Fitting and Compilation of Multiagent Models through Piecewise Linear Functions

David V. Pynadath and Stacy C. Marsella
USC Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292
{pynadath,marsella}@isi.edu

Abstract

Decision-theoretic models have become increasingly popular as a basis for solving agent and multiagent problems, due to their ability to quantify the complex uncertainty and preferences that pervade most nontrivial domains. However, this quantitative nature also complicates the problem of constructing models that accurately represent an existing agent or multiagent system, leading to the common question, "Where do the numbers come from?" In this work, we present a method for exploiting knowledge about the qualitative structure of a problem domain to automatically derive the correct quantitative values that would generate an observed pattern of agent behavior. In particular, we propose the use of piecewise linear functions to represent probability distributions and utility functions with a structure that we can then exploit to more efficiently compute value functions. More importantly, we have designed algorithms that can (for example) take a sequence of actions and automatically generate a reward function that would generate that behavior within our agent model. This algorithm allows us to efficiently fit an agent or multiagent model to observed behavior. We illustrate the application of this framework with examples in multiagent modeling and social simulation, using decision-theoretic models drawn from the alphabet soup of existing research (e.g., MDPs, POMDPs, Dec-POMDPs, Com-MTDPs).

1. Introduction

Decision-theoretic models have become increasingly popular in agent and multiagent domains. Markov Decision Processes (MDPs) [9] and Partially Observable MDPs (POMDPs) [6] provide powerful frameworks for representing and solving single-agent planning problems. Multiagent researchers have begun extending these frameworks to address problems of coordination and teamwork [1, 4, 10]. These frameworks can quantify the expected performance of policies of agent behavior, and their ability to find provably optimal policies provides a natural basis for modeling a rational agent.

However, the application of these frameworks to real-world problems is often difficult, not least because of their prohibitive computational complexity [1, 8, 10]. Furthermore, the quantitative nature of these decision-theoretic frameworks also complicates the problem of constructing the models in the first place. The large parameter space encompassed by an MDP-based model's specification of the world dynamics, observability, preferences, etc. leads to the common question, "Where do the numbers come from?"

We can begin to address these difficulties by exploiting the structure that exists in typical agent domains. Our hypothesis in this work is that the use of piecewise linear functions to represent probability distributions and utility functions provides a valuable structure, without sacrificing much generality. We demonstrate the application of this framework in representing domain examples taken from multiagent modeling and social simulation.

The linearity properties of our representation can potentially form the basis for a suite of algorithms that can exploit this linearity in solving (multi)agent problems. This paper presents two such algorithms that we have developed and implemented. The first addresses the complexity of evaluating the expected reward derived by a particular policy of behavior. While this computational cost is prohibitive in general, the piecewise linear structure of our agent model supports an algorithm that can compile a value function into a decision tree for rapid policy evaluation.

As a second demonstration of this framework, we have also implemented algorithms that can take a sequence of actions and automatically generate a reward function that would generate that behavior within our agent model. This algorithm allows us to efficiently fit an agent or multiagent model to observed behavior. Our method exploits knowledge about the qualitative structure (in the form of our piecewise linearity) of a problem domain to automatically derive the correct quantitative values that would generate an observed pattern of agent behavior.

Section 2 presents the relevant details of the MDP-based frameworks that motivate this work, as well as two illustrative domains that we use as running examples throughout this paper. Section 3 uses these two examples to present our decision tree representation of piecewise linear functions for use in MDP-based frameworks. Section 4 presents an algorithm that exploits this representation to compile an agent model for the efficient evaluation of policy values. Section 5 presents an algorithm that exploits this representation to fit an agent model's reward function to observed behavior. Section 6 concludes with some discussion of the implications of our methodology and opportunities for future applications and extensions.

2. Problem Statement

Our methodology has applications for many decision-theoretic (multi)agent frameworks, starting with MDPs [9], which we denote as tuples, $\langle S, A, P, R \rangle$, where S is the state space, A the action space, P the transition probability function, and R the reward function. The Partially Observable MDP (POMDP) [6, 11] extends the MDP framework to include a function, O, that represents the distribution over the agent's possible observations. Thus, each agent maintains a *belief state*, B, that summarizes its observations so far.

