Energy Efficient Execution of POMDP Policies

Marek Grześ, Pascal Poupart, Xiao Yang, and Jesse Hoey

***Abstract*—Recent advances in planning techniques for partially observable Markov decision processes (POMDPs) have focused on online search techniques and offline point-based value iteration. While these techniques allow practitioners to obtain policies for fairly large problems, they assume that a nonnegligible amount of computation can be done between each decision point. In contrast, the recent proliferation of mobile and embedded devices has lead to a surge of applications that could benefit from state-of-the-art planning techniques if they can operate under severe constraints on computational resources. To that effect, we describe two techniques to compile policies into controllers that can be executed by a mere table lookup at each decision point. The first approach compiles policies induced by a set of alpha vectors (such as those obtained by point-based techniques) into approximately equivalent controllers, while the second approach performs a simulation to compile arbitrary policies into approximately equivalent controllers. We also describe an approach to compress controllers by removing redundant and dominated nodes, often yielding smaller and yet better controllers. Further compression and higher value can sometimes be obtained by considering stochastic controllers. The compilation and compression techniques are demonstrated on benchmark problems as well as a mobile application to help persons with Alzheimer's to way-find. The battery consumption of several POMDP policies is compared against finite-state controllers learned using methods introduced in this paper. Experiments performed on the Nexus 4 phone show that finite-state controllers are the least battery consuming POMDP policies.**

***Index Terms*—Energy-efficiency, finite-state controllers, knowledge compilation, Markov decision processes, mobile applications, partially observable Markov decision processes (POMDPs).**

I. Introduction

PARTIALLY observable Markov decision processes (POMDPs) provide a natural framework for sequential decision making in partially observable domains. Tremendous progress has been made in recent years to develop scalable planning techniques for POMDPs. Point-based value iteration methods for factored and continuous domains can compute good value policies for a wide range of real-world problems [1], [2]. In addition, online resources can be used to perform a search at run time to directly select the next action or refine a precomputed policy [3], [4].

In this paper, we are motivated by an emerging class of applications that pose new challenges for POMDP solvers. We consider monitoring and assistive applications that run on smart-phones, wearable systems, or other mobile devices. While computational resources are rapidly increasing, energy consumption remains an important bottleneck due to limited battery life. This is especially important in monitoring and assistive applications that need to be continuously running, but should be as power efficient as possible. For such applications, online planning is not an option due to the high computational costs. Computed policies that require online belief monitoring at execution time also consume too much energy. While it is sometimes possible to offload computation through cloud solutions, this requires a data connection, which may not always be available or stable, and which has a high battery consumption.

An effective solution can be found by noting that a POMDP policy can be represented very simply by using a finite state controller (FSC) [5], which only requires simple table look-ups during execution. However, controller optimization is notoriously difficult. The nonconvex nature of the optimization makes it difficult for many approaches (e.g., gradient ascent [6], quadratically constrained optimization [7], bounded policy iteration (BPI) [8], and expectation maximization (EM) [9]) to reliably find the global optimum. An exhaustive search of the space of controllers can avoid local optima, but is clearly intractable [10], [11].

In this paper, we describe two novel techniques for compiling an existing POMDP policy (as generated by a point-based method, for example) into a FSC (Section III). The first method requires a policy specified as a set of α-vectors and witness belief points to construct a FSC directly that approximates the given policy. The second method needs only a simulation of the policy to build a controller incrementally by constructing a policy tree and then detecting equivalent conditional plans. We also describe a novel method for compressing a FSC into an equivalent, but smaller, FSC by removing redundant nodes (Section IV). We demonstrate our techniques on a set of large benchmark POMDP problems (Section VI), and we use policies generated by two state-of-the-art point-based techniques, namely GapMin [12] and SARSOP [13]. We show how we can construct very compact controllers that are equivalent, and sometimes better, than the policies they are derived from. We also demonstrate our methods on a set of POMDPs that are used to provide mobile assistance for persons with Alzheimer's disease for way-finding.

This paper is structured as follows. Section II reviews POMDPs. Section III explains our two techniques for policy compilation into a FSC, and Section IV shows how to

Manuscript received January 9, 2014; revised August 7, 2014 and November 7, 2014; accepted November 16, 2014. This work was supported in part by the Ontario Ministry of Research and Innovation, in part by the Natural Sciences and Engineering Research Council of Canada, in part by the Toronto Rehabilitation Institute, and in part by the Alzheimer's Association Grant ETAC-10-173237. This paper was recommended by Associate Editor S. E. Shimony.

Marek Grześ, Pascal Poupart, and Jesse Hoey are with the David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: mgrzes@cs.uwaterloo.ca).

Xiao Yang is with Amazon, Toronto, ON, Canada.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2014.2375817

compress the resulting FSCs by removing redundant nodes. Section V shows experiments about battery consumption for various POMDP policies using a Nexus 4 phone. Section VI reports experiments with benchmark problems and a wayfinding application to assist people with Alzheimer's disease. Section VIII concludes this paper.

II. Background

A POMDP is formally defined by a tuple $\langle S, A, O, T, Z, R, b_0, \gamma \rangle$ that includes a set S of states s, a set A of actions a, a set O of observations o, a transition function $T(s', s, a) = \Pr(s'|s, a)$, indicating the probability of reaching s' after executing a in s, an observation function $Z(o, a, s') = \Pr(o|s', a)$, indicating the probability of observing o after executing a and reaching s', a reward function $R(s, a) \in \Re$, indicating the immediate reward earned after executing a in s, an initial belief $b_0(s) = \Pr(s)$, indicating the probability of starting the process in each state s, and a discount factor $0 \leq \gamma \leq 1$ indicating the rate at which rewards are discounted at each step. In this paper, we assume that the planning horizon is infinite, although the proposed algorithms can be modified easily for finite horizon problems. The goal is to find an optimal policy that maximizes the discounted sum of rewards. A policy determines the choice of action at each time step based on the observable quantities. Since the observable quantities are the past actions and observations, a policy $\pi : H_t \rightarrow A_t$ can be defined as a mapping from histories $H_t \equiv A_0 \times O_1 \times \cdots \times A_{t-1} \times O_t$ of past actions and observations to actions A_t, however, this definition is problematic for an infinite horizon since histories may be arbitrarily long. Two approaches are often used to circumvent this issue: 1) replace histories by finite length sufficient statistics such as beliefs or 2) represent policies as FSCs, which are mappings from cyclic histories to actions.

A belief $b(s)$ is a distribution over states reflecting the decision maker's belief that the process may be in each state s. We can update a belief b after executing a and observing o according to Bayes' theorem

$$b^{ao}(s') \propto \sum_s b(s) \Pr(s'|s, a) \Pr(o|s', a) \ \forall s'. \quad (1)$$

Given the initial belief b_0 and a history $h_t = \langle a_0, o_1, \ldots, a_{t-1}, o_t \rangle$, we can compute the belief b_t at time step t by repeatedly applying the above equation for each action-observation pair in the history. Since beliefs have an $|S|$-dimensional vector representation regardless of the length of the process, they provide a practical alternative to histories. Hence, we can equivalently define policies as mappings $\pi : B \rightarrow A$ from beliefs to actions. The value $V^\pi(b_0)$ of policy π when starting in b_0 is the discounted sum of expected rewards $V^\pi(b_0) = \sum_{t=0}^{\infty} \gamma^t R(b_t, \pi(b_t))$ where $R(b, a) = \sum_s b(s) R(s, a)$.

We can also consider policies represented by a FSC $\pi = \langle N, \phi, \psi \rangle$, which is defined by a set N of nodes n, a mapping $\phi : N \rightarrow A$ indicating which action a to execute in each node n and a mapping $\psi : N \times O \rightarrow N$ indicating that the edge rooted at n and labeled by o should point to n'. A controller is executed by alternating between executing the action $\phi(n)$ of the current node n and moving to the next node $\psi(n, o)$ by following the edge rooted at n that is labeled with the current observation o. The value α_n of the controller when starting in n is an $|S|$-dimensional vector computed as follows:

$$\alpha_n(s) = R(s, \phi(n)) + \gamma \sum_{s', o} \Pr(s'|s, a) \Pr(o|s', a) \alpha_{\psi(n,o)}(s') \ \forall n, s. \quad (2)$$

Here, $\alpha_n(s)$ is an $|S|$-dimensional vector indicating the value of the conditional plan rooted at node n for any starting state s.

Policy optimization algorithms can be classified in two broad categories: 1) offline techniques that precompute a policy before the start of the execution [14]–[18] and 2) online techniques that perform all their computation at run time by searching for the best action to execute after receiving each observation [3]. Online techniques can take advantage of the history so far to focus their computation only on the current belief. When computational resources are not constrained and there is sufficient time between decisions to search for the next action to execute, online techniques can perform very well and can scale to very large problems. In contrast, offline techniques do not scale as well, but permit the deployment of POMDP policies on mobile and/or embedded devices with severe resource constraints due to energy, memory or CPU limitations.

Among the offline techniques, we can further classify algorithms based on the type of policies (belief mapping or FSC) that they produce. Algorithms that produce belief mappings often exploit the fact that the value V^* of an optimal policy satisfies Bellman's equation

$$V^*(b) = \max_a \sum_s b(s) \left[R(s, a) + \gamma \sum_{s', o} \Pr(s'|s, a) \Pr(o|s', a) V^*(b^{ao}) \right] \forall b. \quad (3)$$

In theory, the optimal value function could be computed by value iteration, which repeatedly updates V^* by computing the right hand side of (3). However, the continuous nature of the belief space prevents us from performing value iteration at all beliefs and therefore the important class of point-based techniques performs point-based Bellman backups only at a finite set of beliefs [15]. An approximation of the value function at all beliefs is obtained by computing the gradient in addition to the value at each belief. This allows the formation of a set of linear value functions that are often represented by α-vectors, similar to the value functions of controller nodes. While the details of point-based value iteration are not important for the rest of this paper (see [19] for more information), what is important to know is that they produce a set Γ of $\langle \alpha_i, b_i, a_i \rangle$-tuples that associate each α_i with an action a_i and a witness belief b_i [i.e., belief for which α_i yields the highest value: $\alpha_i(b_i) \geq \alpha_j(b_i) \ \forall j$ where $\alpha(b) = \sum_s b(s) \alpha(s)$]. The policy π induced by Γ is obtained by computing

$$\pi(b) = a_{\text{best}} \text{ where best } = \arg\max_i \ \alpha_i(b). \quad (4)$$

Although point-based value iteration techniques compute the set Γ offline, they still require a certain amount of computation at each decision point. The belief must be updated after each action and observation according to (1) [complexity $O(|S|^2)$] and the best α-vector must be identified according to (4) [complexity $O(|S||\Gamma|)$]. This amount of computation may still be prohibitive when S and Γ are large and there is not enough memory, time, or energy.

