Using Recursive Agent Models Effectively

José M. Vidal and Edmund H. Durfee *

Artificial Intelligence Laboratory, University of Michigan
1101 Beal Avenue, Ann Arbor, Michigan 48109-2110, USA

{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.

1 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 [2] [9]; relying on learned patterns of action [15] risks jumping to incorrect expectations when environmental variations occur; and using deeper models of other agents can be more accurate but extremely time-consuming [7].

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

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

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 more practical[2], 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.

1.1 The Recursive Modeling Method

The basic modeling primitives we use are based on the Recursive Modeling Method (RMM) (see [8] in this Volume) [6] [2] [3]. 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.

An example payoff matrix along with its RMM hierarchy, is shown in Figure 1. This RMM hierarchy represents a situation from agent A1's point of view and, therefore, has her payoff matrix at the root. Assuming that agent A1 knows something about how A2 represents the situation, then A1 will model A2 in order to predict his strategies, which will allow A1 to generate better strategies for herself. These models take the form of payoff matrices and are placed below the root node. The probability associated with each branch captures the uncertainty A1 has about A2. If A1 similarly knows something about what A2 might know about how A1 represents the situation, this could be further captured as more deeply nested payoff matrices, as implied in the left branch of the figure. If A1 knows something about what A2 expects A1 to do in the situation but not how A2 represents A1's thinking, then A1 could associate with A2 a sub-intentional model of A1

[2] There are, of course, other approximation techniques that increase practicality which rely on various abstractions of the problem [1].

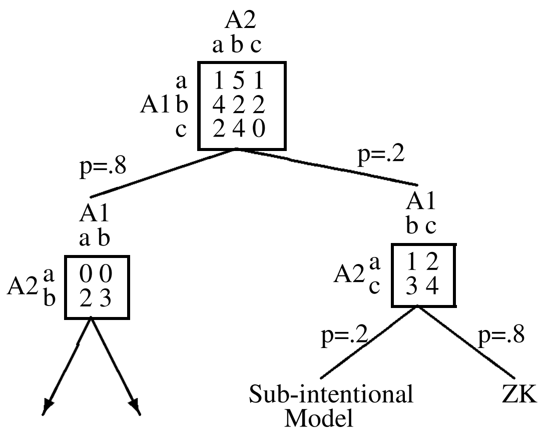

Fig. 1. This is an example RMM hierarchy for two agents A1 and A2. The leaves of the tree will either be Zero Knowledge (ZK) strategy or a sub-intentional model. Note that we consider the ZK strategy and the NO-INFO model in [8], as equivalent.

that summarizes A1's likely actions. Finally, If A1 believes that A2 has no knowledge of A1, this can be captures in a ZK strategy, as shown by the rightmost branch.

1.2 Limited Rationality

Our algorithm is based on ideas borrowed from limited rationality as defined in Russell and Wefald's book [14] which, in turn, was inspired by the work done by Simon [16]. Russell and Wefald define the agent's thinking as the execution of a series of *computational actions*, and use metalevel thinking techniques to determine which computational action, if any, the agent should execute next. We have taken this approach and applied it to the problem of using recursive agent models. The decision made by the metalevel thinking is which model, if any, to expand next. The change of domain required us to define the concept of *gain* since the traditional *utility* concept was not sufficient for the implementation of a successful algorithm.

1.3 The Pursuit Task

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 [10] [17] [11]. 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.

2 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[3] cannot arise in practice. Common knowledge would be represented in our framework by having situations refer back to themselves either directly or transitively. Conversely, cycles or loops of reference between situations imply the presence of common knowledge. They are, therefore, disallowed since common knowledge can not be achieved in many common situations of interest to us, such as in asynchronous message-passing system [4]. Allowing agents to jump to assumptions about common knowledge can simplify coordination reasoning, since agents can use fixed-point equilibrium notions and axioms based on mutual beliefs and plans. While leaping to such assumptions can, at times, be a viable method for keeping coordination practical [1], it also introduces risks that we wish to avoid.

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 be 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.

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

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 for guiding our search through the recursive models. That is, our algorithm will employ it as a heuristic for determining which recursive models to expand. 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. In practice, some simplifying assumptions can be made about how to calculate (i.e. learn) this value, as we shall explain in Section 3.2. We also note that the knowledge encompassed by $f(x)$ is not the real knowledge, which is contained in the matrices, but is the search knowledge used only for guiding the search through the recursive models. There is no guarantee that the predictions produced by $f(x)$ will be correct. The models, on the other hand, are always considered to be correct.

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 $(r = ZK)$.

