Learning Models of Other Agents Using Influence Diagrams

Dicky Suryadi and Piotr J. Gmytrasiewicz
Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX 76019, Box 19015
Phone: (817) 272-3399
{suryadi,piotr}@cse.uta.edu

Abstract

We adopt decision theory as a descriptive and prescriptive paradigm to model rational agents. We use influence diagrams as a modeling representation of agents, which is used to interact with them and to predict their behavior. In this paper, we provide a framework that an agent can use to learn the models of other agents in a multi-agent system (MAS) based on their observed behavior. Since the correct model is usually not known with certainty our agents maintain a number of possible models and assign a probability to each of them being correct. When none of the available models is likely to be correct, we modify one of them to better account for the observed behaviors. The modification refines the parameters of the influence diagram used to model the other agent's capabilities, preferences, or beliefs. The modified model is then allowed to compete with the other models and the probability assigned to it being correct can be arrived at based on how well it predicts the behaviors of the other agent already observed.

Keywords: *learning, decision theory, models of agents, influence diagrams, probabilistic networks.*

1. Introduction

As research and development activities in agent-based systems progress, we can expect to see that in the future much of our computer-related works will be done with the help of agents. Much efforts in the design of these agents are directed toward making them "intelligent". The term here is loosely translated as enabling the agents to actively seek the best way to fulfill their tasks, by taking into accounts the dynamics of their environments. Learning agents fit into this description best, as they are characterized as having the ability to adapt their interaction strategies based on the perceptions of their environment.

In some systems, an agent may be working in the presence of other agents, either automated or human. These types of systems are called multi-agent systems (MAS), and are currently an active area of research. One important motivation behind the research is to find techniques that allow multiple agents to coordinate their actions so that individual rational actions do not adversely affect the overall system efficiency [1]. Effective coordination among agents in dynamic environments may be achieved by extending the agents' learning ability to recognize the capabilities, desires, and beliefs of other agents present in their environment [6].

Several papers have reported variety of techniques for constructing the models of agents. [10] described a rule-based model for plan recognition task in the air combat simulation environment, while [2] explored the use of finite automata to model the opponent agent's strategy. A series of papers reported works on recursive modeling method (RMM) for decision-theoretic agents, which uses deeper, nested models of other agents [7, 15, 5, 4, 12]. RMM represents an agent's decision situation in the form of a payoff matrix. In terms of belief, desire and intention (BDI) architecture, a payoff matrix contains a compiled representation of the agent's capabilities, preferences, and beliefs about the world. Beliefs about other agents are represented in terms of their own payoff matrices. These matrices form a hierarchical modeling structure, which is evaluated in the bottom-up fashion to determine the agent's action that maximize its expected utility.

Another decision-theoretic tool for constructing models of agents is an influence diagram (or decision network), which represents information about an agent's capabilities, preferences, and beliefs in a more explicit form. While RMM payoff matrices can be seen to summarize the information contained in the influence diagram [12], this type of representation may provide better insight to the learning problem as it completely specifies all known random variables in the domain and their dependence relationships.

In our work, we seek to develop a method that can be used by an agent to learn the models of other agents. Our basic paradigm is to use decision-theoretic notion of rationality to describe and predict actions of other agents. Thus, we use influence diagrams as a modeling representation for agents. Given an initial model of an agent and a history of its observed behavior, we construct new models by refining the parameters of influence diagram in the initial model. We use BDI architecture as a framework, and we perform learning in stages, starting with the agent's capabilities, then modifying its preferences, and finally, its beliefs. These stages are ordered in the increasing level of complexity. Since the correct model is not known with certainty, we may have to maintain a number of new models. The probabilities of a model being correct can be assigned based on how well it predicts the history of behavior.

The rest of the paper starts with an overview of influence diagrams, followed by examples on how they can be used to represent decision models of agents. In the next section, we get into the problem on learning the models of other agents. We present a learning method for other agents capabilities and preferences, and we provide an example on how to apply it in a particular MAS domain. Finally, we present our conclusions and directions for further work.

2. INFLUENCE DIAGRAMS

An Influence diagram [9] is a graphical knowledge representation of a decision problem. It may be viewed as an extension to a Bayesian or belief network [13], with additional node types for decisions and utilities. Influence diagrams have three types of nodes: nature node, decision node, and utility node. Just as in belief networks, nature nodes are associated with random variables or features, which represent the agent's possibly uncertain beliefs about the world. Decision nodes hold the choice of actions an agent has, thus represent the agent's capabilities. Utility nodes represent the agent's preferences. The links between the nodes summarize their dependence relationships.

In utility theory, agent's preferences are represented as a real valued number, namely utility, which expresses the desirability of a world state. The mapping from a state to a utility value is called a utility function. In an influence diagram, the value of the utility function is represented in the diamond-shaped utility node, and the features that are the arguments to this function are assigned to be the parents of the utility node. Commonly, the table associated with utility node is a straight tabulation of the agent's utility as a function of the parents.