The Decentralized POMDP (Dec-POMDP) [1] can represent multiple agents in a partially observable environment, where each agent makes independent observations and decisions, but whose actions have potentially dependent effects on the environment. The Communicative Multiagent Team Decision Problem (Com-MTDP) [10] extends the Dec-POMDP to explicitly model communication and the belief state space of the agents. We use the Com-MTDP's illustrative domain from this previous work as an example throughout this paper. This domain models the flight of two helicopters through enemy territory. The uncertainty about the enemy's location, as well as the partial observability, leads to critical tradeoffs between the cost of exchanging messages and the urgency in reaching their destination.

A social simulation using a Com-MTDP-based multiagent system provides our other example. The agents represent different people and groups in a classroom, with a teacher analyzing their simulated behavior to understand the causes and cures for school violence. One agent represents a bully, and another represents the student who is the typical victim of the bully's violence. A third agent represents the group of onlookers, who encourage the bully's exploits by, for example, laughing at the victim as he is beaten up. A final agent represents the class's teacher trying to maintain control of the classroom by doling out punishment in response to the violence. Although each of these individuals has its own goals, we model each agent's decision process as a single-agent problem, where it assumes that all of the other agents are following a fixed policy of behavior.

The application of these various frameworks to realistic problems poses a variety of challenges. Regarding the single-agent case, although the problems of finding optimal policies for MDPs and POMDPs are in P and PSPACE, respectively [8], their complexity is polynomial in the size of the state space, which is in turn exponential in the number of state features. The cost of the multiagent case is even more prohibitive [1, 10], where the problem of finding optimal policies is in NEXP and is thus provably non-polynomial.

Additional difficulties arise in trying to apply MDP-based frameworks to the problem of modeling agents. The large parameter space required by the quantitative specification of the components of these models places a large burden on the model designer. For example, in the school violence simulation domain, the typical user will be a teacher who wishes to prevent the bully from picking on his victim. For example, some bullies may pick on the victim to gain the positive reaction from their classmates, in which case punishing the entire class may be an effective policy. On the other hand, if the bully picks on the victim to lash out at *all* of his classmates, then such a policy may only encourage his violence. Thus, it would be valuable for a teacher to identify the type of bully, which, in the context of our decision-theoretic model, corresponds to identifying the bully's reward function. Unfortunately, the complex interdependencies of an MDP-based agent model make it difficult to predict how modifying a particular parameter will affect the resulting behavior.

It would greatly simplify the simulation setup task if the system could automatically construct a fully parameterized model of the bully based on readily available input, such as observations of his behavior so far. This problem parallels the larger research area of plan recognition. A detailed survey of the field is beyond the scope of this paper, but current plan recognition techniques are unable to provide an automatic mechanism for modeling an MDP-based agent. Recent work in machine learning has provided promising results on learning reward functions from observed behavior of MDP-based agents [2, 7]. However, these approaches require additional knowledge (e.g., basis functions, distribution over reward functions) that may not be readily available and that would be difficult to understand and manipulate by someone who is not an AI expert. We instead would like an algorithm that accomplishes analogous tasks, but does so using structures that are more transparent to a novice user.

3. Piecewise Linear Models

Therefore, there is a need for novel techniques that can exploit structured problem domains to more efficiently solve the modeling and planning tasks in decision-theoretic frameworks. We can draw inspiration for possible solutions from some related work in the literature. Work on learning probabilistic networks has exploited the linearity of the posterior distribution of interest in answering queries to automatically adjust numerical weights to match the desired output [3]. The approach relies on an ability to express a probabilistic query response as a linear combination of the values of nodes in the belief network. In addition, existing multiagent work has gained computation leverage by approximating an agent's value function as a linear combination of basis functions [5]. Linearity is clearly a handy property that one can exploit in many ways in solving agent problems.

Unfortunately, nontrivial agent domains present nonlinearities that prevent us from directly applying such techniques. For example, in the Com-MTDP helicopter domain, the agents receive a positive reward only upon reaching their destination; they receive zero reward while en route. In the school violence simulation, we *can* model the bully's reward as being proportional to the encouragement received from his fellow classmates and negatively proportional to the amount of punishment received from the teacher. However, the *dynamics* are nonlinear, as the policy of the teacher is discontinuous in its specification of punishment upon observing an act of violence and no punishment otherwise.

We propose modeling agent-based systems with *piecewise* linear functions, represented as probabilistic decision trees whose leaf nodes are matrices with constant weights. Each decision tree partitions the state space, with the leaf nodes containing the matrix relevant to the corresponding subset of states. The decision tree branches represent separation by hyperplanes, so we are able to represent nonlinearities while still maintaining some desirable linear properties. The rest of this section describes and illustrates the mechanism of this representation as it pertains to the common features of an MDP-based model specification.