Alternatively, the other group of offline techniques produces policies represented as FSCs [10]. Since the execution of a controller merely consists of a table lookup, they are the most convenient type of policies for deployment in resource constrained applications. A substantial amount of research has, however, been devoted to point-based techniques; therefore, existing point-based methods can either solve larger POMDP instances or are more robust against local optima issues than direct controller optimization. Point-based methods are traditionally robust against local optima because they use upper bounds to guide their search. Except for exact enumeration [11], it is not clear how to use upper bounds efficiently in direct controller optimization when the size of the controller is bounded or fixed [8], [20], i.e., when a small controller is sought. Instead of directly optimizing a FSC, in this paper, we propose two techniques to compile policies into approximately equivalent controllers. This has the benefit that we can use existing scalable algorithms such as point-based value iteration to quickly obtain a good policy. In addition, the controller compilation allows those policies to be executed on devices that are much more constrained.

III. Controller Compilation

Kaelbling *et al*. [5] observed that an optimal controller can be extracted from an optimal value function. Unfortunately, the best value functions found by state-of-the-art algorithms are approximate/suboptimal for most problems. Hansen [21] wrote "it is unclear how to construct suboptimal controllers from [such value functions]." Hence, for the past 15 years, research has focused on directly optimizing controllers. We propose two approaches to compile suboptimal policies into approximately equivalent controllers. The first approach is limited to policies implicitly represented by sets of α-vectors as produced by point-based value iteration techniques. The second approach works with arbitrary policies.

A. Compiling Controllers From Alpha Vectors

As explained in Section II, point-based value iteration techniques produce a set Γ of $\langle \alpha_i, b_i, a_i \rangle$-tuples from which a belief mapping policy is extracted. Algorithm 1 shows how to compile Γ into an approximately equivalent controller $\langle N, \phi, \psi \rangle$. We create a node n_i for each vector α_i (Line 4). Each node n_i is labeled with the action $\phi(n_i) = a_i$ associated with α_i (Line 5). To determine where the edge rooted at n_i and labeled with o should point to, we update the witness b_i of α_i according to (1) based on action a_i and observation o. Let the resulting belief be $b_i^{a_i,o}$. We then find which α-vector has the highest value at $b_i^{a_i,o}$ (Line 9) and assign the corresponding node to $\psi(n_i, o)$ (Line 10). The complexity of this compilation technique is $O(|\Gamma|^2|O||S|^2)$, however, in practice the dependence on $|O|$ and $|S|$ can often be reduced by exploiting sparsity. Note that edges that have zero probability, as computed using $\Pr(o|b,a) = \sum_{s,s'} b(s) \Pr(s'|s,a) \Pr(o|a,s')$, are skipped in Line 8. The time to compile a policy from Γ is typically a fraction of the time taken by point-based value iteration to obtain Γ. The quality of the resulting controller varies. The following guarantee can be made.

Algorithm 1 Compilation of α-Vectors Into an Approximately Equivalent Controller $\langle N, \phi, \psi \rangle$

ALPHA2FSC(Γ)

1: Let Γ be a set of $\langle \alpha_i, b_i, a_i \rangle$-tuples
2: $N \leftarrow \emptyset$
3: **for** $i = 1$ **to** $|\Gamma|$ **do**
4: $\quad N \leftarrow N \cup \{n_i\}$
5: $\quad \phi(n_i) \leftarrow a_i$
6: **for** $i = 1$ **to** $|\Gamma|$ **do**
7: $\quad$ **for all** $o \in O$ **do**
8: $\quad\quad$ **if** $\Pr(o|b_i, a_i) > 0$ **then**
9: $\quad\quad\quad best \leftarrow \arg\max_j \ \alpha_j(b_i^{a_i,o})$
10: $\quad\quad\quad \psi(n_i, o) \leftarrow n_{best}$
11: $\quad\quad$ **else**
12: $\quad\quad\quad \psi(n_i, o) \leftarrow n_i$
13: **return** $\langle N, \phi, \psi \rangle$

Theorem 1: When the initial belief b_0 is the witness of some vector α_0 and the controller starts in the node n_0 associated with α_0, the ALPHA2FSC compilation technique ensures that the actions selected at the first two time steps are identical to that of the policy induced by Γ, but not for the following time steps.

Proof: At the first time step, the policy induced by Γ executes the action associated with α_0 while the controller executes the action associated with n_0. Since the node associated with each α-vector is labeled with the action of that α-vector, the first action is the same in both policies. Let b_0^{ao} be the belief at the second time step. The policy induced by Γ executes the action associated with the maximal α-vector for b_0^{ao}. Since the edge rooted at n_0 and labeled with o is constructed to point to the node associated with the maximal α-vector for b_0^{ao}, the same action is executed at the second time step. We cannot follow the same reasoning at the third timestep since b_0^{ao} may be different than the witness of the maximal α-vector at the second time step. Hence, the policies may execute different actions after the second time step. ■

Note, however, that a much stronger guarantee can be made when the α-vectors in Γ correspond to the optimal value function.

Theorem 2: If the set of α-vectors in Γ corresponds to the optimal value function, then the ALPHA2FSC compilation technique produces an optimal controller.

Proof: Let B_α be the set of beliefs for which α is maximal. When Γ corresponds to the optimal value function, then the action associated with each vector α is optimal for all beliefs in B_α. This ensures that the action associated with each node is also optimal for the same beliefs. Furthermore, when

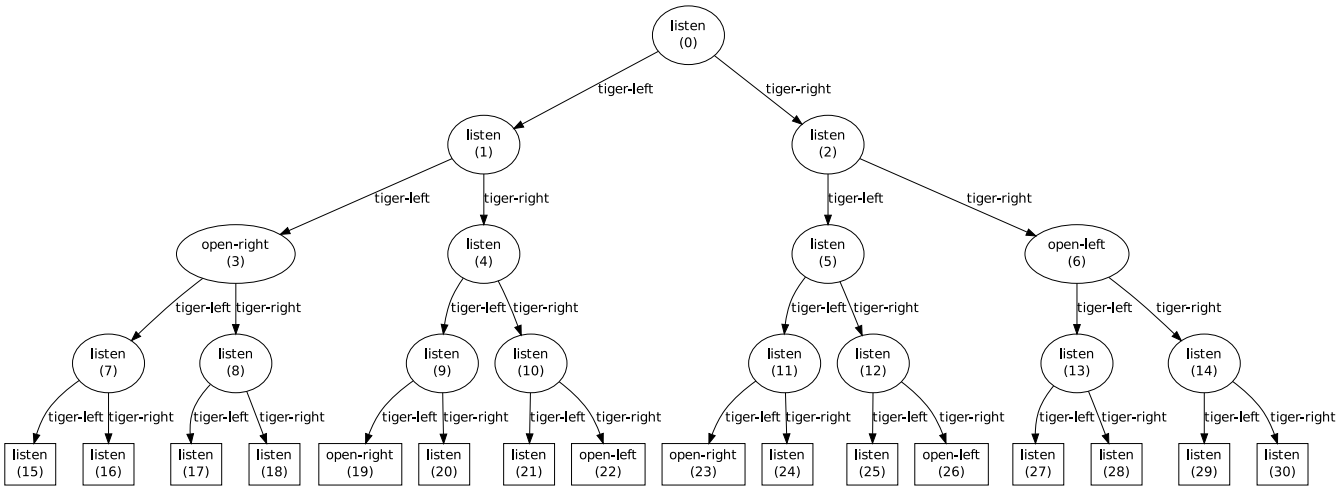

Fig. 1. Policy tree up to a depth of 5 for the classic tiger problem. Nodes are labeled with actions (listen, open-left, or open-right) whereas edges with observations (tiger-left or tiger-right). The numbers in parentheses indicate the order in which the nodes are generated.

Γ is optimal, all beliefs b^{ao} reached by executing the action associated with α from some belief $b \in B_\alpha$ and observing o have the same maximal vector(s). This ensures that the edges constructed by the compilation procedure are optimal. ■

When the set of α-vectors is suboptimal, which is the case most of the time, then actions selected after the second time step may be different than those selected by the policy induced by Γ, leading to a controller that may be better or worse. In the next section, we describe an approach that ensures that the resulting controller is at least as good as the original policy in the limit where the limit is an infinitely deep policy tree used for compilation. Although our goal is to obtain a FSC that is as good or better than the original policy, the reader should note that one can construct a POMDP which would require an infinite number of alpha vectors [22]; equivalently, the POMDP would require an infinite number of nodes in its FSC. The existing literature has shown, however, that bounded memory controllers [12], [14] or policies with a finite number of alpha vectors can yield solutions that are near optimal [8], [13], [23].

B. Compiling Controllers From Arbitrary Policies by Simulation

We describe an approach to compile arbitrary policies into approximately equivalent controllers. The approach simulates the policy up to a certain depth and ensures that the controller will execute the same actions up to that depth. In the limit, with an infinite depth, we obtain a controller that matches the policy exactly. Although, as we show in the experiments, we can often obtain a controller that is at least as good by simulating up to a reasonable depth.

The approach works in two steps: 1) first, we generate a policy tree up to a certain depth and then 2) we compress the policy tree into a controller by detecting matching subtrees. Algorithm 2 shows how to generate a policy tree up to a certain depth by simulating the policy. Since simulation does not require the policy to be in any format, the approach works with arbitrary policies. We just need to generate the next action given the current observation at each time step,

Algorithm 2 Policy Tree Generation

POLICYTREE($\pi, b, depth$)

1: $N \leftarrow \emptyset$
2: $j \leftarrow 1$
3: $queue \leftarrow \{\langle b, 0, j\rangle\}$
4: **while** $\neg isEmpty(queue)$ **do**
5: $\quad \langle b, d, i\rangle \leftarrow removeFirst(queue)$
6: $\quad N \leftarrow N \cup \{n_i\}$
7: $\quad \phi(n_i) \leftarrow \pi(b)$
8: $\quad$ **if** $d = depth$ **then**
9: $\quad\quad \psi(n_i, o) \leftarrow * \; \forall o \in O$
10: $\quad$ **else**
11: $\quad\quad$ **for all** $o \in O$ **do**
12: $\quad\quad\quad j \leftarrow j + 1$
13: $\quad\quad\quad addLast(queue, \langle b^{\phi(n_i)o}, d + 1, j\rangle)$
14: $\quad\quad\quad \psi(n_i, o) \leftarrow n_j$
15: **return** $\langle N, \phi, \psi\rangle$

which is always possible since this is how all policies are executed in practice. To be concrete, Algorithm 2 shows how to generate a policy tree for policies that are belief mappings, but we could easily modify the algorithm to work with policies that are represented as history mappings or any other type of mapping. The algorithm generates a policy tree in breadth first order, which will become handy in the compression step. Since leaves do not have edges, we set $\psi(n, o)$ to $*$ for all edges rooted at a leaf n (Line 9).