Evaluation of the influence diagram is done by setting the value of the decision node to a particular choice of action, and treating the node just as a nature node with a known value that can further influence the values of other nodes. We calculate the action's utility, first by calculating the conditional probabilities for the parents of the utility node using standard inference algorithm, and then feeding the results to the utility function. The algorithm for evaluating influence diagrams are given in [14].

The anti-air defense domain, adapted from [12], presents an example on how to use an influence diagram to represent an agent's decision model in an MAS domain. The scenario shown in figure 1 depicts an MAS with two agents, in the

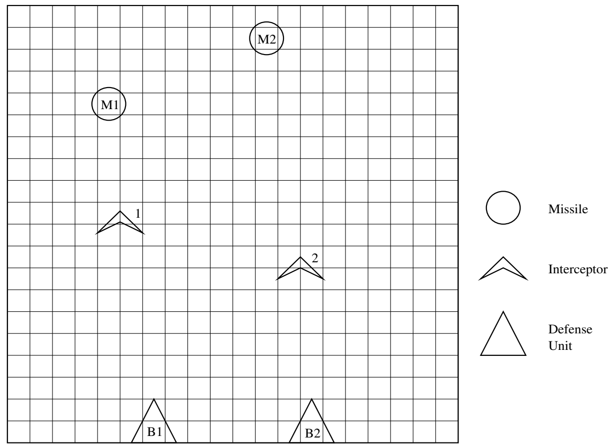

Figure 1. The anti-air defense domain.

roles of the defending units, against two incoming missiles in a 20x20 grid world. Each defending unit has only one interceptor and has to decide, in the absence of communication, which of the incoming missiles to intercept. When a missile reaches the ground, it detonates and inflicts damage in proportion to its warhead size. The goal of each unit is to minimize damage to the overall ground site. To coordinate their interception decisions, RMM suggests that a good way is to have the units model each other to avoid redundant targeting of the same threat.

Figure 2 shows a representation that unit B1 may have to model decision-making of unit B2. The nature nodes represent all the world features that influence the unit's decision. The missile's speed and warhead size, the angle and distance between the missile and the unit, and the type of interceptor the unit has (long range or short range) are the relevant features. The model measures the missile's interceptability, or the likelihood of intercepting a missile, given the type of the interceptor and given the speed, angle, and distance to the missile. The expected damage resulting from the attack is proportional to the missile's interceptability and warhead size. B2's preference is indicated by the link from *Damage* to utility node *U*. The utility function is thus a function of damage.

In RMM, information or knowledge that an agent has about other agents can be represented as other models nested within the agent's own model. The same princi-

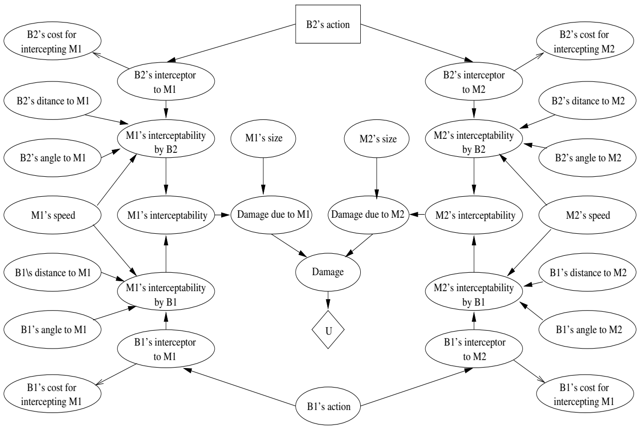

Figure 2. Decision model of defense unit B2.

ple can be applied to our influence diagram representation. Consider *B1's Action* node in figure 2. If B2 has information about B1's decision process, B2 can construct a representation of B1's decision model according to B2's beliefs. This is referred as B2's model of B1. Since B2 may not know for certain the correct model of B1, it may maintain several models, each is assigned the probability of being correct. The conditional probabilities of *B1's Action* can be determined by probabilistically combining evaluation results of all models of B1.

3. LEARNING PROBLEM

The purpose of learning models of other agents is to have models that can predict other agents' behavior correctly, so that the predictions will lead the learning agent to arrive at the best decision in terms of meeting the goal efficiently. The agent should already have a multi-agent reasoning system and maintain initial models of other agents in its knowledge base. We assume that these models are given as the agent's prior knowledge. Due to uncertainty, the agent assigns probabilities to the models, which are updated based on a history of the other agents' observed behavior. If a model is not accurate, its probability will gradually decrease. This signals the need for a better model, which can be obtained by learning.