3.1. System State Vectors

We start, as do most decision-theoretic models (such as MDPs, POMDPs, Dec-POMDPs, Com-MTDPs), with a state space, S, whose elements represent the possible states of an agent's world. It is often convenient to use a factored representation, with each element of the state space being a vector, $\vec{s}$, representing a combination of separate state features. For example, in the helicopter Com-MTDP, the state vector contained three elements: the state of the escort helicopter, transport helicopter, and the enemy radar. In the school violence simulation, the state contains one element for each entity, representing its relative "power" in the scenario. We assume that we can represent such states as real-valued column vectors.

However, the state of the world does not completely capture the state of the overall agent-based system, in that it is not always sufficient for determining behavior and future states. A common definition of the reward function, R, as defined in MDP-based models, is a mapping from both state *and action* to a real value, which indicates that the reward received by the agent(s) depends on the action(s) performed in the time step. Thus, we must extend our state vector to include slots representing those actions.

For example, in the school violence simulation, the bully derives a reward every time his classmates laugh along with his antics. Therefore, there is a slot in the system state indicating the presence (and possibly degree) of such encouragement. Likewise, in the helicopter Com-MTDP, there is a communication cost incurred every time an agent sends a message. We must therefore include a slot that indicates the number of messages sent.

For the completely observable, single-agent MDP, the world state and action are sufficient for determining the behavior of the agent. However, in the partially observable systems represented by POMDPs and Com-MTDPs, an agent receives only an indirect observation of the true world state and instead bases its policy decisions on its *belief state*, B. For example, in the helicopter Com-MTDP, the belief state of the transport helicopter consists of one bit of information: has the enemy been destroyed or not? The belief state of its escort also consists of one bit: does the transport believe that the enemy is destroyed or not?

We denote the vector, $\vec{q}$, to represent the overall state of a system of k agents. Thus, $\vec{q} = [\vec{s}, \vec{a}_1, \ldots, \vec{a}_k, \vec{b}_1, \ldots, \vec{b}_k, 1]$, representing the world state, action selections, and belief states, respectively. The final element is a constant factor that provides for more flexible manipulation of the system state vector, as we show in the following sections. Furthermore, given the typical nondeterminism in agent domains, we are likely to have a probability distribution over system states, $\{\langle \vec{q}, \Pr(\vec{q}) \rangle\}$, rather than a single vector.

In the Com-MTDP helicopter domain, each system state vector, $\vec{q}$, accumulates the world state, action indicators, and belief states as already described, resulting in a vector of seven elements: $[$*escort*, *transport*, *enemy*, *messages*, *transport-belief*, *escort-belief*, $1]$.

In the social simulation model of School violence, the state vector includes the "power" levels of the entities in the scenario, but it also includes boolean components indicating the actions of those entities. In particular, one component is 1 if and only if the bully has attacked the victim in the previous epoch. Another component is 1 if and only if the onlookers laughed in response to such an attack. Regarding the teacher's action, one component is 1 if and only if the teacher has punished the bully. The resulting system state vector has seven elements:

$[$*bully-power*, *victim-power*, *teacher-power*, *bully-violence*, *onlookers-laugh*, *teacher-punish*, $1]$.

Notice that we can quickly recover important expectations from distributions over such state vectors. For example, to compute the expected number of acts of violence in the school violence example, we would simply compute the sum: $E[\textit{bully-violence}] = \sum_{\vec{q}} \Pr(\vec{q})\vec{q}[4]$ where $\vec{q}[i]$ indicates the i^{th} element of the vector. We can perform an analogous summation to compute the expected number of messages sent in the helicopter domain.

3.2. Piecewise Linear Reward Functions

The reward function represents the preferences that an agent has over states of the world and actions that change those states. MDP-based models capture such preferences as a deterministic function that returns a real value for each combination of state of the world and the most recent actions performed by the agent(s). In some domains, the reward function is a weighted sum of the values of the state features. In such cases, we can model the reward function as a row vector containing the weights, which we can then multiply by our system state vector to return the reward received in that state. Unfortunately, most agent domains do not possess such linear reward functions.

