A Language for Opponent Modeling in Repeated Games

Ya'akov Gal and Avi Pfeffer
Division of Engineering and Applied Sciences
Harvard University, Cambridge, MA 02138
{gal,avi}@eecs.harvard.edu

ABSTRACT

Traditional game-theoretic formalisms, commonly used in multi-agent systems, invoke the assumption of common knowledge of rationality to justify a Nash equilibrium solution. It is assumed that all agents know a correct model of the game and are completely rational, and that this is common knowledge. However, real-life agents are partially irrational, they may use models other than the real world to make decisions, and they may be uncertain about their opponents' decision making processes. For modeling boundedly-rational agents, a descriptive approach to game theory is needed, in which agents model their opponents and attempt to predict their behavior using their model. We present Networks of Influence Diagrams (NIDs), a language for descriptive decision and game theory that is based on graphical models. This paper describes NIDs and their syntax, and provides algorithms for solving NIDs and learning NID parameters. We also show that NIDs provide an elegant framework for opponent modeling that is more expressive than current approaches, leads to a better outcome than the Nash equilibrium strategy and is able to capture non-stationary distributions of opponents.

Keywords

Decision-making under uncertainty, Game theory, Opponent modeling

1. INTRODUCTION

In recent years, decision theory and game theory have had a profound impact on artificial intelligence. On a fundamental level, the decision-theoretic approach provides a definition of what it means to build an intelligent agent, by equating intelligence with utility maximization. Meanwhile, game theory has been adopted by many as the basis for building multi-agent systems. More concretely, a wide variety of representations and algorithms have been developed to determine game-theoretic solutions to problems in multi-agent systems.

AAMAS '03 July 14-18, Melbourne, Australia

However, the focus in AI so far has been on the classical, normative approach to decision and game theory, in which the optimal behavior of a *rational* agent is prescribed. In this approach, a game specifies the actions available to the players, and the utility to each player associated with each possible set of actions. The game is then analyzed to determine rational strategies for each of the players. The assumption of rationality also implies that the structure of the game, including the payoff functions and strategies available to each player is known to all of the players, and that all of the players' reasoning about the game is captured in the game structure itself.

However, the assumption of rationality does not hold generally in the real world [2]. A player may be mistaken about the structure of the game, or may have uncertainty about the beliefs of other players about the structure of the game. In addition, agents typically use various data structures and reasoning methods not present in the game itself, such as heuristics, to determine how to play. Thus, there exists a need in AI for tools that describe the reasoning processes of *boundedly rational* agents.

We argue that a *descriptive* approach to modeling multi-agent interaction is needed for this task. In descriptive decision and game theory, one specifies a model of a situation faced by agents and uses the model to predict or explain their behavior.

However, using a descriptive approach to modeling bounded rationality requires us to make a clear distinction between the structure of the game, which determines how the actions of the agents produce effects in the real world, and the mental models used by the agents to make their decisions.

We present a knowledge representation language for games that makes this distinction. In our framework, called *Networks of Influence Diagrams (NIDs)*, there is an explicit model of the real-world game being played, as well as additional mental models of agents playing the game. The language allows for the possibility that an agent's model is different from the real-world model. The language also allows multiple possible mental models for an agent, with uncertainty over which model the agent actually uses. The language is recursive, so that the mental model for an agent may itself contain models of the mental models of other agents, with associated uncertainty.

The NID framework is based on *influence diagrams (IDs)* [11], a representation language based on graphical models for single-agent decision problems, and their recent extension to *multi-agent influence diagrams (MAIDs)* [15]. Graphical models, such as Bayesian networks, provide for natural rep-

resentations of complex situations by explicitly describing the variables involved and the relationships between them. The resulting representations are much more compact than they would be otherwise, and often lead to exponential savings in inference time. MAIDs extended the benefits of graphical models to game theory, but only for the classical normative approach, i.e. it was assumed that all agents know a correct model of the game and are completely rational. Our work provides the benefits of graphical models to modeling bounded-rational agents.

NIDs can be used both descriptively and normatively. In the descriptive approach, they can be used as a tool for describing, in a clear and explicit manner, the reasoning of agents. NID models can be solved to determine the behavior that results from the agents' reasoning processes. The models can thereby be used to predict agents' behavior, or to explain how certain behavior might have arisen.

When used to predict the behavior of other agents, NIDs can form the basis for an opponent modeling approach to game playing. In the normative approach, NIDs are used to model a game, and the beliefs of the agents involved in the game. Solving the NID amounts to computing behavior that is rational *with respect to the beliefs of the agents*. In other words, it is the best that the agents can do, given their beliefs.

In separate papers, we presented a variety of examples demonstrating the expressive power and compactness of NIDs [7] as well as formal semantics for the language [8]. In this paper, we extend our framework to include learning of opponents' models in repeated games.

One of the main approaches to game playing with imperfect information is opponent modeling, in which one tries to learn the patterns exhibited by other players. When playing a repeated game, an agent can use her model to predict the behavior of her opponents and to play a best-response to that prediction. After observing her opponents' play, the agent updates her model and uses it again to predict her opponents' behavior in the next round.