Fig. 1 shows the policy tree generated by Algorithm 2 up to a depth of 5 for the classic tiger problem [5]. In this problem, there are three actions (listen, open-right, and open-left), two observations (tiger-right and tiger-left). Nodes are labeled with actions and edges are labeled with observations. Nodes are also numbered according to the breadth-first order in which they were generated.

In the second step, the policy tree is compressed into a controller by identifying matching conditional plans. Each node of the policy tree is the root of a conditional plan. Conditional plans rooted at each node are compared to conditional plans

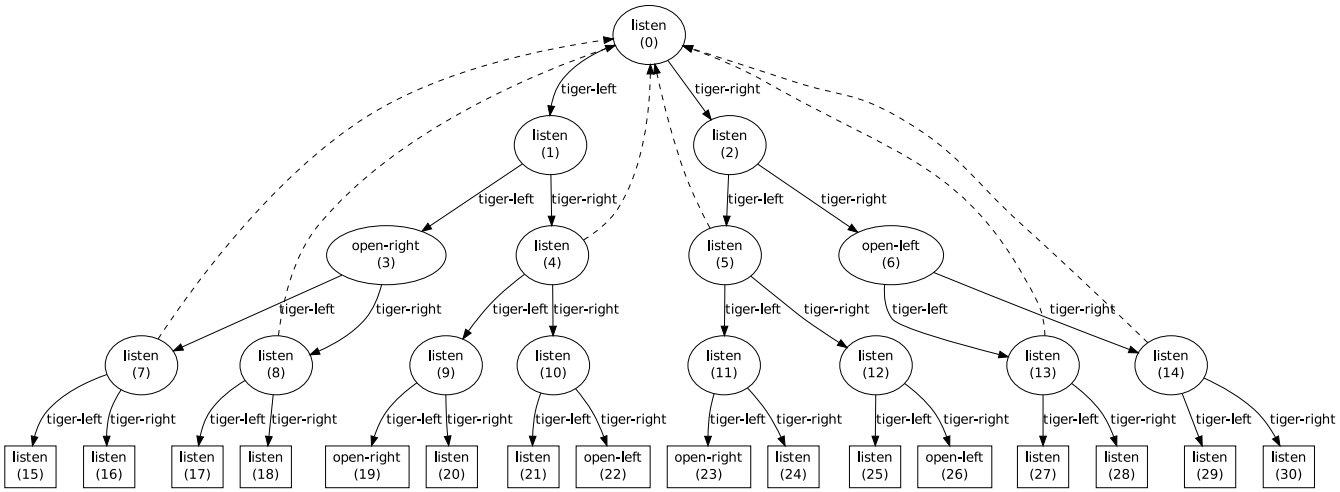

Fig. 2. Policy tree up to a depth 5 for the classic tiger problem with dashed edges indicating nodes whose conditional plans match according to Algorithm 3.

Algorithm 3 Equivalent Conditional Plans

EQUIVALENTCP(n_1, n_2, ϕ, ψ)

if $\phi(n_1) \neq \phi(n_2)$ **then**
  return *false*
for all $o \in O$ **do**
  if $\psi(n_1, o) \neq *$
    and $\neg$EQUIVALENTCP($\psi(n_1, o), \psi(n_2, o), \phi, \psi$)
  then
    return *false*
return *true*

Algorithm 4 Compilation of Arbitrary π Into Approximate Equivalent Controller $\langle N, \phi, \psi \rangle$

POLICY2FSC($\pi, b, depth$)

$\langle N, \phi, \psi \rangle \leftarrow$ POLICYTREE($\pi, b, depth$)
for all $n_i \in N$ in increasing index i **do**
  for all $n_j \in N$ such that $j < i$ **do**
    if EQUIVALENTCP(n_i, n_j, ϕ, ψ) **then**
      $N \leftarrow N \setminus \{n_i, descendents(n_i)\}$
      for all $n \in N, o \in O$ **do**
        if $\psi(n, o) = n_i$ **then**
          $\psi(n, o) \leftarrow n_j$
return $\langle N, \phi, \psi \rangle$

rooted at previous nodes in the breadth-first order. When two conditional plans match, we replace the node with highest breadth-first index by the node with the lowest breadth-first index. Two conditional plans are said to match when they select the same actions in each path up until a leaf is encountered. Hence, conditional plans with different depths can still match since we stop the verification as soon as a leaf is encountered in a path. Algorithm 3 shows how to verify whether two conditional plans match. Algorithm 4 uses this verification procedure to prune nodes whose conditional plans match the conditional plan of an earlier node in the breadth-first order. This process gives rise to a controller that is often much smaller than the original policy tree and yet ensures that the same actions are executed up to the depth of the original policy tree.

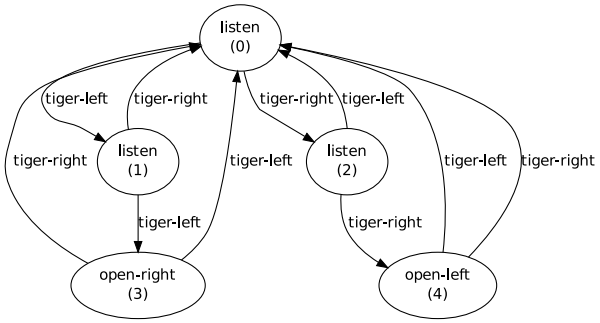

Fig. 3. Controller obtained by reducing a five-step policy tree according to Algorithm 4 for the classic tiger problem.

Fig. 2 shows again the policy tree for the tiger problem with additional dashed edges indicating that the parent node is replaced by the child node due to matching conditional plans. For instance, node 4 will be replaced by node 0 since their conditional plans match. Fig. 3 shows the resulting reduced controller once all node substitutions indicated by dashed edges in Fig. 2 are performed. Since leaf nodes have a trivial one-step conditional plan and they are last in the breadth-first order, they will be replaced by interior nodes as long as there is an interior node with the same action. Since actions eventually repeat in a large enough tree, the compilation procedure generally produces controllers without leaves (i.e., all nodes have a full set of edges). The breadth-first order also ensures that Algorithm 3 terminates since in each pair of conditional plans that we compare, the one rooted at the node with the highest index is necessarily a tree of finite depth (i.e., no loop). In addition, when we replace the node with the highest index we can delete the entire subtree below it since there is no way to reach that subtree other than through the node that is being replaced. This pruning greatly improves the running time. Finally, the breadth first order also helps to produce a small controller since nodes are always replaced by nodes with a lower index and therefore earlier in the tree.

The complexity of Algorithm 4 is quadratic in the size of the policy tree. However, due to the pruning of subtrees each time a node is replaced, we can show that the complexity is really linear in the size of the policy tree times the size of the reduced controller. The experiments show that the reduced controller is often significantly smaller than the policy tree, yielding a substantial speed up. That being said, the linear dependence on the size of the policy tree is still significant since the size of policy trees is exponential in the depth [i.e., $O(|O|^{\text{depth}})$]. We can often reduce the base $|O|$ of the exponential by exploiting sparsity or considering only observations with a probability greater than some threshold.

Theoretical guarantees of the approximate equivalence between the original and compiled polices are shown in the following theorems.

Theorem 3: The POLICY2FSC compilation procedure guarantees that the controller executes the same actions for a number of time steps at least equal to D where D is the depth of the tree in the compilation.

Proof: Whenever a subtree rooted at n_j is replaced by a subtree rooted at n_i, n_i occurs before n_j in the breath-first order. This means that the subtree rooted at n_i is at least as deep as the subtree rooted at n_j. Since the subtree rooted at n_j is replaced by a matching subtree rooted at n_i that is at least as deep, then all actions of n_j are preserved by n_i. This means that all actions of the entire tree of depth D are preserved in the resulting controller. ■

Theorem 4: For a policy tree of depth D, the loss in the value of the initial belief, b_0, due to controller compilation using the POLICY2FSC procedure is bounded by $\gamma^D(R_{\max} - R_{\min})/(1-\gamma)$.

Proof: The maximal loss in any belief is $(R_{\max} - R_{\min})/(1-\gamma)$. Since, the controller is equivalent with the original policy up to depth D, the loss is discounted by γ^D. ■

The theorem shows that the loss goes to zero when depth D goes to infinity. Furthermore, tighter bounds would be possible if the value of the policy is known in all beliefs since the maximal loss in any belief could be much smaller than $(R_{\max} - R_{\min})/(1-\gamma)$.

IV. CONTROLLER COMPRESSION

Once a policy is compiled to a controller, it often contains redundant or dominated nodes. Two nodes are redundant when they have identical α-vectors. In the base case, redundant nodes have identical conditional plans [11]. Additionally, redundant nodes occur when some observations have zero probability, leading to multiple conditional plans with the same value—nodes that lead to different conditional plans can still be redundant. Dominated nodes often occur when the original policy is suboptimal and the compilation process generates suboptimal conditional plans. We describe a technique to compress a controller while ensuring that its value does not decrease and in some cases it increases. The idea is to prune all nodes with α-vectors that are dominated in value by other α-vectors. This approach was first used by Hansen [14] in his policy iteration algorithm. Algorithm 5 describes how to repeatedly compress a deterministic controller until there are no dominated nodes. The approach alternates between policy evaluation and node substitution. The evaluation step computes the α-vector of each node by solving a system of linear equations. Then the α-vector of each node is compared to the α-vectors of the other nodes. When $\alpha_1(s) \leq \alpha_2(s)\ \forall s$ then n_1 can be replaced by n_2.

Algorithm 5 Deterministic FSC Compression

DETERMINISTICFSCCOMPRESSION(N, ϕ, ψ)

```
repeat
  Eval controller by solving (2)
  for each n1 ∈ N do
    for each n2 ∈ N \ {n1} do
      if α_{n1}(s) ≤ α_{n2}(s) ∀s then
        N ← N \ {n1}
        for all n ∈ N, o ∈ O do
          if ψ(n, o) = n1 then
            ψ(n, o) ← n2
        break
      if α_{n1}(s) ≥ α_{n2}(s) ∀s then
        N ← N \ {n2}
        for all n ∈ N, o ∈ O do
          if ψ(n, o) = n2 then
            ψ(n, o) ← n1
until N doesn't change
return ⟨N, φ, ψ⟩
```

Theorem 5: The compression procedure in Algorithm 5 returns a controller with a value equal to or higher than the value of the original controller.

