Proceedings of the First International Conference on Multi-Agent Systems, pages 125-132, 1995.

Recursive Agent Modeling Using Limited Rationality

José M. Vidal and **Edmund H. Durfee***
Artificial Intelligence Laboratory
University of Michigan
Ann Arbor, Michigan 48109-2122.
{jmvidal,durfee}@umich.edu

Abstract

We present an algorithm that an agent can use for determining which of its nested, recursive models of other agents are important to consider when choosing an action. Pruning away less important models allows an agent to take its "best" action in a timely manner, given its knowledge, computational capabilities, and time constraints. We describe a theoretical framework, based on *situations*, for talking about recursive agent models and the strategies and expected strategies associated with them. This framework allows us to rigorously define the *gain* of continuing deliberation versus taking action. The expected gain of computational actions is used to guide the pruning of the nested model structure. We have implemented our approach on a canonical multi-agent problem, the pursuit task, to illustrate how real-time, multi-agent decision-making can be based on a principled, combinatorial model. Test results show a marked decrease in deliberation time while maintaining a good performance level.

Keywords: Algorithms for multi-agent interaction in time-constrained systems; Conceptual and theoretical foundations of multi-agent systems.

Introduction

A rational agent that can be impacted by other agents will benefit if it can predict, with some degree of reliability, the likely actions of other agents. With such predictions, the rational agent can choose its own actions more judiciously, and can maximize its expected payoff in an interaction (or over a set of interactions). To get such predictions, an agent could rely on communication for explicit disclosure by other agents, or on superficial patterns of actions taken by others, or on deeper models of other agents' state. While each of these approaches has its merits, each also involves some cost or risk: effective communication requires careful decision-making about what to say, when to say it, and whether to trust what is heard (Durfee, Gmytrasiewicz, & Rosenschein 1994) (Gmytrasiewicz & Durfee 1993); relying on learned patterns of action (Sen & Durfee 1994) risks jumping to incorrect expectations when environmental variations occur; and using deeper models of other agents can be more accurate but extremely time-consuming (Gmytrasiewicz 1992).

In this paper, we concentrate on coordinated decision-making using deeper, nested models of agents. Because these models allow an agent to fully represent and use all of its available knowledge about itself and others, the quality of its decision-making can be high. As noted above, however, the costs of reasoning based on recursive models can be prohibitive, since the models grow exponentially in size as the number of levels increases. Our goal, therefore, has been to devise an algorithm which can prune portions of the recursive structure that can be safely ignored (that do not alter an agent's choice of action). We have done this by incorporating limited rationality capabilities into the Recursive Modeling Method (RMM). Our new algorithm allows an agent to decrease its deliberation time without trading off too much quality (expected payoff) in its choice of action. Our contribution, therefore, is to show how a principled approach to multi-agent reasoning such as RMM can be made practical, and to demonstrate the practical mechanisms in a canonical multi-agent decision-making problem.

We start by providing a short description of RMM and the Pursuit task, on which we tested our algorithm. Afterward we present *situations* as our way of representing RMM hierarchies, and define their expected *gain*. These concepts form the basis of our algorithm, shown in the Algorithm section, which expands only those nodes with the highest expected gain. The Implementation section presents the results of an implementation of our algorithm, from which we derive some conclusions and directions for further work, discussed in the Conclusion.

RMM and the Pursuit Task: The basic mod-

* Supported, in part, by NSF grant IRI-9158473.

