History-Based Controller Design and Optimization for Partially Observable MDPs

Akshat Kumar
School of Information Systems
Singapore Management University
akshatkumar@smu.edu.sg

Shlomo Zilberstein
School of Computer Science
University of Massachusetts Amherst
shlomo@cs.umass.edu

Abstract

Partially observable MDPs provide an elegant framework for sequential decision making. Finite-state controllers (FSCs) are often used to represent policies for infinite-horizon problems as they offer a compact representation, simple-to-execute plans, and adjustable tradeoff between computational complexity and policy size. We develop novel connections between optimizing FSCs for POMDPs and the dual linear program for MDPs. Building on that, we present a dual mixed integer linear program (MIP) for optimizing FSCs. To assign well-defined meaning to FSC nodes as well as aid in policy search, we show how to associate history-based features with each FSC node. Using this representation, we address another challenging problem, that of iteratively deciding which nodes to add to FSC to get a better policy. Using an efficient off-the-shelf MIP solver, we show that this new approach can find compact near-optimal FSCs for several large benchmark domains, and is competitive with previous best approaches.

1 Introduction

Partially observable Markov decision processes (POMDPs) provide an elegant framework for sequential decision making in partially observable settings (Sondik 1971). The past decade has seen significant improvement in the applicability and scalability of POMDP solvers. In particular, *point-based* methods that restrict value function computations to a subset of the belief space have contributed to rapid improvement of *value iteration* techniques (Smith and Simmons 2004; Spaan and Vlassis 2005; Pineau, Gordon, and Thrun 2006; Shani, Brafman, and Shimony 2007; Poupart, Kim, and Kim 2011; Shani, Pineau, and Kaplow 2013). There has also been significant improvement in *policy iteration* techniques that use finite-state controller (FSC) to represent solutions (Hansen 1998). Finite-state controllers are particularly useful because executing a FSC-based policy requires a simple table lookup without any belief updates. Such resource-effective execution is desirable in many settings, such as battery-constrained mobile and wearable device applications (Grześ, Poupart, and Hoey 2013a; 2013b). In addition, controllers can provide significantly more semantic information about the policy than alpha-vectors used by value iteration approaches. This is particularly important when human understanding of the policy is required, for example in assistive healthcare applications of POMDPs (Hoey et al. 2012).

Optimizing controllers for POMDPs is challenging (Vlassis, Littman, and Barber 2012). Several approaches have been developed to optimize *stochastic* controllers, such as bounded policy iteration (Poupart and Boutilier 2003), formulations based on nonlinear programming (Amato, Bernstein, and Zilberstein 2007; 2010; Charlin, Poupart, and Shioda 2007), stochastic local search (Braziunas and Boutilier 2004) and expectation-maximization (Toussaint, Harmeling, and Storkey 2006; Pajarinen and Peltonen 2011). There has been recent progress with branch-and-bound search methods for finding optimal deterministic controllers (Meuleau et al. 1999; Grześ, Poupart, and Hoey 2013b).

In our work, we develop a mixed integer linear programming (MIP) formulation for optimizing deterministic controllers. This MIP formulation is heavily influenced by the dual LP formulation of MDPs (Puterman 1994), underscoring that optimization techniques developed for MDPs can be adapted to POMDPs. The MIP formulation is advantageous over nonlinear programming (NLP) based formulations (Amato, Bernstein, and Zilberstein 2007; 2010; Charlin, Poupart, and Shioda 2007) as most off-the-shelf MIP solvers provide an upper bound that can be used to check the optimality gap, in contrast to NLP solvers that typically do not provide quality bounds. Given enough time, MIP solvers can provide an optimal fixed-size controller, which may not be possible with non-convex programming solvers, that could get stuck in local optima. Indeed, we show empirically that our approach finds near-optimal fixed-size controllers for commonly used benchmark instances.

The importance of MIP based formulations for obtaining quality bounded solutions has been recognized by the planning under uncertainty and partial observability community. For example, a MIP based approach to obtain optimal policy for *finite-horizon* decentralized POMDPs (Dec-POMDPs) has been developed in (Aras, Dutech, and Charpillet 2007; Aras and Dutech 2010). This approach works by incorporating both the world state and all the possible observation histories for every time step to form an extended state-space, and then defining a MIP over this extended space. A similar idea of using all possible observation histories for *finite-horizon* POMDPs has been discussed in (Witwicki 2011, Chapter 5). The main disadvantage of such approaches is that the size of the observation history increases exponentially with the horizon, severely impacting the scalability of the MIP. Furthermore, such observation-history based non-stationary policy does not address the infinite-horizon case. In contrast, our approach is based on optimizing a FSC-based stationary policy for infinite-horizon POMDPs. Furthermore, the size of the MIP in our case is polynomial in the FSC size and is therefore far more scalable than previous MIP approaches.

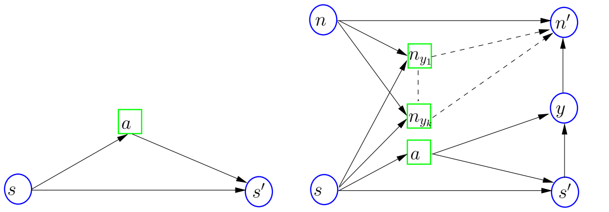
In previous work on optimizing controllers (Meuleau et al. 1999; Amato, Bernstein, and Zilberstein 2010; Grześ, Poupart, and Hoey 2013b), it is often hard to determine the "right" size of the controller in terms of the number of nodes. Larger controllers are preferable as they better approximate the optimal policy, but optimizing them is more challenging. Hence, we develop techniques that are synergistic with our dual MIP formulation to find a good quality controller via an iterative process. We start with a fixed-size controller, for example a reactive controller with one node per available observation. After optimizing this controller in iteration zero, our approach uses the information-theoretic measure of entropy, readily available from the dual MIP solution, to determine how to add nodes to this controller, and again optimizes the larger controller to improve the quality of the solution in an iterative fashion. To further aid the optimization engine in the policy search as well as provide useful semantic interpretation for the resulting policy, we associate each controller node with well-defined aspects of the observation history. For example, such aspects can include associating each node with the last observation received, or the last action and observation. The node addition strategy is designed to preserve the semantic structure of the controller while expanding its size.

