Isomorph-free Branch and Bound Search for Finite State Controllers

Marek Grześ, Pascal Poupart and **Jesse Hoey**
Cheriton School of Computer Science, University of Waterloo
200 University Avenue West, Waterloo, Ontario N2L 3G1, Canada
{mgrzes, ppoupart, jhoey}@uwaterloo.ca

Abstract

The recent proliferation of smart-phones and other wearable devices has lead to a surge of new mobile applications. Partially observable Markov decision processes provide a natural framework to design applications that continuously make decisions based on noisy sensor measurements. However, given the limited battery life, there is a need to minimize the amount of online computation. This can be achieved by compiling a policy into a finite state controller since there is no need for belief monitoring or online search. In this paper, we propose a new branch and bound technique to search for a good controller. In contrast to many existing algorithms for controllers, our search technique is not subject to local optima. We also show how to reduce the amount of search by avoiding the enumeration of isomorphic controllers and by taking advantage of suitable upper and lower bounds. The approach is demonstrated on several benchmark problems as well as a smart-phone application to assist persons with Alzheimer's to wayfind.

1 Introduction

In this work we describe an approach to generate isomorph-free Moore finite state automata. This is an important problem since deterministic finite state controllers (FSCs) for partially observable Markov decision processes (POMDPs) are Moore finite state automata. The optimization of a POMDP policy can be cast as a search for the best FSC of a given size. However, since there are many controllers that encode the same policy, there is a need for techniques that can generate only the controllers that correspond to different policies. We denote by "equivalent" the controllers that encode the same policy. Controllers can be thought as deterministic finite automata for which there exist several minimization techniques to identify smaller equivalent automata [Huffman, 1954; Moore, 1956; Hopcroft, 1971]. Building on this work we show how to generate only non-equivalent minimal FSCs.

Meuleau et al. [1999] previously designed a branch and bound technique to search for the best controller of a given size that avoids the generation of some symmetric controllers. Two controllers are symmetric when a permutation of the actions of the first controller yields the second controller. Controller symmetry is a special type of controller equivalence, but does not cover all forms of controller equivalence. We show that a significant speed up can be obtained by generating only non-equivalent controllers. We also show how to further prune the search space by using tighter upper bounds in branch and bound.

The optimization of POMDP policies by branch and bound in the space of fixed-size deterministic controllers is particularly useful in applications where some performance guarantees are desired and online computation costs need to be minimized. For instance, the emerging class of monitoring and assistive applications on mobile or wearable devices can be cast as POMDPs. While these applications need to be continuously running, they also need to minimize energy consumption to maximize battery life. Controllers are ideally suited since there is no online planning or belief monitoring. Due to the non-convex nature of controller optimization, alternative approaches such as bounded policy iteration [Poupart and Boutilier, 2003], stochastic local search [Braziunas and Boutilier, 2004], quadratically constrained linear programming [Amato *et al.*, 2009], gradient ascent [Meuleau *et al.*, 1999] and expectation maximization [Toussaint *et al.*, 2006] may get stuck in arbitrarily bad local optima (although some techniques to escape local optima have been proposed [Poupart *et al.*, 2011b]). In contrast, branch and bound is guaranteed to find the best deterministic controller of a given size with enough time.

The paper is structured as follows. Sec. 2 reviews POMDPs and finite state controllers. Sec. 3 describes related work on controller optimization, including branch and bound. Sec. 4 shows how to detect equivalent controllers and how to efficiently generate isomorph-free controllers. Sec. 5 explains how to improve branch and bound by generating only non-equivalent controllers, using tighter upper bounds, prioritizing the search and by limiting the number of edges. Sec. 6 reports experiments with benchmark problems and a wayfinding application to assist people with Alzheimer's disease. Sec. 7 concludes.

2 Background

Partially observable Markov decision processes are formally defined by a tuple $<S, A, O, T, Z, R, \gamma>$ where S is the set of

states s, A is the set of actions a, O is the set of observations o, $T(s', s, a) = \Pr(s'|s, a)$ defines the transition probabilities, $Z(o, a, s') = \Pr(o|a, s')$ defines the observation probabilities, $R(s, a)$ defines the reward function and $0 < \gamma < 1$ is the discount factor. The goal is to find a policy $\pi : H \to A$ that maps histories $h \in H$ of previous actions and observations to the next action. Since the length of histories grows with time and the number of histories grows exponentially with time, there is a need for a bounded representation. One option is to use belief states (i.e., distributions over states) which can be thought as a sufficient statistic that encodes the same information as histories. Alternatively, we can restrict policies to finite state controllers, which define a compact mapping from histories to actions.

A finite state controller (FSC) consists of a set of nodes labeled with actions and edges labeled with observations. An FSC is parametrized by an action mapping ϕ, which assigns an action to each node (i.e., $\phi : A \to N$) and a node mapping ψ, which indicates which node each edge maps to (i.e., $\psi : N \times O \to N$). The policy encoded by a controller is executed by performing the action associated with each node traversed and by following the edge associated with each observation received. This execution requires a negligible amount of computation at each step to update the current node and lookup the corresponding action. Hence, it is ideal for mobile applications, especially those with severe computation or energy constraints.