eling primitives we use are based on the **Recursive Modeling Method** (RMM) (Gmytrasiewicz 1992; Gmytrasiewicz, Durfee, & Wehe 1991; Durfee, Gmytrasiewicz, & Rosenschein 1994; Durfee, Lee, & Gmytrasiewicz 1993). RMM provides a theoretical framework for representing and using the knowledge that an agent has about its expected payoffs and those of others. To use RMM, an agent is expected to have a payoff matrix where each entry represents the payoffs the agent expects to get given the combination of actions chosen by all the agents. Typically, each dimension of the matrix corresponds to one agent, and the entries along it to all the actions that agent can take. The agent can (but need not) recursively model others as similarly having payoff matrices, and them modeling others the same way, and so on.... The recursive modeling only ends when the agent has no deeper knowledge. At this point, a Zero Knowledge (ZK) strategy can attributed to the particular agent in question, which basically says that, since there is no way of knowing whether any of its actions are more likely than others, all of the actions are equally probable. If an agent does have reason to believe some actions are more likely than others, this different probability distribution can be used. RMM provides a method for propagating strategies from the leaf nodes to the root. The strategy derived at the root node is what the agent performing the reasoning should do.

For example, a simplified payoff matrix for the pursuit task is shown in Figure 3. An RMM hierarchy would have such a matrix for one of the agents (say P1) at the root. If that agent knows something about how the other agents represent the situation (their payoffs), then it will model them in order to predict their strategies, so it can generate its own strategy. If it knows something about what the others know about how agents model the situation, another nested layer can be constructed. When it runs out of knowledge, it can adopt the ZK strategy at those leaves, yielding a hierarchy like that shown abstractly in Figure 1.

To evaluate our algorithm, we have automated the construction of payoff matrices for the **pursuit task**, in which four predators try to surround a prey which moves randomly. The agents move simultaneously in a two-dimensional square grid. They cannot occupy the same square, and the game ends when all four predators have surrounded the prey on four sides ("capture"), when the prey is pushed against a wall so that it cannot make any moves ("surround") or when time runs out ("escape"). The pursuit task has been investigated in Distributed AI (DAI) and many different methods have been devised for solving it (Korf 1992) (Stephens & Merx 1990) (Levy & Rosenschein 1992). These either impose specific roles on the predators, spend much time computing, or fail because of lack of coordination. RMM provides another method for providing this coordination but, to avoid the "time-consuming" aspects of game-theoretic approaches, we must devise a theory to support selective expansion of the hierarchy.

Theory

The basic unit in our syntactical framework for recursive agent modeling is the *situation*. At any point in time, an agent is in a situation, and all the other agents present are in their respective situations. An agent's situation contains not only the situation the agent thinks it is in but also the situations the agent thinks the others are in (these situations might then refer to others and so on ...). In this work, we have adopted RMM's assumption that common knowledge[1] cannot arise in practice, and so it is impossible for a situation to refer back to itself either directly or transitively.

A situation has both a physical and a mental component. The physical component refers to the physical state of the world and the mental component to the mental state of the agent, i.e. what the agent is thinking about itself and about the other agents around it. Intuitively, a *situation* reflects the state of the world from some agent's point of view by including what the agent perceives to be the physical state of the world and what the agent is thinking. A situation evaluates to a *strategy*, which is a prescription for what action(s) the agent should take. A strategy has a probability associated with each action the agent can take, and the sum of these probabilities must always equal 1.

Let S the the set of situations an agent might encounter, and A the set of all other relevant agents. A particular situation s is recursively defined as:

$$s = (M, f, W, \{\{(p, r, a) | \sum p = 1,\\ r \in (S \cup \text{ ZK})\} | a \in A\}) \in S$$

The matrix M has the payoff the agent, in situation s, expects to get for each combination of actions that all the agents might take. The matrix M can either be stored in memory as is, or it can be generated from a function (which is stored in the agent's memory) that takes as inputs any relevant aspects of the physical world, and previous history. The relevant aspects of the physical world are stored in W, the *physical* component of the situation. The rest of s constitutes the *mental* component of the situation.

[1]Common knowledge about a fact x means that everybody knows that everybody knows... about x.