We test our approach on a number of large POMDP domains. Using an efficient MIP solver (CPLEX), we show that our approach can find compact optimal or near-optimal FSCs for commonly-used large benchmarks, and is highly competitive with the best existing controller optimization approaches.

2 The POMDP Model

A POMDP is described using a tuple $\langle S, A, T, Y, O, R, \gamma\rangle$. The set S is the set of states s, A is the set of actions a, Y is the set of observations y. The transition function is defined as $T(s', s, a) = \Pr(s'|s, a)$ and the observation function as $O(y, a, s') = \Pr(y|a, s')$, where s' is the resulting state when the last action taken was a. The function $R(s, a)$ defines the immediate reward received when action a is taken in state s, and $0 < \gamma < 1$ is the reward discounting factor. We consider the infinite-horizon setting in which decision making unfolds over an infinite sequence of stages. In each stage, the agent selects an action, which yields an immediate reward, and receives an observation that depends on the outcome. The agent must choose actions based on history of observations it obtains. The objective of the agent is to maximize the expected discounted sum of rewards received.

Finite-state controllers (FSC) provide an elegant way of representing POMDP policies using finite memory. A FSC is parameterized as $\langle N, \phi, \psi\rangle$, where N is a set of nodes n with each node labeled with an action. Edges represent controller transitions and are labeled with observations. The action mapping $\phi : N \to A$ assigns an action to each FSC node. The node mapping $\psi : N \times Y \to N$ maps the current node and observation pair to a new node. Executing a policy represented by a given controller is straightforward. The agent first executes the action $\phi(n)$ associated with the current node n. It then receives an observation y and transitions to the resulting node $\psi(n, y)$. Such table-lookup based plan execution is ideal for resource-constrained devices, such as mobile phones, because no computationally expensive operations are required at execution time. The expected value of executing a controller starting in node n and state s according to policy $\pi = \langle\phi, \psi\rangle$ is:



(a) MDP DBN (b) POMDP DBN

Figure 1: Dynamic Bayesian Networks (DBNs) for (PO)MDPs.

$$\forall n, s:\ V(n, s; \pi) = R(s, \phi(n)) + \gamma \sum_{s',y} T(s', s, \phi(n))O(y, a, s')V(\psi(n, y), s'; \pi)$$

We assume without loss of generality that the controller starts in node n_1. So, the value of a controller given an initial belief b_0 is $V(b_0; \pi) = \sum_s b_0(s)V(n_1, s; \pi)$. Our goal is to find an optimal policy $\pi^\star$ such that $V(b_0; \pi^\star)$ is maximized. It has been shown that optimizing fixed-size deterministic controllers is NP-Hard (Vlassis, Littman, and Barber 2012). Even optimizing a reactive controller with one node per observation is NP-Hard (Littman 1994).

3 The Dual MIP for POMDPs

Our dual MIP formulation for optimizing controllers is based on the dual LP formulation for optimizing MDP policies. Therefore, we first describe relevant aspects of dual LP for MDPs.

The linear program for finding an optimal MDP policy is often represented as follows (Puterman 1994):

$$\max_{\{x(\cdot,\cdot)\}} \sum_s \sum_a R(s, a)x(s, a) \tag{1}$$

$$\sum_a x(j, a) - \sum_s \sum_a \gamma\, P(j|s, a)x(s, a) = b_0(j)\ \forall j \in S$$

where the variable $x(s, a)$ intuitively denotes the total discounted amount of time the environment state is s and action a is taken. Therefore, the total reward corresponding to being in state s and taking action a is $R(s, a)x(s, a)$. This describes the objective function of the LP formulation. The constraints represent the flow conservation principle.

Definition 1. *The variable $x(s,a)$ is defined as follows (Puterman 1994):*

$$x(s,a) = \sum_{j \in S} b_0(j) \sum_{t=1}^{\infty} \gamma^{t-1} P(s_t = s, a_t = a \mid s_1 = j) \qquad (2)$$

3.1 Cross-Product MDP for a POMDP

Our goal is to develop a mathematical programming formulation analogous to the LP formulation of MDPs to optimize FSCs. We can interpret the POMDP as an MDP defined over states of the form (s,n), where s is a world state and n is a state of the FSC, as also shown in (Meuleau et al. 1999). The LP formulation for this cross-product MDP will not necessarily result in policies that are executable in a partially observable setting because action may depend in part on the world state. To remedy this, our key contribution is to introduce constraints in the LP formulation of this cross-product MDP such that the resulting policy depends only on the information available to the agent at runtime. This results in transforming the *dual LP* for MDPs to the *dual MIP* for POMDPs. Our strategy is different from previous nonconvex and nonlinear programming formulations of FSC optimization (Charlin, Poupart, and Shioda 2007; Amato, Bernstein, and Zilberstein 2010), which are analogous to the *primal* LP formulation of MDPs, directly optimizing $V(n,s)$, the value of starting a controller in node n and state s using NLP solvers. In contrast, our approach works with *occupancy distributions* $x(\cdot)$, similar to the ones introduced in Def. 1.

We first define the cross-product MDP for a POMDP as shown in Fig. 1(b), similar to (Meuleau et al. 1999). The state of this MDP is the joint controller state and environment state $(n,s) \in N \times S$. The action space has two components. First, action $a \in A$ of the agent is part of the action space. In addition, the controller transition function ψ is also part of the action space. This is required as we not only need to find the action mapping ϕ, but also the node transition function ψ. We achieve this by creating a new random variable n_{y_i} for each observation y_i of the agent. The domain of each random variable n_{y_i} is N or the set of all possible next nodes reachable upon receiving observation y_i. In n_{y_i}, the observation y_i is not a variable, rather the variable is which node the control transitions to given y_i.