Proof: Suppose that n_1 is replaced by n_2 in Algorithm 5. Since the value of n_2 (measured by its vector α_2) is at least as good as that of n_1 (measured by its vector α_1) in all states, then pruning n_1 and replacing it by n_2 does not lower the value of the controller. The value will go up if there is an s such that $\alpha_2(s) > \alpha_1(s)$. We prove this formally by induction.

Consider (2) for policy evaluation. This system of linear equations can be solved by dynamic programming by repeatedly computing the right hand side

$$\alpha_n^{i+1}(s) \leftarrow R(s, \phi(n)) + \gamma \sum_{s',o} \Pr(s', o|s, a)\alpha^i_{\psi(n,o)}(s')\ \forall n, s. \quad (5)$$

Here, the superscript i indicates the step in dynamic programming. As $i \to \infty$, the α-vectors converge. Let us initialize each α_n^0 with the values of the nodes in the original controller. When replacing n_1 by n_2, we are effectively replacing all instances where $\psi(n, o) = n_1$ by $\psi'(n, o) = n_2$. We will show that

$$\alpha_n^i(s) \geq \alpha_n^0(s)\ \forall s, n, i \quad (6)$$

and therefore when $i \to \infty$ the compressed controller does not decrease in value in comparison to the original controller.

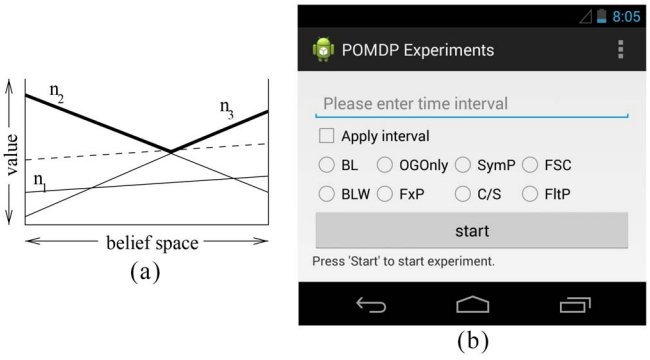

Fig. 4. (a) n_1 is jointly dominated by n_2 and n_3. (b) Application screenshot.

Let $i = 0$ be the base case. Suppose (6) holds for i, then at $i+1$ we have

$$\begin{aligned}\alpha_n^{i+1}(s) &= R(s,\phi(n)) + \gamma \sum_{s',o} \Pr(s',o|s,a)\alpha^i_{\psi'(n,o)}(s') \ \forall s,n \\ &\geq R(s,\phi(n)) + \gamma \sum_{s',o} \Pr(s',o|s,a)\alpha^0_{\psi'(n,o)}(s') \ \forall s,n \\ &\geq R(s,\phi(n)) + \gamma \sum_{s',o} \Pr(s',o|s,a)\alpha^0_{\psi(n,o)}(s') \ \forall s,n \\ &= \alpha_n^0(s) \ \forall s,n. \end{aligned}$$

■

The complexity of the policy evaluation step in Algorithm 5 is $O(|N|^3|S|^3|O|)$, however, sparsity often allows to reduce the dependence on $|S|$ and $|O|$. The complexity of the pruning step is $O(|N|^2|S|)$. Overall, compression time is a small fraction of compilation time.

The above compression technique can only detect nodes that are dominated by a single node. In some cases, the α-vector of a node is not entirely dominated by any single α-vector, but it is dominated by the upper surface of several α-vectors. In that case, it is possible to remove the dominated node and replace any link to that node by stochastic links to the dominating nodes [8]. Consider the α-vector of n_1 in Fig. 4(a). It is jointly dominated by α-vectors of n_2 and n_3. The following linear program computes a convex combination of α-vectors that uniformly increases the value of a dominating node by the largest δ possible:

$$\begin{aligned}\max_{\delta,\ \Pr(n')\forall n'} \delta \text{ s.t. } \alpha_n(s) + \delta &\leq \sum_{n'} \Pr(n')\alpha_{n'}(s) \ \forall s \\ \sum_{n'} \Pr(n') = 1, \quad \Pr(n') &\geq 0 \ \forall n'. \end{aligned} \quad (7)$$

Here, $\Pr(n')$ denotes the probability of choosing n' in the convex combination of dominating nodes. In Fig. 4(a), the value of the convex combination of n_2 and n_3 that uniformly increases the value of n_1 the most is denoted by the dashed line.

We can use the above linear program to define a more aggressive compression technique that yields stochastic controllers. In fact, it is well known that the optimal controller for a fixed number of nodes may be stochastic [8]. The reader should realize that stochastic FSCs require a random number generator to implement stochastic edges or stochastic actions.

Algorithm 6 Stochastic FSC Compression

STOCHASTICFSCCOMPRESSION(N, ϕ, ψ)

repeat
  Eval controller by solving (8)
  for each $n \in N$ **do**
    Compute δ and $\Pr(n')$ by solving the LP in (7)
    if $\delta > 0$ **then**
      $N \leftarrow N \setminus \{n\}$
      $\psi(n'|n'',o) \leftarrow \psi(n'|n'',o) + \Pr(n')\psi(n|n'',o) \forall n' n'' o$
until N doesn't change
return $\langle N, \phi, \psi \rangle$

While devices such as mobile phones usually have software that includes pseudo-random number generators by default, it may be challenging to access random number generators on wearable or other small devices, although techniques for embedded and hardware random generators exist [24], [25] and energy efficient solutions may still be implemented.

A stochastic controller is parametrized by probability distributions over actions and reachable nodes. We abuse notation by using $\phi(a|n)$ and $\psi(n'|n,o)$ to denote those distributions. The value of a stochastic controller can be computed by solving a system of linear equations similar to the one in (2)

$$\begin{aligned}\alpha_n(s) = \sum_a \phi(a|n)[R(s,a) + \gamma \sum_{s',o} \Pr(s'|s,a) \Pr(o|s',a) \\ \times \sum_{n'} \psi(n'|n,o)\alpha_{n'}(s')] \ \forall n,s. \end{aligned} \quad (8)$$

Algorithm 6 describes a compression technique that yields stochastic controllers. It uses the above system of linear equations to compute the α-vectors of each node and the LP in (7) to detect nodes dominated by a convex combination of several nodes. Links to dominated nodes are replaced by stochastic links to the dominating nodes.

Theorem 6: The compression procedure in Algorithm 6 returns a controller with a value equal to or higher than the value of the original controller. Furthermore, Algorithm 6 finds controllers with value at least as high and size at least as small as that of the deterministic controllers found by Algorithm 5.

Proof: Algorithm 6 replaces a node by a convex combination of nodes such that the value of the convex combination is $\delta (\geq 0)$ higher than the value of the original node according to the linear program in (7). We can then prove by induction that the value of the compressed controller does not decrease in the same way as the proof for Theorem 5.

Since pruning nodes dominated by a single node is a special case of domination by a convex combination of nodes, Algorithm 6 will produce controllers that are at least as small as those produced by Algorithm 5. Furthermore, since the algorithm finds convex combinations with values at least as high as the value of singly dominating nodes, Algorithm 6 will produce controllers with value at least as high as the value of the controllers produced by Algorithm 5. ■

Note that there is a significant time cost to solve linear programs and therefore the running time of Algorithm 6 is much longer in practice than for Algorithm 5.

V. Experiments About Battery Consumption

We start with a battery consumption experiment that shows that controllers are important when energy efficient POMDP execution is required. The battery consumption experiment demonstrates that different types of policies will drain a battery at different rates. In particular, FSCs emerge as the most energy efficient type of policy, which motivates our experiments about controller compilation in Section VI. The need for battery efficient execution of POMDP policies is important for mobile devices such as mobile phones and wearable sensors. In this paper, we consider an application on a smart phone that assists people with cognitive disabilities to find their way to a destination. The use of an energy efficient policy is critical since the battery must not run out before the end of the wayfinding task. The next section introduces the application domain.

A. LaCasa Domain

Wandering is a common behavior among people with dementia (PwD). It is also one of the main concerns of caregivers since it can cause the person to get lost and injured. The frequency and manner in which a person wanders is highly influenced by the person's background and contextual factors specific to the situation. We developed a POMDP model for a mobile application called "LaCasa" [11], [26] that estimates the risk faced by the PwD and decides on the appropriate action to take, such as prompting the PwD or calling the caregiver. The model was designed and instantiated using our technique for engineering POMDPs [27]. Contextual information gathered from sensors is integrated into the model, including current location, battery power, and proximity to the caregiver. The system can reason about the costs of sensors (e.g., battery charge) and the relative costs of different types of assistance. A preliminary version of the system has been instantiated in a wandering assistance application for mobile devices running on an Android platform. However, in the current system, the necessary POMDP belief updates and policy queries are computationally too demanding and are done on a remote server that communicates with the smartphone using simple XML messages. This is a problem since the server communications can be expensive battery-wise, and rely on a data connection. The FSCs we find using the method proposed in this paper alleviate this problem, allowing the policy to run directly and cheaply on the smartphone. Additionally, it is not necessarily the case that persons with dementia will be able to carry a smartphone, and may require a much smaller, embedded or wearable device. In such cases, the memory and computation power available becomes a more serious constraint, making the use of FSCs imperative. We experimented with three LaCasa versions of different sizes. The battery consumption experiment was conducted on the largest version of LaCasa (lacasa4.batt) whereas compilation and compression were performed on all three versions and those compilation results are reported in the next section.

B. Experimental Design

In order to examine how a mobile phone battery is drained by different POMDP policies, we conduct a series of experiments and record the time per 1% battery depletion for each implementation involving the configurations shown in Table I. All those configurations—except client/server—run entirely on a mobile phone. For the client/server configuration, the POMDP policy is executed on the server whereas observations are generated on the mobile phone. Observations and actions are communicated over a WIFI connection. The mobile phone obtains observations from its own sensors and then acts like a client and queries actions from the server. The process of querying actions involves updating the belief state on the server.