Our hypothesis is that many agent domains have reward functions that are linear in a *piecewise* fashion. In other words, we model the reward as a decision tree partitioning the state space, with the leaf nodes representing the weighted combination applicable in that subset of states. We partition the space with a set of hyperplanes, where we define each plane, p_i, as a set of weights, $\vec{w}_i \equiv w_{i1}, w_{i2}, \ldots, w_{in}$, and a threshold, θ_i. We can then define a decision tree branch, D_i, as a function over state vectors:

$$D_i(\vec{q}) = \begin{cases} D_{iL}(\vec{q}) & \text{if } \vec{w}_i \cdot \vec{q} < \theta_i \\ D_{iR}(\vec{q}) & \text{if } \vec{w}_i \cdot \vec{q} \geq \theta_i \end{cases} \quad (1)$$

The definition is recursive, so that if the state, $\vec{q}$, is to the left (right) of the dividing hyperplane, we follow the corresponding decision tree branch, D_{iL} (D_{iR}). Leaf nodes return a unique value, rather than another node.

For a reward function, the leaf node value is a row vector, which we can then multiply (dot product) with a state vector to compute a real value. In other words, if D_R represents the reward function and $\vec{q}$ is the current state of the system, then $D_R(\vec{q})$ returns the row vector that represents the reward weights for the relevant portion of the state space. Thus, $D_R(\vec{q}) \cdot \vec{q}$ represents the current reward received by the agent(s). To simplify notation, we will write this expression as simply $D_R \cdot \vec{q}$ when there is no confusion.

If an agent's reward is proportional to the components of our system state vector, then the reward decision tree, D_R, is a single leaf node. For example, in the school violence simulation, the bully's reward is proportional to his power, negatively proportional to the power of the victim and teacher, and proportional to the encouragement (e.g., laughter) elicited from his onlooking classmates. In other words, D_R returns a row vector, $[r_1, r_2, r_3, 0, r_5, 0, 0]$, for some constant weights, $r_1 > 0, r_2 < 0, r_3 < 0$, and $r_5 > 0$.

Although most reward structures are not so linear, we can represent many of them within our decision tree structure. We can easily represent achievement goals with such a reward function structure. For example, in the helicopter Com-MTDP, the agents receive a positive reward when they reach the destination, but no reward while they are in transit. Thus, the reward's decision tree must first branch on whether each helicopter has reached the destination. The transport's corresponding hyperplane, p, has weights, $\vec{w} = [0, 1, 0, 0, 0, 0, 0]$ (i.e., a weight of 1 for the transport's position), and threshold, $\theta = d$, where d is the position of the destination. The left branch results in a leaf node with only the communication cost as a reward ($[0, 0, 0, r_4, 0, 0, 0]$), while the right results in a leaf node with the communication cost and the successful arrival reward ($[0, 0, 0, r_4, 0, 0, r_7]$), for constants $r_4 < 0$ and $r_7 > 0$.

3.3. Piecewise Linear Dynamics

Agents typically base their decisions on some sort of planning process that examines the possible future outcomes of their actions. To represent the dynamics of the state of the world, decision-theoretic models typically use a transition probability function, P, that maps a state and an action into a distribution over possible new states. We represent this transition probability function as a *set* of decision trees, one for each action, with each decision tree returning a transformational matrix that represents the dynamics in the relevant portion of the state space. In other words, we define a decision tree, $D_{\vec{a}}$, to represent the dynamics of a combined action, $\vec{a}$. We apply these dynamics in a similar fashion to the reward function, so that if $\vec{q}$ is the system state at time t, then $D_{\vec{a}}(\vec{q}) \times \vec{q}$ (again denoted in shorthand as simply $D_{\vec{a}} \times \vec{q}$) produces a system state that represents the world state and action indicators at time $t + 1$. Section 3.4 discusses the projection of future belief states.

In the school violence simulation, we model acts of violence (or punishment) as generally decreasing the power of the object of that action and increasing the power of the actor. However, the effects change if the actor is less powerful than the object of his/her violence. For example, in the decision tree representing the dynamics when the bully beats up his victim, there is a top-level branching hyperplane with weights $[1, -1, 0, 0, 0, 0, 0]$ and threshold $\varepsilon > 0$. To the right of that branch, the bully is at least marginally stronger than his victim, and he inflicts damage proportional to his advantage, leading to the 7×7 matrix in Table 1.