Traditional models of learning in games [6] assume that strategies exist a-priori and that opponents' strategies do not change over time. However, real agents often use rules, heuristics, patterns or tendencies when making decisions. Furthermore, agents often reason about the workings of other agents. Consider, for example, a baseball manager ordering a pitch-out in a certain situation because the opposing manager has a tendency to order a stolen base in that situation. In addition, agents frequently change their strategy over time.

In this paper, we show that NIDs provide an elegant and precise tool for opponent modeling, and that using NIDs, it is possible to model non-stationary distributions over opponents' strategies.

The rest of this paper is organized as follows. In Section 2 we present the syntax of NIDs as well as a general algorithm for solving a-cyclic NIDs. Section 3 presents an algorithm for learning NID parameters, and Section 4 puts the algorithm to the test in an interesting domain, where we show that opponent modeling using NIDs does better than just playing the Nash equilibrium solution. Section 5 discusses future work and concludes.

2. NID SYNTAX

We begin the description of NIDs by describing the basic building blocks, *influence diagrams (IDs)* [11]. An ID consists of a directed graph with three types of nodes: chance nodes, drawn as circles, represent random variables; decision nodes, drawn as rectangles, represent decision points; and value nodes, drawn as a diamonds, represent the agent's utility which is to be maximized. There are two kinds of edges in the graph. Edges leading to chance and value nodes represent probabilistic dependence, in the same manner as edges in a Bayesian network. Edges leading into decision variables represent information that is available to the agent at the time the decision is made.

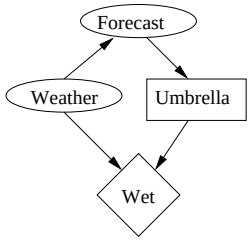


Figure 1: real-world ID of umbrella scenario

For example, consider the following scenario: Waldo is about to leave his house, and needs to decide whether or not to take an umbrella. Waldo's objective is to remain dry, but carrying an umbrella around is annoying. The ID in Figure 1 describes Waldo's decision problem as we see it. There is a prior distribution over the weather, for which the forecast is a noisy sensor. Waldo gets to observe the forecast, and decides whether or not to take an umbrella, which, depending on the weather, influences his final utility.

Solving an ID requires computing an optimal strategy for the agent. Such a strategy specifies what the agent should do, given her available information. In our case, a strategy will specify whether or not the agent should take an umbrella, given that it is or is not raining. Note that in some cases an influence diagram may contain several sequential decisions to be made by an agent. In such cases, a strategy specifies what the agent should do at each of her decisions. It is standard to assume that all information that was known for earlier decisions is available to later decisions, so the strategy will specify what an agent should do for each decision, conditioned on all prior information. After the strategies for an ID have been computed, a Bayesian network can be produced in which the decision nodes are replaced by chance nodes representing the optimal strategies for the agent. This network can then be used to predict what might actually happen in the situation. In our example, if we have determined that Waldo's strategy is to take an umbrella if the forecast says rain, we can then query the probability that Waldo will get wet.

Multi-agent influence diagrams (MAIDs) [15] extend the framework of ID to game-theoretic situations. Syntactically, MAIDs are almost the same as IDs. The only difference is that each decision and utility node is now associated with a particular agent. A MAID represents a game in which each agent gets to choose the decisions associated with it. Once all decisions have been fixed, values of all utility nodes are determined, and each agent's utility is the sum of the values of the utility nodes associated with it. Solving a MAID requires computing a Nash equilibrium of strategies for each

of the players.

To motivate the definition of NIDs, let us return for now to the previous single-agent example. Suppose now that Waldo's view of the world is slightly different from what we believe to be the correct model, in that Waldo is more trusting of weather forecasters, and uses a different sensor model for the forecast variable. Structurally, the ID representing Waldo's view is the same as ours, but the values of the parameters are different. We would like to be able to answer a query such as "What is the probability, according to our view of the world, that Waldo will get wet?" This requires reasoning with both Waldo's view and our view. We can determine Waldo's decision function by maximizing his expected utility relative to his own model. We can then plug his decision function into our model, and compute our subjective belief in the event that he will get wet. We can also ask, "What is the cost to Waldo of using his flawed model?", by comparing the utility he gets using his decision function with the one he would get if he used the correct model to determine his decisions.

In the previous example, Waldo's model differed from our own only in the conditional probabilities, but it can differ in any aspect. For example, we might imagine that Waldo does not believe the forecast to depend on the actual weather at all. We might imagine that instead, Waldo listens to the Dow Jones index, which he believes to be a barometer of the weather, in order to decide whether to take an umbrella. We also might have uncertainty as to which model Waldo uses.

We introduce the following language to allow for all of these possibilities:

An *Network Of Influence Diagrams (NID)* is a rooted directed acyclic graph, in which each node is a MAID. To avoid confusion with the internal nodes of each MAID, we will call the nodes of an NID *blocks*. The root of the graph is called the *top-level model* and represents the real world from the modeler's point of view. Edges in the graph from block U to block V are labeled $\{i, \mathbf{D}\}$, where $\mathbf{D}$ is a set of decision variables in U belonging to agent i. Intuitively, such an edge means that in model U, we view agent i as using mental model V to make decisions $\mathbf{D}$. We say that the decisions $\mathbf{D}$ in U are *modeled* by V.