2.1 Notation and Formalisms

Our implementation of limited rationality and our notation closely parallels Russell and Wefald's work [14], 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 2.

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:

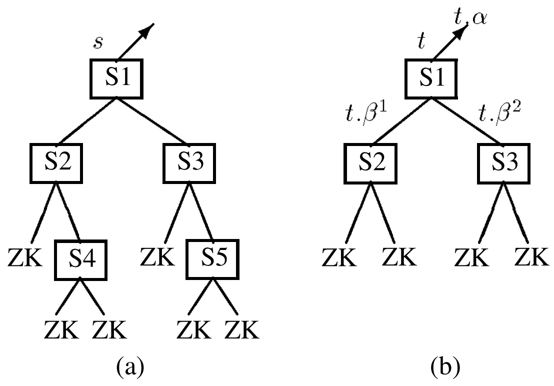

Fig. 2. These are graphical representations of (a) a situation s (the agent in s expects to get the payoffs given by S1 and believes that its opponents will get payoffs based on S2 and S3, respectively), and (b) a partially expanded situation t that corresponds to s.

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

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. $t' \leftarrow \mathrm{PE}(t, l)$. 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.[4]

Unfortunately, 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. This reasoning is better understood with the help of an example. We contrast our problem with the case of using minimax for playing a board game, as studied in [14]. There, if a player realizes that taking move A will lead to a set of possible outcomes, none of which have higher utility than he gets when taking move B, then the player should not consider move A anymore. Within our framework, however, this result does not apply. Imagine a predator who is very close to surrounding the prey. It might initially decide that the others will be taking moves so that, if he moves North, the prey will be captured. This state has a very high utility and is very desirable to the predator. However, the predator also realizes that if he thinks some more about the predator to his left, he will probably change his mind about what the predator to the left will do, so he will then prefer to go West. The resulting situation will not be a capture and, therefore, will have smaller utility. The question then is: should the predator stop thinking about the other one to its left, just because doing so will lead him to think his prospects for the future are worse? We do not believe so. The predator should continue to think more deeply about others as long as doing so makes him change his mind about which move to take, even when it leads him to the realization that things are worse than he initially believed[5]. Thus, we need a different notion of what an agent can "gain" by examining more deeply nested knowledge.

The driving force for 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

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

[5] We can anthropomorphize this advice to say that, it does not help to pretend things are better than they probably are; we must determine exactly where we stand and take the optimal action based on this assessment.

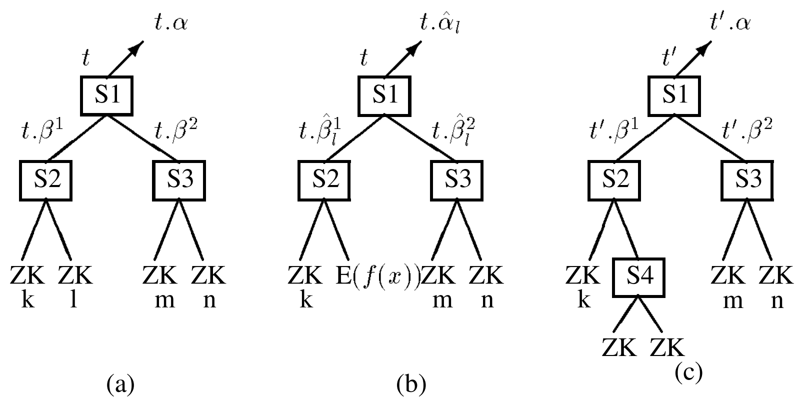

Fig. 3. Here we can see how the algorithm proceeds when expanding a situation. We start with t in (a). For each leaf we calculate the expected gain. (b) shows how this is done for leaf l. If we do decide to expand leaf l we will end up with a new t', as seen in (c).

t', $G(t')$, is the amount of payoff an agent gains if it previously was in situation t and, after expanding leaf $l \in t$.leaves, is now in situation t'. Since $t' \leftarrow$ 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.[6] The formal definitions of gain and expected gain are:

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

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.

3 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

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

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 $t.\alpha$.

```
Expand_Situation(t)
  max_gain ← −∞; max_sit ← nil; t' ← t
  For l ∈ t.leaves /* l is a structure */
    t'.leaf ← l
    l.α ← Expected_Strategy(l)
    t.α̂_l, t.β̂_l^{1···n} ← Propagate_Strategy(t, l)
    t'.gain ← P(t.α̂_l, t.β̂_l^{1···n}) − P(t.α, t.β̂_l^{1···n}) − TC(t, l)
    If t'.gain > max_gain Then
      max_gain ← t'.gain
      max_sit ← t'
  If max_gain > K Then
    t ← Propagate_Expand(max_sit)
    Return(Expand_Situation(t))
  Return(t.α)

Propagate_Expand(t) /* l is included in t.leaf */
  leaf ← t.leaf
  leaf.α ← Expand(leaf)
  t.α, t.β^{1···n} ← Propagate_Strategy(t, leaf)
  t.leaves ← New_Leaves(t)
  Return(t)

Propagate_Strategy(t, l)
  Propagates l.α 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.
```

Fig. 4. Limited Rationality RMM (LR-RMM) Algorithm

The algorithm, shown in Figure 4, is started by calling `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 the theory that it is better to have some information rather than none. This is the classic time versus memory tradeoff.