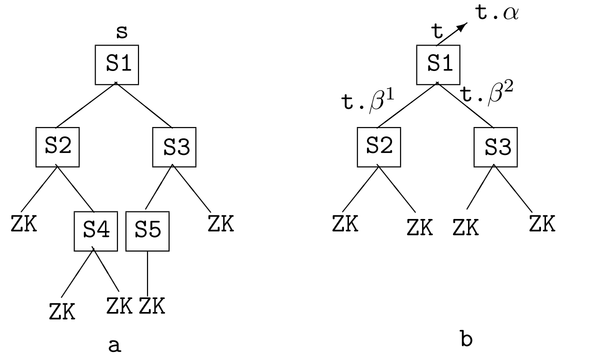

Figure 1: These are graphical representations of (a) a situation s (the agent in s expects to get the payoffs given by S1 and believes that its opponents will get S2 and S3, respectively), and (b) a partially expanded situation t that corresponds to s. ZK stands for the zero knowledge strategy.

The probabilistic distribution function $f(x)$ gives the probability that the strategy x is the one that the situation s will evaluate to. It need not be a perfect probabilistic distribution; it is merely used as a useful approximation. We use $f(x)$ to calculate the strategy that the agent in the situation is expected to choose, using standard expected value formulas from probability theory. The values of this function are usually calculated from previous experience. A situation s also includes the set of situations which the agent in s believes the other agents are in. Each agent a is believed to be in r, with probability p. The value of r can either be a situation ($r \in S$) or, if the modeling agent has no more knowledge, it can be the Zero Knowledge strategy ($r = ZK$).

Notation and Formalisms: Our implementation of limited rationality and our notation closely parallels Russell and Wefald's work (Russell & Wefald 1991), although with some major differences as will be pointed out later. We define a *partially expanded situation* as a subset of a situation where only some of the nodes have been expanded. That is, if t is a partially expanded situation of s, then it must contain the root node of s, and all of t's nodes must be directly or indirectly connected to this root by other nodes in t. The situation t might have fewer nodes than s. At those places where the tree was pruned, t will insert the zero knowledge strategy, as shown in Figure 1.

We also define our basic computational action to correspond to the expansion of a leaf node l in a partially expanded situation t, plus the propagation of the new strategy up the tree. The expansion of leaf node l corresponds to the creation of the matrix M, and its placement in l along with $f(x)$, W, and pointers to the children. The children are set to the ZK strategy because we have already spent some time generating M and we do not wish to spend more time calculating a child's matrix, which would be a whole other "step" or computational action. We do not even want to spend time calculating each child's expected strategy since we wish to keep the computation fairly small. The strategy that results from propagating the ZK strategies past M is then propagated up the tree of t, so as to generate the new strategy that t evaluates to. The whole process, therefore, is composed of two stages. First we Expand l and then we Propagate the new strategy up t. We will call this process Propagate_Expand(t, l), or $PE(t, l)$ for short. The procedure $PE(t, l)$ is our basic computational action.

We define the time cost for doing $PE(t, l)$ as the time it takes to generate the new matrix M, plus the time to propagate a solution up the tree, plus the time costs incurred so far. This time cost is:

$$\begin{aligned} TC(t,l) = {} & (c \cdot \text{size of matrix } M \text{ in } l) + \\ & (d \cdot \text{propagate time}) + g(\text{time transpired so far}) \end{aligned}$$

where c and d are constants of proportionality. The function $g(x)$ gives us a measure of the "urgency" of the task. That is, if there is a deadline at time T before which the agent must act, then the function $g(x)$ should approximate infinity as x approximates T. We now define some more notation to handle all the strategies associated with a particular partially expanded situation.

t: Partially expanded situation, all of the values below are associated with it. Corresponds to the fully expanded situation s.

$t.\alpha$: The strategy the agent currently favors in the partial situation t.

$t.\beta^{1\cdots n}$: The strategies the agent in t believes the n opponents will favor.

t.leaves: These are the leaf nodes of the partially expanded situation t that can still be expanded (i.e. those that are not also leaf nodes in s).

$t.\hat{\alpha}_l$: The strategy an agent expects to play given that, instead of actually expanding leaf l (where $l \in t$.leaves) he simply uses its expected value, which he got from $f(x)$, and propagates this strategy up the tree.