All the experiments were conducted on the same smartphone (Nexus 4, Android 4.2). Each experiment started with a battery level above 95% and kept running for 3 h, during which battery changes were recorded. We closed as many unrelated user applications as possible. No SIM card was installed on the smartphone in order to avoid uncontrollable interference. WIFI was turned on only when necessary (baselines with WIFI and client/server). The screen was on and set to fixed brightness throughout every 3-h experiment. Since Android OS considers screen-off as a signal of low usage, it may slow down the CPU and turn off WIFI, which would impact the results and

TABLE I
Configurations Evaluated in the Battery Consumption Experiment

	Method	Description
Baselines	OS only	The Android 4.2 operating system only.
	OS with WIFI	The same as above but with WIFI on.
	Observation generator	Observations of the environment are generated randomly on the phone, but no POMDP policy is executed.
	Constant policy	Constant POMDP policy is executed on the phone which always selects the same action and receives observations from the observation generator.
Policies	FSC	The finite-state controller is executed on the phone and receives observations from the observation generator.
	Client/Server via WIFI	The POMDP policy is executed on the server. Observations are generated on the phone. WIFI is on in this experiment.
	Flat policy	The policy based on sparse matrices is executed on the phone and receives observations from the observation generator.
	Symbolic Perseus	The policy based on algebraic decision diagrams (instead of sparse matrices) is executed on the phone and receives observations from the observation generator.

TABLE II
BATTERY CONSUMPTION RESULTS ON THE NEXUS 4 PHONE

	Experiment/Policy	0.1 Hz		0.5 Hz		1 Hz		8 Hz		Time of one Query (seconds)
		1% Battery Depletion Time (minutes)	Standard Error	1% Battery Depletion Time (minutes)	Standard Error	1% Battery Depletion Time (minutes)	Standard Error	1% Battery Depletion Time (minutes)	Standard Error	
Baselines	OS only	6.74	0.13	6.74	0.13	6.74	0.13	6.74	0.13	n.a.
	OS with WIFI	6.69	0.08	6.69	0.08	6.69	0.08	6.69	0.08	n.a.
	Observation generator	6.56	0.23	6.45	0.26	6.40	0.26	6.34	0.25	$< 10^{-3}$
	Constant policy	6.30	0.20	6.42	0.27	5.82	0.21	5.81	0.19	$< 10^{-3}$
Policies	FSC	6.30	0.12	6.51	0.29	5.71	0.18	5.86	0.24	$< 10^{-3}$
	Client/Server via WIFI	6.03	0.17	5.83	0.21	5.52	0.22	n.a.	n.a.	0.898
	Flat policy	6.02	0.12	4.82	0.15	4.10	0.11	n.a.	n.a.	0.472
	Symbolic Perseus	5.67	0.24	4.43	0.08	3.91	0.15	n.a.	n.a.	0.669

prevent a fair comparison between different policies. All comparisons were performed with the same screen configuration. A screenshot of the application is shown in Fig. 4(b).

The execution frequency (i.e., the number of policy queries per second) has a significant impact on battery consumption. For example, reading observations from the sensors, updating the belief state, querying the policy, and executing the action selected every 10 s consumes much less energy than executing all these steps several times per second. Therefore, we experiment with frequencies. The program runs in an infinite loop where policy queries are made at different time intervals: 10 s (0.1 Hz), 2 s (0.5 Hz), 1 s (1 Hz), and 0.125 s (8 Hz). During each evaluation, the program obtains observations, updates its belief state, queries a policy, executes the action and then sleeps for the rest of the interval. For example, if it takes 0.1 s to finish the above sequence of steps, the program will then sleep 9.9 s for the 10-s interval experiment, 1.9 s in the 2-s interval experiment. For some policies (e.g., client/server, flat or symbolic Perseus policies), it may take longer than some intervals to finish a round. For example, it may take 0.3 s, which means that shorter intervals, such as 0.125 s, are infeasible. In such a case, the program raises a flag and "n.a." is indicated in the corresponding cells of Table II.

C. Results

The lacasa4.batt POMDP with 2880 states, six actions, and 72 observations is evaluated. The finite-state controller for this POMDP was computed using the POLICY2FSC method introduced in Algorithm 4 and contained 87 deterministic nodes. The flat policy was computed using SARSOP [13] and contained 27 α-vectors. The SARSOP policy was used in the flat experiment where the model is represented using sparse matrices. The symbolic Perseus planner [28] was used to compute a factored policy (symbolic Perseus in results) and the number of factored α-vectors was bounded to 27 in order to have the same number of α-vectors in both flat and factored methods.

The results for lacasa4.batt are in Table II where eight configurations from Table I are compared. Fig. 5 shows the same battery depletion results graphically. For every frequency and every configuration, two numbers are reported: the average time measured in minutes for the battery to deplete itself by 1% and the standard error of this average. Across all frequencies, the flat and symbolic Perseus policies are the most energy intensive policies, which is natural since they do both belief updates and policy queries on the phone. The symbolic

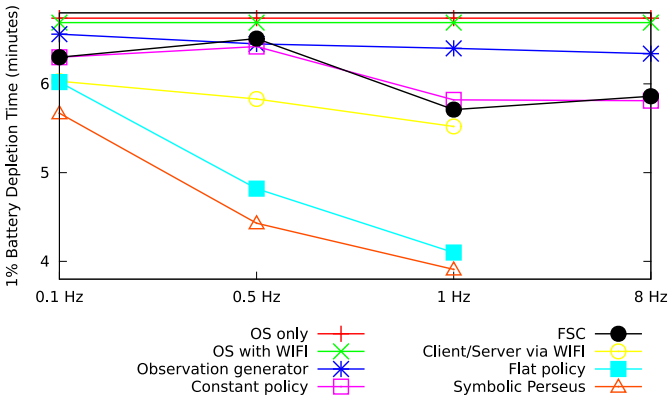

Fig. 5. Battery consumption results on the Nexus 4 phone. Missing values indicate that the policy did not return fast enough for a given frequency.

Perseus policy consumed more energy than the flat policy. The fact that this factored policy is more resource demanding than a flat policy represented by sparse α-vectors shows that a factored policy needs a substantial amount of structure (i.e., conditional and context-specific independence) to outweigh the overhead introduced by algebraic decision diagrams in order to consume less energy than a flat policy. The lacasa problem has a natural factored representation, but the amount of structure was not sufficient for the factored representation to payoff. The client/server solution was more battery efficient than both flat and factored policies since belief updates and policy queries happened on the dedicated server. The finite-state controller was the most energy efficient POMDP policy across all frequencies and with a frequency of 8 Hz, it is the only method that is sufficiently fast to return an action within 0.125 s (n.a. in columns that correspond to a frequency of 8 Hz means that the policy was too slow for the required frequency). The last column in Table II shows how long, on average, it takes to process one query for each policy. Here, the processing time of the finite-state controller is comparable with baselines that do not do any computation related to policy execution.

It should be noted that in the above experiments, both flat and symbolic Perseus policies had a considerably small number (27) of α-vectors. In many applications, this number is much larger (e.g., 100–10 000), which slows down the process of querying a policy since each policy query costs N dot products, where N is the number of α-vectors. The computational cost of querying a finite-state controller does not

depend on the size of the controller, which suggests that the gap between the results for finite-state controllers and policies based on α-vectors will widen as the number of α-vectors increases.

VI. Experiments on Controller Compilation and Compression

We evaluate our compilation and compression methods with policies computed by two state-of-the-art point-based POMDP algorithms: GapMin [12] and SARSOP [13]. GapMin returns $\langle \alpha_i, b_i, a_i \rangle$-tuples and, therefore, we can compile its policies into finite-state controllers using both of our compilation methods. SARSOP was used to compute policies for the largest POMDP benchmarks, however, it returns only α-vectors, which is sufficient to apply POLICY2FSC (Algorithm 4), but not ALPHA2FSC (Algorithm 1). Witness beliefs are also needed, but SAROP's interface does not expose them. The experiments are conducted with some benchmark problems and the lacasa POMDPs for smart phones. The running time of compilation algorithms—reported in the column time—corresponds to the time of actual compilation whereas the time to compute initial policies before compilation can be found in parentheses in the same column. High time limits were selected in order to compute policies of high quality. Thus, this time could be considerably shorter if one stops the planning algorithms as soon as a policy of sufficient quality is obtained. This could lead to a substantial reduction of the planning/initialization time since longer planning times (e.g., 10^4 s instead of 10^3 s in the case of SARSOP [13]) do not usually lead to dramatically improved policies.

A. Results

Tables III and IV compare the results obtained by compiling policies produced by GapMin and SARSOP, respectively, to five techniques that directly optimize controllers: BPI with escape [8], quadratically constrained linear programming for Moore (QCLP) [7] and Mealy automata (QCLP-Mealy) [29], EM with forward search [30], and branch&bound (B&B) with isomorph pruning [11]. POLICY2FSC was used in an iterative deepening fashion, starting from depth 2, up to a depth where the resulting controller was at least as good as the original policy or a time limit was exceeded. Hence, the time reported for POLICY2FSC is the cumulative time (seconds) to process all compilations from depth 2 up to the depth reported in column depth. Tree size is the size of the policy tree for that depth (note that edges with zero probability reduce the size of the policy tree considerably). Column nodes displays the number of nodes in the final controller after compression (before compression in the parentheses). Column value shows the value of controllers after compression (analogously, before compression in the parentheses). Column "c" indicates the number of iterations of the compression until there is no compression possible. QCLP and QCLP-Mealy experiments that did not complete on the NEOS[1] server are indicated with n.a. as well as experiments on EM and BPI when 3 GB of memory was not sufficient. A "*" besides the value of B&B indicates that B&B did not complete its search in 24 h and that the value reported is for the best controller found in 24 h. Since our POLICY2FSC procedure can be applied with any policy, we could also compile GapMin upper bound policies into finite-state controllers.

Table III compares our two compilation methods for policies computed by GapMin. Method GM-LB stands for POLICY2FSC applied to the GapMin lower bound policy whereas GM-UB to the upper bound policy. Note that on domains where QMDP [31] or the fast informed bound [31] are near optimal, POLICY2FSC can be used with such policies and no extra planning is required. Results confirm that our methods are successful in compiling POMDP policies into finite-state controllers of approximately equivalent quality. The highest value found for each problem is bolded. ALPHA2FSC compiles $|lb|$ α-vectors into controllers with similar value, though sometimes the value is significantly worse (e.g., lacasa4.batt and machine). In contrast, POLICY2FSC finds better controllers by simulating the input policy to a larger depth, but this takes more time. It was stopped as soon as the value of the controller matches GapMin's lower bound or 1 h was reached. In many cases, the number of nodes is still less than or equal to the size of the input policy (e.g., 4x5x2.95, cheese-taxi, lacasa2, machine). The direct optimization techniques (B&B, QCLP, QCLP-Mealy, EM, BPI) generally take much longer and/or do not consistently produce good controllers.