3.1 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 5). 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.

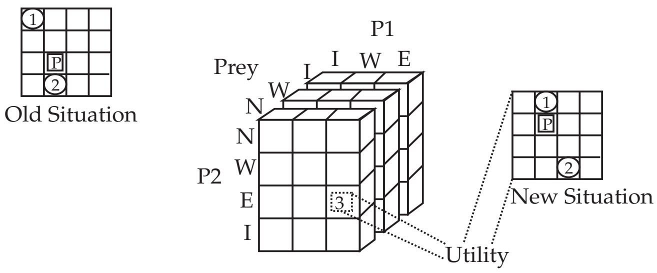

Fig. 5. A very simplified version of the Pursuit Problem with only 3 agents and in a 4 by 4 grid. Given the old situation, each one of the elements of the matrix corresponds to the payoff in one of the many possible next situations. There is one for each combination of moves the agents might take (e.g. North, East, Idle). 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[7]. 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 times[8].

3.2 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. In our case we chose to represent situations from the agent's point of view, without any references to the cardinal points or any other fixed landmark. This meant that situations were automatically equivalent under rotations of 90 degrees. Also, if an agent was more than a Manhattan distance of two from the prey, we only considered the quadrant in which it was located (with respect to the agent observing it), not the actual distance. The intuition was that agents that are far from the prey do not influence the other agents' move decision.

4 Implementation of Pursuit Task

The original algorithm has been implemented in a simulation of the pursuit task, using the MICE system [12]. The experiments showed some promising results. The overall

[7] In a pure strategy one action is chosen with probability 1, the rest with probability 0.
[8] Some basic calculations, along these lines, are shown in Section 4.1.

performance of the agents was maintained while the total number of expanded nodes was reduced by an order of magnitude.

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.

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 [5, 11].

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.

4.1 Results

Our tests were run on an 20 by 20 grid with four predators randomly placed and one prey that always started in the middle of the grid. We started with tests where the predators used 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

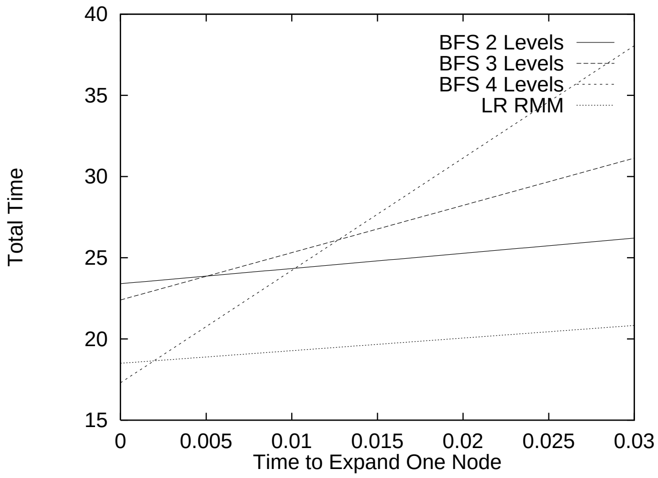

Fig. 6. Plot of the total time that would, on average, be spent by the agents before capturing the prey, for each method. The x axis represents the amount of time it takes to expand a node as a percentage (more or less) of the time it takes the agent to take action.

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 6. 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.

5 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 three main lessons. Firstly that it is possible and beneficial to ignore some of the deeper models. Secondly, that the question of which models can safely be ignored, and which ones should not be ignored, can only be answered at run time. Our data showed that the actual nodes chosen for expansion differ from situation to situation (i.e. depending on the relative locations of the predators and prey). Thirdly, 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.