$t.\hat{\beta}_l^{1\cdots n}$: The strategies an agent expects the other agents to play given that, instead of actually expanding leaf l, he simply uses its expected value.

$P(t.\alpha, t.\beta^{1\cdots n})$: This is the payoff an agent gets in his current situation. To calculate it, use the root matrix and let the agent's strategy be $t.\alpha$ while the opponents' strategies are $t.\beta^{1\cdots n}$.

Given all this notation, we can finally calculate the utility of a partially expanded situation t. Let t' be situation t after expanding some leaf node l in it, i.e. $t' \leftarrow \text{PE}(t, l)$. The utility is the payoff the agent will get, given the current evaluation of t'. The expected utility of t' is the utility that an agent in t *expects* to get if he expands l so as to then be in t'.

$$U(t') = P(t'.\alpha, t'.\beta^{1\cdots n})$$

$$E(U(t')) = P(t.\hat{\alpha}_l, t.\hat{\beta}_l^{1\cdots n})$$

Note that the utility of a situation depends on the shape of the tree below it since both $t.\alpha$ and $t.\beta$ are calculated by using the whole tree. There are no utility values associated with the nodes by themselves.[2] Unfortunately, therefore, we cannot use the utilities of situations in the traditional way, which is to expand leaves so as to maximize expected utility. Such traditional reasoning in a recursive modeling system would mean that the system would not introspect (examine its recursive models) more deeply if it thought that doing so might lead it to discover that it in fact will not receive as high a payoff as it thought it would before doing the introspection. Ignoring more deeply nested models because they reduce estimated payoff is a bad strategy, because ignoring them does not change the fact that other agents might take the actions that lead to lower payoffs. Thus, we need a different notion of what an agent can "gain" by examining more deeply nested knowledge.

The point of the agent's reasoning is to decide what its best course of action is. Thus, expanding a situation is only useful if, by doing so, the agent chooses a different action than it would have before doing the expansion. And the degree to which it is better off with the different action rather than the original one (given what it now knows) represents the *gain* due to the expansion. More specifically, the Gain of situation t', $G(t')$, is the amount of payoff an agent gains if it previously was in situation t and, after expanding leaf $l \in t.\text{leaves}$, is now in situation t'. Since $t' \leftarrow \text{PE}(t, l)$, we could also view this as $G(PE(t, l))$. The Expected Gain, $E(G(t'))$, is similarly $E(G(PE(t, l)))$, which is approximated by G(Propagate(E(Expand(t, l)))) in our algorithm. We can justify this approximation because Propagate() simply propagates the strategy at the leaf up to the root, and it usually maps similar strategies at the leaves to similar strategies at the root. So, if the expected strategy at the leaf, as returned by E(Expand(t, l)), is close to the real one, then the strategy that gets propagated up to the root will also be close to the real one.[3] The formal definitions of gain and expected gain are:

$$\begin{aligned} G(t') &= P(t'.\alpha, t'.\beta^{1\cdots n}) - P(t.\alpha, t'.\beta^{1\cdots n}) \\ &\quad - TC(t, l) \\ E(G(t')) &= P(t.\hat{\alpha}_l, t.\hat{\beta}_l^{1\cdots n}) - P(t.\alpha, t.\hat{\beta}_l^{1\cdots n}) \\ &\quad - TC(t, l) \end{aligned}$$

Notice that $E(G(t'))$ reaches a minimum when $t.\alpha = t.\hat{\alpha}_l$, since $t.\hat{\alpha}_l$ is chosen so as to maximize the payoff given $t.\hat{\beta}_l^{1\cdots n}$. In this case, $G(t') = -TC(t, l)$. This means that the gain will be negative if the possible expansion l does not lead to a different and better strategy.