For this to make sense, every decision variable $D \in \mathbf{D}$ must appear in V. However, we do not require that it be a decision for agent i in V; it may be a chance variable instead. In both cases, we require that the parents of D in V be a subset of the informational parents of D in U. In other words, agent i cannot have more information when making decision D in V than she did in U. If she did, we could not use V to model her decision in U, because the decision rule in V would require her to examine information she does not have access to in U.

If the variable D is a chance variable for i in V, we are viewing agent i as behaving like an automaton with respect to the decision D. For example, agent i may be following a pattern or social convention. This allows one important form of bounded rationality to be captured. Agents do not always optimize all of their decisions, but may perform some of them according to fixed rules or heuristics.

There may be more than one edge out of a block U labeled by an agent i. For example, there may be two different models V_1 and V_2, with edges $\{i, \mathbf{D}_1\}$ from U to V_1 and $\{i, \mathbf{D}_2\}$ from U to V_2. In this situation, we are using two different models to model agent i's decision making processes with regard to decisions $\mathbf{D}_1$ and $\mathbf{D}_2$. We will see later how this capability is also useful for modeling boundedly rational agents. We also allow for uncertainty over which model agent i uses for a decision. This is represented by multiple edges labeled $\{i, \mathbf{D}\}$ leaving U.

In order for the model to be coherent, we do require the following property: if $\{i, \mathbf{D}_1\}$ and $\{i, \mathbf{D}_2\}$ are two edges leaving U, then either $\mathbf{D}_1 = \mathbf{D}_2$ or $\mathbf{D}_1 \cap \mathbf{D}_2 = \emptyset$. This means that if we have uncertainty over whether some set of decisions is modeled by V, either all of them will be modeled by V or none of them will. The condition also implies that for any individual decision D for agent i at model U, there is at most one set of decisions to which it belongs appearing in the right hand side of the label on any edge leaving U.

As an example, let us revisit our umbrella scenario and suppose that an analyst has uncertainty over which model Waldo uses to decide whether to take an umbrella, when the top-level model, denoted I_r is given in Figure 1. The analyst is unsure whether Waldo uses model I_f, where he observes the weather forecast, or model I_j, where he looks at the Dow Jones index. These models make up our NID in Figure 2(a).

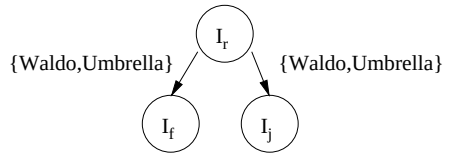
When there are multiple edges leaving U labeled with the same label $\{i, \mathbf{D}\}$, we must quantify our uncertainty over which model agent i uses for deciding decisions $\mathbf{D}$. This is captured by introducing a node Mod[$\mathbf{D}$] into the influence diagram U. Mod[$\mathbf{D}$] is a chance node, that may be influenced by other nodes of U, just like any other chance node. The values of Mod[$\mathbf{D}$] range over the blocks V such that there is an edge from U to V labeled $\{i, \mathbf{D}\}$. In addition, Mod[$\mathbf{D}$] can take on the value U itself. Intuitively, when Mod[$\mathbf{D}$] takes value V it means that decisions $\mathbf{D}$ are actually modeled by V, and when Mod[$\mathbf{D}$] has value U it means the decisions are actually taken according to U. (The correct model, from the point of view of U). Mod[$\mathbf{D}$] has a conditional probability distribution, just like any other chance node.

Following our example, suppose the analyst believes Waldo to use model I_j with probability 0.7, model I_f with probability 0.2, and the real-world model I_r with probability 0.1. The top-level model I_r will now include a chance node labeled Mod[$Umbrella$] whose only child is $Umbrella$. The conditional probability table(CPT) for Mod[$Umbrella$] includes entries for I_j, I_f and I_r with corresponding probabilities $0.7, 0.2$, and 0.1.

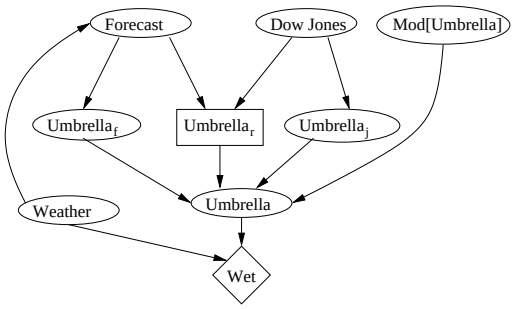
2.1 Solving NIDs

Solving an NID means computing a decision rule for every decision in every MAID in the network. For acyclic NIDs, this can be done easily in a bottom-up fashion, from the leaves of the graph to the root. The leaves are simply MAIDs, and can be solved using the MAID algorithm [15]. In the case that they only involve decisions of a single agent, they can be solved using a standard ID algorithm such as [17, 5]. For an internal block U, once all of its children have been solved, decision rules are available for all of the nodes in U modeled by one of U's descendants.

We then transform U into a new MAID U'. All chance and utility nodes of U become nodes of U', with the same parents and conditional probability distributions. For each decision D in U, we introduce a decision node D_U into U', with the same informational parents as D in U. We also introduce a chance node D_V for each V such that D may



(a) umbrella NID



(b) transformed top-level ID

Figure 2: Umbrella scenario revisited

be modeled by V. The parents of D_V are the same as its informational parents in V, which as we stated earlier must exist in U. The conditional probability distribution for d_V is the decision rule computed for d in V. Finally, D is made into a chance node in U', with parents Mod[$\mathbf{D}$], D_U and all the D_V. Its CPT is a multiplexer, with the value of Mod[$\mathbf{D}$] determining which of the possible decisions gets assigned to D. The children of D are the children of the original decision D in the original U. Once this process has been done for every set of decisions, U' is a MAID containing the models for decisions taken outside of U and incorporating the uncertainty about which models are used. It can then be solved to obtain decision rules for the decisions that are made in U.

Returning to our example, after solving the leaf models and transforming the ID at the root, we will end up with the ID depicted in Figure 2(b). We then solve this ID, converting it to a Bayesian network.

Once we have computed a probability distribution over all of the models at the top-level of the NID, we are able to answer a wide variety of queries. For example, we can find the probability of Waldo getting wet by "plugging in" the CPT for umbrella from Waldo's model into the top-level model and inferring from it the probability $P(wet = T)$. Similarly, we can compute the cost to Waldo of using the wrong model by computing $E[U^{RW}_{RW}] - E[U^{waldo}_{RW}]$ where $E[U^{A}_{RW}]$ is defined as the expected utility under the real-world random variables given the decisions as in model A.

3. LEARNING NID PARAMETERS

Until now, we have assumed that NID parameters are known to the agents prior to solving the model and that these parameters do not change over time. We now relax these assumptions and allow NIDs to include blocks in which some nodes are missing their CPTs. In order for the model to be coherent, we must provide an algorithm for learning the values of missing CPTs by incorporating observations over time.

We present such an algorithm for a repeated game setting, in which agents get to observe their opponents' behavior at every round of the game. Agents use the observations at each round to learn a better model of their opponents, and then exploit that model in order to predict their opponents' behavior at the next round. A good learning algorithm will increase the accuracy of the model as the game progresses and more observations are obtained.

For repeated games, NIDs provide a natural framework for learning the distribution over strategies and chance variables in blocks where nodes are missing their CPTs. As we will show, NIDs can also learn models of opponents that change over time.

We do make the following restrictions. Given any NID model, we require that any variable M that is missing its CPT in some block must also appear at the top-level block. If M is a chance node, this implies that an agent or modeler cannot learn a variable that appears in some mental model of one of the agents, but does not exist in the real-world. If M is a Mod node, this implies that any agent or modeler can only learn strategies that are played out in the real-world.

In addition, if any node D appearing on an edge leading to U depends on the CPT of M, we require that D also appear in the top-level block. This is to ensure that any dependencies between M and any variable that are described in an inner block are also present in the top-level block. To verify whether D depends on the CPT of M, we use Geiger et. al's [9] definition of a *requisite node*:

Let G be a Bayesian network and let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be sets of variables in G. A node Z is a *requisite probability node* for the query $P(\boldsymbol{X}|\boldsymbol{Y})$ if there exists two Bayesian Networks G_1 and G_2, that are identical to G except in the CPT they assign to Z, but $P(\boldsymbol{X}|\boldsymbol{Y})$ in G_1 does not equal $P(\boldsymbol{X}|\boldsymbol{Y})$ in G_2.

Returning to our NID, if D is a chance variable, then if M is a requisite node to the query $P(D)$, any change to the CPT of M will affect D, and therefore both M and D must be present in the top-level block. If D is a decision variable, we use a similar argument, based on a slightly modified version of Koller and Milch's [15] definition of *strategic relevance*:

The node M is strategically relevant to D iff M is a requisite node for the probability $P(\mathsf{U}_D|D, Pa(D))$, where U_D is the utility node associated with D.

Intuitively, a decision D is strategically relevant to a chance node M if optimizing the decision rule of D depends on the CPT of M.

Returning to our NID, if D is a decision variable, then if M is strategically relevant to D, we require that both M and D appear in the top-level block.

Once a NID meets these requirements, we can assign an arbitrary CPT to any missing variable in a NID block, and solve the NID bottom-up, using the algorithm described in Section 2, before the game ensues. Now we have a top-level block incorporating all of the agents' beliefs. Next, we denote a missing CPT entry for the variable X_i taking value x once its parents take value u as Θ_{iux}.

We outline an algorithm that computes the maximum likelihood estimate for missing NID parameters in a repeated game setting. For any X_i variable at the top-level block that

is missing its CPT, let N_{iux} denote the number of times X_i takes value x while its parents take value u. We let $E_\Theta[N_{iux}]$ denote the expectation over that quantity. We begin by initializing $N_{iux} = 0$ and $E_\theta[N_{iux}] = 0$ for all i, x and u.

If X_i corresponds to a variable that is directly observed, then at each round, we estimate the appropriate CPT entry Θ_{iux} to be

$$P(X_i = x|Pa(X_i) = u) = \frac{N_{iux}}{N_{iu}}$$

where $N_{iu} = \sum_x N_{iux}$. This is an on-line version of the method described in [10] for estimating parameters in Bayesian networks in the case of complete data.

If X_i corresponds to an unobserved variable, then at each round we compute the expected sufficient statistics for every parameter as

$$E_\theta[N_{iux}] = E_\theta[N_{iux}] + P(X_i = x, Pa(X_i) = u|\bar{x})$$

where $\bar{x}$ is the observation at the current round. Here, our algorithm is based on an on-line version of the EM-algorithm [16]. We proceed to compute Θ_{iux} as in the complete data case.

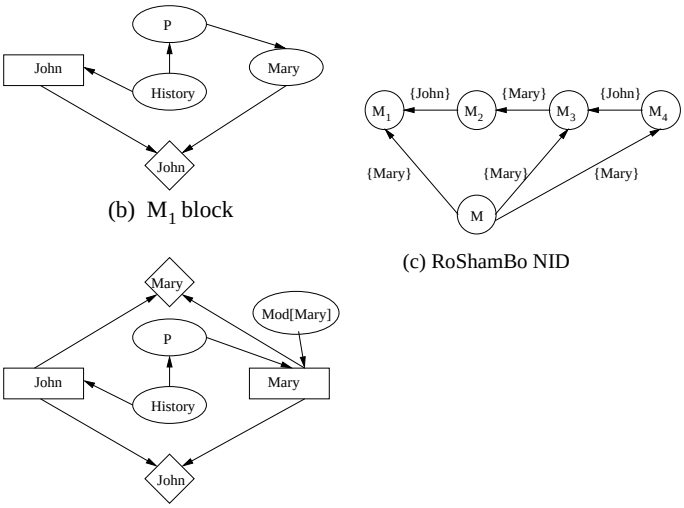


Figure 3: Simple RoShamBo model

4. OPPONENT MODELING USING NIDS

We now show that the representational benefits of NIDs, when coupled with a learning algorithm for parameter values, provide a solid, coherent foundation for opponent modeling. We will also show that they provide a smooth integration between the opponent modeling approach and classical game theory, by allowing uncertainty over whether an agent is computing its decisions heuristically or rationally.

4.1 The RoShamBo Domain

Consider the game of RoShamBo (commonly referred to as Rock-Paper-Scissors). In a single round of the game, two players simultaneously choose one of *rock*, *paper*, or *scissors*. If they choose the same item, the result is a tie; otherwise rock crushes scissors, paper covers rock, or scissors cut paper, as shown in the matrix below.

	$ROCK$	$PAPER$	$SCISSORS$
$ROCK$	$(0,0)$	$(-1,1)$	$(1,-1)$
$PAPER$	$(1,-1)$	$(0,0)$	$(-1,1)$
$SCISSORS$	$(-1,1)$	$(1,-1)$	$(0,0)$

The game has a single Nash Equilibrium in which both players play a mixed strategy over $\{rock, paper, scissors\}$ with probability $\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\}$. Therefore, if both players do not deviate from their equilibrium strategy, they are guaranteed an expected payoff of zero. In fact, it is easy to verify that a player who always plays his equilibrium strategy is guaranteed to get an expected zero payoff regardless of the strategy of his opponent. In other words, sticking to the equilibrium strategy guarantees not to lose a match, but it also guarantees not to win it!

Now consider a situation in which two players play repeatedly against each other. If a player is able to pick up the tendencies of a sub-optimal opponent, it might be able to defeat it, assuming the opponent continues to play sub-optimally. In a recent competition [1], programs competed against each other in matches consisting of 1000 games of RoShamBo. As one might expect, Nash equilibrium players came in the middle of the pack because they broke even against every opponent. It turned out that the task of modeling the opponent's strategy can be surprisingly complex, despite the simple structure of the game itself. This is because sophisticated players will attempt to counter-model their opponents, and will hide their own strategy to avoid detection. The winning program, called Iocaine Powder [4], did a beautiful job of modeling its opponents on multiple levels. Iocaine Powder considered that its opponent might play randomly, according to some heuristic, or it might try to learn a pattern used by Iocaine Powder, or it might play a strategy designed to counter Iocaine Powder learning its pattern, or several other possibilities. In short, it is Iocaine Powder's ability to consider many possible strategies of his opponent, as well as change his own strategy over time, that led him to win the match. the strategies employed by Iocaine Powder changes over time.

4.2 Building a RoShamBo Player

Inspired by Iocaine Powder, we constructed an NID for a player that is playing a match of RoShamBo and is trying to model his opponent.

Suppose that John wishes to model Mary's play using an NID. The top-level model of the NID, shown in Figure 3(a), is a "rational" block, which is simply a MAID depicting a RoShamBo round between John and Mary. Both players have access to the history of the game, which might be summarized by some statistics. Also, the history determines some signal P, which is available to Mary to use in her decision making process. In the rational block, Mary will ignore P, and play the Nash Equilibrium strategy. However, there are several alternative models of Mary's decision. According to block M_1, shown in Figure 3(b), John believes Mary to be an automaton that follows some predictive algorithm that is dependent on the signal P. We can then solve M_1 to determine John's best response to Mary. For example, if John thinks, based on the history, that P is most likely to tell Mary to play *rock*, then John would play *paper*. Let us denote this strategy as $BR(P)$.

According to block M_2 that is a rational block that mod-

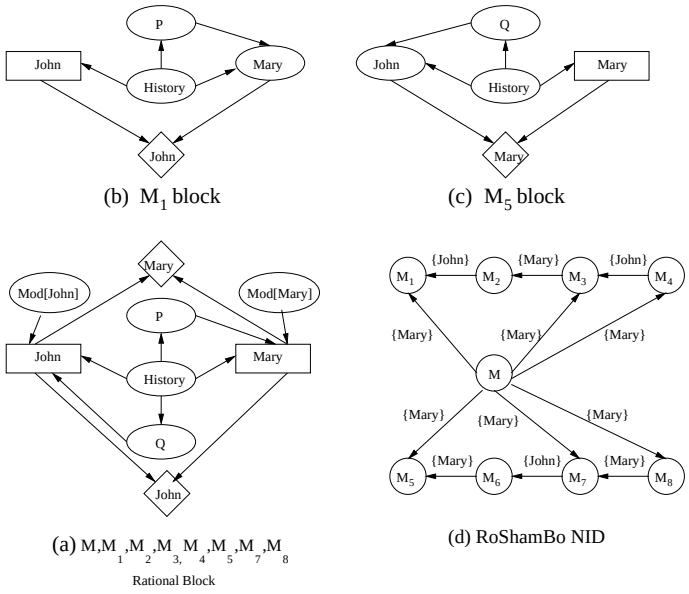


Figure 4: A more complex RoShamBo model

els Mary's beliefs, John's play is modeled by M_1. In other words, Mary believes that John plays $BR(P)$, as a result of John's belief that Mary plays P. Therefore, she will rationally (with respect to her model) play a best response to $BR(P)$, thereby second-guessing John. Mary's strategy in M_2 is thus $BR(BR(P))$. Following our example, in M_2 Mary would not play *rock* at all, but *scissors*, in order to beat $BR(P)$. Now, solving for John's strategy in M_3 that is a rational block that models M_2, we obtain $BR(BR(BR(P)))$. Again, following the example, this would prompt John to play *rock*, in order to beat $BR(BR(P))$. We can continue this to a third level at M_4, in which Mary believes John plays $BR(BR(BR(P)))$, and so
plays $BR(BR(BR(BR(P))))$.

The entire NID is shown in Figure 3(c). In the root block M, John models Mary as playing at one of the possible levels M_1, M_3 or M_4. The uncertainty is captured in the Mod[$Mary$] variable, which also has some probability for Mary playing according to the rational block M. John can then compute his best possible play, according to his beliefs about Mary's possible strategies.

We have shown how to compute three different possible meta-strategies for Mary and for John, based on our assumption that John and Mary are both reacting to an algorithm P that predicts how Mary will behave in the next round. Let us now enhance the complexity of the model by introducing an algorithm Q that predicts how John will behave in the next round. Using a similar rationale, this will result in three more meta-strategies for Mary and John (for example, for John, we will add Q, $BR(BR(Q))$ and $BR(BR(BR(BR(Q)))))$. The resulting NID is depicted in Figure 4(d). Note that each player's strategy depends on his own predictor.

4.3 Learning the Model

So far, our model can describe several different meta-strategies for both John and Mary given P and Q. Can it also learn the distribution over the meta-strategies, even if this distribution changes over time? To find out, we decided to compare the model's performance against the automatic RoShamBo entries who are also meta deliberators.

We chose an entry submitted by John Beal called Simple Modeler. At each round of the game, this program maintains frequencies of moves given past history for both players. For any given history, it computes a distribution over the past moves of both players and plays a move that counters the predicted play of the most extreme distribution. Therefore, Simple Modeler exhibits both meta deliberation and a strategy that changes over time.

Suppose now that in a RoShamBo match consisting of 3,000 rounds, Mary is using Simple Modeler as her model of John and John is using a NID as in Figure 4(d) as his model of Mary. Using the NID learning algorithm described in the previous section, John should be able to learn Mary's model over time, predict her moves and subsequently win the match. Following the procedure described in the previous section, John solves the NID in Figure 4(d), before the match ensues, to end up with the MAID in figure 5.