Table IV summarizes the results for some problems that are among the largest available benchmarks for point-based value iteration techniques that do not exploit factored representations. In this case, SARSOP was used to obtain a lower bound policy that is then compiled by POLICY2FSC. Even though SARSOP returned value functions with thousands of α-vectors, we compiled those policies into considerably smaller controllers (up to three orders of magnitude reduction) of the same or better quality (e.g., underwaterNav) demonstrating that our method scales to large problems. POLICY2FSC produced the best value for all problems except underwaterNav where the direct optimization techniques produced better controllers. This simply indicates that the policy compiled from SARSOP was not the best as opposed to any weakness in POLICY2FSC.

B. Stochastic Finite-State Controllers

The compilation and compression techniques tested in the previous section produce deterministic controllers, however, the space of stochastic controllers is larger and the optimal controller for a fixed number of nodes may be stochastic. In fact, some of the techniques that directly optimize controllers (e.g., BPI, QCLP, QCLP-Mealy, and EM) work in the space of stochastic controllers for this reason. In this section, we investigate to what extent the deterministic controllers found so far can be improved by expanding the search to stochastic controllers. We experiment with the compression technique in Algorithm 6 since it produces stochastic controllers that are potentially smaller and better than the deterministic controllers produced by Algorithm 5. We also initialized BPI with the

[1] http://www.neos-server.org which imposes the following restrictions: 3 GB RAM, 8 h CPU time, and 16 MB task size in AMPL.

TABLE III
COMPILATION OF GAPMIN POLICIES USING (1) ALPHA2FSC APPLIED TO GAPMIN LOWER BOUND α-VECTORS; (2) POLICY2FSC APPLIED TO GAPMIN LOWER BOUND POLICY (GM-LB); AND (3) POLICY2FSC APPLIED TO GAPMIN UPPER BOUND POLICY (GM-UB). THE NUMBERS IN PARENTHESES IN COLUMNS "NODES" AND "VALUE" INDICATE THE NUMBER OF NODES AND A VALUE OF CONTROLLERS BEFORE COMPRESSION. IN COLUMN "TIME," VALUES IN PARENTHESES SHOW THE TIME REQUIRED TO COMPUTE THE INITIAL GAPMIN POLICY THAT IS USED FOR COMPILATION BY ALPHA2FSC AND POLICY2FSC

POMDP	GapMin	method	depth	tree size	nodes	value	time	c
4x5x2.95	GM-lb=2.08	alpha2fsc			47 (58)	2.08 (2.08)	0.29 (4.96)	2
$\|S\|$=39, $\|A\|$=4	GM-ub=2.08	**GM-LB**	8	287	10 (17)	**2.08** (1.85)	0.30 (4.96)	1
$\|O\|$=4, $\gamma = 0.95$	$\|lb\|$=58	**GM-UB**	8	287	10 (17)	**2.08** (1.85)	0.29 (4.96)	1
	$\|ub\|$=243	B&B			5	2.02	639.9	
		EM			10	2.01 ± 0.02	66.8	
		QCLP			10	1.74 ± 0.11	7.7	
		QCLP-Mealy			5	1.51 ± 0.12	1.2	
		BPI			8	0.71 ± 0.09	0.72	
aloha.10	GM-lb=533.4	alpha2fsc			137 (158)	533.2 (533.2)	11.5 (5223)	1
$\|S\|$=30, $\|A\|$=9	GM-ub=544.1	**GM-LB**	11	29525	390 (1116)	**537.6** (537.5)	83.6 (5223)	2
$\|O\|$=3, $\gamma = 0.999$	$\|lb\|$=158	GM-UB	11	29525	402 (1148)	537.6 (537.6)	94.5 (5223)	1
	$\|ub\|$=406	B&B			10	529.0*	24h	
		EM			40	534.8 ± 0.25	2739	
		QCLP			25	534.37 ± 0.52	99.2	
		QCLP-Mealy			13	532.32 ± 1.8	142.7	
		BPI			5	112.4 ± 1.59	0.69	
chainOfChains3	GM-lb=157	**alpha2fsc**	10	10	10 (10)	**157** (157)	0.26 (0.86)	0
$\|S\|$=10, $\|A\|$=4	GM-ub=157	**GM-LB**	11	11	10 (10)	**157** (157)	0.42 (0.86)	0
$\|O\|$=1, $\gamma = 0.95$	$\|lb\|$=10	**GM-UB**	11	11	10 (10)	**157** (157)	0.26 (0.86)	0
	$\|ub\|$=1	**B&B**			10	**157**	1.69	
		EM			10	0.17 ± 0.06	6.9	
		QCLP			10	0 ± 0	0.16	
		QCLP-Mealy			10	33.1 ± 2.3	0.1	
		BPI			10	25.7 ± 0.77	4.25	
cheese-taxi	GM-lb=2.481	**alpha2fsc**			17 (22)	**2.476** (2.476)	0.29 (1.88)	1
$\|S\|$=34, $\|A\|$=7	GM-ub=2.481	**GM-LB**	15	167	17 (24)	**2.476** (2.476)	0.56 (1.88)	1
$\|O\|$=10, $\gamma = 0.95$	$\|lb\|$=22	**GM-UB**	15	167	17 (24)	**2.476** (2.476)	0.55 (1.88)	1
	$\|ub\|$=13	B&B			10	-19.9*	24h	
		EM			17	-12.16 ± 2.08	337.9	
		QCLP			17	-18.22 ± 1.77	227.4	
		QCLP-Mealy			9	-10.1 ± 3.9	578.2	
		BPI			16	-18.1 ± 0.39	7.18	
lacasa2a	GM-lb=6714.6	alpha2fsc			5 (5)	6714.0 (6714.0)	5.15 (54)	0
$\|S\|$=320, $\|A\|$=4	GM-ub=6717.6	**GM-LB**	5	22621	106 (421)	**6715.0** (6715.0)	933.9 (54)	1
$\|O\|$=12, $\gamma = 0.95$	$\|lb\|$=5	GM-UB	5	22621	100 (517)	6714.1 (6714.1)	256.4 (54)	1
	$\|ub\|$=14	B&B			3	6710.0	493.8	
		EM			11	6710 ± 0.11	6485	
		QCLP			2	6699.9 ± 5.5	181	
		QCLP-Mealy			2	6714.6 ± 0.0	5497	
		BPI			26	6709.3 ± 0.2	121.5	
lacasa3.batt	GM-lb=293.4	alpha2fsc			25 (26)	292.4 (292.4)	399.7 (5386)	1
$\|S\|$=1920, $\|A\|$=6	GM-ub=294.7	GM-LB	4	12601	47 (60)	293.1 (292.7)	1451 (5386)	2
$\|O\|$=36, $\gamma = 0.95$	$\|lb\|$=26	**GM-UB**	4	12697	41 (48)	**293.2** (293.1)	1030 (5386)	2
	$\|ub\|$=48	B&B			5	287.0*	24h	
		EM			5	**293.2** ± 0.03	13331	
		QCLP			n.a.	n.a.	n.a.	
		QCLP-Mealy			n.a.	n.a.	n.a.	
		BPI			9	**293.2** ± 0.12	2102	
lacasa4.batt	GM-lb=291.1	alpha2fsc			10 (10)	285.5 (285.5)	302 (8454)	0
$\|S\|$=2880, $\|A\|$=6	GM-ub=292.6	GM-LB	3	745	19 (22)	287.3 (287.1)	3652 (8454)	1
$\|O\|$=72, $\gamma = 0.95$	$\|lb\|$=10	**GM-UB**	4	23209	87 (94)	**290.8** (290.8)	3681 (8454)	1
	$\|ub\|$=23	B&B			10	285.0*	24h	
		EM			3	290.2 ± 0.0	19920	
		QCLP			n.a.	n.a.	n.a.	
		QCLP-Mealy			n.a.	n.a.	n.a.	
		BPI			6	290.6 ± 0.2	4124	
hhepis6obs_woNoise	GM-lb=8.64	**alpha2fsc**			14 (18)	**8.64** (8.64)	0.49 (2.6)	1
$\|S\|$=20, $\|A\|$=4	GM-ub=8.64	**GM-LB**	12	21	14 (18)	**8.64** (8.64)	0.89 (2.6)	1
$\|O\|$=6, $\gamma = 0.99$	$\|lb\|$=18	**GM-UB**	12	21	14 (18)	**8.64** (8.64)	0.74 (2.6)	1
	$\|ub\|$=7	**B&B**			8	**8.64**	4.48	
		EM			14	0.0 ± 0.0	49.2	
		QCLP			14	0.16 ± 0.10	26	
		QCLP-Mealy			7	0.81 ± 0.0	7	
		BPI			13	0.0 ± 0.0	1.68	
machine	GM-lb=62.38	alpha2fsc			5 (39)	54.61 (54.09)	5.53 (3784)	1
$\|S\|$=256, $\|A\|$=4	GM-ub=66.32	GM-LB	9	376	26 (41)	62.92 (62.84)	18.5 (3784)	1
$\|O\|$=16, $\gamma = 0.99$	$\|lb\|$=39	**GM-UB**	12	2864	11 (159)	**63.02** (60.29)	86.8 (3784)	2
	$\|ub\|$=243	B&B			6	62.6	52100	
		EM			11	62.93 ± 0.03	1757	
		QCLP			11	62.45 ± 0.22	4636	
		QCLP-Mealy			9	53.69 ± 9.21	17011	
		BPI			10	35.7 ± 0.52	2.14	

deterministic controllers found in previous experiments to see whether it could improve them by searching in the space of stochastic controllers.