In most situations, information about other agents only comes from our observation of their behavior. Let us define a history of an agent's behavior as a set of its observed actions during a particular time frame, in which the data of the world states are known. Given only a history of behavior, how do we learn a better model? The idea is to modify the initial model by refining parameters in the influence diagram that are associated with capabilities, preferences and beliefs of other agent, based on the history of its behavior. The refinement can be done in stages with the order as above, according to the increasing level of complexity. The learning process is similar to a hill climbing search in a space of possible models. There can be a number of modified models that can be generated. We will allow the modified models to compete with each other and other models maintained by the learning agent. The probability of each model being correct can be determined based on how well the model predicts the history of behavior.

As mentioned in Section 2, influence diagrams are used for agent modeling. From the point of view of learning, influence diagrams inherit the main feature of learning in belief networks that boils down to finding the correct network structure and the local probability distributions, given the data about the world. Additional aspects of learning in influence diagrams relate to the character of decision and utility nodes. As a modeling representation tool, an influence diagram is able to express an agent's capabilities, preferences, and beliefs, which are required if we want to predict the agent's behavior. Whereas the learning problem in belief networks is limited to learning the beliefs, influence diagrams allow us to extend the description with learning the capabilities, that is, finding the correct possible value assignments for the decision node, and learning the preferences, which is finding the correct utility function.

The rest of this Section describes the method used to perform the refinement in an influence diagram. The methods for learning the capabilities and the preferences are given, followed by the discussion on the strategy for learning the beliefs. Our preliminary idea on the issue of when to commence a certain method of learning is to do the simplest method first, which is learning the capabilities. If the learning fails to produce a good model, we commence the method for learning the preferences. If the learning still fails, we will attempt to learn the beliefs. We regard this issue as an open problem, and will be addressed in our future work.

3.1. Capabilities

An agent's capabilities are the alternative courses of actions it can execute to interact with the world. In an influence diagram, they are represented as possible value assignments to the agent's decision node. These values need to be exhaustive, which means that a model of another agent should cover all the possibilities of what the agent can do, even if they are not known or very unlikely. To avoid the obvious difficulty in listing all the possibilities, a knowledge engineer can specify *Other* action value to represent all the unknown or unlikely actions not explicitly represented. At times, the agent may be observed to be executing an action that is not explicitly present in our current model. In this case, we modify the model by specifying the action as explicit possible value of the decision node, and by updating the CPT's of the nodes that are influenced by the decision node.

For example, consider the agent as a defense unit B1 in the anti-air defense domain. It may be that initially B1 thinks that B2 is not equipped with long range interceptors. It would then model the capabilities of B2 as a set of decision node's values {*SR_M1, SR_M2, Other*} in B1's model of B2, where SR_M1 and SR_M2 denote the actions of launching short range interceptors at missile M1 and M2, respectively. At some time, B1 could observe that B2 launches long range interceptor at one of the targets. Given this observation, B1 will modify its model of B2; the refined set of decision node's values is {*SR_M1, SR_M2, LR_M1, LR_M2, Others*}, where LR_M1 and LR_M2 explicitly represent the actions of launching long range interceptors at M1 and M2, which were thought unlikely before.

Conversely, sometimes the other agent's capabilities may need to be collapsed and included as part of the *Other* value. This may happen, for example, when B2 has become incapacitated and can no longer perform certain actions. From B1's perspective, it may notice that certain actions are missing from B2's history of behavior, especially when B1 believes that the missing action is the action B2 should take, if it is capable of it. To guarantee that the situation is caused only by an inaccuracy in modeling B2's capabilities, we need to identify the next preferable action given the model. If it agrees with the history of behavior, we can reason that the model still captures B2's beliefs and preferences except that somehow B2 cannot do the action that is missing from the history of behavior. B1's model of B2 is modified by removing the value of the missing action from the decision node.

3.2. Preferences

When a model of other agent cannot explain the history of its behavior, one of the possible reasons is that the agent's preferences are not accurately represented by the model. Our strategy will be to modify the model by refining the utility function so that every action in the history of behavior always maximizes utilities of the resulting states. Let $U(S)$ denote the utility of state S, which is given by a utility function.

The general structure of utility function is:

$$U(S) = f(X_1, ..., X_N) \tag{1}$$

where $X = \{X_1, ..., X_N\}$ is a set of features that directly influence the agent's preferences. In influence diagram, X is the set of parents of the utility node. The utility function f, is commonly postulated in multi-attribute utility theory [11, 14] to be a weighted sum of the factors of values of features X_k, $k = 1, ..., N$. For simplicity, we assume that weighted factors depend linearly on the features X_k. This is a strong approximation, for example, in case of the utility of money, St. Petersburg paradox shows that it is not linear. We use this assumption here to simplify the learning procedure.

Therefore, we rewrite an agent's utility function as follows:

$$U(S) = w_1 x_1 + ... + w_N x_N \tag{2}$$

where each weight w_k corresponds to a feature X_k and represents the feature's measure of influence on the agent's utility, and x_k is the value of the feature X_k in state S. Our method of learning the agent's preferences will modify the weights w_k.

