored the incorrect fact that it has stayed still and not moved *East*. (c) **predator 2** pushes out **predator 1**, and **predator 1** learns the incorrect case that if its best action is to stay still in the configuration shown in (a), then it should select its next best action.

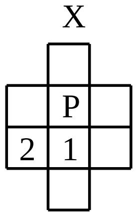



Figure 9: Case window for **predator 1**.

Figure 8(a), is encountered, the case will fire and **predator 1** will consider its next best action. With the enhanced rules, this will be to move *North*.

The same case can represent several actual configurations of the domain being modeled. If the case windows are going to map to more than one physical situation, then clearly the issue is how to find the most relevant general case. If we limit the case window to represent the potential conflicts that can occur "after" the agent selects a move based on the default rules or learned case, then we can utilize the case windows shown in Figure 9. Since the agent decides where to move first, we can utilize rotational symmetry to map four potential cases into one.

# 5 Experimental Setup and Results

The initial configuration consists of the prey in the center of a 30 by 30 grid and the predators placed in random non–overlapping positions. All agents choose their actions simultaneously. The environment is accordingly updated and the agents choose their next action based on the updated environment state. If two agents try to move into the same location simultaneously, then randomly one is "bumped back" to its prior position and learns a case (this corresponds to the physical analogue of one of the contenders randomly being assigned the physical resource). One predator can push another predator (but not the prey) if the latter decided not to move. The prey does not move 10% of the time; effectively making the predators travel faster than the prey. The grid is toroidal in nature, and only orthogonal moves are allowed. All agents can sense the positions of all other agents. Furthermore, the predators do not possess any explicit communication skills: two predators cannot communicate to resolve conflicts or negotiate a capture strategy. The case window employed is that depicted in Figure 9. We have also identified two enhancements to break ties caused by the default rules employed in the MD metric: look ahead and least conflict (Haynes et al., 1996). Look ahead breaks ties in which two moves are equidistant via MD, the one which is potentially closer to the prey in two moves is selected. If look ahead also results in a tie, then the move which conflicts with the least number of possible moves by other predators is selected to break the tie.

Initially we were interested in the ability of predator behavioral rules to effectively capture the Still prey. We tested three behavioral strategies: MD – the basic MD algorithm, MD–EDR – the MD modified with the enhancements discussed in Section 3.3, and MD–CBL – which is MD–EDR utilizing a case base learned from training on 100 random simulations. The results of applying these strategies on 100 test cases are shown in Table 1. It should be noted that the MD algorithm is the best hand-coded behavioral strategy to–date in the predator–prey domain in the absence of explicit communication between the agents. While the enhancement of the behavioral rules does increase capture, the addition of learning via negative cases leads to capture in almost every simulation.

We also conducted a set of experiments in which the prey used the Linear algorithm as its behavioral rule. Again we tested the three predator behavioral strategies of MD, MD–EDR, and MD–CBL. The MD–CBL algorithm was trained on the Still prey. We trained on a Still prey because, as shown in Section 3, the Linear prey typically degrades to a Still prey. We have also presented the results of training the MD–CBL on the Linear prey (MD–CBL$^\star$). The results for the Linear prey are presented in Table 2.

| Algorithm | Captures | Ave. Number of Steps |
|---|---|---|
| MD | 3 | 19.00 |
| MD–EDR | 46 | 21.02 |
| MD–CBL | 97 | 23.23 |

Table 1: Number of captures (out of a 100 test cases) and average number of steps to capture for the Still prey.

| Algorithm | Captures | Ave. Number of Steps |
|---|---|---|
| MD | 2 | 26.00 |
| MD–EDR | 20 | 24.10 |
| MD–CBL | 54 | 27.89 |
| MD–CBL* | 66 | 26.45 |

Table 2: Number of captures (out of a 100 test cases) and average number of steps to capture for the Linear prey. MD–CBL* denotes a test of the MD–CBL when trained on a Linear prey.

In Table 3 we present the number of case learned during the training and the number of cases utilized during the testing trials. Note that the learning is evenly distributed across the agents. The increase in learning from Still to Linear can be explained by the interactions caused by the agents chasing the prey: the dynamic movement of the prey forces the predators into configurations not seen in the static case.

With both prey algorithms, the order of increasing effectiveness was MD, MD–EDR, and MD–CBL. Clearly the addition of MCBL to this multiagent system is instrumental in increasing the effectiveness of the behavioral rules. There is some room for improvement, as the results from the Linear prey indicate. A majority of the time spent in capturing the Linear prey is spent chasing it. Only after it is blocked do interesting conflict situations occur.