The joint action for the underlying cross-product MDP is $\langle a, n_{y_1}, \ldots, n_{y_k} \rangle$, assuming there are $k = |Y|$ observations. The middle layer of the POMDP DBN in Fig. 1(b) shows the joint action variables. Each action variable has two parents, one for each joint-state component. For clarity, we use following notations:

$\langle n_{y_1}, ..., n_{y_k} \rangle \equiv \langle n_y \rangle$ and $\langle ..., n_{y_i} = n', ... \rangle \equiv \langle ..., n'_{y_i}, ... \rangle$

The transition function for this cross-product MDP is:

$$P(n', s' \mid n, s, a, \langle n_y \rangle) = \sum_{y_r \in Y} P(n' \mid n, \langle n_y \rangle, y_r) \\ O(y_r, a, s') T(s', s, a) \qquad (3)$$

where distribution $P(n' \mid n, \langle n_y \rangle, y_r)$ is defined as:

$$P(n' \mid n, \langle n_y \rangle, y_r) = \begin{cases} 1 & \text{if } n_{y_r} = n' \\ 0 & \text{otherwise} \end{cases} \qquad (4)$$

Intuitively, the above distributions encode the fact that a complete assignment to variables $\langle n_y \rangle$ defines a deterministic controller transition function ψ. Similar to the MDP linear program, we create occupancy distribution variables $x(n, s, a, n_{y_1}, \ldots, n_{y_k})$ by noting that the state in the cross-product MDP is (n,s) and the action is $\langle a, n_{y_1}, \ldots, n_{y_k} \rangle$. To simplify the notation, we denote this distribution compactly as $x(n, s, a, \langle n_y \rangle)$. Notice that the total number of these variables is exponential in the size of the observation space. Fortunately, we can exploit the independence structure present in the DBN of Fig. 1(b) to reason with the marginalized version of these variables as shown next.

3.2 Mixed-Integer Linear Program for POMDPs

We are now ready to write the dual mathematical program for the cross-product MDP shown in Fig. 1 and develop additional constraints that guarantee that the resulting policy is valid in partially observable setting. This program is presented in table 1. The binary variables in (24) represent the controller policy $\pi = \langle \phi, \psi \rangle$. We next provide the justification for constraints in table 1. First, we formulate the flow conservation constraint analogous to Eq. (2). The *incoming* flow into the state (n', s') is defined as:

$$b_0(n', s') + \gamma \sum_{n,s,a,\langle n_y \rangle} P(n', s' | n, s, a, \langle n_y \rangle) x(n, s, a, \langle n_y \rangle)$$

Using the transition function definition in Eq. (3), we represent the incoming flow as follows:

$$b_0(n', s') + \gamma \sum_{n,s,a,\langle n_y \rangle} \sum_{y_r \in Y} P(n' \mid n, \langle n_y \rangle, y_r) \\ O(y_r, a, s') T(s', s, a) x(n, s, a, \langle n_y \rangle)$$

From the definition of the transition distribution (Eq. (4)), n_{y_r} must be equal to n', otherwise the transition function is zero. Using this, we simplify the incoming flow as follows:

$$b_0(n', s') + \gamma \sum_{n,s,a,y_r,\langle n_{y \setminus y_r} \rangle} O(y_r, a, s') T(s', s, a) x(n, s, a, n'_{y_r}, \langle n_{y \setminus y_r} \rangle)$$

We can marginalize $x(n, s, a, n'_{y_r}, \langle n_{y \setminus y_r} \rangle)$ to get:

$$x(n, s, a, n'_{y_r}) = \sum_{\langle n_{y \setminus y_r} \rangle} x(n, s, a, n'_{y_r}, \langle n_{y \setminus y_r} \rangle) \qquad (5)$$

Using this relation, we further simplify the incoming flow:

$$b_0(n', s') + \gamma \sum_{n,s,a,y_r} O(y_r, a, s') T(s', s, a) x(n, s, a, n'_{y_r}) \qquad (6)$$

The above equation represents the total incoming flow as shown on the RHS of constraint (15) of the POMDP MIP. The LHS of (15) is simply the total discounted amount of time the controller is in joint-state $\langle n', s' \rangle$. Hence, constraint (15) is the analogue of the MDP flow constraint (2). We define the variables $x(\cdot)$ for POMDPs as follows.

Definition 2. *The variable $x(n, s, a, \langle n_y \rangle)$ is defined as:*

$$\sum_{j \in S, k \in N} b_0(j,k) \sum_{t=1}^{\infty} \gamma^{t-1} P_t(n, s, a, \langle n_y \rangle \mid s_1 = j, n_1 = k)$$

We use the convention that $P(u_t = u, v_t = v|\cdot)$ can be concisely represented as $P_t(u, v|\cdot)$, where u_t and v_t denote random variables for the time slice t and u, v are particular assignments to these random variables; $\cdot$ represents conditioning on the appropriate variables; subscripts denote time. Analogously, we can define the marginalized variable $x(n, s, a, n'_{y_i})$ as follows.

Definition 3. *The variable $x(n, s, a, n'_{y_i})$ is defined as:*

$$\sum_{j \in S, k \in N} b_0(j,k) \sum_{t=1}^{\infty} \gamma^{t-1} P_t\big(n, s, a, n_{y_i} = n' \,|\, s_1 = j, n_1 = k\big)$$

Definition 3 is a simple consequence of marginalizing every probability in Definition 2. We can solve the cross-product MDP just with constraint (15). But this would not lead to a valid policy as action selection depends on the underlying world state, which is not observable. To remedy this, we introduce additional constraints (16) to (24) that result in a proper policy for a POMDP. We explore these constraints below.

Theorem 1. *The MIP constraint* (20) *along with the integrality constraints of* (24) *guarantee a valid action mapping in a partially observable setting. That is,*

$$x(n, a^\star) = x(n) \quad \textit{if}\, x(a^\star|n) = 1 \quad \forall n \in N \tag{7}$$

$$x(n, a) = 0 \qquad \forall a \in A \backslash \{a^\star\} \;\; \forall n \in N \tag{8}$$

Proof. We know from the MIP constraints that:

$$x(n) = \sum_a x(n, a) \;\forall n \tag{9}$$

$$x(n) \geq x(n, a) \;\forall a \tag{10}$$

As the variable $x(a|n)$ is an integer, we consider a particular action $a^\star$ such that $x(a^\star|n) = 1$. The rest of the action selection variables will be zero because $\sum_a x(a|n) = 1$. Substituting $x(a^\star|n) = 1$ in constraint (20), we get:

$$x(n) - x(n, a^\star) \leq 0 \tag{11}$$

$$x(n) \leq x(n, a^\star) \tag{12}$$

From inequalities (10) and (12), we have $x(n) = x(n, a^\star)$. That is, whenever the controller is in state n, it executes the action $a^\star$ corresponding to the variable $x(a^\star|n) = 1$. This is exactly how a policy in a partially observable environment should look like. The action selection ϕ depends only on the controller state and not on the world state.