Let A^* denote an action that maximizes expected utility:

$$A^* = \arg\max_{a_i} \sum_{j=1}^{J} P(S_j|a_i, E) \times U(S_J) \tag{3}$$

where $A = \{a_1, ..., a_M\}$ is a set of the agent's alternative actions, and S_j are the possible outcome states given a, possibly non-deterministic, action a_i, with j ranging over J different outcomes. The background evidence E in the conditional probabilities represents the known data of the world which are provided in the history of behavior.

Using Equation 2 and 3, we obtain:

$$A^* = \arg\max_{a_i} \sum_{j=1}^{J} P(S_j|a_i, E) \sum_{k=1}^{N} w_k x_k^j \tag{4}$$

where x_k^j is the value of the feature X_k in the state S_j. We can show that this equation is equivalent to:

$$A^* \quad = \quad \arg\max_{a_i} \sum_{k=1}^{N} w_k \chi_k^i \tag{5}$$

where χ_k^i denote the expected value of feature X_k given a_i and background evidences E, given by the summation of all possible values of X_k: $(x_{k,1}, ..., x_{k,r_k})$ times their conditional probabilities:

$$\chi_k^i = \sum_{l=1}^{r_k} x_{k,l} \times P(X_k = x_{k,l}|a_i, E) \tag{6}$$

From the history of other agent's behavior, we have a set of the agent's observed actions during a given time frame. Let $D = \{D(1), ..., D(T)\}$ denotes such set, where T is the total number of actions, and $D(t) \in A$, $t = 1, ..., T$. The refinement of utility function is accomplished by having an initial utility function derived from the initial model, and subsequently adjusting the weights w_k based on the set D, so that $D(t)$ is equal to A^* for every situation t.

The choice of initial utility function depends on the initial knowledge the learning agent has about the other agent.

For example, if agent B1 knows that B2's preferences are only influenced by the features *Damage* and *Cost*, although it doesn't know exactly how B2 computes them, B1 can model B2's utility function only with the two features, and learn the correct weights. But in the extreme case where B1 has absolutely no knowledge on B2's preferences, it may have to put all the known features in B2's utility function and assign zero to all the weights.

To adjust the weights we follow a procedure common in neural network learning. In the context of neural network, our learning is best described as a supervised learning for performing the task of function approximation. We can render Equation 5 into neural network structure as shown in Figure 3. The network represents a decision process at

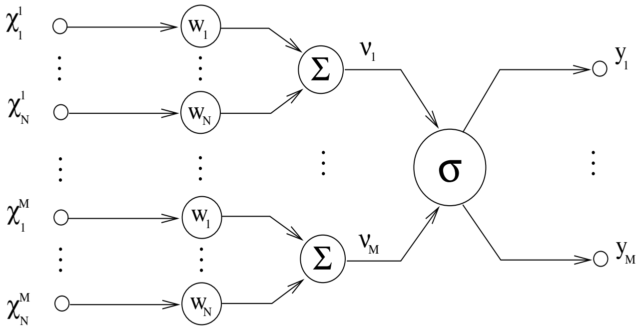

Figure 3. Network for learning the weights.

a decision situation t, where $\sigma(t)$ is the activation function that constitutes the output decision rule [8]:

$$y_i(t) = \begin{cases} 1 & v_i(t) > v_j(t) \quad for\ all \quad j \neq i \\ 0 & otherwise \end{cases} \tag{7}$$

where $i = 1, ..., M$. Let $d_i(t)$ represents the desired output value from $D(t)$ as follows:

$$d_i(t) = \begin{cases} 1 & D(t) = a_i \\ 0 & otherwise \end{cases} \tag{8}$$

Weight adjustment is done by applying a well-known gradient descent technique, namely *delta rule* [16]. The idea is to minimize a cost function based on the error signal $e_i(t)$ so that each weight adjustment brings the actual output value closer to the desired value. A commonly used cost function is the *mean-square-error* criterion:

$$E(t) = \frac{1}{2} \sum_{i=1}^{M} e_i^2(t) \tag{9}$$

where $e_i(t)$ is the error signal which is defined by:

$$e_i(t) = d_i(t) - y_i(t) \tag{10}$$

According to delta rule, the weight adjustment is proportional to the product of the error signal and the input unit. In our network, we use weight sharing to indicate the symmetry of the utility computation. We settle the adjustment for a shared weight as the sum of the correction of weights for all output units. We require normalization on χ before presenting it to the network, so that the resulting weights are normalized. The normalization is given by:

$$\aleph_k^i = \frac{\chi_k^i}{\sqrt{\sum_{s=1}^{M} \sum_{t=1}^{N} (\chi_t^s)^2}} \tag{11}$$

Thus, the final rule is given by:

$$\Delta w_k(t) = \eta \sum_{i=1}^{M} e_i(t) \aleph_k^i(t) \tag{12}$$

where $k = 1, ..., N$ and η is a positive constant that determines the rate of learning. The probabilities required to compute χ^i can be determined by evaluating the influence diagram model of the agent for every a_i.