Algorithm

Our algorithm starts with the partially expanded root situation, which consists only of the payoff matrix for the agent. ZK strategies are used for the situations of other agents, forming the initial leaves. The algorithm proceeds by expanding, at each step, the leaf node that has the highest expected gain, as long as this gain is greater than K. For testing purposes, we set $K = 0$, but setting K to some small negative number would allow for the expansion of leaves that do not show an immediate gain, but might lead to some gains later on. The function `Expected_Strategy(l)` takes as input a leaf node situation l and determines the strategy that its expansion, to some number of levels, is *expected* to return. It does this by calculating the expected value of the corresponding $f(x)$. The strategy is propagated up by the function `Propagate_Strategy(t,l)`, which returns the expected $t.\hat{\alpha}_l$ and $t.\hat{\beta}_l^{1\cdots n}$. These are then used to calculate the expected gain for leaf l. The leaf with the maximum expected gain, if it is greater than K, is expanded, a new strategy propagated, and the whole process is repeated. Otherwise, we stop expanding and return the current $t.\alpha$.

The algorithm, shown in Figure 2, is started with a call to `Expand_Situation(t)`. Before the call, t must be set to the root situation, and $t.\alpha$ to the strategy that t evaluates to. The strategy returned by `Expand_Situation` could then be stored away so that it can be used, at a later time, by `Expected_Strategy`. The decision to do this would depend on the particular implementation. One might only wish to remember those strategies that result from expanding a certain number of nodes or more, that is, the "better" strategies. In other cases, it could be desirable to remember all previous strategies that correspond to situations that the agent has never experienced before, on

[2]This is in contrast to Russell's work, where each node has a utility associated with it.

[3] Test results show that this approximation is, indeed, good enough.

Expand_Situation(t)
  max_gain $\leftarrow -\infty$; max_sit $\leftarrow$nil; $t' \leftarrow t$
  For $l \in t$.leaves /* l is a structure */
    t'.leaf $\leftarrow l$
    $l.\alpha \leftarrow$ Expected_Strategy(l)
    $t.\hat{\alpha}_l$, $t.\hat{\beta}_l^{1\cdots n} \leftarrow$ Propagate_Strategy(t, l)
    t'.gain $\leftarrow P(t.\hat{\alpha}_l, t.\hat{\beta}_l^{1\cdots n}) - P(t.\alpha, t.\hat{\beta}_l^{1\cdots n}) - TC(t, l)$
    If t'.gain $>$ max_gain Then
      max_gain $\leftarrow t'$.gain
      max_sit $\leftarrow t'$
  If max_gain $> K$ Then
    $t \leftarrow$ Propagate_Expand(max_sit)
    Return(Expand_Situation(t))
  Return($t.\alpha$)

Propagate_Expand(t) /* l is included in t.leaf */
  leaf $\leftarrow t$.leaf
  leaf.$\alpha \leftarrow$ Expand(leaf)
  $t.\alpha$, $t.\beta^{1\cdots n} \leftarrow$ Propagate_Strategy(t, leaf)
  t.leaves $\leftarrow$ New_Leaves(t)
  Return(t)

Propagate_Strategy(t, l)
  Propagates $l.\alpha$ all the way up the tree and returns the best strategy for the root situation, and the strategies for the others.

Expand(l)
  Returns the strategy we find by first calculating the matrix that corresponds to l, and then plugging the zero-knowledge solution for all the children of l.

New_Leaves(t)
  Returns only those leaves of t that can be expanded into full situations.

Figure 2: Limited Rationality RMM Algorithm

the theory that it is better to have some information rather than none. This is the classic time versus memory tradeoff.

Time Analysis: For the algorithm to work correctly, the time spent in metalevel thinking must be small compared to the time it takes to do an actual node expansion and propagation of a new solution (i.e. a $PE(t, l)$). Otherwise, the agent is probably better off choosing nodes to expand at random without spending time considering which one might be better.