These experiments show the remarkable effectiveness of individual learning for contributing to group performance. Whereas the addition of communication and other sophisticated reasoning mechanisms definitely improve group performance in this and other complex domains, one can certainly make the point that individual learning can often compensate for inadequacies in prefabricated behavioral strategies.

| | Cases learned | | Utilization of cases | |
|---|---|---|---|---|
| Predator | Still | Linear | Still | Linear |
| 1 | 27 | 51 | 80 | 118 |
| 2 | 27 | 40 | 78 | 76 |
| 3 | 29 | 47 | 80 | 78 |
| 4 | 33 | 53 | 82 | 92 |

Table 3: Number of cases learned while trained on the Still and Linear preys. (Of 100 random trials, the Still prey was caught 100 times, and the Linear prey was caught 63.) Also, the number of times the learned cases were utilized in the test trials.

# 6 Discussion

Agent interactions can be represented as games (Rosenschein and Genesereth, 1985). In the extensive representation of a game, both minmax and alpha-beta search techniques can be used to determine which strategy to take at a decision point. Without knowing the behavioral strategy, or utility function, employed by other agents, a common model of others is to assume that they are rational and will take the move that maximizes their payoff.

A problem with this assumption is that agents can have different skills and proficiencies in those skills. An expert plays differently against another expert than an intermediate opponent. The expert opponent is expected to maximize its utility, making few mistakes, and the intermediate opponent is not expected to perceive all of its options, making more mistakes. Identifying another agent's model is crucial in effectively playing against it (Gmytrasiewicz and Durfee, 1995; Tambe and Gmytrasiewicz, 1996).

How does an agent determine the skill level of an unknown opponent, or how does it determine which model applies? The agent can assume that the opponent utilizes the same behavioral strategy as itself. As the opponent takes moves that are not "optimal" as predicted by that strategy, a model of the skill level can be learned. If the opponent wins, then it is more skilled. If it looses, then it is less skilled.

Clearly an agent should remember those situations in which the more skilled opponent took an action $a'$ different than the predicted one $a$. When in the same situation, the action $a'$ will be preferentially ranked above $a$. Consider that the opponent's reasoning or perception is superior; it may have reacted to something the agent either is not considering or is not able to detect or compute. The opponent might take a better action, but a lower bound is the action already seen by the agent. An agent should also remember both those actions and situations that the less skilled opponent utilized. While the alpha-beta heuristic is guaranteed to maximize its utility against the best action selected by an opponent, the agent can do better if its utility evaluation can detect that the opponent will not take the best action available. So, MCBL can help an agent to improve its performance both against more and less skilled opponents.

# 7 Conclusions

We have shown that case–based learning can be effectively applied to multi-agent systems. We have taken a difficult problem of group problem-solving from DAI literature and shown how MCBL can significantly improve on the performance of agent groups utilizing fixed behavioral rules. Our results, however, suggests interesting avenues for future research. Some of the critical aspects of MCBL in agent groups that we plan to further investigate are the following:

**Changing agent models :** A potential problem with this algorithm is that as Agent $A_i$ is learning to model the group behavior, the other agents in the group are likewise refining their models of the group interactions. This learning is dynamical, and the model Agent $A_i$ constructs of $A_j$ may be invalidated by the model of $A_j$ of $A_i$. In the environment state $E_l$, agent $A_i$ learns that $A_j$ will select action $a_y$. It might be the situation that when the environment is again at $E_l$, $A_j$ does not select $a_y$, but instead $a_z$. Is this an exception to the exception? Or is it just a re-learning of Agent $A_i$'s model of $A_j$?

If we return to our cat example presented earlier, we can see a situation in which group learning occurs when Daylight Savings Time takes effect. The time the alarm clock is set for is pushed back an hour. No one has informed Buster of this change in his environment. Adam's model of the cat is that Buster will try to wake him up "early" on weekday mornings. As predicted, Buster tries to wake up Adam. Adam refuses to get out of bed until the alarm sounds. After a week of not being able to wake Adam, Buster changes his routine by waiting until the new time before he tries to wake Adam.

**Diversity of experience :** In order for agents to significantly improve performance through learning it is essential that they be exposed to a wide array of situations. In some domains, agents can deliberately experiment to create novel interaction scenarios which will allow them to learn more about other agents in the group.

**Forgetting :** We believe that in order to further improve the performance of the presented system, it is essential to incorporate a structured mechanism for deleting or overwriting cases that are recognized to be ineffective. This is particularly important in multiagent systems because as multiple agents concurrently adapt their behavior, a particular agent's model of other agents is bound to get outdated. In effect, "the person I knew is not the same person any more!" To modify learned cases, we need to store more information about which agent caused us to learn the case, and what is our expectation of the behavior of that particular agent. We are currently working on developing a representation for the above without exploding the search space.

## Acknowledgments:

This work has been supported, in part, by the National Science Foundation under a Research Initiation Award IRI-9410180 and a CAREER award IRI-9702672.