In general, the gradient descent technique can reach a local minima, making the weights converge to a non-optimal values. However in our case, the function in equation 3.5 which our learning network represents can be shown as linearly separable, therefore the weight space will not have any local minima. The learning will converge to the correct set of weights, given the learning rate parameter is not so large to cause "overshooting" [14].

Example

For the demonstration of our method, let us consider a simple situation in the anti-air defense domain. Say that B1 knows there are at most two features that influence preferences of the unit B2. That is, the set of features that can influence B2's utility is $\{Damage, Cost\}$, where the possible value assignments for $Damage$ are $(0, 100, 500)$ and $Cost$ $(0, 1000, 5000)$.

B1 initially models B2's utility in inverse proportion to $Damage$ and independent to $Cost$. Therefore, the initial assignment of weights to features is:

$$w_1 = -1.0, \quad w_2 = 0.0$$

This means that B1 thinks B2 only cares about minimizing damage to the ground, and not cost of intercepting the missiles. In this example we show how these weight are modified gives observation of B2's behavior that were different from the ones predicted by the initial model.

Let $A = \{I\_M1, I\_M2\}$ is a set of B2's possible actions, where I_M1 stands for intercept missile M1, and I_M2 intercept missile M2. In this case, assume that intercepting missile M1 would require a more sophisticated and more accurate interceptor. The following conditional probability distributions for $Damage$ and $Cost$:

$$\begin{array}{lcl}
\mathbf{P}(Damage|I_M1,E) & = & <.10,.10,.80> \\
\mathbf{P}(Damage|I_M2,E) & = & <.20,.30,.50> \\
\mathbf{P}(Cost|I_M1,E) & = & <0,.30,.70> \\
\mathbf{P}(Cost|I_M2,E) & = & <0,.60,.40>
\end{array}$$

The above values reflect that using the advanced interceptor for missile M1 is more likely to reduce damage, but also more likely to incure a higher cost. The values of expected damage and cost are computed using Equation 6:

$$\begin{array}{lclcl}
\chi_1^1 & = & 0\times.1+100\times.1+500\times.8 & = & 280 \\
\chi_2^1 & = & 0\times0+1000\times.3+5000\times.7 & = & 3800 \\
\chi_1^2 & = & 0\times.2+100\times.3+500\times.5 & = & 410 \\
\chi_2^2 & = & 0\times0+1000\times.6+5000\times.4 & = & 2600
\end{array}$$

where χ_1^1 and χ_2^1 are expected damage and cost given M1, and χ_1^2 and χ_2^2 are expected damage and cost given M2. The next step is to normalize χ, by applying Equation 11. The normalized expected damages and costs given M1 and M2 are as follow:

$$\begin{array}{l}
\aleph_1^1 = 0.060 \\
\aleph_2^1 = 0.821 \\
\aleph_1^2 = 0.089 \\
\aleph_2^2 = 0.561
\end{array}$$

Using the initial weights that indicate that B2 prefers to minimize damages but does not care about the costs, B1 arrives at a prediction that B2 would choose to intercept M1 and execute the action I_M1. Therefore, the values of network outputs according to B1's model of B2 are:

$$y_1 = 1, \quad y_2 = 0$$

In the example case, however, B2 does something B1 did not expect and intercepts missile M2. Given that, B1 observes B2 takes action I_M2, the values of desired outputs, and the error signals are:

$$\begin{array}{ll}
d_1 = 0, & d_2 = 1 \\
e_1 = -1, & e_2 = 1
\end{array}$$

Using Equation 12 with the learning rate of $\eta = .1$, we have:

$$\begin{array}{lcl}
\Delta w_1 & = & .1\times(-1\times0.060+1\times0.089) = 0.029 \\
\Delta w_2 & = & .1\times(-1\times0.821+1\times0.561) = -0.260
\end{array}$$

The weights for the utility function in B1's model of B2 become:

$$w_1 = -0.971, \quad w_2 = -0.260$$

As the result of considering this unexpected behavior of B2, therefore, B1 adjusted the weights it uses to model B2's utility function. Intuitively, the adjusted weights indicate that B2 does not care about the damages as much as B1 initially thought, and that it cares abut the costs of the defense operation as well.

In our experiments we simulated 100 decision situations, in which the positions and the sizes of the attacking missiles were generated randomly for each situation, and B2's behavior was guided by it's preference to minimize both damages and costs in equal proportion. After these iterations, the weights were further to the values:

$$w_1 = -0.777, \quad w_2 = -0.561$$

As the result of considering this unexpected behavior of B2, therefore, B1 adjusted the weights it uses to model B2's utility function. Intuitively, the adjusted weights indicate that B2 does not care about the damages as much as B1 initially thought, and that it cares abut the costs of the defense operation as well.