A node expansion involves the creation of a new matrix M. If we assume that there are k agents and each one has n possible actions, then the number of elements in the matrix will be n^k. Each one of these elements represents the utility of the situation that results after all the agents take their actions. In order to calculate this utility, it is necessary to simulate the new state that results from all the agents taking their respective actions (i.e. as dictated by the element's position in the matrix) and then calculate the utility, to some particular agent, of this new situation (see Figure 3). The calculation of this utility can be an arbitrarily complex function of the state, and must be performed for each of the n^k elements in the matrix.

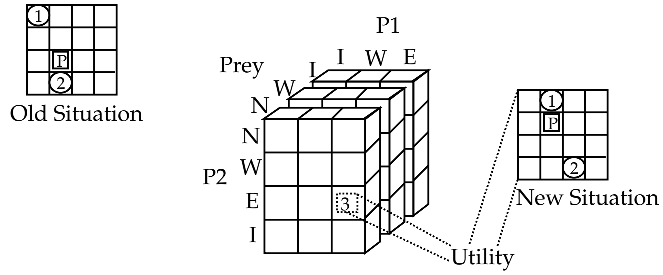

Figure 3: Given the old situation, each one of the elements of the matrix corresponds to the payoff in one of the many possible next situations. Notice that, sometimes, the moves of the predators might interfere with each other. The predator calculating the matrix has to resolve these conflicts when determining the new situation, before calculating its utility.

The next step is the propagation of the expected strategy. If we replace one of the ZK leaf strategies by some other strategy, then the time to propagate this change all the way to the root of the tree depends both on the distance between the root and leaf, and the time to propagate a strategy past a node. Because of the way strategies are calculated, it is almost certain that any strategy that is the result of evaluating a node will be a pure strategy[4]. If all the children of a node are pure strategies, then the strategy the node evaluates to can be found by calculating the maximum of n numbers. In other words, the k-dimensional matrix collapses into a one-dimensional vector. If c of the children have mixed strategies, we would need to add n^{c+1} numbers and find the maximum partial sum. In the worst case, $c = k - 1$, so we need to add n^k numbers.

We can conclude that propagation of a solution will require a good number of additions and max functions, and these must be performed for each leaf we wish to consider. However, these operations are very simple since, in most instances, the propagation past a node will consist of one *max* operation. This time is small, especially when compared to the time needed for the simulation and utility calculation of n^k different possible situations. A more detailed analysis can only be performed on an application-specific basis, and it would have to take into account actual computational

[4] In a pure strategy one action is chosen with probability 1, the rest with probability 0.

Method	Avg. Nodes Expanded	Avg. Time to Capture/Surround
BFS 1 Level	1	∞/Never Captured
BFS 2 Levels	4	23.4
BFS 3 Levels	13	22.4
BFS 4 Levels	40	17.3
LR RMM	4.2	18.5
Greedy	NA	41.97

Table 1: Results of implementation of the Pursuit problem using simple BFS to various levels (i.e. regular RMM), using our Limited Rationality Recursive Modeling algorithm, and using a simple greedy algorithm, in which each predator simply tries to minimize its distance to the prey.

times[5].

Implementation Strategies: A strategy we adopt to simplify the implementation is to classify agents according to *types*. Associated with each type should be a function that takes as input the agent's physical situation and generates the full mental situation for that agent. This technique uses little memory and is an intuitive way of programming the agents. It assumes, however, that the mental situation can be derived solely from the physical. This was true in our experiments but need not always be so, such as when the mental situation might depend on individual agent biases or past experiences. The function `Expected_Strategy`, which calculates the expected value of $f(x)$, can then be implemented by a hash table for the agent type. The keys to the table would be *similar* physical situations and the values would be the last N strategies played in those situations. Grouping similar situations is needed because the number of raw physical situations can be very big. The definition of what situations are similar is heuristic and determined by the programmer.

Implementation of Pursuit Task

The original algorithm has been implemented in a simulation of the pursuit task, using the MICE system (Montgomery & Durfee 1990). The experiments showed some promising results. The overall performance of the agents was maintained while the total number of expanded nodes was reduced by an order of magnitude.