From our experience with the algorithm, the test results, and our ongoing work, we arrived at some important conclusions. First of all, recursive models appear very frequently when working with Multi-Agent Systems (for another example see [18]). They are not just artifacts of our modeling techniques but are, in fact, dictated by the fact that "smart" agents when interacting with other agents, will usually chose to build models of them. Moreover, if the agents do a reasonable job in producing these models, they will find them useful for predicting the actions of others and, therefore, for determining the best action to take. The success of these models leads to the building of recursive models for predicting the actions of others. This escalation in recursive modeling quickly leads to models that are too unwieldy to use, as argued before. However, our second conclusion points to the fact that one can not pick an arbitrary model size and say that agents will only build recursive models smaller than that, even when given a fixed environment. The problem is that there are times when using deeper models will be profitable and times when the best models are the shallow and quick ones. We see our algorithm as a first step in developing a principled method that an agent can use to decide the depth of the models its should use.

Experience has also shown us that there is a definite set of challenges that lie ahead. We have seen that, in order to apply our techniques for reducing the amount of information the agent considers, we actually need more information. For our experiments, we made some ad-hoc decisions as to what extra information to gather and how to store it. We believe that such decisions will always have to be made and will be domain-dependent. That is why we are trying to divorce our theory from the particulars of the modeling/learning techniques and develop a decision framework that would allow an agent, given its own modeling/learning techniques, to determine what particular models to use and when to use them. Finally, we are looking for ways to apply or ideas to other modeling techniques. Our algorithm is based on the RMM framework which is a formal and clean modeling technique that made it easier for us to formalize our ideas. Unfortunately, these benefits do not come for free, namely, RMM suffers from a combinatorial explosion since it requires knowledge of the payoffs for all the combinations of all the possible actions. We are currently trying to apply the ideas distilled from this work and form a more general approach that can work with different agent modeling techniques such as those based on Beliefs, Desires and Intentions logics [13].

References

1. Edmund H. Durfee. Blissful ignorance: Knowing just enough to coordinate well. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 406–413, 1995.
2. Edmund H. Durfee, Piotr J. Gmytrasiewicz, and Jeffrey S. Rosenschein. The utility of embedded communications and the emergence of protocols. In *Proceedings of the 13th International Distributed Artificial Intelligence Workshop*, 1994.
3. Edmund H. Durfee, Jaeho Lee, and Piotr J. Gmytrasiewicz. Overeager reciprocal rationality and mixed strategy equilibria. In *Proceedings of the eleventh National Conference on Artificial Intelligence*, 1993.
4. Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.
5. Les Gasser, Nicholas F. Rouquetter, Randall W. Hill, and John Lieb. Representing and using organizational knowledge in distributed ai systems. In Les Gasser and Michael N Huhns, editors, *Distributed Artificial Intelligence*, volume 2, pages 55–78. Morgan Kauffman Publishers, 1989.
6. Piotr J. Gmytrasiewics and Edmund H. Durfee. A rigorous, operational formalization of recursive modeling. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 125–132, 1995.
7. Piotr J. Gmytrasiewicz. *A Decision-Theoretic Model of Coordination and Communication in Autonomous Systems (Reasoning Systems)*. PhD thesis, University of Michigan, 1992.
8. Piotr. J. Gmytrasiewicz. On reasoning about other agents. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents Volume II — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996. (In this volume).
9. Piotr J. Gmytrasiewicz and Edmund H. Durfee. Toward a theory of honesty and trust among communicating autonomous agents. *Group Decision and Negotiation*, 2:237–258, 1993.
10. Richard E. Korf. A simple solution to pursuit games. In *Proceedings of the 11th International Distributed Artificial Intelligence Workshop*, 1992.
11. Ran Levy and Jeffrey S. Rosenschein. A game theoretic approach to the pursuit problem. In *Proceedings of the 11th International Distributed Artificial Intelligence Workshop*, 1992.
12. T. A. Montgomery and Edmund H. Durfee. Using mice to study intelligent dynamic coordination. In *Proceedings of IEEE Conference on Tools for AI*, 1990.
13. Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 312–319, 1995.
14. Stuart Russell and Eric Wefald. *Do The Right Thing*. The MIT Press, Cambridge, Massachusetts, 1991.

15. Sandip Sen and Edmund H. Durfee. Adaptive surrogate agents. In *Proceedings of the 13th International Distributed Artificial Intelligence Workshop*, 1994.
16. Herbert A. Simon. *Models of Bounded Rationality*, volume 2. MIT Press, 1982.
17. Larry M. Stephens and Matthias B. Merx. The effect of agent control strategy on the performance of a DAI pursuit problem. In *Proceedings of the 9th International Distributed Artificial Intelligence Workshop*, 1990.
18. M. Tambe and P. S. Rosenbloom. Agent tracking in real-time dynamic environments. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents Volume II — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996. (In this volume).
19. José M. Vidal and Edmund H. Durfee. Agent modeling methods using limited rationality. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, page 1495, 1994.