From the constraint $x(n) = \sum_a x(n, a)$ we can see clearly that if $x(n) = x(n, a^\star)$, then the rest of the $x(n, a)$ variables must be zero. Using this fact about $x(n, a)$ variables for all actions a with $x(a|n) = 0$, we can simplify constraint (20) for such actions a as:

$$x(n) \leq \frac{1}{1-\gamma} \tag{25}$$

This inequality also holds because the maximum value the variable $x(n)$ can have is $\frac{1}{1-\gamma}$. Thus, constraint (20) holds for every action $a \in A$. Therefore, constraint (20) succinctly encodes the partial observability constraint into the cross-product MDP. □

Variables: $x(n, s, a),\; x(n, s, a, n'_y), x(n, a), x(n), x(n, n'_y),$

$$x(a|n), x(n'|n, y) \;\forall\; n, s, a, y, n' \tag{13}$$

$$\text{Maximize: } \sum_{n,s} \sum_a R(s, a) x(n, s, a) \tag{14}$$

Subject to:

$$\sum_a x(n', s', a) = b_0(n', s') + \gamma \sum_{n,s} \sum_{a,y} O(y, a, s') T(s', s, a) x(n, s, a, n_y = n') \;\forall (n', s') \tag{15}$$

$$x(n, s, a) = \sum_{n'} x(n, s, a, n'_y) \;\forall (n, s, a, y) \tag{16}$$

$$x(n, a) = \sum_s x(n, s, a) \;\forall (n, a) \tag{17}$$

$$x(n) = \sum_a x(n, a) \;\forall n \tag{18}$$

$$x(n, n'_y) = \sum_{s,a} x(n, s, a, n'_y) \;\forall (n, y, n') \tag{19}$$

$$x(n) - x(n, a) \leq \frac{1 - x(a|n)}{1-\gamma} \;\forall (n, a) \tag{20}$$

$$x(n) - x(n, n'_y) \leq \frac{1 - x(n'|n, y)}{1-\gamma} \;\forall (n, y, n') \tag{21}$$

$$\sum_a x(a|n) = 1 \;\forall n \tag{22}$$

$$\sum_{n'} x(n'|n, y) = 1 \;\forall (n, y) \tag{23}$$

$$x(a|n) \in \{0, 1\}, x(n'|n, y) \in \{0, 1\} \tag{24}$$

Table 1: Mixed-integer program for optimizing a single agent POMDP policy. The 0-1 binary variables are shown in Eq. (24). The rest are continuous, positive variables: $x(\cdot) \geq 0$. The policy is denoted using binary variables $x(a|n)$ and $x(n'|n, y)$.

Theorem 2. *The MIP constraint* (21) *along with the integrality constraints of* (24) *guarantee a valid node mapping in a partially observable setting. That is,*

$$x(n, n^\star_y) = x(n) \quad \textit{if}\, x(n^\star|n, y) = 1, \;\forall y \in Y, \forall n \in N \tag{26}$$

$$x(n, n'_y) = 0 \qquad \forall n' \in N \backslash n^\star, \;\forall y \in Y, \forall n \in N \tag{27}$$

Proof. We proceed by interlinking constraints (16) to (19). We know from constraint (19) that

$$x(n, n'_y) = \sum_{s,a} x(n, s, a, n'_y) \tag{28}$$

$$\sum_{n'} x(n, n'_y) = \sum_{n',s,a} x(n, s, a, n'_y) \tag{29}$$

We know from constraint (16) that

$$x(n, s, a) = \sum_{n'} x(n, s, a, n'_y).$$

Substituting this result into equation (29), we get:

$$\sum_{n'} x(n, n'_y) = \sum_{s,a} x(n, s, a) \tag{30}$$

Using constraints (17) and (18) for the above equation:

$$\sum_{n'} x(n, n'_y) = x(n) \;\forall y, n \tag{31}$$

Using the above relation, we have the inequality that:

$$x(n) \geq x(n, n'_y) \;\forall n', y, n \tag{32}$$

Setting the variable $x(n^\star|n, y) = 1$ in constraint (21) we get:

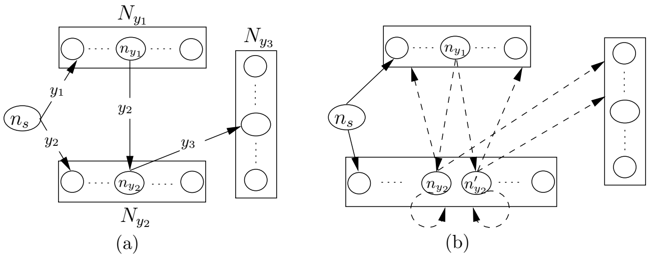
$$x(n) - x(n, n^\star_y) \leq 0 \tag{33}$$

Combining equations (32) and (33), we get $x(n) = x(n, n^\star_y)$. Using equation (31), we deduce that $x(n, n'_y) = 0 \;\forall n' \in N \backslash n^\star$. This completes the proof. □

Complexity The total number of variables in the dual MIP is $O(|N|^2|S||A||Y|)$. The total number of constraints is $O(|N||S||A||Y| + |N|^2|Y|)$. The number of binary variables is $O(|N||A| + |N|^2|Y|)$. The number of binary variables, which greatly influences the complexity of solving MIPs, corresponds only to the policy functions ϕ and ψ. No other auxiliary variable is binary. Therefore, our MIP for POMDP is an efficient encoding to optimize FSCs containing minimal number of binary variables.

Practical Issues Despite the advantages of the MIP formulation, a naive implementation that tries to optimize controllers of increasing size using this MIP may not scale well. Finding optimal fixed-size controllers remains a challenging NP-Hard problem. First, the complexity of solving MIP is influenced by the number of continuous and integer variables, both increase quadratically with the controller size $|N|$. Therefore, large controller sizes and the overall problem representation (state, action and observation space) will affect the scalability of MIP solvers. The second barrier to scalability is somewhat subtle. Semantically, each controller node *summarizes* some relevant aspect of the observation history. Therefore, while optimizing a fixed-size controller, the underlying optimization engine must decide what information should be memorized by each controller node as well as what actions should be taken. Optimizing simultaneously the variables that govern these two separate decisions is what makes the optimization problem so hard. One way to mitigate this is by assigning some nodes of the controller fixed actions (Amato, Bernstein, and Zilberstein 2010), but that only pays off under certain conditions

We develop a more disciplined approach to address this challenge, taking a cue from the concept of *feature selection* in machine learning, which improves model interpretability and enhances algorithmic performance (Guyon and Elisseeff 2003). In our case, the main intuition is to associate some specific interpretation (or feature) with each controller node. For example, one can associate each node with the last observation received or the last action and observation. Such interpretations help keep the size of the MIP small and aid the optimization engine. Earlier efforts to associate some prescribed meaning with controller nodes have shown good promise (Amato and Zilberstein 2008), but what space of possible features should be considered? And how could we explore it automatically in an efficient manner? In the next section we propose a novel approach to these questions.