# References


Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6(1):37–66.

Benda, M., Jagannathan, V., and Dodhiawala, R. (1986). On optimal cooperation of knowledge sources - an empirical investigation. Technical Report BCS–G2010–28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington.

Bond, A. H. and Gasser, L., editors (1988). *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann.

Cardie, C. (1993). Using decision trees to improve case–based learning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 25–32. Morgan Kaufmann Publishers, Inc.

Garland, A. and Alterman, R. (1995). Preparation of multi-agent knowledge for reuse. In Aha, D. W. and Ram, A., editors, *Working Notes for the AAAI Symposium on Adaptation of Knowldege for Reuse*, Cambridge, MA. AAAI.

Gasser, L., Rouquette, N., Hill, R. W., and Lieb, J. (1989). Representing and using organizational knowledge in DAI systems. In Gasser, L. and Huhns, M. N., editors, *Distributed Artificial Intelligence*, volume 2 of *Research Notes in Artificial Intelligence*, pages 55–78. Pitman.

Gmytrasiewicz, P. J. and Durfee, E. H. (1995). A rigorous, operational formalization of recursive modeling. In Lesser, V., editor, *Proceedings of the First International Conference on Multi–Agent Systems*, pages 125–132, San Francisco, CA. MIT Press.

Golding, A. R. and Rosenbloom, P. S. (1991). Improving rule–based systems through case–based reasoning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 22–27.

Halpern, J. and Moses, Y. (1990). Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587. A preliminary version appeared in *Proc. 3rd ACM Symposium on Principles of Distributed Computing*, 1984.

Hammond, K., Converse, T., and Marks, M. (1990). Towards a theory of agency. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 354–365, San Diego. Morgan Kaufmann.

Haynes, T., Lau, K., and Sen, S. (1996). Learning cases to compliment rules for conflict resolution in multiagent systems. In Sen, S., editor, *Working Notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, pages 51–56, Stanford University, CA.

Haynes, T. and Sen, S. (1996). Evolving behavioral strategies in predators and prey. In Weiß, G. and Sen, S., editors, *Adaptation and Learning in Multi–Agent Systems*, Lecture Notes in Artificial Intelligence, pages 113–126. Springer Verlag, Berlin.

Haynes, T., Sen, S., Schoenefeld, D., and Wainwright, R. (1995). Evolving multiagent coordination strategies with genetic programming. Technical Report UTULSA-MCS-95-04, The University of Tulsa.

Kolodner, J. L., editor (1988). *Proceedings of a Workshop on Case–Based Reasoning (DARPA).* Morgan Kaufmann.

Kolodner, J. L. (1993). *Case–Based Reasoning.* Morgan Kaufmann Publishers.

Korf, R. E. (1992). A simple solution to pursuit games. In *Working Papers of the 11th International Workshop on Distributed Artificial Intelligence*, pages 183–194.

Lesser, V. R. (1995). Multiagent systems: An emerging subdiscipline of AI. *ACM Computing Surveys*, 27(3):340–342.

Ordeshook, P. C. (1995). *Game Theory and Political Theory: An Introduction.* Cambridge University Press.

Prasad, M. V. N., Lesser, V. R., and Lander, S. (1995). Reasoning and retrieval in distributed case bases. *Journal of Visual Communication and Image Representation, Special Issue on Digital Libraries.* Also as UMASS CS Technical Report 95-27, 1995.

Rosenschein, J. S. and Genesereth, M. R. (1985). Deals among rational agents. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 91–99, Los Angeles, CA. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 227–234, Morgan Kaufmann, 1988.).

Sen, S. (1996). Adaptation, coevolution and learning in multiagent systems. Technical Report SS-96-01, AAAI Press, Stanford, CA.

Singh, M. P. (1990). The effect of agent control strategy on the performance of a DAI pursuit problem. In *Working Papers of the 10th International Workshop on Distributed Artificial Intelligence.*

Stephens, L. M. and Merx, M. B. (1989). Agent organization as an effector of DAI system performance. In *Working Papers of the 9th International Workshop on Distributed Artificial Intelligence.*

Stephens, L. M. and Merx, M. B. (1990). The effect of agent control strategy on the performance of a DAI pursuit problem. In *Proceedings of the 1990 Distributed AI Workshop.*

Sycara, K. (1987). Planning for negotiation: A case-based approach. In *DARPA Knowledge-Based Planning Workshop*, pages 11.1–11.10.

Tambe, M. and Gmytrasiewicz, P., editors (1996). *Working Notes of the AAAI-96 Workshop on Agent Modeling*, Portland, OR.

Weiß, G. and Sen, S., editors (1996). *Adaptation and Learning in Multi–Agent Systems.* Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin.