Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)

Efficient Robot Planning for Achieving Multiple Independent Partially Observable Tasks That Evolve over Time

Anahita Mohseni-Kabir,[1] **Manuela Veloso,**[1] **Maxim Likhachev**[1]
[1]School of Computer Science, Carnegie Mellon University
{anahitam, maxim, mmv}@cs.cmu.edu

Abstract

We focus on domains where a robot is required to accomplish a set of tasks that are partially observable and evolve independently of each other according to their dynamics. An example domain is a restaurant setting where a robot waiter should take care of an ongoing stream of tasks, namely serving a number of tables, including delivering food to the tables and checking on customers. An action that the robot should take next at any point of time typically depends on the duration of possible actions, the state of each table, and how these tables evolve over time, *e.g.*, the food becomes cold after a few time steps. As most of these domains are dynamic and tasks are frequently being added and removed, the robot typically needs to plan for a short h-step horizon to quickly decide on the next action and replans at each time step. A conventional approach to deal with this problem is to combine all the tasks' states and robot actions into one large model and to compute an h-step optimal policy for this combined model. For the problems that we are interested in, the number of tasks, *e.g.*, the number of tables in the restaurant domain, can be large making this planning approach computationally impractical. The observation that we make however is that in many domains the number of tasks that the robot can accomplish within h-steps is very limited. We present an algorithm that exploits this observation and decomposes the problem into a series of much smaller planning problems, the solution to which gives us an optimal solution. We demonstrate the efficiency of our algorithm on the restaurant domain.

Introduction

Many robotics applications, such as waiting tables in a restaurant and robots in search and rescue, involve a robot acting in a stochastic environment under partial observability while completing multiple independent tasks. For instance, consider the restaurant setting where a robot is waiting multiple tables. The robot waiter should take care of an ongoing stream of tasks, including taking food orders, delivering food and drinks, and checking on customers. The tasks are partially observable as the robot cannot directly observe what people want and their degree of satisfaction. Although in such domains the dynamics of each task is independent of one another, they all share a single robot that attends to them, and an optimal policy for the robot should consider all tasks at each step. Many of these multi-task domains can be very dynamic with tasks being added and removed at each time step, *e.g.*, a customer leaves or a table wants to order an extra dish. Thus, the robot only needs to plan for a short horizon of actions as any long-term plan quickly becomes sub-optimal or even infeasible after a few time steps.

A conventional planning approach for such domains is to combine all the tasks' states and actions into one large model and compute the h-step optimal policy for the combined model at each time step. However, this approach is impractical if the number of tasks are large. In this work, we leverage the structure of the problem, namely the independency between the tasks, and the observation that given a short horizon only a subset of tasks can be accomplished within this horizon.

We utilize Partially Observable Markov Decision Process (POMDP) representation. POMDP is a powerful mathematical tool to model the robot's sequential decision making under uncertainty (Cassandra 1998). However, planning algorithms for POMDPs can only handle small state and action spaces and consequently do not scale well as the number of tasks increase. The combined model of the robot and all tasks grow as the number of tasks grow.

Many works have proposed approaches to speed up POMDP solvers by using point-based methods (Shani, Pineau, and Kaplow 2013), hierarchical planning (Theocharous and Kaelbling 2004), clustering and compression of belief space (Roy, Gordon, and Thrun 2005; Smith, Thompson, and Wettergreen 2007), factored representation (Shani et al. 2008), and online POMDP approaches (Ross et al. 2008). We leverage online POMDP approaches which only compute the optimal policy for the current information state and a small horizon of contingency plans. We are interested in domains in which a robot has to attend to multiple independent tasks whereas the above approaches do not make the independency assumption and address the combined model directly.

Our algorithm decomposes the problem into a series of smaller planning problems. In particular, a robot attending to a single task can be represented as a standalone smaller

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

POMDP. We show how to compute lower and upper bounds on the cost of an optimal solution involving N tasks. Using these insights, we develop an algorithm that searches over possible subsets of N tasks, solving each optimally until a provably globally optimal solution is found. We test our approach on a simplified restaurant environment in simulation. We present how we model the waiting table task as a robot planning problem and show the effectiveness of our approach compared to the combined model, a hierarchical POMDP approach, and a related paper (Shani 2013).

The paper is structured as follows. We begin by discussing the difference between our current work and the related work. We then introduce a motivating example, namely the restaurant domain. We then describe our notation, provide a pseudo code for the algorithm, and prove the optimality of the approach. Finally, we discuss the performance of the algorithm compared to the existing approaches.

Related Work

There is extensive research on speeding up POMDP solvers using different variations of the point-based value iteration method (Pineau et al. 2003). These methods (Shani, Pineau, and Kaplow 2013) focus on various aspects of the point-based value iteration, specifically the selection of the belief space subset and the order of value function updates of the belief space. It has been shown that these approaches generate good, approximate policies for large domains.

Other methods for scaling up POMDPs leverage factored representations in the form of decision trees (Boutilier and Poole 1996) or graphs (Bahar et al. 1997; Shani et al. 2008), specifically Algebraic Decision Diagrams (ADDs). Even though ADDs expedite planning by utilizing the limited dependencies between the state variables, they fail to compactly represent the POMDP when the policy is dependent on all the variables (Shani 2013).

Some research for scaling up POMDPs compresses the state space (Poupart and Boutilier 2003; Roy, Gordon, and Thrun 2005) by mapping each high-dimensional belief state into low-dimensional compressed belief or by bounding the number of non-zero values within each belief point (Wray and Zilberstein 2017). (Li, Cheung, and Liu 2009) proposes an approach to cluster belief states and combine it with belief compression to further improve POMDP tractability.

Research on hierarchical POMDP planning (HPOMDP) includes learning how to perform a set of subtasks independently, and then learning a high-level policy to sequence the subtasks (Theocharous and Kaelbling 2004). A method (Foka and Trahanias 2007) focuses on clustering the belief space to decompose a flat POMDP into a HPOMDP that has coarser discretization at higher levels for both state and action space and then solving the HPOMDP. Another method uses hierarchical finite-state controllers to scale-up planning and to provide theoretical guarantees on the quality of the computed policy (Hansen and Zhou 2003). Different from all these approaches, the tasks in our domains are independent and the robot can execute them in any order, *i.e.*, no task provides a precondition for another task. Our approach not only considers executing the tasks in a sequence but also interleaving the tasks' execution when more rewarding.

All the above approaches expedite planning in some way by using approximation methods, using compression or clustering techniques, or assuming some factored or hierarchical structure in the domain. We consider a structure in our domain, namely the independency among the tasks, that differs from the assumptions made in the previous methods. Our approach can leverage the above methods to further expedite planning when finding solutions for subsets of tasks.

The restless multi-arm bandit problem (RMAB) (Whittle 1988; Weber and Weiss 1990) concerns the optimal allocation of resources over time among a collection of bandits (or tasks) which are in competition. At each time step, an algorithm should decide which bandits should be active, *i.e.*, follow their optimal policy, and which bandits should be passive, *i.e.*, evolve to a new state. The optimization is to find a policy for sequential selection of active bandits. These problems can be modeled as Markov Decision Process (MDP), are intractable (Glazebrook, Ruiz-Hernandez, and Kirkbride 2006), and have shown to have near-optimal heuristic solutions on real-world problems (Liu and Zhao 2010; Deo et al. 2013). Differently, our approach 1) takes into account the partial observablity of the state space, 2) is fast and provably optimal given a short horizon, and 3) does not limit the robot's action to be passive or active; the robot's action set is a union of all POMDPs' action sets.

The closest work to ours is by (Shani 2013), who developed an algorithm to decompose a factored POMDP into a set of restricted POMDPs (or tasks). They solve each identified task separately, create a set of models with all possible combinations of the subsets of the tasks, solve them, and combine the policies of the smaller models into a policy for the complete model. Their work mostly focuses on how to decompose a factored POMDP into a set of smaller POMDPs, whereas our work assumes that the robot has a predefined set of tasks, efficiently removes subsets of tasks that have a low solution quality, and then creates a set of smaller models from the remaining subsets and solves them. We compare against this method in the experiments section.

Motivating Domain: Waiting Tables in a Restaurant

As illustrated in Fig. 1, we consider a restaurant setting with one robot and multiple tables which go through the dining process independently of each other. The robot has its own state variables such as position and can perform services such as *go to a table*, and *deliver food to a table*. The robot can only execute one action at each time that depends on the duration of possible actions, the state of each table, and how these tables evolve, *e.g.*, the table becomes unsatisfied if it is not served soon. The state of the table can include both observable variables such as wait time and partially observable variables such as satisfaction. To enable the robot to perform the waiting tables task, we model each table, the state of the robot and the actions that can be applied to this particular table as a POMDP. This POMDP representation enables the robot to reason about the uncertainty in humans' internal state and its own actions and how the dining process evolves over time after a sequence of action executions.

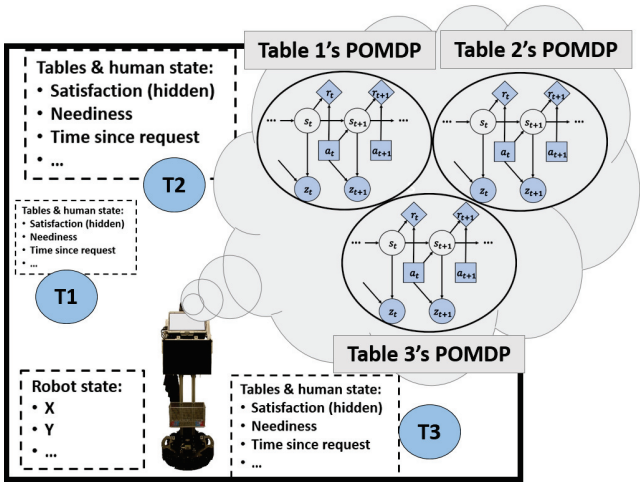

Figure 1: The robot operates in a restaurant with 3 tables, *T1*, *T2*, and *T3*. The robot builds a POMDP model for each table in the restaurant by using its own state and service actions, the table's state and the human's state. The robot has N POMDP models for a restaurant with N tables.

Problem Formulation

We focus on domains where one robot is addressing a set of N independent tasks. At each time step the robot decides what action should be executed with respect to which task. We model each task, the robot's state and the actions that can be applied to the task as a POMDP and call it *client POMDP*. In this section, we first explain how we represent the client POMDP. We then discuss how the N *client POMDPs* are combined into one large POMDP model called an *agent POMDP*. Finally, we discuss how we use the independence property among the N client POMDPs to compute the agent POMDP's solution.

Client POMDP

The client POMDP for task i is represented as a tuple $(S_i, A_i, Z_i, T_i, O_i, R_i, \gamma, H)$, where $S_i = SR \times SC_i$ denotes the state space, $sr \in SR$ denotes the robot's state (it is shared between the client POMDPs), and $sc_i \in SC_i$ represents the other state variables that are specific to task i. For example, in our restaurant domain, sr contains the robot's position, and sc_i contains state variables such as level of satisfaction. A_i denotes the robot's action space which contains a special action called *no operation* (*no op*). Z_i denotes the robot's observation space which assumes there is no partial observability on the robot's state and only contains task i's observation space ZC_i, $Z_i = ZC_i$. For example, zc_i contains table i's neediness. The robot takes an action $a \in A_i$ and transitions from a state $s \in S_i$ to $s' \in S_i$ with probability $T_i(s, a, s') = \Pr(sc_i'|sc_i, a)\Pr(sr'|sr, a)$ where $s = (sr, sc_i)$ and $s' = (sr', sc_i')$; the robot's actions are deterministic with respect to the robot's state so $\Pr(sr'|sr, a)$ is either 0 or 1. The robot makes an observation $z \in Z_i$, and receives a reward equal to $R_i(sr, sc_i, a)$. The probability function $O_i(s', a, z) = \Pr(z_i|sc_i', a)$ models noisy sensor observations. The discount factor γ specifies how much immediate reward is favored over more distant reward, and H denotes the robot's horizon.

The robot's objective is to choose actions at each time step to maximize its expected future discounted reward: $\mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r_{i,t}\right]$, where $r_{i,t}$ is the reward gained at time t from POMDP i. In POMDP planning, the robot keeps a probability distribution over the states S_i, which is called a belief state B_i. POMDP planning searches for a policy $\pi : B_i \to A$ that maximize the expected future discounted reward at each belief $b \in B_i$. After executing an action, the robot's belief is updated by Eq. 1, where $\Pr(z|b_i, a)$, the probability of observing z after doing action a in belief b_i, is a normalizing constant.

$$b_i(s') = \frac{O_i(z|s', a) \sum_{s \in S_i} T_i(s'|s, a) b_i(s)}{\Pr(z|b_i, a)} \tag{1}$$

The optimal return at stage t, $V_{i,t}^*(b)$, can be iteratively computed by Eq. 2.

$$\begin{aligned} Q_{i,t}^*(b_i, a) &= \sum_{s \in S_i} b_i(s) R_i(s, a) + \gamma \sum_{z \in Z_i} \Pr(z|b_i, a) V_{i,t-1}^*(b_{i,z}^a) \\ V_{i,t}^*(b_i) &= \max_{a \in A_i} \left[Q_{i,t}^*(b_i, a) \right] \end{aligned} \tag{2}$$

The value of following a deterministic trajectory τ at belief state b_i and continuing according to the rest of τ for the remaining $t-1$ steps is computed by Eq. 3.

$$V_{i,t}^{\tau}(b_i) = \sum_{s \in S_i} b_i(s) R_i(s, \tau_t) + \gamma \sum_{z \in Z_i} \Pr(z|b_i, \tau_t) V_{i,t-1}^{\tau}(b_{i,z}^{\tau_t}) \tag{3}$$

For the *no op* action, the reward function only depends on the state variables that are specific to the client POMDP $R_i(sr, sc_i, \text{no op}) = R_i(sc_i, \text{no op})$.

Agent POMDP

We call a POMDP created from multiple client POMDPs *agent POMDP* (or *robot POMDP*). Formally, the agent POMDP for a domain with N tasks is represented by $(N, S, A, Z, T, O, R, \gamma, H)$ where $S = SR \times SC_1 \times SC_2 \times \ldots \times SC_N$, $s \in S$ represents the agent POMDP's state. Let $P = \{i \in \mathbb{N} : i \leq N\}$. The robot's action set A (Eq. 4) contains vectors of length N in which except one element, all other elements are *no op*. The observation space is $Z = Z_1 \times Z_2 \times \ldots \times Z_N$. The robot's probability distribution over the states S is $b \in B$ where $B = B_1 \times B_2 \times \ldots \times B_N$. We assume that the agent POMDP's reward function is additive in terms of its underlying tasks $\mathbb{E}\left[\sum_{i=1}^{N} \sum_{t=0}^{H} \gamma^t r_{i,t}\right]$, where $r_{i,t}$ is the reward gained at time t from task i.

$$A = \bigcup_{i \in P} \bigcup_{a \in A_i} \overbrace{[\text{no op...no op}, \underbrace{a}_{i\text{th element}}, \text{no op...no op}]}^{\text{length N}} \tag{4}$$

The properties in Def. 1 should hold for a set of N client POMDPs to be independent. The N client POMDPs can only share robot's state space and the *no op* action. The transition and observation functions of different client POMDP models are independent of one another.

Definition 1 We call a set of N client POMDPs independent iff $\forall i, j \in P$, $a \in A_i$ and $i \neq j$, the following holds:

1. $SC_i \cap SC_j = \emptyset$
2. $Z_i \cap Z_j = \emptyset$
3. $(A_i \setminus \{no\ op\}) \cap (A_j \setminus \{no\ op\}) = \emptyset$
4. $\Pr(sc'_i|sc_1, sc_2, \ldots, sc_N, a) = \Pr(sc'_i|sc_i, a)$
5. $\Pr(z_i|sc'_1, sc'_2, \ldots, sc'_N, a) = \Pr(z_i|sc'_i, a)$

Given that the tasks are independent and the reward is additive, $R(s,a) = \sum_{i \in P} R_i(s_i, a[i])$, the optimal return at stage t, $V_t^*(b)$, can be iteratively computed as follows:

$$\Pr(z|b,a) = \prod_{k \in P} \Pr(z_k|b_k, a[k]) \tag{5}$$

$$V_t^*(b) = \max_{a \in A} \Big[\sum_{i \in P} \sum_{s \in S_i} b_i(s) R_i(s, a[i]) + \gamma \sum_{z \in Z} \Pr(z|b,a) V_{t-1}^*(b_z^a) \Big] \tag{6}$$

Definition 2 We introduce a parameter k^* which represents the maximum number of tasks that the robot can potentially attend to within H steps. For example, k^* can be set to $\lceil \frac{H}{l} \rceil$ where $\forall i, j \in P$ and $i \neq j$, l is the minimum number of time steps that the robot takes to transition from task i to task j and affect the task j's state variables sc_j.

In the next section, we provide a pseudocode for the algorithm by assuming an arbitrary k. We then show that our approach is optimal for $k \geq k^*$ and discuss its performance with different values for k compared to the other methods.

Approach

We exploit the observation that in some domains the number of tasks that the robot can accomplish within h-steps is limited. We present an algorithm that exploits this observation and decomposes the problem into a series of much smaller planning problems, the solution to which gives us an optimal solution. We denote a subset of k tasks out of P as tpl or k-tuple, $tpl \subseteq P$. We refer to a set including all combinations of k out of N tasks, $\binom{N}{k}$ tasks, as $tpls$ or k-tuples $tpls = \{tpl \in \mathcal{P}(P) : |tpl| = k\}$. The issue is the planner does not know apriori which k tasks it should consider.

Proposed Method

Alg. 1 provides a pseudo code of the main loop of our approach. We follow the online POMDP planning framework where the planning and execution steps are interleaved until all the tasks are terminated (line 2). P represents a set of POMDPs. During the planning phase, the algorithm computes the best action to execute given the POMDPs' belief state (line 3). The execution step executes the selected action (line 4) and updates the belief state of the POMDPs based on the obtained observation (line 5). The robot replans after each action execution. All the baseline algorithms that we compare against modify *SelectAction* in some way.

Algorithm 1: Online Planner for Multiple Independent Tasks (env,P,H,k)

```
1 MultiTaskPOMDPPlanner (env, P, H, k)
2     while not AllTasksDone() do
3         a ← SelectAction(P, H, k)
4         observations ← Step(env, a)
5         UpdateBeliefs(P,observations)
```

Algorithm 2: Our Method (P,H,k)

```
1  SelectAction (P,H,k)
2      τ ← array [1..H] filled with no op; Q_best ← −∞
3      V*,V^τ, Q* ← empty array [1..|P|]
4      for p ∈ P do
5          V*[p],V^τ[p],Q*[p] ← SolvePomdp(p,H)
6      LB ← LowerBound(V*,V^τ)
7      tpls ← BestKTuples(P,V^τ,Q*,LB,k)
8      for tpl ∈ tpls do
9          a_max, Q_max ← SolveAgentPOMDP(tpl,H)
10         if Q_max > Q_best then
11             a_best ← a_max; Q_best ← Q_max
12     return a_best
13 BestKTuples (P, V^τ, Q*, LB, k)
14     tpls ← a set with all combinations of k out of P
15     for tpl ∈ tpls do
16         UB_tpl ← max_{a∈A_tpl} (Σ_{p∈tpl} Q*[p, a[p]]) + Σ_{q∈P\tpl} V^τ[q]
17         if UB_tpl < LB then
18             remove tpl from tpls
19     return tpls
20 LowerBound (V*, V^τ)
21     return max_{p∈P} (V*[p] + Σ_{q∈P\{p}} V^τ[q])
```

Overview of the algorithm (Alg. 2) We first solve each client POMDP separately (lines 4-5). We use the solutions of the client POMDPs to compute a lower bound on the optimal value of the agent POMDP with N tasks (Function $LowerBound$, line 6). We consider a set with all possible combinations of k tasks (k-tuples). We use the solutions of the client POMDPs to compute an upper bound on the value of an agent POMDP created from a k-tuple and use the lower bound to remove the ineffective candidate k-tuples (Function $BestKTuples$). For each remaining candidate k-tuple (line 8), we build and solve the agent POMDP model created from the k-tuple optimally to get an action and the Q-value associated with it (line 9). The action from the k-tuple with the maximum Q-value is selected (lines 10-11) by the robot.

Compute lower bound To compute a lower bound on the optimal value of the agent POMDP with N tasks (Function $LowerBound$), the robot only considers one client POMDP in its horizon H and will perform *no op* on the other POMDPs. For a client POMDP p out of N POMDPs, the algorithm sums up the optimal V-value of the pth POMDP (V_p^*) and the value of performing *no op* on the other POMDPs ($\sum V_q^\tau$, line 21). The sum with the maximum value is returned. This calculation provides a lower bound on the value of the agent POMDP since it does not take into account that 1) the optimal policy might involve switching from one task to another, or 2) an action other than a table's optimal action might be optimal in the agent POMDP.

Find best k-tuples To remove the ineffective POMDP tuples, we compute an upper bound on the value of each k-tuple. We start with all $\binom{N}{k}$ k-tuples (line 14). For each k-tuple (line 15) if its computed upper bound UB_{tpl} is lower than the lower bound (line 17), we remove it from the candidate k-tuples set (line 18).

To compute a k-tuple's upper bound, we assume that the robot only considers the selected k tasks and performs *no op* on the other POMDPs ($\sum V_q^\tau$, line 16). For the selected k-tuple, the robot executes one of the actions from the k-tuples' set of valid actions A_{tpl} which only considers the actions associated with the POMDPs in tpl (Eq. 7). We assume that after executing the first action, each of the k POMDPs follow their optimal policies $Q_p^*(b, a[p])$. This breaks the assumption that the robot cannot address all the tasks in parallel and gives an upper bound on the value of the k-tuple.

Optimality Proofs

Given Def. 1 and an assumption that we define later in this section, we prove that Alg. 2 computes an optimal solution for the agent POMDP with N tasks.

Some notation:

- $V_{p,t}^*$: the optimal value of the client POMDP p at time t.
- $V_{P,t}^*$: the optimal value of the agent POMDP created from the N tasks at time t (Eq. 6).
- A_{tpl}: only considers the actions associated with the POMDPs in a given k-tuple and performs *no op* on the other POMDPs, Eq. 7. In this equation, the union is only over $tpl \subseteq P$, so $A_{tpl} \subseteq A$.

$$A_{tpl} = \bigcup_{i \in tpl} \bigcup_{a \in A_i} \overbrace{[\text{no op...no op}, \underbrace{a}_{i\text{th element}}, \text{no op...no op}]}^{\text{length N}} \quad (7)$$

- $V_{tpl,t}^*$: the optimal value of the agent POMDP created from only the client POMDPs in tpl at time t.

$$V_{tpl,t}^*(b_{tpl}) = \max_{a \in A_{tpl}} Q_{tpl,t}^*(b_{tpl}, a) \quad (8)$$
$$= \max_{a \in A_{tpl}} \Big[\overbrace{\sum_{i \in tpl} \sum_{s \in S_i} b_i(s) R_i(s, a[i])}^{\text{immediate reward}} + \gamma \sum_{z \in Z_{tpl}} \Pr(z | b_{tpl}, a) V_{tpl,t-1}^*(b_{tpl,z}^a) \Big]$$

- $U_{tpl,t}^*$: the optimal value of the agent POMDP created from the client POMDPs in P at time t with action set A_{tpl}. Intuitively, $V_{tpl,t}^*$ considers the value of the client POMDPs in tpl, but $U_{tpl,t}^*$ also considers the utility of performing *no op* on the POMDPs that are not in tpl.

$$U_{tpl,t}^*(b) = \max_{a \in A_{tpl}} \Big[\sum_{i \in P} \sum_{s \in S_i} b_i(s) R_i(s, a[i]) + \gamma \sum_{z \in Z} \Pr(z | b, a) U_{tpl,t-1}^*(b_z^a) \Big] \quad (9)$$

Assumption 1 *The robot has a short horizon H and can only consider k^* tasks in its horizon (Def. 2). Under this assumption, the optimal value for the agent POMDP is called $\hat{V}_P^*$.*

As mentioned earlier, $U_{tpl,t}^*$ considers the action sets of all the POMDPs that are in tpl and performs *no op* on the POMDPs that are not in tpl (Eq. 9). Given Def. 1 and Ass. 1, to compute $\hat{V}_{P,t}^*$, the robot can take a maximum over $U_{tpl,t}^*$ where $|tpl| = k, k \geq k^*$ for all possible $tpl \in tpls$.

$$\hat{V}_{P,t}^*(b) = \max_{tpl \in tpls} U_{tpl,t}^*(b) \quad (10)$$

Lemma 1 *Eq. 11 provides a lower bound on the value of the agent POMDP created from set P.*

$$\max_{p \in P} (V_{p,t}^*(b_p) + \sum_{q \in P \setminus \{p\}} V_{q,t}^\tau(b_q)) \leq \hat{V}_{P,t}^*(b) \quad (11)$$

Proof: We compute $U_{\{p\},t}^*$ (or $U_{p,t}^*$) by using Eq. 9 where the robot only considers POMDP p for horizon H, $tpl = \{p\}$, and performs *no op* on the other POMDPs over that horizon. In Eq. 9, the maximum is taken over A_p, whereas in Eq. 6, the maximum is taken over A. We know $A_p \subseteq A$ as it follows from Eq. 7, thus $U_{p,t}^*$ is a lower bound on $\hat{V}_{P,t}^*$, and Eq. 12 holds. We will show that $U_{p,t}^*$ is in fact $V_{p,t}^*(b_p) + \sum_{q \in P \setminus \{p\}} V_{q,t}^\tau(b_q)$.

$$\forall p \in P : U_{p,t}^*(b) \leq \hat{V}_{P,t}^*(b) \Rightarrow (\max_{p \in P} U_{p,t}^*(b)) \leq \hat{V}_{P,t}^*(b) \quad (12)$$

Using Eq. 5, we expand Eq. 9 as follows:

$$U_{p,t}^*(b) = \max_{a \in A_p} \Big[\sum_{i \in P} \sum_{s \in S_i} b_i(s) R_i(s, a[i]) + \gamma \underbrace{\sum_{z_1 \in Z_1} \Pr(z_1 | b_1, \text{no op}) \dots \sum_{z_N \in Z_N} \Pr(z_N | b_N, \text{no op})}_{\text{does not include } \sum_{z_p \in Z_p}} \sum_{z_p \in Z_P} \Pr(z_p | b_p, a[p]) U_{p,t-1}^*(b_z^a) \Big] \quad (13)$$

The proof goes by mathematical induction. If $H = 1$ and assuming that $U_{p,0}^*(b) = 0$, the following equation holds.

$$U^*_{p,t}(b) = \max_{a \in A_p} \Big[\sum_{i \in P} \sum_{s \in S_i} b_i(s) R_i(s, a[i]) \Big] = V^\tau_{1,1}(b_1) + \ldots + V^*_{p,1}(b_p) + \ldots + V^\tau_{N,1}(b_N)$$

If $H = t - 1$, we assume Eq. 14 where τ is a trajectory consisting of only *no op*s, $\tau =$ no op$[1..H]$, and show that Eq. 14 also holds for $H = t$. Intuitively, Eq. 14 holds since the reward is additive, the POMDPs are independent, and the *no op* actions are executed in parallel while the robot addresses POMDP p.

$$U^*_{p,t-1}(b) = V^\tau_{1,t-1}(b_1) + \ldots + V^*_{p,t-1}(b_p) + \ldots + V^\tau_{N,t-1}(b_N) \tag{14}$$

We substitute Eq. 14 in Eq. 13. Given Def. 1, for a specific Z_i, we can marginalize out the sum over Z_js ($j \neq i$) to obtain:

$$\begin{aligned} U^*_{p,t}(b) &= V^*_{p,t}(b_p) + \underbrace{V^\tau_{1,t}(b_1) + \ldots + V^\tau_{N,t}(b_N)}_{\text{does not include } V_p} \\ &= V^*_{p,t}(b_p) + \sum_{q \in P \setminus \{p\}} V^\tau_{q,t}(b_q) \end{aligned} \tag{15}$$

Thus, Eq. 15 holds for every $H = t$.

Lemma 2 *Under Ass. 1, the optimal value of the agent POMDP created from the set P can be computed by Eq. 16.*

$$\begin{aligned} \hat{V}^*_{P,t}(b) &= \max_{tpl \in tpls} U^*_{tpl,t}(b) \\ &= \max_{tpl \in tpls} (V^*_{tpl,t}(b_{tpl}) + \sum_{q \in P \setminus tpl} V^\tau_{q,t}(b_q)) \end{aligned} \tag{16}$$

Proof: If we show Eq. 17 holds, under Ass. 1, Eq. 16 follows from Eq. 10 and Eq. 17.

$$U^*_{tpl,t}(b) = V^*_{tpl,t}(b_{tpl}) + \sum_{q \in P \setminus tpl} V^\tau_{q,t}(b_q) \tag{17}$$

We consider an agent POMDP with the set of tasks tpl and call it P_{tpl}. We then build a new set of client POMDPs as follows: $\mathcal{A} = \{P_{tpl}\} \cup \{q | q \in P \setminus tpl\}$. Since all the members of P follow Def. 1, the POMDPs in the set $\mathcal{A}$ also follow Def. 1 and are independent; thus, Eq. 17 follows from Eq. 15.

Lemma 3 *For a given tpl, Eq. 18 provides an upper bound on the value of $U^*_{tpl,t}$.*

$$U^*_{tpl,t}(b) \leq \max_{a \in A_{tpl}} (\sum_{p \in tpl} Q^*_{p,t}(b_p, a[p])) + \sum_{q \in P \setminus tpl} V^\tau_{q,t}(b_q) \tag{18}$$

Proof: Substituting U^*_{tpl} from Eq. 17 in Eq. 18:

$$V^*_{tpl,t}(b_{tpl}) \leq \max_{a \in A_{tpl}} (\sum_{p \in tpl} Q^*_{p,t}(b_p, a[p])) \tag{19}$$

Thus, we only need to show that Eq. 19 holds. We use Eq. 2 to compute the Q^* of POMDP p and take a sum of the Q^*s over all members of tpl:

$$\begin{aligned} \sum_{p \in tpl} Q^*_{p,t}(b_p, a[p]) = \overbrace{\sum_{p \in tpl} \sum_{s \in S_p} b_p(s) R_p(s, a[p])}^{\text{immediate reward } (IR)} \\ + \gamma \sum_{p \in tpl} \sum_{z \in Z_p} \Pr(z | b_p, a[p]) V^*_{p,t-1}(b^{a[p]}_{p,z}) \end{aligned} \tag{20}$$

The immediate reward IR operand in Eq. 20 and Eq. 8 are equal, so we only need to show the inequality for the second operand. For time step $t-1$, if the robot could address all the k tasks in parallel, we could compute an upper bound on the value of $V^*_{tpl,t-1}(b^a_{tpl,z})$ by summing over the optimal value of each POMDP in tpl, $(m, n, o, \ldots) \in tpl$, Eq. 21. This can be proved using a similar induction procedure that we used earlier.

$$\begin{aligned} V^*_{tpl,t-1}(b^a_{tpl,z}) \leq V^*_{m,t-1}(b^{a[m]}_{m,z_m}) + \ldots + V^*_{n,t-1}(b^{a[n]}_{n,z_n}) + \\ \ldots + V^*_{o,t-1}(b^{a[o]}_{o,z_o}) = \sum_{q \in tpl} V^*_{q,t-1}(b^{a[q]}_{q,z_q}) \end{aligned} \tag{21}$$

Substituting Eq. 21 in Eq. 8:

$$\begin{aligned} Q^*_{tpl,t}(b_{tpl}, a) &\leq IR + \sum_{p \in tpl} \sum_{z \in Z_p} \Pr(z | b_p, a[p]) V^*_{p,t-1}(b^{a[p]}_{p,z}) \\ &= \sum_{p \in tpl} Q^*_{p,t}(b_p, a[p]) \end{aligned} \tag{22}$$

Thus, $V^*_{tpl,t}(b_{tpl}) \leq \max_{a \in A_{tpl}} \sum_{p \in tpl} Q^*_{p,t}(b_p, a[p])$.

Therefore, Alg. 2 computes an optimal solution for the agent POMDP created from set P. The proof should follow from Lemma 1, 2 and 3.

Experiments

We call our approach *method-k*, *method-2 (A)* or *method-3 (F)*, as we evaluate it for different values for k (2 or 3). We compare our method against the following baselines.

- **Agent POMDP (B)**: We use the agent POMDP model that we described in the approach section.
- **N-samples-k (C,G)**: We use the approach by (Shani 2013) where they select N k-tuples from all possible combinations of k-tuples and solve them optimally. They iterate over the set P; for each task they randomly select $k-1$ additional tasks from set P. We refer to the *N-samples-2* method as *C* and the *N-samples-3* method as *G*.

- **Hierarchical POMDP or HPOMDP (D)**: We represent each task as a macro action; in total there are N macro actions for all the N tasks. We use the agent POMDP model and replace its action set with the set of macro actions. During planning when a macro action i, m_i, gets selected, the agent POMDP evolves according to the POMDP i's action set, while the other POMDPs evolve as *no op* has been executed on them. Each macro action is atomic and takes $min(\text{horizon}, \#\text{ time steps left till } m_i \text{ terminates})$ time steps to get executed.
- **Greedy (E)**: The robot solves each client POMDP separately and selects an action according to $\arg\max_{a \in A}(\sum_{p \in P} Q_p^*(b_p, a[p]))$. This approach assumes that after the first action execution, for the remaining horizon, the tasks can be executed at the same time in parallel.

Restaurant Model

We run experiments in restaurant scenarios with 2 to 12 tables. The table's model is described below where the value in parenthesis shows the range of the variable. For simplicity, we model only the *satisfaction* as an unobservable variable. All other variables are observable.

- **States** S: $S = \{SR, SH, ST\}$.
  - Robot's state SR contains *x* and *y* (11×11 grid).
  - Human's state SH contains *satisfaction* (6). The highest satisfaction is $sat_{max} = 5$.
  - Table's state ST contains the following: *food* (4), *water* (4), *time since food or water has been served*, *cooking status* (3), *time since food is ready*, *current request* (8), *hand raise* (2), and *time since the hand has been raised*. The time related variables have $time_{max} = \#\text{ tables} \times sat_{max}$ values which accounts for having more time to service the customers when there are more tables. When the food is ready to be served, the *time since request* variable is equal to *time since food is ready*, but when the food is not fully cooked *time since request* $= 0$.
- **Actions** A: The full action space is the concatenation of all actions $A = \{AN, AC, AS\}$.
  - Navigation actions AN such as *go to table* i.
  - Communication actions AC such as *food is not ready*.
  - Service actions AS such as *fill a water glass*, *serve food*, and *get the bill*. Depending on the table's current request, the appropriate service action gets executed.
- **Transition function** T: The outcome of executing an action is deterministic for the observable variables and stochastic for the *satisfaction*. Each table goes through a consecutive sequence of 8 requests, such as *wanting food* and *wanting the bill*. The *time since request* variable resets to 0 when a table is served. The *food* and *water* variables represent the table's eating and drinking process and increase after a couple of time steps until they reach their maximum value. The *satisfaction* variable goes down by one after $\frac{time_{max}}{sat_{max}}$ time steps; if a customer is very satisfied at the beginning and does not get served within $time_{max}$, she becomes very unsatisfied. The actions increase the time-related variables of the state by 1 except the navigation actions which can take 1, 2, or 3 time steps depending on where the robot is with respect to the tables.
- **Reward function** *R*: The reward function is as follows. Servicing a table has a positive reward inversely proportional to the table's satisfaction (sat) to give higher priority to unsatisfied tables. If the table is unsatisfied and waiting to be served, a negative reward is given.

$$time = min(\text{time since request}, 10)$$

$$R(s, \text{serve}) = 5 * (sat_{max} - sat' + 1); R(s, \text{go to}) = \frac{-\text{dist}}{3}$$

$$R(s, \text{other actions}) = \begin{cases} -2^{time} & sat' = 0 \\ -1.7^{time} & sat' = 1 \\ -1.4^{time} & sat' = 2 \\ 1 & sat' = 3, 4, 5; sat' > sat \\ 0 & \text{otherwise} \end{cases}$$

Given this model description, if $horizon \leq 4$, the robot can only address two tasks in its 4-step limited horizon, so our algorithm computes an optimal solution if it considers all the 2-tuples. If $horizon = 5$, the robot can address 3 tasks, so the algorithm should consider all the 3-tuples.

Simulation Setup

For each algorithm, we run 30 episodes each for 20 actions. Each episode starts with a random initialization of the state variables with its belief probability set to 1. The random initialization for each episode is the same across all algorithms.

Quantitative Results We compare our method in terms of planning time and average reward against the baselines. For each episode, we compute the average time that the planner takes to plan over 20 actions. The reward for each action execution is the expected reward over the belief state distribution $r(b, a) = \sum_{s \in S} b(s)R(s, a)$. We report the average reward over 20 actions. To remove the variations that results from the initial randomization, for each episode we take the difference between the average reward of *method-2* and other approaches.

Fig. 2 shows a comparison of the mean and standard deviation of the planning time for *method-2* against the baselines. As the number of tables increase, the *agent POMDP* approach has a higher positive slope than other approaches. The planning time for different approaches mostly follow $B > A > C > D > E$.

Fig. 3 shows the mean and standard deviation of the difference between the average reward of *method-2* and that of other approaches. We compare all the baselines against a zero line which represents our approach. We proved earlier that our approach is optimal, but an optimal action for horizon H might not result in higher average return for an episode because 1) an action that is optimal for a short horizon might not be optimal for a longer horizon, and 2) different approaches have different tie-breaking strategies; *i.e.*, actions with equal average returns for a short horizon would

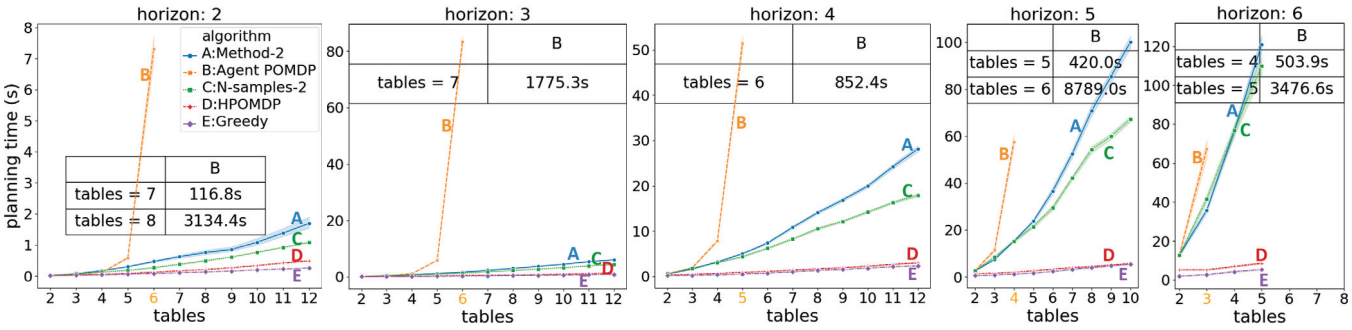

Figure 2: Planning times for different horizons and number of tables. We could only run the *agent POMDP* for 30 episodes up to a certain number of tables ≤ 6 (shown by the orange label on the x-axis). Beyond that we run the *agent POMDP* approach for one episode and report the results in a table on each figure.

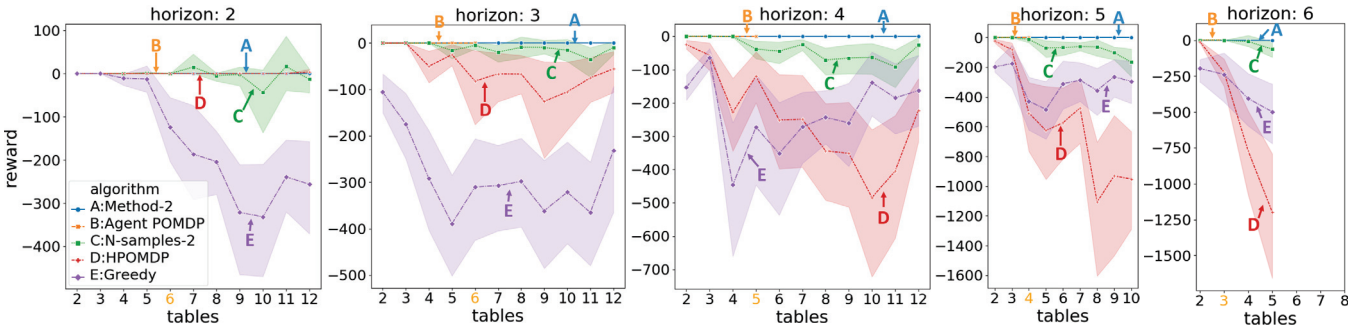

Figure 3: Difference between the average reward of our method and the other methods.

have different average returns for a longer horizon. This is why for horizon 2 our approach performs exactly the same as the *agent POMDP* approach, but other methods can perform better. For the horizons other than 2, the average reward for different approaches mostly follow $B \approx A > C$.

We also compare the reward for each episode with the same initialization across multiple algorithms. We report the results in terms of the percentage of the episodes (out of 30) that the rewards are equal or one is better. For different horizons, our approach's reward is exactly the same as the *agent POMDP*'s reward for 2,4,5, and 6 tables. For 3 tables and horizons 3 and 6, the *agent POMDP* is better in 3% of the episodes (one episode) because of having a different tie-breaking strategy. For 5 tables and horizon 3, our approach is better in 3% of the episodes.

Comparing the reward of *method-2* against *N-samples-2* for each episode, we observe that for 2 and 3 tables and for different horizons, the rewards are exactly the same. For horizon 2, if $tables = 7$, the rewards are exactly the same in 87% of the episodes and *N-samples-2* is better in 10% of the episodes. For 11 tables, the rewards are exactly the same in 77% of the episodes and our approach performs better in 13% of the episodes. For a longer horizon 3, if $tables = 8$, the rewards of *method-2* and *N-samples-2* are exactly the same 50% of the times and *method-2* is better 27% of the times. Thus, for horizon 2, our approach has a similar performance as *N-samples-2*. Our algorithm mostly performs better than *N-samples-2* for horizons ≥ 3. For horizon 2, our method does not benefit from considering 2-tuples, so the *HPOMDP* approach provides similar average reward as our algorithm. The *HPOMDP* approach performs better than our approach when horizon is 2 and $tables > 8$ in 3% of the episodes because of the tie-breaking strategy.

We run the algorithms on a simpler version of the restaurant that includes *current request*, *hand raise*, and *time since request* as the table's state ST, and has all the actions except *food is not ready yet* and compare its performance with different k values against other methods. As can be seen in Fig. 5, given horizon 5, the planning time for different approaches mostly follow $B > F > G > A > C$. Comparing *method-2* and *method-3*, for horizon 5, if $tables = 4$, the rewards are exactly the same. If $tables = 6, 7$, the rewards are exactly the same in 67% of the cases and *method-3* is better in 23% of the episodes. If $tables = 8$, the rewards are exactly the same in 77% of the cases and *method-2* is better in 13% of the cases. Both *method-2* and *method-3* perform better that the other baselines.

In summary, we observe that our approach results in a similar average reward compared to the *agent POMDP* approach while being significantly faster. Although our approach has a higher planning time compared to some of the baselines, it has a higher average reward than them.

Qualitative Results Fig. 4 shows a sample output policy for 5 tables. The leftmost figure shows the restaurant configuration at time step 11 after Table 0 is served. Table 3

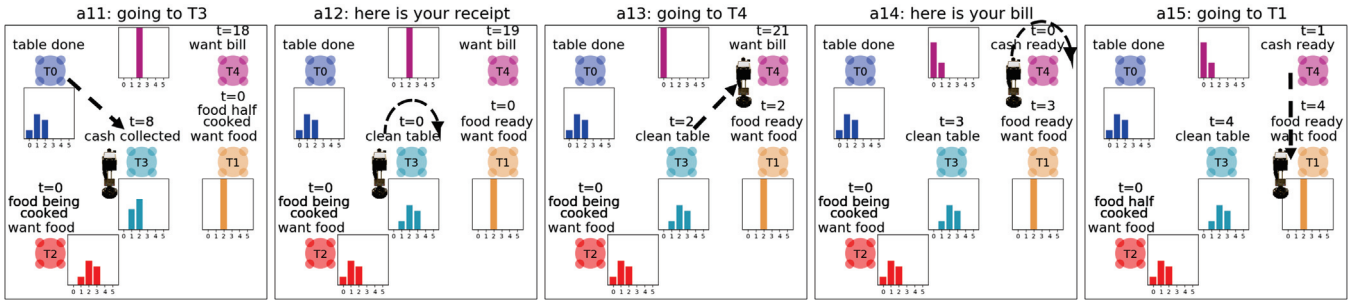

Figure 4: Example output policy for $H = 4$ and 5 tables. The histograms show the belief over satisfaction for different tables. The leftmost bar is 0 (very unsatisfied) and the rightmost is 5 (very satisfied). The robot's action is shown on top of each figure. Each table's request and the amount of time they have been waiting is shown above it.

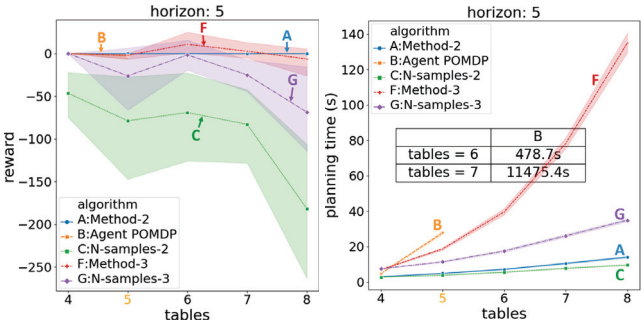

Figure 5: Performance comparison between our approach and other baselines when $k = 2, 3$. We run the algorithms on a simpler version of the restaurant model.

has been waiting for 8 time steps and compared to others is less satisfied, so the robot services it to increase the table's satisfaction. The robot then goes to Table 4 since it soon becomes very unsatisfied. The robot then services Table 1 as their food is ready before going to Table 2 to update them that their food is not ready. The *greedy* approach selects the *no op* action at time step 11 as it assumes that after 1 time step, it can service all tasks in parallel. Two consecutive *go to* actions appear frequently in the output policy of the *greedy* approach, *e.g.*, *greedy* selects *go to T4* at time step 12 instead of serving T3.

Discussion

Here, we provide a discussion on how much the effectiveness of our approach depends on the parameters of the model. In general if we apply the approach on a different planning problem of similar complexity, the planning time would still be similar to the planning time that we computed for the restaurant model, and the optimality guarantees of our approach would still hold. The *HPOMDP*, *N-samples-k*, and *greedy* approaches do not have any optimality guarantees. On a different application domain which requires switching among the tasks, we expect our method to perform better in terms of average reward than the *HPOMDP* approach since the *HPOMDP* approach does not consider switching between the tasks in each planning step. We also expect our approach to perform better than *N-samples-k* since *N-samples-k* samples from the subset of tasks whereas our method finds an optimal solution by computing upper and lower optimal value bounds for subsets of tasks to prune the subsets.

The parameters of the reward function does not affect our approach's planning time and the optimality of our approach. However, the parameters will change the output policy of the robot, and how much better our approach is compared to the other baseline approaches. For example, if the negative reward for going from one table to another table is high, the robot would prefer to stay as much as it can at the current table, even if the other tables are very unsatisfied. This way of defining the reward function would make our approach just a bit better than the *HPOMDP* approach in terms of the average reward. The parameters are engineered such that we can get a sensible and interesting output policy as shown in Fig. 4 where the robot switches among multiple tables instead of just servicing one table mostly.

The *greedy* approach sometimes outperforms the *HPOMDP* approach since the domain requires the robot to switch between the tables to keep the customers satisfied. The *HPOMDP* approach assumes that only one table can be serviced in the given horizon. When this assumption is valid, for example for horizon 2, the *HPOMDP* approach performs much better than the *greedy* approach. For longer horizons (and more tables), this assumption becomes less valid and the *HPOMDP* approach performs poorly. The *greedy* approach does not provide an accurate estimate, but can consider more than one task in the short horizon since it assumes that the tasks can be executed at the same time in parallel after the first action execution.

Conclusion

We proposed an algorithm to speed up POMDP planning for domains where a robot is required to accomplish a set of tasks that are partially observable and evolve independently of each other. We exploited the observation that the number of tasks that the robot can accomplish within a short horizon is limited and presented an algorithm that leverages the solutions to much smaller POMDP models to optimally solve the combined model with all the tasks. We proved the optimality of our approach and evaluated it on a restaurant setting.

Acknowledgments

This work was partially supported by Sony AI. The views contained in this document are those of the authors only.

References

Bahar, R. I.; Frohm, E. A.; Gaona, C. M.; Hachtel, G. D.; Macii, E.; Pardo, A.; and Somenzi, F. 1997. Algebric decision diagrams and their applications. *Formal methods in system design*.

Boutilier, C., and Poole, D. 1996. Computing optimal policies for partially observable decision processes using compact representations. In *AAAI*.

Cassandra, A. R. 1998. A survey of pomdp applications. In *AAAI fall symposium on planning with partially observable Markov decision processes*.

Deo, S.; Iravani, S.; Jiang, T.; Smilowitz, K.; and Samuelson, S. 2013. Improving health outcomes through better capacity allocation in a community-based chronic care model. *Operations Research*.

Foka, A., and Trahanias, P. 2007. Real-time hierarchical pomdps for autonomous robot navigation. *Robotics and Autonomous Systems*.

Glazebrook, K. D.; Ruiz-Hernandez, D.; and Kirkbride, C. 2006. Some indexable families of restless bandit problems. *Advances in Applied Probability*.

Hansen, E. A., and Zhou, R. 2003. Synthesis of hierarchical finite-state controllers for pomdps. In *ICAPS*.

Li, X.; Cheung, W. K.; and Liu, J. 2009. Improving pomdp tractability via belief compression and clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*.

Liu, K., and Zhao, Q. 2010. Indexability of restless bandit problems and optimality of whittle index for dynamic multichannel access. *IEEE Transactions on Information Theory*.

Pineau, J.; Gordon, G.; Thrun, S.; et al. 2003. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*.

Poupart, P., and Boutilier, C. 2003. Value-directed compression of pomdps. In *NeurIPS*.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-Draa, B. 2008. Online planning algorithms for pomdps. *JAIR*.

Roy, N.; Gordon, G.; and Thrun, S. 2005. Finding approximate pomdp solutions through belief compression. *JAIR*.

Shani, G.; Poupart, P.; Brafman, R. I.; and Shimony, S. E. 2008. Efficient add operations for point-based algorithms. In *ICAPS*.

Shani, G.; Pineau, J.; and Kaplow, R. 2013. A survey of point-based pomdp solvers. *AAMAS*.

Shani, G. 2013. Task-based decomposition of factored pomdps. *IEEE transactions on cybernetics*.

Smith, T.; Thompson, D. R.; and Wettergreen, D. 2007. Generating exponentially smaller pomdp models using conditionally irrelevant variable abstraction. In *ICAPS*.

Theocharous, G., and Kaelbling, L. P. 2004. Approximate planning in pomdps with macro-actions. In *NeurIPS*.

Weber, R. R., and Weiss, G. 1990. On an index policy for restless bandits. *Journal of Applied Probability (JAP)*.

Whittle, P. 1988. Restless bandits: Activity allocation in a changing world. *JAP*.

Wray, K. H., and Zilberstein, S. 2017. Approximating reachable belief points in pomdps. In *IROS*.