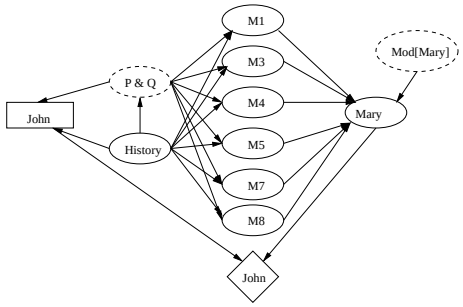


Figure 5: RoShamBo top-level

Nodes that are outlined with a dotted line represent variables with missing CPTs. In order to learn Mary's model, we attempted to estimate the parameters for P, Q and Mod[$Mary$], by using the on-line estimation techniques while playing the game. If our model is correct, then we should converge on parameters values for Mod[$Mary$] that correspond to the meta deliberation employed by Simple Modeler. Our empirical methodology consisted of running ten matches of 3,000 rounds each.

However, our NID model was only occasionally able to beat Mary's Simple Modeler program. Specifically, it lost five matches out of the ten, and did not exhibit a consistent learning curve for those that it won. Moreover, we discovered that while parameter values for P were being learned by the NID model, the initial parameter values chosen for Mod[$Mary$] and for Q did not change. In other words, we could not learn the distribution over Mary's strategies nor over John's predictive algorithm.

The reason for this is that the learning algorithm's preference bias for a "simple" explanation folded all of John's uncertainty over Mary's behavior into the predictor P. Because of this preference, the distribution over John's predictor Q and the meta distribution over Mary's strategies Mod[$MAry$] were not being learned.

In other words, our model assumed that Mary was always playing according to her predictor. Therefore, our current NID model could not learn meta-strategies that change over

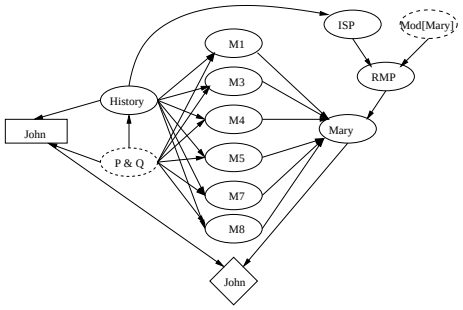


Figure 6: RoShamBo top-level 2

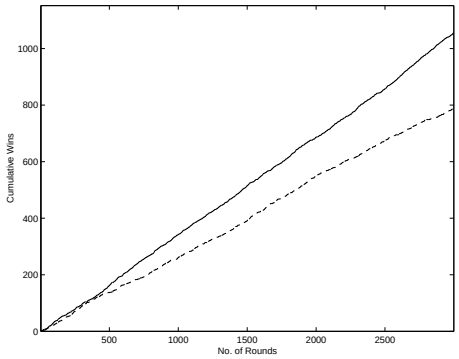


Figure 7: Typical Run

time !

In order to allow our learning algorithm to learn non-stationary strategies, we increased the inductive bias of our model by making the following assumptions regarding Mary's play.

- John believes that Mary can either be playing *pattern* or some meta-strategy.
- If Mary is playing pattern, she is merely an automaton that follows her predictor.
- If Mary is playing a meta-strategy, she is playing one of the strategies $M_1, M_3, M_4, M_5, M_7, M_8$ according to the value of Mod[$Mary$].
- John believes that Mary believes that John always plays the pattern dictated by the predictor Q.
- At each round, John examines the prior moves incorporated in the *history* node. If that particular history has not been seen before, John believes that Mary's current move was pattern. Otherwise, John believes Mary to be playing some meta-strategy with some decaying probability that depends on the number of occurrences of the history.

We incorporate our assumptions into our model by adding to new nodes RMP and ISP to the top-level block. The boolean variable ISP represents John's belief over whether or not Mary is playing according to pattern. In our experiments, its CPT assigned an arbitrary probability of 0.7 to True and 0.3 to False. The variable RMP is a multiplexor node. It equals $M1$ if ISP is True, i.e. Mary is playing her pattern. (see top-level block in Figure **??**. Otherwise, it is equal to one of the blocks $\{M2, \ldots, M6\}$, according the strategy that is attributed by the value of $ModMary$.

Using the model described in Figure 6, John was able to learn a viable model of Mary's behavior. We ran ten different games of 3,000 rounds each. In all of them, the NID model used by John outperformed Simple Modeler that was used by Mary. A sample run is shown in Figure 7, where the solid and dashed line correspond to John and Mary's performance respectively.

Note that John's lead continues to grow as more rounds are played. This behavior was consistent in all of the games we ran, and imply that the accuracy of John's model is increasing as more observations are collected. John's NID model never took more than 560 turns to catch up with Mary's Simple Modeler, and finished every match by a lead of over 300 games.

Recall that the program Simple Modeler meta-deliberates between choosing two strategies — a best response to her prediction of John, and a best-response to John's best-response to his prediction of Mary. In our model, these predictors correspond to the nodes P and Q. Indeed, even though John's hypothesis space consisted of six possible meta-strategies, examining the CPT for the node Mod[$Mary$] after each game was over confirmed that a correct model was learned. This was because the probabilities that were assigned to the appropriate two entries corresponding to Mary's meta-deliberation were significantly higher than the other CPTs. This behavior was consistent in all of the games we ran.