The first row produces an increase in the bully's power, and the second a decrease in the victim's. The third row

$$\begin{bmatrix} 1.1 & -.1 & 0 & 0 & 0 & 0 & 0 \\ -.2 & 1.2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Table 1. Dynamics matrix for bully's act of violence toward his victim, when bully is the more powerful.

leaves the teacher's power unchanged. The fourth row activates the indicator for *bully-violence*, while the next two rows leave the indicators for *onlookers-laugh* and *teacher-punish* off. The last row simply maintains the constant factor in the state vector. The matrix for the branch when the bully is not more powerful than his victim is similar in structure, except that the numbers in the upper left 2×2 submatrix change so that the bully gets the worst of the fight.

In the helicopter Com-MTDP, the transport has two options for action: flying at a normal altitude and flying nap-of-the-earth (NOE), which is slower but safer. The dynamics decision tree for flying NOE branches when the helicopter reaches the destination, which we model as a hyperplane with weights $\vec{w} = [0, 1, 0, 0, 0, 0, 0]$ and threshold, $\theta = d$, where d is the destination. The right-hand side is a leaf node that returns a matrix whose second row (corresponding to the transport's position) is $[0, 1, 0, 0, 0, 0, 0]$, so that the transport's position remains unchanged upon reaching the destination. The left-hand side branches a second time on whether the helicopter has already been destroyed (denoted by a negative state value). In particular, we have a hyperplane with the same weights as above ($[0, 1, 0, 0, 0, 0, 0]$), but with a threshold of 0. The left-hand side again returns a matrix whose second row is $[0, 1, 0, 0, 0, 0, 0]$, so that the transport's state remains unchanged upon being shot down. The right-hand side returns a 7×7 matrix whose second row is $[0, 1, 0, 0, 0, 0, 0.5]$. In other words, if the transport has neither reached the destination nor been destroyed, it moves forward by 0.5. The dynamics for flying at a normal altitude are more complex, because there is the additional risk of being detected by the enemy radar. Thus, there is an additional branch testing whether the radar has already been destroyed (again denoted by a negative state value).

In a multiagent domain, the dynamics of the agents' actions have a cumulative effect on the state of the world. In the two example domains we present here, we can model the dynamics of the individual agents' actions as independent. Thus, we can define separate decision trees for each agent's actions. The decision tree representing the dynamics of a combined action of the set of agents is then the sum of the individual decision trees (Section 4.1 presents the algorithm for computing such a sum). If the effect of the agents' actions are not independent then we must define the decision trees for each combined action directly.

In addition, the dynamics of an agent's world is sometimes nondeterministic. We can model such nondeterminism by introducing chance nodes into our decision tree structures, where each such chance node represents a probabilistic branch according to some fixed distribution. However, for the purposes of the algorithms in Sections 4 and 5, we assume that the decision trees have only one probabilistic branch and that it is the top node of the tree. We can rewrite any tree with internal probabilistic branches as having only a single such branch, but the number of branches for that single top node will be exponential in the number of internal probabilistic nodes.

3.4. Piecewise Linear Belief Update

In partially observable domains, the dynamics of our system state vectors must also include the dynamics of the agents' belief states. For example, in the helicopter domain, when the escort helicopter destroys the enemy radar, the transport helicopter observes the destruction with some fixed probability, ρ_O. Similarly, in an alternate formulation of the school violence simulation, the teacher may not always observe the fight between the bully and the victim.

Each agent updates its belief state based on the observation it receives (as well as any messages sent, in the Com-MTDP framework). For example, in the helicopter domain, the transport believes the enemy radar has been destroyed if it either observes the fact or if it receives such a message from its escort. We can thus model the belief update as a decision tree with a top-level probabilistic branch, with both branches leading to 7×7 matrices whose entries are all 0, except for the fifth row, which represents *transport-belief*. With probability ρ_O, we follow the left branch, where the fifth row of the matrix is $[0, 0, 0, 0, 0, 0, 1]$, meaning that the transport now believes the radar to be destroyed. With probability $1 - \rho_O$, we follow the right branch, where the fifth row of the matrix is $[0, 0, 0, 1, 1, 0, 0]$, meaning that the transport definitely believes the radar to be destroyed if it receives a message to that effect; otherwise, it maintains its current level of belief.

The dynamics of the system depend on the actions chosen by the agents. Following the convention of POMDP-based frameworks, agents follow a policy, $\pi : B \rightarrow A$, which maps a belief state into an action. We can model such policies as a decision tree, where the "leaves" are actually the root nodes of an action dynamics decision tree. For example, the transport helicopter's policy is to fly NOE when it does not believe the enemy to be destroyed and to fly at its normal altitude otherwise. This condition translates into a hyperplane branch with weights $[0,0,0,0,1,0,0]$ and threshold 1. The left branch leads to the decision tree model of the dynamics for flying normally, while the left branch leads to the decision tree model of the flying NOE dynamics. In a multiagent scenario, we can create a decision tree, $D_{\vec{\pi}}$, to represent the dynamics of the agents' joint policy, $\vec{\pi}$, by adding up the decision trees corresponding to the individual agent's policies.