We designed a function that takes as input a physical situation and a predator in that situation and returns the payoff matrix that the predator expects to get given each of the possible moves they all take. This function is used by all the predators (that is, we assumed all predators were of the same type). A predator's payoff is the sum of two values. The first value is the change in the distance from the predator to the prey between the current situation and the new situation created after all agents have made their moves. The second value is 5^k, where k is the number of quadrants around the prey that have a predator in them in the new situation. The quadrants are defined by drawing two diagonal lines across the prey (Gasser *et al.* 1989; Levy & Rosenschein 1992).

[5]Some basic calculations, along these lines, are shown in the results section.

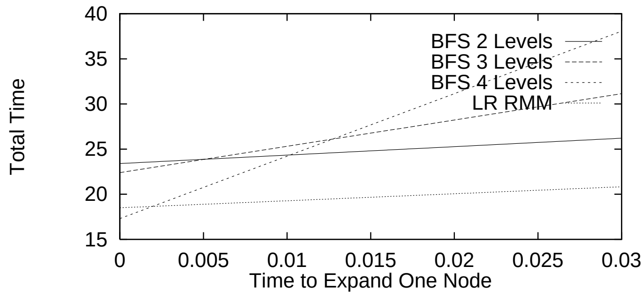

Figure 4: Plot of the total time that would, on average, be spent by the agents before capturing the prey, for each method. The x axis represents the amount of time it takes to expand a node as a percentage (more or less) of the time it takes the agent to take action.

Since the matrices take into account all possible combinations of moves (5 for each predator and 4 for the prey), they are five-dimensional with a total of 2500 payoff entries. With matrices this big, even in a simple problem, it is easy to see why we wish to minimize the number of matrices we need to generate. We defined the physical situation which the predator is in as its relative position to the prey and to the other predators. These situations were then generalized such that distances greater than two are indistinguishable, except for quadrant information. This generalization formula served to shrink the number of buckets or keys in our hash table to a manageable number (from $4.1 \cdot 10^{11}$ to around 3000). Notice also that the number of buckets remains constant no matter how big the grid is.

Results: We first ran tests without our algorithm and using simple Breadth First Search of the situation hierarchy. When using BFS to one level, the predators could not predict the actions of the others and so, they never got next to the prey. As we increased the number of levels, the predators could better predict what the others would do, which made their actions more coordinated and they managed to capture the prey. The goal for our algorithm was to keep the same good results while expanding as few nodes as possible. For comparison purposes, we also tested a very simple greedy algorithm where each predator simply tries to minimize its distance to the prey, irrespective of where the other predators are.

We then ran our Limited Rationality RMM algorithm on the same setup. We had previously compiled a hash table which contained several entries (usually around 5, but no more than 10) for almost all situations. This was done by generating random situations and running RMM to 3 or 4 levels on them to find out the strategy. The results, as seen in Table 1, show that our algorithm managed to maintain the performance of a BFS to 4 levels while only expanding little more than four nodes on the average. All the results are the averages of 20 or more runs.

Another way to view these results is by plotting the total time it takes the agents, on average, to surround the prey as a function of the time it takes to expand a node. If we assume that the time for an agent to make a move is 1 unit and we let the time to expand a node be x, the average number of turns before surrounding the prey be t_s, and the average number of nodes N, then the total time T the agents spends before surrounding the prey is approximately $T = t_s \cdot (N \cdot x + 1)$, as shown in Figure 4. LR RMM is expected to perform better than all others except when the time to expand a node is much smaller (.002 times smaller) than the time to perform an action.

Conclusions

We have presented an algorithm that provides an effective way of pruning a recursive model so that only a few critical nodes need to be expanded in order to get a quality solution. The algorithm does the double task of delivering an answer within a set of payoff/time cost constraints, and pruning unnecessary knowledge from the recursive model of a situation. We also gave a formal framework for talking about strategies, expected strategies, and expected gains from expanding a recursive model.