5. DISCUSSION AND RELATED WORK

A crucial difference between our NID model and all of the RoShamBo entries is that our NID model can only directly observe the last two rounds of the game, whereas the other RoShamBo entries had access to the entire history of the match. Since Iocaine Powder was not only a good meta-deliberator, but also a sophisticated "context sniffer", performing statistical analysis on the entire match history, our NID model could not compete with it. The reason for this is that the number of parameters to learn in our NID model is exponential in the size of the observed history, so we could not expand the window of our learner to account for the complete history. Still, our NID model could beat the other entries in the tournament, even though its history window was limited.

NIDs share a close relationship with the classic game theoretic formalism of *Bayesian games* [13]. Any Bayesian game can be reduced to a NID, and computing the Nash equilibrium [14] of the Bayesian game reduces to the same strategies that are obtained by solving the NID. In a separate paper [8], we show that in fact, NIDs are a superset of Bayesian games. In this paper, we show that NIDs are better suited to knowledge representation than Bayesian games, providing representations that are more compact, descriptive and explicit, thereby making them preferable for AI systems.

Hu and Welman [12] present a model for learning about other agents in a simulated double auction market. In their model, some agents do not model others, some employ a standard linear regression for modeling others, and some model others who model others using linear regression. Their

model showed that on average, learning agents perform better than agents that do not use information about others. NIDs allow agents to express uncertainty over the rationality of their opponents in a controlled manner. If, in a particular game, no model of an agent's model is provided, we invoke a rationality assumption for that agent. If, on the other hand, one agent's model specifies the behavior of another agent, the first agent chooses a best response. The key point is that our language allows for the agents to have models of each other just as much as is desired.

In work by Suryadi and Gmytrasiewicz [3], each agent embodies the decision process of all other agents together in a separate ID, where decisions of other agents are modeled as chance variables and given a prior CPT. A neural network is then used for updating CPTs of variables in an agent's model from observations. In this model, the representation of agents' model is limited to a single influence diagram for each agent. The approach can be viewed as a special case of the one in this paper, together with an algorithm for agents to update the model parameters.

There is plenty to be done for future work. First, we are constructing algorithms for computing the Nash equilibrium of cyclic NIDs that take advantage of the compactness of Bayesian networks the graph as well as the relationships that hold between the NID blocks. Second, we are currently working on applying NIDs to learning models of behavior in an interesting negotiation game, in which agents might be boundedly rational. The game includes both human and automatic players, and exhibits a high degree of uncertainty over game mechanics as well as the reasoning processes of the players. We feel that this type of domain would be an ideal test bed for NIDs.

Acknowledgments

This work was supported by NSF Career Award #IIS-0091815.

6. REFERENCES

[1] D. Billings. The first international RoShamBo programming competition. *International Computer Games Association Journal*, 23(1), March 2000.

[2] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron. Improved opponent modeling in poker. In *International Conference on Artificial Intelligence*, 2000.

[3] D.Suryadi and P.J. Gmytrasiewicz. Learning models of other agents using influence diagrams. In *Proc. Seventh International Conference on User Modeling (UM-99)*, pages 223–232, 1999.

[4] D. Egnor. Iocaine Powder. *International Computer Games Association Journal*, 23(1), March 2000.

[5] F.V.Jensen F.Jensen and S. L Dittmer. From influence diagrams to junction trees. In *UAI*, pages 367–373, 1994.

[6] D. Fudenberg and D. K. Levine. *The Theory of Learning in Games*. MIT Press, 1998.

[7] Y. Gal and A. Pfeffer. A language for modeling agents' decision making processes in games. In *AAMAS*, 2003. To appear.

[8] Y. Gal and A. Pfeffer. Modeling agents' beliefs using networks of influence diagrams. In *TARK IX*, 2003. Submitted.

[9] D. Geiger, T. Verma, and J. Pearl. Identifying independence in Bayesian Networks. *Networks*, 20:507–534, 1990.

[10] D. Heckerman. A tutorial on learning with Bayesian Networks. Technical report, Microsoft Research, 1996.

[11] R.A. Howard and J.E. Matheson. Influence diagrams. In *Readings on the Principles and Applications of Decision Analysis*, pages 721–762, 1984.

[12] J. Hu and M.P. Wellman. Learning about other agents in a dynamic multiagent system. *Cognitive Systems Research*, 2:67–79, 2001.

[13] J.C.Harsanyi. Games with incomplete information played by 'Bayesian' players. *Management Science*, 14:159–182,320–334,486–502, 1967-68.

[14] J.Nash. Equilibrium points in n-person games. *Proc. National Academy of Sciences of the USA*, 36:48–49.

[15] D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. In *IJCAI*, 2001.

[16] R.M. Neal and G. Hinton. *Learning in graphical models*, chapter A view of the EM algorithm that justifies incrememntal, sparse and other variants, pages 355–368. MIT Press, 1999.

[17] R. D. Shachter. Evaluating influence diagrams. *Operations Research*, 34:871–882, 1986.