4. Compilation of Policy Evaluation

Once we have a piecewise linear model of our domain, we can exploit its structure to design novel algorithms that more efficiently compute the expected value of agent policies. Section 4.1 describes the basic computational manipulations that allow us to combine multiple piecewise linear functions. Section 4.2 describes algorithms for pruning these functions so as to maximize efficiency. Section 4.3 then applies these capabilities to create an algorithm for more efficient forward projection of agent behavior.

4.1. Decision Tree Arithmetic

We have already described the method for performing a dot product of a decision tree and a state vector. To perform lookahead, we must sum rewards over a finite horizon. To do so, we can follow Algorithm 1, which simply adds the branches of one tree to all of the leaf nodes to the other, and then adds the values on the leaf nodes together.

Algorithm 1 $D_1 + D_2$

```
1: if $D_1$ is leaf then
2:     if $D_2$ is leaf then
3:         return $D_1 + D_2$
4:     else
5:         return $\langle p_2, D_1 + D_{2L}, D_1 + D_{2R} \rangle$
6: else
7:     return $\langle p_1, D_{1L} + D_2, D_{1R} + D_2 \rangle$
```

We must also compute products of decision trees. For example, we can represent the state dynamics over two time steps as the square of our dynamics decision tree twice. Algorithm 2 computes the product of two decision trees. It is nearly identical to Algorithm 1, but there is the added wrinkle that multiplication modifies the branches of the first term by scaling the state vector, so we must scale the weights on our hyperplane accordingly.

4.2. Pruning Decision Trees

Addition and multiplication return decision trees with many more branches than their input terms. In fact, the number of branches will increase exponentially with each operation. Fortunately, we can prune away many of the resulting branches as being either redundant or contradictory. For example, if we add a decision tree, D_1, with one branching hyperplane, p_1, and a decision tree, D_2, with one plane, p_2, then Algorithms 1 and 2 will return a decision tree with four leaf nodes, representing all combinations of branches along the two planes. However, suppose $p_2 \leq p_1$ (i.e., $\forall \vec{q}$, if $\vec{w}_1 \cdot \vec{q} > \theta_1$ then $\vec{w}_2 \cdot \vec{q} > \theta_2$). Then the leaf node corresponding to the combination $\vec{q} > p_1 \wedge \vec{q} < p_2$ is impossible, and the branch at that point is irrelevant. It is also possible that the arithmetic operation produces branches that are completely identical. This commonly occurs with probabilistic branches where the operation has marginalized out one of the probabilistic conditions. We can simply remove the branch, thus merging the two equivalent branches together. We have implemented a pruning function that prunes the results of Algorithms 1 and 2 as a post-processing step. Although, this pruning has no effect in the worst case, we have found it to be quite effective in practice.

Algorithm 2 $D_1 \times D_2$

```
1: if $D_2$ is leaf then
2:     if $D_1$ is leaf then
3:         return $D_1 \times D_2$
4:     else
5:         return $\langle p_1 \times D_2, D_{1L} + D_2, D_{1R} + D_2 \rangle$
6: else
7:     return $\langle p_2, D_1 \times D_{2L}, D_1 \times D_{2R} \rangle$
```

4.3. Forward Projection

Now that we have algorithms capable of combining our decision trees, we can examine the possibility of compiling our domain-level specification into efficient representations of a value function. We can define the value of a particular policy, $\vec{\pi}$, over a finite horizon, T, as follows:

$$V_{\vec{\pi}}^T = E\left[\sum_{t=0}^{T} R^t\right] = \sum_{\vec{q}} \Pr(\vec{q}) \sum_{t=0}^{T} \left(D_R \times D_{\pi}^t\right) \cdot \vec{q} \quad (2)$$

By the properties of our decision tree formulation and the dot product, we can distribute the multiplication and addition as follows:

$$V_{\vec{\pi}}^T = \sum_{\vec{q}} \Pr(\vec{q}) \left[\sum_{t=0}^{T} \left(D_R \times D_{\pi}^t\right)\right] \cdot \vec{q} \quad (3)$$

The summation within the brackets produces a decision tree whose output is the cumulative reward. Significantly, it does not depend on the current system state, $\vec{q}$, so we can therefore pre-compute it once. We thus effectively compile the lookahead into a single decision tree, denoted D_V, which we can then apply to different system states to compute the expected value the agent(s) will achieve.