3.3. Beliefs

An agent's beliefs are represented in influence diagram as the nature nodes and the probabilistic dependence relationships that exist among them. Given that we have knowledge on other agent's capabilities and preferences, and that our model of it still cannot predict its behavior, it is reasonable to assume that the model may not have the correct dependence links or probability assignments. We modify the model by refining the links and the probability assignments. The obvious difficulty is to determine where to begin the refinement. In a complex model, there can be a large number of possibilities that can account for the problem. What we need is to have a learning method that, given a history of the agent's behavior, can trace down the elements in the belief system which need refinement.

Although there are some good algorithms for learning the structure and local probability distributions, such as K2 algorithm [3], we see a difficulty in applying such algorithms to our problem. For example, K2 requires a database of cases on features for constructing the network. Translated to our problem, it means that a learning agent has to have a database of cases on features according to the other agent in order to construct the probabilistic model of it. Given that all the learning agent has is maybe only the history of other agent's behavior, it may not be feasible to apply the algorithm directly.

Our preliminary idea for approaching the problem is to apply similar learning method that we have reported in the previous subsection, only now we set the weights fixed and learn the expected values of the features which influence the utility. By using this process, it is possible to determine which of the feature nodes that are likely to be the source of inaccuracy in the model. We let Φ denote the set of of such feature nodes. There are several strategies we can take to modify the model. The strategies are based on exploring the possible reasons that cause the model's failure to predict the behavior.

The first strategy assumes that the reason is due to incorrect local probability distributions of the features in Φ. Based on this assumption, the model is modified by refining CPT's on the feature nodes in Φ.

The second strategy assumes that the reason is due to incorrect dependence relationships involving the features in Φ. The model is modified by refining the structure of influence diagram. The alternatives are to remove links from certain parent nodes, or to add more links from other nodes.

The third strategy assumes that the problem is caused, not by the features in Φ, but by their parents. It is possible that certain parent nodes have incorrect probability distributions or incorrect dependence relationships. The model is modified by performing these strategies at the level of the parent nodes of Φ. Note that this strategy has a recursive property. It allows the refinement goes bottom up until there are no more parents or the evidence nodes are reached.

For example, consider the situation where B1 observes B2 intercepts missile M1. B1 knows for certain that B2's preference is to minimize damage to the ground. Given that B1 knows that damage would have been minimized had B2 intercepted M2, B1 reasons that it doesn't correctly model B2's beliefs on the values of the node *Damage*. In this case, we assign $\Phi = \{Damage\}$. Let B1's model of B2 be similar to the one in Figure 2. According to the model, damage is measured by the missile's size and interceptability, shown by the links from *Size* and *Interceptability* to *Damage*.

The first strategy reflects a possible reason that B2 really considers intercepting M1 as minimizing damage. This would mean that B1's model of B2 doesn't correctly represent the way B2 evaluates damage. The model is modified by refining local probability distribution in the node $Damage$.

The second strategy reflects another possible reason that B1 incorrectly models B2's beliefs about the dependence relationships from size and interceptability to damage. For example, maybe B2 only considers size as a single factor which influence damage, or maybe B2 considers other factors that affect damage, besides size and interceptability. The model can be modified by removing some links to $Damage$, or adding new links from other nodes to $Damage$.

The third strategy reflects a reason that B1 incorrectly models B2's beliefs about the dependence relationships at the level of the nodes that influence $Damage$. For example, maybe B2's beliefs about the size of incoming missiles are not represented in B1's model of B2. The model can be modified in several ways according to the strategies we described, but now they are done at the level of the nodes $Size$ and $Interceptability$.

These strategies attempt to narrow down the search space of possible models. However, the number of models we come up with can still be unreasonably high, and we are yet to develop a method that can decrease the number further.

4. EXPERIMENTS

The purpose of the experiments is to measure the performance of the learning methods described in Section 3. The experiment is conducted in the anti-air defense domain. We define an experiment run as a decision making situation which involves two defense agents, each has a capability to launch exactly one interceptor against one of the two incoming missiles. The set of decision options for a fully functional agents is $\{$*M1_SR, M1_LR, M2_SR, M2_LR, None*$\}$, which denote actions to intercept missile M1 or M2 with short range or long range interceptors, and an action not to intercept at all.

The experiment runs are generated by an anti-air defense simulator [12]. For each run, the location, speed, and warhead size of the missiles are assigned with random values. Using RMM payoff matrices, each agent then evaluates the models it has about other agent to arrive at an optimum decision. After a certain number of runs, the simulator generates a history of behavior file to be passed on to the learning routine.

The learning routine utilizes the history of behavior and the initial models the learning agent has about the other agent to generate a new model. The new model is then passed back to the simulator. Given a new set of runs, the simulator assigns the new model a probability of being the correct model by means of a Bayesian belief updating mechanism. We use this probability value as the performance indicator for the learning methods.