The results of the evaluation are shown in Table V, where all deterministic FSCs from the previous section are reported in the first group of columns named deterministic FSC. Those

TABLE IV
COMPILATION AND COMPRESSION OF SARSOP POLICIES. THE NUMBERS IN PARENTHESES IN COLUMNS NODES AND VALUE INDICATE THE NUMBER OF NODES AND A VALUE OF CONTROLLERS BEFORE COMPRESSION. IN COLUMN TIME, VALUES IN PARENTHESES SHOW THE TIME REQUIRED TO COMPUTE THE INITIAL SARSOP POLICY THAT IS USED FOR COMPILATION BY POLICY2FSC

POMDP	SARSOP	method	depth	tree size	nodes	value	time	c
baseball	$\|\alpha\|$ =1415	**policy2fsc**	7	175985	10 (47)	**0.641** (0.641)	78.22 (122.7)	1
$\|S\|$=7681, $\|A\|$=6	UB=0.642	B&B			5	0.636*	24h	
$\|O\|$=9, $\gamma = 0.999$	LB=0.641	EM			2	0.636 ± 0.0	48656	
		QCLP			n.a.	n.a.	n.a.	
		QCLP-Mealy			n.a.	n.a.	n.a.	
		BPI			9	0.636 ± 0.0	445	
elevators_inst_pomdp_1	$\|\alpha\|$ =78035	**policy2fsc**	11	419	20 (24)	**-44.41** (-44.41)	1357 (11228)	1
$\|S\|$=8192, $\|A\|$=5	UB=-44.31	B&B			10	-149.0*	24h	
$\|O\|$=32, $\gamma = 0.99$	LB=-44.32	EM			n.a.	n.a.	n.a.	
		QCLP			n.a.	n.a.	n.a.	
		QCLP-Mealy			n.a.	n.a.	n.a.	
		BPI			n.a.	n.a.	n.a.	
tagAvoid	$\|\alpha\|$ =20326	**policy2fsc**	28	7678	91 (712)	**-6.04** (-6.04)	582.2 (10073)	1
$\|S\|$=870, $\|A\|$=5	UB=-3.42	B&B			10	-19.9*	24h	
$\|O\|$=30, $\gamma = 0.95$	LB=-6.09	EM			9	-6.81 ± 0.12	19295	
		QCLP			2	-19.99 ± 0.0	12.9	
		QCLP-Mealy			2	-9.46 ± 0.81	15798	
		BPI			88	-12.42 ± 0.13	1808	
underwaterNav	$\|\alpha\|$ =26331	policy2fsc	51	1242	52 (146)	745.3 (745.3)	5308 (10222)	1
$\|S\|$=2653, $\|A\|$=6	UB=753.8	B&B			10	747.0*	24h	
$\|O\|$=103, $\gamma = 0.95$	LB=742.7	**EM**			5	**749.9** ± 0.02	31611	
		QCLP			n.a.	n.a.	n.a.	
		QCLP-Mealy			n.a.	n.a.	n.a.	
		BPI			49	748.6 ± 0.24	14758	
rockSample-7_8	$\|\alpha\|$ =12561	**policy2fsc**	31	2237	204 (224)	**21.58** (21.58)	1291 (10629)	1
$\|S\|$=12545, $\|A\|$=13	UB=24.22	B&B			10	11.9*	24h	
$\|O\|$=2, $\gamma = 0.95$	LB=21.50	EM			n.a.	n.a.	n.a.	
		QCLP			n.a.	n.a.	n.a.	
		QCLP-Mealy			n.a.	n.a.	n.a.	
		BPI			5	7.35 ± 0.0	78.8	

TABLE V
DETERMINISTIC FINITE-STATE CONTROLLERS FROM TABLES III AND IV (SHOWN HERE IN THE "DETERMINISTIC FSC" COLUMN) CONVERTED TO STOCHASTIC FINITE-STATE CONTROLLERS; THE RESULT IS NOT SHOWN WHEN NEITHER THE SIZE WAS REDUCED NOR THE QUALITY OF THE STOCHASTIC CONTROLLER WAS INCREASED. COLUMNS "STOCHASTIC FSC" ARE FOR STOCHASTIC CONTROLLERS OBTAINED USING ALGORITHM 6 THAT DETECTS NODES DOMINATED BY MORE THAN ONE NODE AND THEN REMOVES THEM. COLUMNS "FSC FROM BPI" SHOW STOCHASTIC FINITE-STATE CONTROLLERS OBTAINED WHEN BPI [8] IS INITIALIZED WITH CORRESPONDING DETERMINISTIC CONTROLLERS. THE VALUES ARE PRINTED IN BOLD WHEN STOCHASTIC CONTROLLERS YIELD CONSIDERABLE IMPROVEMENT

	deterministic FSC			stochastic FSC					FSC from BPI				
POMDP	compil.	det-FSC	det-FSC	stoch-FSC	stoch-FSC	time	% stoch	% stoch	BPI	BPI	time	% stoch	% stoch
POMDP	method	value	# nodes	value	# nodes	(sec)	actions	edges	value	# nodes	(sec)	actions	edges
aloha.10	alpha2fsc	533.2	137	533.2	137	8.0	0	0	533.3 ± 0	137 ± 0	6.4 ± 0	0.73	0.16
aloha.10	GM-LB	537.6	390	537.7	280	54.0	0	20.6	537.7 ± 0	390 ± 0	78.2 ± 0.9	9.11	6.12
aloha.10	GM-UB	537.6	402	537.7	260	62.0	0	23.1	537.7 ± 0	402 ± 0	88.9 ± 2.5	9.78	6.8
lacasa2a	alpha2fsc	6714.0	5	6714.6	5	3.4	0	0	**6715.6** ± 0	17 ± 0	320.6 ± 7	38.6	8.0
lacasa2a	GM-LB	6715.0	106	6715.6	101	96.0	0	1.5	**6715.7** ± 0	118 ± 0	3209 ± 103	22.3	8.6
lacasa2a	GM-UB	6714.1	100	6714.7	90	99.9	0	1.4	**6715.7** ± 0	112 ± 0	3864 ± 68.1	23.1	8.8
lacasa3.batt	alpha2fsc	292.4	25	292.5	24	280.2	0	0.1	**293.4** ± 0	37 ± 0	2862 ± 64	20.8	3.7
lacasa3.batt	GM-LB	293.1	47	293.2	43	595.0	0	0.8	293.5 ± 0	59 ± 0	5248 ± 0	18.6	4.2
lacasa3.batt	GM-UB	293.2	41	293.2	36	448.0	0	1.4	293.5 ± 0	53 ± 0	4885 ± 119	31.0	4.4
lacasa4.batt	alpha2fsc	285.5	10	285.5	10	175.8	0	0	**291.4** ± 0	14 ± 0.4	15871 ± 1074	3.87	0.0
lacasa4.batt	GM-LB	287.3	19	**289.2**	17	620.1	0	0.2	**291.3** ± 0	23 ± 1	6195 ± 1685	30.8	2.9
lacasa4.batt	GM-UB	290.8	87	290.8	86	3015	0	0.1	**291.4** ± 0	87 ± 0	15369 ± 946	26.3	3.1
machine	alpha2fsc	54.61	5	54.62	5	0.8	0	0	**62.76** ± 0	17 ± 0	167 ± 5	22.4	1.4
machine	GM-LB	62.92	26	62.93	26	3.7	0	0	62.95 ± 0	34 ± 0.4	165 ± 7	0.0	0.33
machine	GM-UB	63.02	11	63.03	11	1.2	0	0	63.03 ± 0	11 ± 0	0.64 ± 0	0.0	0.0
rockSample-7_8	policy2fsc	21.59	204	21.59	198	2202	0	0	21.59 ± 0	204 ± 0	6965 ± 82	0.0	0.0

results are repeated from Tables III and IV for convenience. The columns under stochastic FSC and FSC from BPI show, respectively the results of the compression technique described in Algorithm 6 and BPI [8] initialized with the deterministic controllers of the first set of columns.

Before the results are discussed in detail, we note that our deterministic finite-state controllers are already optimal for several POMDPs, i.e., for 4x5x2.95, chainOfChains3, hhepis6obs_woNoise and no improvement is expected from stochastic finite-state controllers in terms of value, although the stochastic controllers can be smaller due to the use of stochastic actions and/or stochastic edges. The percentage of stochastic actions and stochastic edges for each stochastic controller is shown in the columns "% stochastic actions" and "% stochastic edges."

Let us first look at the columns under stochastic FSC in Table V. A comparison with deterministic FSCs shows that there is only one case where removing nodes jointly dominated by several nodes yields a nonnegligible increase in the quality of the policy—lacasa4.batt GM-LB. For POMDPs where deterministic controllers are optimal, the compression procedure did not remove any node except for rockSample-7_8.

For POMDPs, where the deterministic controllers are not optimal, the procedure reduced the size of some controllers by introducing stochastic nodes. The biggest reduction in controller size was for aloha.10 GM-LB and GM-UB.

The second group of results under FSC from BPI shows the average of 21 executions of BPI initialized with our deterministic FSCs. BPI is a planning algorithm that can improve our deterministic policies with its optimization module that creates improved nodes in the controller leading to controllers of larger size when existing nodes are not dominated by the new nodes. BPI improved most policies for the LaCasa POMDPs where both a larger numbers of nodes in the stochastic controllers, and stochastic actions and edges were utilized. In other cases, e.g., aloha.10, even if the quality of the policy could not be improved, the size of the stochastic finite-state controllers is substantially smaller benefiting from stochastic actions and edges.

Overall, our thorough evaluation of methods for stochastic finite-state controllers indicates that deterministic finite-sate controllers are usually very good. This is in line with Hansen [32] who also found that the stochastic FSCs found by BPI are mostly deterministic. Our methods for controller compilation depend on policies computed by point-based value iteration or other methods. If these algorithms are improved and yield better policies, then our compilation and compression techniques will naturally produce better controllers. Since point-based solvers do not always find near optimal policies for large problems, a translation to stochastic controllers can be seen as a way to potentially improve the quality (and the size) of policies before deployment in practice.

VII. Discussion and Related Work

Before we conclude, we discuss several relevant lines of research that were not critical to understand this paper, yet they examine similar challenges and apply related algorithmic approaches.

The goal of this paper was to compile existing policies into FSCs where a policy can be queried on demand for different execution trajectories. There is a growing interest—especially within theoretical computer science and computational linguistics—in the related problem of automata identification [33]. Those communities investigate a more challenging problem than policy compilation because the number of trajectories may be large, those trajectories may contain repeated transitions, yet not cover all possible realizations of the automaton, and the process of merging those trajectories introduces an additional computational challenge when constructing or approximating a policy tree [33]. Trajectory and sample-based approaches become useful for POMDP planning, however, when the policy is not available as shown in [34] where FSCs are compiled from a set of trajectories sampled according to an exploration policy (e.g., a random policy). A large set of trajectories is sampled first, and after that the trajectories that yield highest rewards are considered to be realizations of the desirable policy, and they are merged into a FSC. A similar approach with a richer policy representation was investigated subsequently in [35].

The algorithm in [34] seeks a policy, and it tries to compile it into a FSC at the same time whereas the transition probabilities of the underlying Markov decision process are available, and they can be used at all stages of the algorithm. In contrast, reinforcement learning aims to find a policy—which can be in the form of a FSC as well—when transition probabilities of the Markov process are not known [36]. The absence of those probabilities introduces yet another level of complexity. To this end, various reinforcement learning algorithms have been proposed as shown in [37]. A particular approach that is relevant to this discussion is batch reinforcement learning where the algorithm has access to a set of trajectories sampled from a real or a simulated environment [38]. One of the most recent implementations of this approach for POMDPs was proposed in [39] and [40]. The combination of a missing policy that is sought, missing transition probabilities, and partial observability make the task particularly difficult. As a result, the current algorithms are most efficient when rewards of the underlying Markov decision process are sparse, i.e., rewards are zero for most states except for the goal state.