(a) (b)

Figure 2: Part (a) shows an example of a HBC with three observations. Part (b) shows the warm start strategy with dotted lines corresponding to variables to be optimized.

4 History-Based Controller Design

We introduce the notion of *history-based* controllers (HBCs), associating clear history-related properties with each controller node. Such association offers a useful semantic meaning to controller nodes, and it helps keep the MIP size small. We also propose an efficient way to explore the space of HBCs and determine the desired controller size, striking a good balance between solution quality and computational tractability. We start with defining a HBC.

Definition 4. *A history-based controller (HBC) consists of a set of nodes $\{n_s\} \cup \mathcal{N}$, where n_s denotes a unique start node. The set $\mathcal{N}$ is partitioned according to observations: $\mathcal{N} = \cup_{y \in Y} N_y$, where N_y represents the set of nodes that encode the information that the last observation received was y. The action and node mappings are defined as:*

$\phi : \{n_s\} \cup \mathcal{N} \rightarrow A$, *and* $\psi_{y'} : (\{n_s\} \cup \mathcal{N}) \times \{y'\} \rightarrow N_{y'} \;\forall y' \in Y$

Intuitively, the node mapping $\psi_{y'}$ for a HBC needs to take into account the last received observation y' to determine the target set $N_{y'}$. Figure 2(a) shows an example of a HBC with three observations. All the incoming edges to each node in a set N_{y_i} are labeled with the corresponding observation y_i.

Proposition 1. *Finding optimal fixed-size HBC is NP-Hard.*

The proof of this proposition is straightforward by noting that a *reactive* controller is a special case of a HBC with $|N_y| = 1 \;\forall y$, and optimizing a reactive controller is NP-Hard (Littman 1994). In addition to associating the last observation received with a node, other possible types of HBCs may be based on remembering the last action and observation or remembering the last k observations. For ease of exposition and consistency, we focus primarily on HBCs based on the last observation received.

A key advantage of a HBC is the immediate reduction in the computational complexity of solving the dual MIP. In an arbitrary FSC, the total number of *binary* variables for node mapping is $O(|N|^2|Y|)$, whereas for a HBC, node mapping requires only $O(m|\mathcal{N}||Y|)$, where m denotes the largest size of any set N_y. In fact, we show empirically that even for large POMDP benchmarks, the maximum size of any set $|N_y| \leq 2 \ll |\mathcal{N}|$ provides optimal or near-optimal solutions. Thus, a HBC results in a simpler MIP with markedly fewer binary variables due to its observation-based structure. Furthermore, a key issue in performing branch-and-bound search over generic cyclic controllers is the need to include additional checks for symmetry breaking (multiple controllers encoding the same policy) (Grześ, Poupart, and Hoey 2013b). For HBCs, the observation based underlying structure reduces such symmetry in the search space, and without the need for additional checks, our MIP based approach can provide good solutions.

As highlighted previously, optimizing large size controllers in a single attempt may not be a scalable strategy for any branch-and-bound solver. Thus, there is a need to grow the controller in size by a deliberate addition of nodes. Therefore, we employ an iterative strategy, where we first optimize a default fixed-size controller, such as a reactive controller. We then employ the following node addition strategy that identifies, using a rigorous heuristic based on the notion of entropy, how to best add nodes to the controller that are likely to significantly increase solution quality.

Entropy-Based Node Splitting The importance of adding nodes to a fixed-size controller to yield a better policy has been recognized in the POMDP community (Hansen 1997; Poupart and Boutilier 2003; Poupart, Lang, and Toussaint 2011). In Hansen (1997), the size of the controller may increase exponentially after each node addition step. The BPI approach of (Poupart and Boutilier 2003) can grow the controller, but the node addition strategy is not focused on the initial belief state. Furthermore, BPI optimizes a stochastic controller whereas our MIP formulation is for deterministic controllers. Poupart, Lang, and Toussaint (2011) propose a number of techniques to add nodes to a controller within the context of the EM algorithm (Toussaint, Harmeling, and Storkey 2006). They propose a forward search technique that is specific to the local optima of the EM algorithm and EM's particular update strategy. The node splitting strategy proposed in (Poupart, Lang, and Toussaint 2011), based on the idea of state-splitting while using EM for HMMs (Siddiqi, Gordon, and Moore 2007), seems to be the most generic strategy that does not depend on the underlying algorithm used to optimize the FSC. We propose a similar node splitting strategy. The key novelty in our work is the development of an entropy based heuristic that uses the readily available information from the dual MIP of Table 1 to identify the controller node to split. Our entropy based approach is computationally much cheaper than the approach of (Poupart, Lang, and Toussaint 2011) which evaluates every node split to finally choose the node split that leads to highest increase in quality. We show empirically that our approach provides similar increase in quality as the *bestSplit* approach of (Poupart, Lang, and Toussaint 2011), but significantly faster.