We design two scenarios to experiment with the learning methods, shown in figure 4. In both scenarios, B1 is a fully functional agent and its utility is influenced only by $Damage$. B1 thinks B2 as a fully functional agent that considers only damage with a probability of 0.8. For 0.2 probability, B1 thinks it has no information about B2. In scenario (a), B2 considers only damage, but it is not a fully functional agent, and can only launch long range interceptors. In scenario (b), B2 is a fully functional agent, but its utility is influenced not only by $Damage$, but also by $Cost$ with equal ratio. In both scenarios, B2 correctly models B1 and assigns the model with 0.8 probability, and no information with 0.2. In this experiment, one hundred runs are generated for each scenario.

In scenario (a), agent B1 runs the method for learning B2's capabilities. After one hundred runs, the method results in a new model for B2 where the decision options are limited to $\{M1\_LR, M2\_LR, None\}$. The new model is passed to the simulator, where its probability is computed.

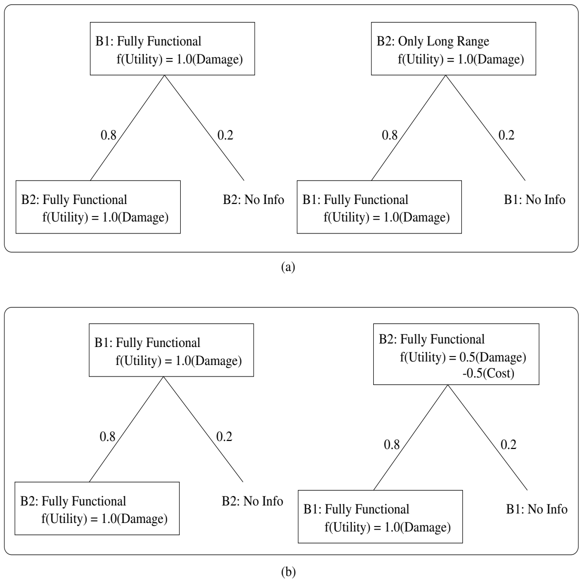

Figure 4. Two scenarios for the learning experiments.

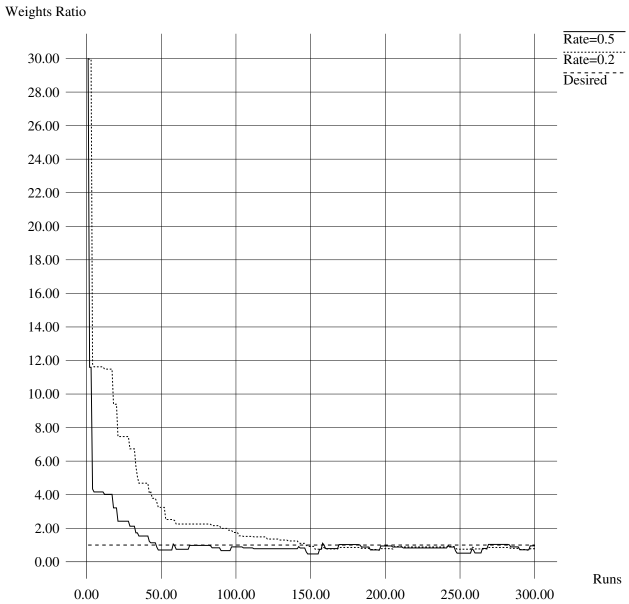

Figure 5. The result of the second scenario.

The resulting probability value is very close to 1.0, which is to be expected, since the new model and the correct model of B2 is similar.

In scenario (b), agent B1 runs the method for learning B2's preferences. Three iterations of one hundred runs are applied to the learning method, using the learning rates 0.5 and 0.2. To plot the weights after each run, we compute the weight ratio as the weight for damage divided by the weight for cost. The result is shown in figure 5. As seen in the diagram, as more runs are applied to the learning method, the weights converge to the correct weights of the model, with the learning rate determines how fast they converge.

We use the result after the first iteration as the new model, where the weights for damage and cost set to -0.47 and -0.53, respectively. The probability for the new model according to the simulator is very close to 1.0, which shows that the new model, although slightly different from the correct model, is acceptable.

We take this experiment further by comparing the average damage to the defense area after one hundred more runs between the agents with the initial model and the new model. The overall performance of agents is measured by how much they can minimize the average damage. A decrease in the average damage as a result of the agent using a different model may indicate that the model provides an improvement in the agent's competence in interacting with the other agent. The diagram, which is shown in figure 6, includes as a measurement point the average damage when all the agents consider only $Damage$ and have the correct model of each other.

In the event where the agents consider $Damage$ and have the correct model of the other agent, the diagram shows that they achieve the best performance. However, when an agent changes its preferences and the other one does not know about it, the performance decreases significantly. When the agent performs learning and uses the resulting model, the performance improves. This fact demonstrates that the learning is able to produce a new model that can increase the learning agent's interaction competence.