VIII. Conclusion

We presented two novel methods to compile policies for POMDPs into approximately equivalent FSCs. Our motivation is that these FSC representations are very useful in resource-constrained applications such as on mobile or wearable devices. Methods that can create FSC policies open up new possibilities for using POMDP controllers on these devices, where battery, computation, and memory resources are at a premium. We showed how we can get very compact policy representations that are equivalent to those generated by two state-of-the-art offline planners. We also performed battery consumption experiments on a real device—Nexus 4 smartphone—which confirmed that finite-state controllers are the most energy efficient POMDP policies and they can increase battery life significantly in comparison with existing methods. The battery consumption experiment was set up in a way that was challenging for finite-state controllers because alternative policies had a small number of α-vectors. For larger POMDPs, the number of α-vectors will likely increase, which will also increase the performance gap and finite-state controllers will display further battery savings. Increases in battery life are always welcome by users of mobile devices.

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments and C. Amato for help with running the QCLP algorithm.

References

[1] J. D. Williams and S. Young, "Scaling POMDPs for spoken dialog management," *IEEE Audio, Speech, Language Process.*, vol. 15, no. 7, pp. 2116–2129, Sep. 2007.

[2] J. Hoey *et al.*, "People, sensors, decisions: Customizable and adaptive technologies for assistance in healthcare," *ACM Trans. Interact. Intell. Syst.*, vol. 2, no. 4, p. 20, Dec. 2012.

[3] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa, "Online planning algorithms for POMDPs," *J. Artif. Intell. Res.*, vol. 32, no. 1, pp. 663–704, May 2008.

[4] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *Proc. Adv. Neural Inform. Process. Syst. (NIPS)*, pp. 2164–2172, 2010.
[5] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, nos. 1–2, pp. 99–134, 1998.
[6] D. Braziunas and C. Boutilier, "Stochastic local search for POMDP controllers," in *Proc. 19th Nat. Conf. Artif. Intell. (AAAI)*, 2004, pp. 690–696. [Online]. Available: citeseer.ist.psu.edu/braziunas04stochastic.html
[7] C. Amato, D. Bernstein, and S. Zilberstein, "Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs," *J. Auton. Agents Multi-Agent Syst.*, vol. 21, no. 3, pp. 293–320, 2009.
[8] P. Poupart and C. Boutilier, "Bounded finite state controllers," in *Proc. Adv. Neural Inform. Process. Syst. (NIPS)*, 2003, pp. 823–830.
[9] M. Toussaint, S. Harmeling, and A. Storkey, "Probabilistic inference for solving (PO)MDPs," School Inform., Univ. Edinburgh, Edinburgh, U.K., Tech. Rep. EDI-INF-RR-0934, 2006.
[10] N. Meuleau, K.-E. Kim, L. P. Kaelbling, and A. R. Cassandra, "Solving POMDPs by searching the space of finite policies," in *Proc. 15th Conf. Uncertainty Artif. Intell. (UAI)*, 1999, pp. 417–426.
[11] M. Grześ, P. Poupart, and J. Hoey, "Isomorph-free branch and bound search for finite state controllers," in *Proc. 23rd Int. Joint Conf. Artif. Intell. (IJCAI)*, 2013, pp. 2282–2290.
[12] P. Poupart, K.-E. Kim, and D. Kim, "Closing the gap: Improved bounds on optimal POMDP solutions," in *Proc. 21st Int. Conf. Autom. Plan. Sched. (ICAPS)*, 2011, pp. 194–201.
[13] H. Kurniawati, D. Hsu, and W. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Proc. Robot. Sci. Syst. (RSS)*, 2008, pp. 65–72.
[14] E. Hansen, "An improved policy iteration algorithm for partially observable MDPs," in *Proc. Adv. Neural Inform. Process. Syst. (NIPS)*, 1998, pp. 1015–1021.
[15] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proc. 23rd Int. Joint Conf. Artif. Intell. (IJCAI)*, 2003, pp. 1025–1032.
[16] M. T. J. Spaan and N. Vlassis, "Perseus: Randomized point-based value iteration for POMDPs," *J. Artif. Intell. Res.*, vol. 24, no. 1, pp. 195–220, 2005.
[17] G. Shani, R. I. Brafman, and S. E. Shimony, "Prioritizing point-based POMDP solvers," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 38, no. 6, pp. 1592–1605, Dec. 2008.
[18] G. Shani, "Evaluating point-based POMDP solvers on multicore machines," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 40, no. 4, pp. 1062–1074, Aug. 2010.
[19] G. Shani, J. Pineau, and R. Kaplow, "A survey of point-based POMDP solvers," *J. Auton. Agents Multi-Agent Syst.*, vol. 27, no. 1, pp. 1–51, 2013.
[20] C. Amato, D. Bernstein, and S. Zilberstein, "Solving POMDPs using quadratically constrained linear programs," in *Proc. 20th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2007, pp. 2418–2424.
[21] E. A. Hansen, "Finite-memory control of partially observable systems," Ph.D. dissertation, Dept. Comp. Sci., Univ. Massachusetts Amherst, Amherst, MA, USA, 1998.
[22] E. Sondik, "The optimal control of partially observable decision processes over the infinite horizon: Discounted cost," *Oper. Res.*, vol. 26, no. 2, pp. 282–304, 1978.
[23] T. Smith and R. Simmons, "Point-based POMDP algorithms: Improved analysis and implementation," in *Proc. Uncertainty Artif. Intell. (UAI)*, 2005, pp. 542–555.
[24] V. Fischer and M. Drutarovsky, "True random number generator embedded in reconfigurable hardware," in *Proc. 4th Int. Workshop Cryptographic Hardware Embedded Syst. (CHES)*, 2002, pp. 415–430.
[25] R. Davies, "Hardware random number generators," presented at the 15th Australian Statistics Conference, Adelaide, SA, Australia, 2000.
[26] J. Hoey, X. Yang, E. Quintana, and J. Favela, "LaCasa: Location and context-aware safety assistant," in *Proc. 6th Int. Conf. Pervasive Comp. Techn. Healthcare*, San Diego, CA, USA, 2012, pp. 171–174.
[27] M. Grześ *et al*., "Relational approach to knowledge engineering for POMDP-based assistance systems as a translation of a psychological model," *Int. J. Approx. Reason.*, vol. 55, no. 1, pp. 36–58, 2014.
[28] P. Poupart, "Exploiting structure to efficienty solve large scale partially observable Markov decision processes," Ph.D. dissertation, Dept. Comp. Sci., Univ. Toronto, Toronto, ON, Canada, 2005.
[29] C. Amato, B. Bonet, and S. Zilberstein, "Finite-state controllers based on Mealy machines for centralized and decentralized POMDPs," in *Proc. 24th Conf. Artif. Intell. (AAAI)*, 2010, pp. 1052–1058.
[30] P. Poupart, T. Lang, and M. Toussaint, "Analyzing and escaping local optima in planning as inference for partially observable domains," in *Proc. Eur. Conf. Mach. Learn. Knowl. Discovery Databases (ECML/PKDD)*, 2011, pp. 613–628.
[31] M. Hauskrecht, "Value-function approximations for partially observable Markov decision processes," *J. Artif. Intell. Res.*, vol. 13, no. 1, pp. 33–94, 2000.
[32] E. A. Hansen, "Sparse stochastic finite-state controllers for POMDPs," in *Proc. 24th Conf. Uncertainty Artif. Intell. (UAI)*, 2008, pp. 256–263.
[33] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009.
[34] C. Amato and S. Zilberstein, "Achieving goals in decentralized POMDPs," in *Proc. 8th Int. Conf. Auton. Agents Multiagent Syst.*, 2009, pp. 593–600.
[35] J. Pajarinen and J. Peltonen, "Periodic finite state controllers for efficient POMDP and DEC-POMDP planning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 2636–2644.
[36] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA, USA: Athena Scientific, 1996.
[37] R. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[38] S. Lange, T. Gabel, and M. Riedmiller, "Batch reinforcement learning," in *Reinforcement Learning: State of the Art*, M. Wiering and M. van Otterlo, Eds. Berlin, Germany: Springer, 2011.
[39] H. Li, X. Liao, and L. Carin, "Multi-task reinforcement learning in partially observable stochastic environments," *J. Mach. Learn. Res.*, vol. 10, pp. 1131–1186, Jan. 2009.
[40] M. Liu, X. Liao, and L. Carin, "The infinite regionalized policy representation," in *Proc. Int. Conf. Mach Learn.*, Bellevue, WA, USA, 2011, pp. 769–776.

Marek Grześ received the M.Sc. degree in software engineering from the Bialystok University of Technology, Bialystok, Poland, and the Ph.D. degree in computer science from the University of York, York, U.K.

He is a Post-Doctoral Research Fellow with the University of Waterloo, Waterloo, ON, Canada. His current research interests include artificial intelligence, decision-theoretic planning, probabilistic reasoning, and applications thereof.

Pascal Poupart received the B.Sc. degree in mathematics and computer science from McGill University, Montreal, QC, Canada, in 1998, and the M.Sc. and Ph.D. degrees in computer science from the University of British Columbia, Vancouver, BC, Canada, and the University of Toronto, Toronto, ON, Canada, in 2000 and 2005, respectively.

He is an Associate Professor with the Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada. His current research interests include the development of algorithms for decision-theoretic planning and machine learning with application to assistive technologies and natural language processing.

Xiao Yang received the M.Math. degree from the University of Waterloo, Waterloo, ON, Canada, supervised by Prof. Hoey.

He is a Software Development Engineer at Amazon, Toronto, ON, Canada. His current research interests include leveraging the advantages of mobile devices in health informatics.

Jesse Hoey received the B.Sc. degree in physics from McGill University, Montreal, QC, Canada, the M.Sc. degree in physics, and the Ph.D degree in computer science from the University of British Columbia in Vancouver, BC, Canada, in 1992, 1995, and 2004, respectively.

He was a Post-Doctoral Researcher at the Department of Computer Science and Occupational Science and Occupational Therapy, University of Toronto, Toronto, ON, Canada, from 2004 to 2006. From 2004 to 2010, he was a Lecturer (assistant professor) at the School of Computing, University of Dundee, Dundee, Scotland. He is an Associate Professor with the Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada. He is also an Adjunct Scientist with the Toronto Rehabilitation Institute, Toronto, ON, Canada, where he is currently a Co-Leader of the AI and Robotics Research Team. His current research interests include artificial intelligence and health informatics.