The key intuition in our approach is that we quantify the uncertainty about the world state associated with every HBC node, then split the node with the highest uncertainty. To measure the uncertainty, we use the $x(\cdot)$ variables computed by solving the dual MIP:

$$x(n,s) = \sum_a x(n,s,a) \text{ , and } x(s|n) = \frac{x(n,s)}{x(n)} \quad (34)$$

$$H(n) = -\sum_s x(s|n) \ln x(s|n) \quad (35)$$

Algorithm 1: Dual MIP Based Controller Optimization

```
$i \leftarrow 0$ // Initialize iteration number
$C^i \leftarrow$ optimizeReactiveController()
// Iteratively grow default controller
repeat
    Terminate $\leftarrow$ true
    $WH[n] \leftarrow$ computeWeighteEntropy($n$) $\forall n \in \mathcal{N}^i$
    sorted $\leftarrow$ descendingSort( $WH$[])
    // Split nodes in dec. order of entropy
    for $k = 0$ to $|\mathcal{N}|^i$ do
        Let $n_y \leftarrow$ sorted[$k$]
        Split $n_y$ to create single node $n'_y$
        $C^{i+1} \leftarrow$ optimizeWithWarmStart($C^i, n'_y$)
        if $n'_y$ = cloneOf($n_y$) then
            Discard $C^{i+1}$; continue for loop
        else
            Terminate $\leftarrow$ false; break for loop
    $i \leftarrow i + 1$
until Terminate = true
```

Note that $x(s|n)$ is simply the fraction of time the world is in state s *given* that the HBC's state is n. Thus, $x(s|n)$ can be regarded as a probability distribution. We next calculate the entropy, $H(n)$, of this distribution in Eq. (35). The lower the entropy, the more indicative the HBC node is of the world state; when a node n corresponds to a unique world state s, the entropy $H(n)$ is zero. Intuitively, the higher the entropy, the more likely we can benefit from splitting the node and re-optimizing the more refined policy.

Incidentally, splitting nodes based exclusively on the highest entropy $H(n)$ did not work that well. However, a closely related idea turned out to be quite effective. We observed that nodes n in which the HBC spends the most time should be given more weight. More precisely, we define the notion of *weighted entropy*: $WH(n) = x(n) \cdot H(n)$.

Empirically, we found that splitting the node with highest weighted entropy provided excellent empirical performance. Furthermore, the weighted entropies can be computed as a byproduct of the dual MIP with minimal extra computation. Thus, our node addition technique is highly efficient.

Re-optimizing the HBC after Node Splitting Once we identify a node n_y in HBC C, we split it to create another node n'_y with the same feature y. We can re-optimize the larger controller C' with extra node n'_y from scratch. However, this is computationally inefficient. Instead, we employ a *warm-start* strategy to optimize C'. Figure 2(a) and (b) illustrate this warm start strategy.

In Figure 2(a), assume that the node n_{y_2} is being split to create a node n'_{y_2}. Figure 2(b) shows the larger controller C' with the added node n'_{y_2}. The following variables in C' need to be re-optimized:

- The action mappings ϕ of both nodes n_{y_2} and n'_{y_2} as variables to be re-optimized
- Consider a node n_{y_1} in Figure 2(a) that had a transition to the node n_{y_2} in controller C. We make this particular

node mapping of the node n_{y_1} as variable with two possible choices n_{y_2} and n'_{y_2}. This is shown using outgoing dotted lines from the node n_{y_1} in Figure 2(b). Such a process is done for every node that had a transition to the node n_{y_2} being split

- The only remaining variables to be re-optimized are the node mappings for nodes n_{y_2} and n'_{y_2}. We make all the outgoing node mappings of nodes n_{y_2} and n'_{y_2} variables. This is shown as outgoing dotted edges from both the nodes n_{y_2} and n'_{y_2} in Figure 2(b).

Other than the above variables, all the rest of the policy variables are fixed in C' as in C. Thanks to the warm-start strategy, the new dual-MIP has far fewer integer variables. Thus, it can be optimized quickly even for large problems, using little time per iteration. We note that such a warm-start strategy may not provide the best controller after the addition of the node n'_{y_2}. Nonetheless, empirically we found that this strategy was quite effective in increasing the solution quality. Re-optimizing the expanded controller from scratch led to slightly better quality per iteration, but it was significantly more time consuming, which limited its usability.

Algorithm 1 summarizes our strategy for optimizing a POMDP controller. We start by optimizing a default controller in iteration zero (e.g., a reactive controller, line 2). We then attempt to increase the size of the default controller iteratively by adding one node at a time. Lines 7 to 14 show the logic behind adding a node. We start by attempting to add a node with the highest weighted entropy. The expanded controller C^{i+1} is re-optimized in line 10. We include a check to see if node n'_y is a clone with the same action mapping and same node mapping of the original node n_y after re-optimization. If so, then we discard the controller C^{i+1} as adding n'_y does not result in a better policy. We then proceed by trying to split the next node in descending order of weighted entropy. When it is not longer possible to add any node to the controller C^i, the algorithm terminates.

5 Experiments

We compare our dual MIP approach with the nonlinear programming (NLP) approach (Amato, Bernstein, and Zilberstein 2010) and the optimal branch-and-bound search IsoBNB (Grześ, Poupart, and Hoey 2013b) for optimizing FSCs. To put the results in perspective, we also provide the best solution quality achieved by HSVI (Smith and Simmons 2005), the state-of-the-art point-based solver. We used CPLEX 12.6 as the MIP solver on a 2.8GHz machine with 4GB RAM. We used a publicly available executable version of IsoBnB; for the NLP approach, we took the best known quality and timing results from (Amato, Bernstein, and Zilberstein 2010; Grześ, Poupart, and Hoey 2013b) that use the SNOPT solver on the NEOS server.

Both NLP and IsoBnB have multiple parameters that affect the final solution quality. This complicates the comparison. For example, for the NLP approach, controller size and the number of random restarts are input parameters. For the IsoBnB approach, controller size and the number of edges in controller are input parameters. For the problems tested in (Grześ, Poupart, and Hoey 2013b), we use the exact same setting to reproduce the earlier results. For other problems, we fixed the controller size to the number of actions for IsoBnB. The performance of IsoBnB is sensitive to the number of edges in the controller, so we varied that number from 5 to 30 (total of three settings) and also experimented with no edge restrictions. We used a time limit of 10,000 seconds for each IsoBnB run and report the best quality achieved and when over all four variants.

	dualMIP (fixed-size)			IsoBnB		NLP	
Problem	Qual.	Time	Opt.	Quality	Time	Quality	Time
lacasa1 $(\lvert S\rvert = 16$ $\lvert A\rvert = 2, \lvert O\rvert = 3)$ HSVI = 294.3	293.1	0.14	Y	294.0	38.2	293.8	1.76
lacasa3 $(\lvert S\rvert = 640$ $\lvert A\rvert = 5, \lvert O\rvert = 12)$ HSVI = 294.9	292.5	895	Y	292	340	*	*
lacasa3-ext $(\lvert S\rvert = 1920$ $\lvert A\rvert = 5, \lvert O\rvert = 3)$ HSVI = 295.6	285.0	73.5	Y	291	1325	*	*
underwaterNav$(\lvert S\rvert = 2653$ $\lvert A\rvert = 6, \lvert O\rvert = 102)$ HSVI = 749	747.7	56.2	Y	?	?	*	*
baseball$(\lvert S\rvert = 7681$ $\lvert A\rvert = 6, \lvert O\rvert = 9)$ HSVI = 0.64	0.64	17.2	Y	0.64	5224	*	*

Table 2: Quality and runtime (in sec.) comparisons for different controller optimization algorithms. The point-based solver HSVI is used for reference. A '*' denotes the algorithm was unable to run due to a large number of constraints; '?' denotes parsing error.