5. Conclusion and Further Work

This work reports the development of a framework that can be used by a rational agent in a multi-agent system to learn models of other agents. The framework utilizes influence diagrams as a modeling representation tool. The work addresses three strategies to create a new model of the other agent, which are based on learning its capabilities, preferences, and beliefs, given an initial model and the agent's behavior data.

The methods for learning other agent's capabilities and preferences are presented along with the discussion on the strategies for learning other agent's beliefs. The method for learning the capabilities is based on the observation of the occurrence of other agent's actions. The method for learning the preferences seeks to find the other agent's utility function by incorporating a neural network learning technique, which involves the presentation of the history of other agent's observed behavior as the training set and a

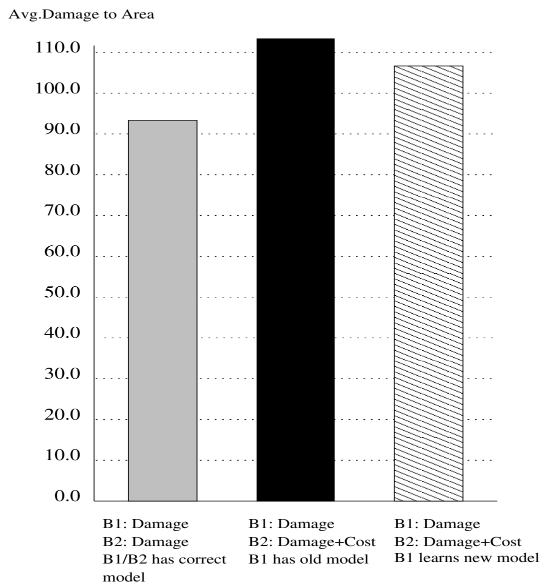

Figure 6. The performance diagram of different models.

series of weight adjustments. Both these methods result in a creation of a new model of the other agent. To assign the probability to the new model being correct, we allow it to compete with other models we have by presenting the history of behavior and performing probabilistic update based on how well each model predicts the behavior.

The experiments to test these methods were done by using the anti-air defense domain, where the learning agent is one of the defense units which protect the area against incoming missiles. The results of the experiments demonstrate the effectiveness of the methods in our chosen domain. By applying the learning mechanisms, the agent is able to significantly increase its interaction competency.

For the future of this work, there are several directions that can be taken. One is to resolve the issue of when a certain learning method should be commenced. Another is to extend the learning to the next level of nested modeling, that is, to learn the models the other agent has about others. The third one is to explore the issue of learning about agents that have the ability to learn themselves. The last one is to devise the method for learning the beliefs based on the discussion in the beliefs Section.

References

[1] A. H. Bond and L. Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[2] D. Carmel and S. Markovitch. Learning models of intelligent agents. *AAAI/IAAI*, 1:62–67, 1996.

[3] G. H. Cooper and E. H. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[4] P. J. Gmytrasiewicz. An approach to user modeling in decision support systems. In *Proceedings of the Fifth International Conference on User Modeling*, pages 121–127, Jan. 1996.

[5] P. J. Gmytrasiewicz and E. H. Durfee. A rigorous, operational formalization of recursive modeling. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS)*, pages 125–132, June 1995.

[6] P. J. Gmytrasiewicz and E. H. Durfee. Rational interaction in multiagent environments: coordination. *Submitted for publication, available at http://www-cse.uta.edu/ piotr/piotr.html*, 1998.

[7] P. J. Gmytrasiewicz, E. H. Durfee, and D. K. Wehe. A decision-theoretic approach to coordinating multiagent interactions. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 166–172, July 1991.

[8] S. Haykin. *Neural Network: A Comprehensive Foundation*. Macmillan College Publishing Company, New York, 1994.

[9] R. A. Howard and J. E. Matheson. Influence diagrams (article dated 1981). *Howard, R.A. and Matheson, J.E. (Eds.), Readings on the principles and applications of decision analysis*, 2:719–762, 1984.

[10] G. Kaminka, M. Tambe, and C. Hopper. The role of agent-modeling in agent robustness. In *Proceedings of the Conference on AI Meets the Real World*, 1998.

[11] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley, New York, 1976.

[12] S. Noh and P. J. Gmytrasiewicz. Agent modeling in antiair defense. In *Proceedings of the Sixth International Conference on User Modeling*, pages 389–400, June 1997.

[13] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kauffman, San Mateo, CA, 1988.

[14] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[15] J. M. Vidal and E. H. Durfee. Recursive agent modeling using limited rationality. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS)*, pages 376–383, June 1995.

[16] B. Widrow and M. E. Hoff Jr. Adaptive switching circuits. *IRE WESCON Convention Record*, pages 96–104, 1960.