For example, in the original Com-MTDP work, the expected value of a policy was computed by repeatedly performing lookahead from each of the possible starting states. We have applied Equation 3 to the candidate policies to produce decision trees with only eight leaf nodes (corresponding to the possible positions of the enemy radar). We can then take our input prior probability distribution over the possible enemy locations and compute the expected value of a given policy by invoking its decision tree.

5. Fitting Agent Models

Recent work on belief networks has developed algorithms for automatically modifying the network parameters to achieve a desired query result [3]. The basis for these algorithms is the insight that one can rewrite the query result as a linear combination of the parameters to be adjusted. While we cannot rewrite MDP-based agent behavior as a linear combination of the parameters, our decision-tree representation *is* linear at the leaf nodes. Therefore, it may be possible to exploit piecewise linearity in our agent models to develop automatic behavior fitting algorithms, analogous to those for probabilistic networks.

We have developed one such algorithm, motivated by a problem in our school violence simulation domain, where a teacher wishes to identify the type of bully (i.e., his reward function). In our model of the school violence scenario, we assume that the bully derives a reward that is a weighted sum of the relative power levels of himself, his victim, and his teacher, as well as the amount of laughter elicited from his classmates. To simplify the presentation, we first present the fitting algorithm for such cases, where the reward decision tree, D_R, is a leaf node. We then present the extension to handle the more general case.

If D_R is a leaf node, then we can redistribute the multiplication and addition of Equation 3 as follows:

$$V_{\vec{\pi}}^T = \sum_{\vec{q}} \Pr(\vec{q}) D_R \cdot \left(\left[\sum_{t=0}^{T} D_{\pi}^t \right] \times \vec{q} \right) \tag{4}$$

If we assume that we know a distribution over the possible system states, $\vec{q}$, then we can evaluate everything but D_R:

$$V_{\vec{\pi}}^T = \sum_{\vec{q}} D_R \cdot \vec{w}_{\pi}, \text{ where } \vec{w}_{\pi} = \Pr(\vec{q}) \left[\sum_{t=0}^{T} D_{\pi}^t \right] \times \vec{q} \tag{5}$$

Our goal is to find a D_R such that the observed policy of behavior (e.g., of the bully) has a higher value than all of the other candidate policies. For the purposes of illustration, we begin with a model of the bully as having two such policies of action: committing violence against his victim (π_v) or not doing so ($\pi_{\neg v}$). If we observe the bully selecting the latter, we have the following constraint:

$$\sum_{\vec{q}} D_R \cdot (\vec{w}_{\pi_{\neg v}} - \vec{w}_{\pi_v}) \geq 0 \tag{6}$$

If we denote the elements of the vectors, D_R, $\vec{w}_{\pi_{\neg v}}$, and $\vec{w}_{\pi_v}$ as r_i, $w_{i\neg v}$, and w_{iv}, respectively:

$$\sum_i r_i \sum_{\vec{q}} (w_{i\neg v} - w_i) \geq 0 \tag{7}$$

There are obviously many reward weights that satisfy this constraint, and a variety of methods can select a good solution. For example, existing work uses a prior distribution over reward functions to find the most likely solution [2].

However, we take a different approach, one that is analogous to that taken in tuning probabilistic networks [3]. We assume that we start with an initial reward function that we wish to modify in some minimal way to match the observed behavior. Within our social simulation tool, we have default models of the possible scenario individuals, and one such model provides an appropriate starting reward function for a classroom bully. If this reward function already matches the observed behavior, then no change is necessary.

If, on the other hand, the current reward function violates the constraint in Equation 7, then we must modify the reward weights accordingly. We have constructed a set of heuristics aimed at finding a reasonable selection of changes that would correct the error in reward. For example, the fitting algorithm does not consider modifying a reward weight so much that its sign changes. In other words, we assume that the current reward structure is correct *qualitatively*, so that if we believe that a particular feature is being maximized by the agent, we do not ever modify our model so that the agent is minimizing it, and vice versa.