The experimental results proved that this algorithm works, in the example domain. We expect that the algorithm will provide similar results in other problem domains, provided that they show some correspondence between the physical situation and the strategy played by the agent. The test runs taught us two main lessons. Firstly that there is a lot of useless information in these recursive models (i.e. information that does not influence the agent's decision). Secondly, they showed us how much memory is actually needed for handling the computation of expected strategies, even without considering the agent's mental state. This seems to suggest that more complete implementations will require a very smart similarity function for detecting which situations are similar to which others if we want to maintain the solution quality. We are currently investigating which techniques (e.g. reinforcement learning, neural networks, case-based reasoning) are best suited for this task.

Given what we have learned from our tests, there is a definite set of challenges that lie ahead in terms of improving and expanding the algorithm. First of all, we have to determine how we can include the mental state of an agent into the hashing function, which now contains only the generalized physical situation. This enhancement, along with the addition of more hash tables for different agent types, would be very useful for moving the algorithm to more complex domains. Also, we are studying possible methods that an agent can use for determining which strategy to play when it has little or no knowledge of it's opponents' behavior, except for whatever observations it has managed to record.

References

[Durfee, Gmytrasiewicz, & Rosenschein 1994] Durfee, E. H.; Gmytrasiewicz, P. J.; and Rosenschein, J. S. 1994. The utility of embedded communications and the emergence of protocols. In *Proceedings of the 13th International Distributed Artificial Intelligence Workshop.*

[Durfee, Lee, & Gmytrasiewicz 1993] Durfee, E. H.; Lee, J.; and Gmytrasiewicz, P. J. 1993. Overeager reciprocal rationality and mixed strategy equilibria. In *Proceedings of the Eleventh National Conference on Artificial Intelligence.*

[Gasser *et al.* 1989] Gasser, L.; Rouquetter, N. F.; Hill, R. W.; and Lieb, J. 1989. Representing and using organizational knowledge in distributed AI systems. In Gasser, L., and Huhns, M. N., eds., *Distributed Artificial Intelligence*, volume 2. Morgan Kauffman Publishers. 55–78.

[Gmytrasiewicz & Durfee 1993] Gmytrasiewicz, P. J., and Durfee, E. H. 1993. Toward a theory of honesty and trust among communicating autonomous agents. *Group Decision and Negotiation* 2:237–258.

[Gmytrasiewicz, Durfee, & Wehe 1991] Gmytrasiewicz, P. J.; Durfee, E. H.; and Wehe, D. K. 1991. A decision-theoretic approach to coordinating multiagent interactions. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence.*

[Gmytrasiewicz 1992] Gmytrasiewicz, P. J. 1992. *A Decision-Theoretic Model of Coordination and Communication in Autonomous Systems (Reasoning Systems).* Ph.D. Dissertation, University of Michigan.

[Korf 1992] Korf, R. E. 1992. A simple solution to pursuit games. In *Proceedings of the 11th International Distributed Artificial Intelligence Workshop.*

[Levy & Rosenschein 1992] Levy, R., and Rosenschein, J. S. 1992. A game theoretic approach to the pursuit problem. In *Proceedings of the 11th International Distributed Artificial Intelligence Workshop.*

[Montgomery & Durfee 1990] Montgomery, T. A., and Durfee, E. H. 1990. Using MICE to study intelligent dynamic coordination. In *Proceedings of IEEE Conference on Tools for AI.*

[Russell & Wefald 1991] Russell, S., and Wefald, E. 1991. *Do The Right Thing.* Cambridge, Massachusetts: The MIT Press.

[Sen & Durfee 1994] Sen, S., and Durfee, E. H. 1994. Adaptive surrogate agents. In *Proceedings of the 13th International Distributed Artificial Intelligence Workshop.*

[Stephens & Merx 1990] 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 9th International Distributed Artificial Intelligence Workshop.*

[Vidal & Durfee 1994] Vidal, J. M., and Durfee, E. H. 1994. Agent modeling methods using limited rationality. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1495.