Our approach has relatively *fewer* parameters and ones that are *easier* to set. Importantly, the initial controller size is not a parameters in our approach. We use a default reactive controller in iteration zero. The maximum time limit for iteration zero is set to 900 seconds. Each subsequent iteration that grows the controller is limited to 350 seconds for re-optimizing the larger controller using a warm start. These time limits were derived from our initial experiments with some large POMDP instances.

Optimizing Fixed-Size Controllers Table 2 shows results for our approach 'dualMIP', the branch-and-bound solver 'IsoBnB' (Grześ, Poupart, and Hoey 2013b) and the NLP approach (Amato, Bernstein, and Zilberstein 2010). We compare the solution quality and runtime for fixed-size controllers. For dualMIP, a reactive controller was sufficient for all these instances. Notice that in Algorithm 1, the node addition strategy is unable to add any nodes to the reactive controller for these instance as all addition attempts lead to clones, which is detected in Algorithm 1. For 'IsoBnB', we used the same controller settings as in (Grześ, Poupart, and Hoey 2013b). Table 2 clearly shows that dualMIP is able to find provably optimal reactive controllers even for large POMDP instances[1] ('baseball' and 'underwaterNav'). The runtime of dualMIP also compares favorably with the IsoBnB approach. The NLP solver was unable to run due to the large size of these instances. Furthermore, the quality

[1] IsoBnB is not restricted to the class of reactive controllers, therefore it can sometimes achieve slightly better quality

	dualMIP			IsoBnB		NLP	
	Reactive	MaxEnt					
Problem	Qual.	Qual.	Time	Qual.	Time	Qual.	Time
tiger.95 $\big(\lvert S\rvert = 2$ $\lvert A\rvert = 3, \lvert O\rvert = 2\big)$ HSVI = 19.3	-20(opt)	19.3	0.44	19.3	1.4	-0.63	3.79
hallway $\big(\lvert S\rvert = 60$ $\lvert A\rvert = 5, \lvert O\rvert = 21\big)$ HSVI = 0.52	0.38(0.42)	0.46	3345	0.19	8170	0.47	362
hallway2$\big(\lvert S\rvert = 93$ $\lvert A\rvert = 5, \lvert O\rvert = 17\big)$ HSVI = 0.35	0.24(0.29)	0.28	5700	0.15	5616	0.28	420
machine$\big(\lvert S\rvert = 256$ $\lvert A\rvert = 4, \lvert O\rvert = 16\big)$ HSVI = 63	33.2(42.1)	62.54	950	62.6	41200	62.4	2640
tag$\big(\lvert S\rvert = 870$ $\lvert A\rvert = 5, \lvert O\rvert = 30\big)$ HSVI = -6.37	-17.2(-2.2)	-7.81	6000	?	?	-13.94	5596

Table 3: Quality and runtime (in sec.) comparisons for different controller optimization algorithms. 'Time' for 'MaxEnt' includes the time for optimizing the reactive controller.

achieved by dualMIP is very close to that of HSVI. It supports the case for HBCs showing that optimizing compact observation-based controllers can provide good quality even for large instances. The underlying structure in HBC also contributes to the faster runtime of the MIP solver.

In addition to reactive controllers, we also investigated other types of HBCs, such as controllers that memorize last k observations, and controllers that memorize last action taken and the last observation received. These types of controllers were highly effective for small and moderate sized problems. For '4x5x2.95' ($|S| = 39, |A| = 4, |O| = 4$), last action-observation based controller optimized by dualMIP achieved a quality of 2.01 in 0.64 sec. In contrast, IsoBnB achieved a quality of 2.02 in 625 sec, and NLP achieved a quality of 1.43 in 0.75 sec. These results again highlight that optimizing controllers with history based structure can provide high quality policies significantly faster than previous approaches.

Node Addition Using MaxEnt Table 3 highlights the gain provided by the entropy-based node addition approach ('MaxEnt'). For instances in this table, we optimized the reactive controller for a maximum of 900 seconds and then started the iterative node addition process. The column 'Reactive' shows the quality provided by the reactive controller, the number in brackets denote the upper bound provided by CPLEX. That is, for 'hallway', $.38(0.42)$ denotes the quality of reactive controller being .38, the upper bound on the optimal reactive controller being 0.42. As can be seen from this table, the 'MaxEnt' approach's deliberate addition of nodes was quite successful in significantly increasing the solution quality over a reactive controller. Furthermore, the final quality by 'MaxEnt' was very close to the quality provided by HSVI. The dualMIP approach also compares favorably w.r.t.quality with the NLP approach.

The runtime of dualMIP also compares favorably with the runtime of IsoBnB. For 'tiger' and 'machine', we use the same controller setting as in (Grześ, Poupart, and Hoey 2013b) without any time limit. For other instances, the best setting for IsoBnB is not known. Therefore, we used multiple settings for IsoBnB as highlighted earlier and report the best one w.r.t. quality. For most of these instances, IsoBnB was unable to provide a good solution quality within the time limit of 10,000 seconds.

Problem	tiger	hallway	hallway2	machine	tag
$\lvert Y\rvert + 1$	3	22	18	17	31
Size	10	36	35	20	56

Table 4: Final controller size, including both the size of the initial reactive controller (=$|Y| + 1$) and the number of nodes added by MaxEnt.

Table 4 shows the final controller size for dualMIP for different instances. Notice that the final size includes the size of the reactive controller ($= |Y| + 1$) and the subsequent nodes added using the MaxEnt approach. Other than the smaller problems such as 'tiger', the MaxEnt approach added less than 2 nodes per observation for a given instance. For example, for 'machine', the observation space is 16, and MaxEnt only added three more nodes in order to obtain good quality. This further supports that including the observation based structure in the controller can result in compact high-quality controllers.