Our algorithm first considers modifying only a single reward weight. For each current weight, r_i, we can instantly compute a correcting change, Δr_i, such that substituting $r_i + \Delta r_i$ into the original reward function will generate the observed behavior:

$$\Delta r_i = \frac{\left| \sum_i r_i \sum_{\vec{q}} (w_{i\neg v} - w_i) \right|}{\sum_{\vec{q}} (w_{i\neg v} - w_i)} \tag{8}$$

Our first heuristic modifies the r_i with the minimal Δr_i, subject to the constraint forbidding a change of sign of r_i. If no such r_i exists, then we can consider changing combinations of two weights. The algorithm for the case of a D_R with branches follows a similar derivation. However, the number of weights available increases with the number of branches, as we could potentially adjust weights along any of the leaf nodes of D_R.

One can easily apply a distance metric on the reward weights to find a more rigorously defined “closest” reward function. However, we have implemented the current set of heuristics so as to more naturally interact with a human user (e.g., a teacher) in tuning the multiagent model. Thus, our algorithm can also present users with the possible weight changes and allow them to select the ones that seem most appropriate. It is instructive to notice that we can also use Equation 7 to analyze the sensitivity of the agent’s policy to the specific weights on the reward function. This additional property of our algorithm is not surprising, given that the analogous algorithm for tuning probabilistic networks also supported such sensitivity analysis [3].

6. Discussion

We have a presented a novel representational framework for expressing the quantitative specification of decision-theoretic agent models as a set of piecewise linear functions in the form of probability distributions over decision trees. We have successfully encoded two example domains within this representation, and we believe that our framework supports the encoding of many realistic domains. Furthermore, common usage of piecewise linear functions as an approximation in other areas encourages us to think that the representation will be useful even in domains that do not exactly match its structural assumptions.

The decision tree representation also allows extension into the underlying agent model itself. For example, MDP-based models typically assume a deterministic reward function and agent policies, but there is no such limitation on our representation itself. Therefore, there is potential for extending our algorithms to handle cases that lie beyond the typical usage of these models.

The incentive behind modeling an agent domain in our framework comes in the form of the novel algorithms that one can then apply to common problems. We have provided two illustrative examples of such algorithms that provide efficient mechanisms for compiling policy evaluation and for fitting reward functions to observed behavior. These algorithms have proved useful in our two example domains, but we believe a few simple extension will make them even more powerful. For example, our pruning algorithm is rather coarse, in that it makes very safe changes to the decision tree. However, we can easily apply the many approaches in the decision tree literature for minimizing tree complexity (e.g., balancing). Furthermore, our approach to manipulating probabilities within these trees is through brute-force enumeration. Cleverer manipulations will provide even more improvement in the efficiency of our decision trees and their manipulation.

Beyond the two algorithms we present in this work, we believe that there is a wide range of possible algorithms that can exploit the specific structure of our piecewise linear framework. Both algorithms exploit the linearity property in the same way: they isolate a single feature (state in the case of compilation, reward in fitting) and then combine all of the other features into a single decision tree. By isolating different features in the same way, we can potentially generate additional algorithms to solve the corresponding agent problems. In addition, as suggested in Section 5, these algorithms can, at the same time, provide a sensitivity analysis over the parameter space. Thus, we believe that the structured framework of our piecewise linear models will form the basis for a valuable suite of algorithms that will expand the applicability of decision-theoretic agent methodologies to real-world domains.

Acknowledgments This work was supported in part by the Office of the Assistant Secretary of Defense (SO/LIC).

References

[1] Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman. The complexity of decentralized control of Markov decision processes. In *Proceedings of the Conference on Uncertainty in Artifi cial Intelligence*, pp. 32–37, 2000.

[2] Urszula Chajewska, Daphne Koller, and Dirk Ormoneit. Learning an agent’s utility function by observing behavior. In *Proceedings of the International Conference on Machine Learning*, pp. 35–42, 2001.

[3] Hei Chan and Adnan Darwiche. When do numbers really matter? *Journal of Artifi cial Intelligence Research*, 17:265–287, 2002.

[4] Claudia V. Goldman and Shlomo Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 137–144, 2003.

[5] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems*, pp. 1523–1530, 2001.

[6] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artifi cial Intelligence*, 101:99–134, 1998.

[7] Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 663–670, 2000.

[8] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operation Research*, 12(3):441–450, Aug 1987.

[9] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[10] David V. Pynadath and Milind Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artifi cial Intelligence Research*, 16:389–423, 2002.

[11] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a fi nite horizon. *Operations Research*, 21:1071–1088, 1973.