6 Conclusion

Optimizing finite-state controllers for POMDPs is a challenging problem in probabilistic planning. We present a number of advances in this area. By exploiting the connection between an MDP and the cross-product MDP for a POMDP, we introduce a mixed-integer linear program for optimizing FSCs. Such dual MIP can be solved using efficient off-the-shelf MIP solvers. We make several additional contributions, namely, providing well defined semantics to FSC nodes derived from observations, and automatically determining the "right" controller size so as to obtain good solution quality with tractable computational complexity. We believe that such a *feature-based* planning approach holds significant promise for efficiently solving POMDPs by *helping* the optimization engine with the additional structure, as well as providing semantically meaningful plans.

Empirically, our approach worked quite well, providing solution quality on par with the best point-based solvers, yet using highly compact controllers. Our approach also scaled significantly better than previous FSC optimization methods such as nonlinear programming and provided better solution quality.

In future work, we plan to examine ways to extend this approach to different types of controllers (Amato, Bonet, and Zilberstein 2010) and to decentralized POMDPs (Amato, Bernstein, and Zilberstein 2010). Solving decentralized POMDPs is a fundamentally more complex problem and we expect our controller design and optimization method to be quite effective in that case as well.

Acknowledgments

Support for this work was provided in part by NSF Grants IIS-1116917 and IIS-1405550.

References

Amato, C., and Zilberstein, S. 2008. What's worth memorizing: Attribute-based planning for DEC-POMDPs. In *Proceedings of the ICAPS Workshop on Multiagent Planning*.

Amato, C.; Bernstein, D. S.; and Zilberstein, S. 2007. Solving POMDPs using quadratically constrained linear programs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2418–2424.

Amato, C.; Bernstein, D. S.; and Zilberstein, S. 2010. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Autonomous Agents and Multi-Agent Systems* 21(3):293–320.

Amato, C.; Bonet, B.; and Zilberstein, S. 2010. Finite-state controllers based on Mealy machines for centralized and decentralized POMDPs. In *Proceedings of the 24th Conference on Artificial Intelligence*, 1052–1058.

Aras, R., and Dutech, A. 2010. An investigation into mathematical programming for finite horizon decentralized POMDPs. *Journal of Artificial Intelligence Research* 37:329–396.

Aras, R.; Dutech, A.; and Charpillet, F. 2007. Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling*, 18–25.

Braziunas, D., and Boutilier, C. 2004. Stochastic local search for POMDP controllers. In *Proceedings of the 19th National Conference on Artificial Intelligence*, 690–696.

Charlin, L.; Poupart, P.; and Shioda, R. 2007. Automated hierarchy discovery for planning in partially observable environments. In *Advances in Neural Information processing Systems*, 225–232.

Grześ, M.; Poupart, P.; and Hoey, J. 2013a. Controller compilation and compression for resource constrained applications. In *Algorithmic Decision Theory*, volume 8176 of *Lecture Notes in Computer Science*, 193–207.

Grześ, M.; Poupart, P.; and Hoey, J. 2013b. Isomorph-free branch and bound search for finite state controllers. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 2282–2290.

Guyon, I., and Elisseeff, A. 2003. An introduction to variable and feature selection. *Journal of Machine Learning Research* 3:1157–1182.

Hansen, E. A. 1997. An improved policy iteration algorithm for partially observable MDPs. In *Advances in Neural Information Processing Systems*, 1015–1021.

Hansen, E. A. 1998. Solving POMDPs by searching in policy space. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 211–219.

Hoey, J.; Yang, X.; Quintana, E.; and Favela, J. 2012. LaCasa: Location and context-aware safety assistant. In *Proceedings of the 6th International Conference on Pervasive Computing Technologies for Healthcare*, 171–174.

Littman, M. 1994. Memoryless policies: Theoretical limitations and practical results. In *Proceedings of the International Conference on Simulation of Adaptive Behavior*, 238–245.

Meuleau, N.; Kim, K.-E.; Kaelbling, L. P.; and Cassandra, A. R. 1999. Solving POMDPs by searching the space of finite policies. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, 417–426.

Pajarinen, J., and Peltonen, J. 2011. Periodic finite state controllers for efficient POMDP and DEC-POMDP planning. In *Advances in Neural Information Processing Systems*, 2636–2644.

Pineau, J.; Gordon, G.; and Thrun, S. 2006. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research* 27:335–380.

Poupart, P., and Boutilier, C. 2003. Bounded finite state controllers. In *Advances in Neural Information Processing Systems*.

Poupart, P.; Kim, K.; and Kim, D. 2011. Closing the gap: Improved bounds on optimal POMDP solutions. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling*, 194–201.

Poupart, P.; Lang, T.; and Toussaint, M. 2011. Analyzing and escaping local optima in planning as inference for partially observable domains. In *Proceedings of the European Conference on Machine Learning*, 613–628.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1st edition.

Shani, G.; Brafman, R. I.; and Shimony, S. E. 2007. Forward search value iteration for POMDPs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2619–2624.

Shani, G.; Pineau, J.; and Kaplow, R. 2013. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems* 27(1):1–51.

Siddiqi, S.; Gordon, G.; and Moore, A. 2007. Fast state discovery for HMM model selection and learning. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics*, 492–499.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for POMDPs. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 520–527.

Smith, T., and Simmons, R. G. 2005. Point-based POMDP algorithms: Improved analysis and implementation. In *Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*, 542–547.

Sondik, E. J. 1971. *The Optimal Control of Partially Observable Markov Processes*. Ph.D. Dissertation, Stanford University.

Spaan, M., and Vlassis, N. 2005. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research* 24:195–220.

Toussaint, M.; Harmeling, S.; and Storkey, A. 2006. Probabilistic inference for solving (PO)MDPs. Technical Report EDIINF-RR-0934, University of Edinburgh, School of Informatics.

Vlassis, N.; Littman, M. L.; and Barber, D. 2012. On the computational complexity of stochastic controller optimization in POMDPs. *ACM Transactions on Computation Theory* 4(4):12:1–12:8.

Witwicki, S. J. 2011. *Abstracting Influences for Efficient Multiagent Coordination Under Uncertainty*. Ph.D. Dissertation, Department of Computer Science, University of Michigan, Ann Arbor.