We consider the problem of finding the best controller with a fixed number of nodes. The more nodes there are the better the controller may be, but past experience has shown that for many POMDPs there exist small controllers that are quite good [Poupart *et al.*, 2011b]. Hence, by fixing the number of nodes we can limit the search space and still find a good controller most of the time. The search space consists of all possible ϕ's and ψ's, which is exponentially large. The value $V^\pi(s, n)$ of starting a controller $\pi = <\phi, \psi>$ in node n at state s can be computed as follows:

$$V^\pi(s, n) = R(s, \phi(n)) + \gamma \sum_{s', o} \Pr(s', o|s, \phi(n)) V^\pi(s', \psi(n, o)) \ \forall s, n$$

Without loss of generality, we assume that the policy of a controller always starts in the first node n_1. Hence the value of a controller at initial belief b_0 is $V^\pi(b_0) = \sum_s b_0(s) V^\pi(n_1, s)$.

3 Related Work

The optimization of POMDP controllers is notoriously difficult [Vlassis *et al.*, 2012]. Hansen [Hansen, 1998] first proposed a policy iteration algorithm that gradually constructs a controller by adding nodes. However, the size of the controller tends to grow exponentially with the number of time steps. Subsequently, several approaches were proposed to optimize controllers of a fixed size [Poupart and Boutilier, 2003; Braziunas and Boutilier, 2004; Amato *et al.*, 2009; Meuleau *et al.*, 1999; Toussaint *et al.*, 2006]. These approaches often consider stochastic controllers, allowing them to employ techniques from continuous optimization. Unfortunately, due to the non-convex nature of the optimization, they may get trapped in arbitrarily bad local optima. Some approaches have been augmented with escape techniques, but this is usually done by increasing the size of the controller [Poupart and Boutilier, 2003; Poupart *et al.*, 2011b]. Alternatively, an exhaustive search in the space of deterministic controllers can guarantee that the best deterministic controller is found. To that effect, Meuleau et al. [1999] proposed a branch and bound technique to efficiently search the space of controllers. We briefly review Meuleau's approach since we will improve it in the next sections.

Branch and bound performs a depth-first backtracking search in the space of controller parameters. This space consists of $|N|$ action variables (one for each node mapping $\phi(n)$) and $|N||O|$ node variables (one for each edge mapping $\psi(n, o)$). The search incrementally builds a controller by assigning values to the variables until a full controller is obtained or we backtrack at a partial controller. At each partial controller, an upper bound on the value of all possible completions of this partial controller is computed. If this upper bound is lower than the value of the best controller visited so far, then the search is cut. Alg. 1 describes a generic branch and bound algorithm for controllers. It requires a method to compute partial controller upper bounds and heuristics to determine the order in which the variables and their values should be enumerated. In addition, a pruning technique may be used to detect equivalent controllers.

Meuleau et al. [1999] prune symmetric controllers by imposing a lexicographic ordering on the actions assigned to the nodes. Assuming that actions are numbered from 1 to $|A|$, then controllers whose nodes are not assigned actions in increasing order are rejected. This prunes factorially many equivalent controllers that would result from permuting nodes with different actions. However, nodes with identical actions may still be permuted to yield syntactically different controllers that are equivalent. Other non-obvious forms of equivalence will also go undetected. We explain how to detect and prune *all* forms of equivalence in the next section.

Algorithm 1 Branch and Bound

```
BRANCHANDBOUND(π, LB)
 1  if PRUNE(π) then return LB
 2  if π is fully specified then return max(V^π(b_0), LB)
 3  UB ← UPPERBOUND(π)
 4  if UB < LB then return LB
 5  Select next variable V to instantiate
 6  while some domain values have not been tried
 7      Select next value v to try
 8      Let π' be π extended with V = v
 9      if ¬PRUNE(π')
10          LB ← BRANCHANDBOUND(π', LB)
11  return LB
```

For the upper bound, Meuleau et al. [1999] adapt the QMDP upper bound to partial controllers. The idea is to let the choice of action and next node in the ϕ and ψ mappings depend on the underlying state s even if it is not observable. Because decisions are based on more information than what is normally available, the resulting value is necessarily higher, yielding an upper bound. When ϕ and ψ are completely un-

specified, the QMDP upper bound $\bar{V}$ satisfies:

$$\bar{V}(s,n) = \max_a [R(s,a) + \gamma \sum_o \max_{n'} \sum_{s'} \Pr(o,s'|s,a)\bar{V}(s',n')] \forall_{s,n} \quad (1)$$

The above equation can easily be adapted to partially specified controllers by replacing the maximization over actions by the action selected for each specified action mapping and by replacing the maximization over next nodes by the selected next node for each specified next node mapping. The upper bound is computed with dynamic programming by repeatedly computing the right-hand-side of Eq. 1. In Sec. 5, we explain how to improve this bound by considering the fast informed bound and an augmented POMDP.

4 Detecting Equivalent Controllers

Two FSCs are said to be equivalent when they encode the same policy. The policy encoded by a controller is a set of conditional plans rooted at each node. Here a conditional plan is a tree that alternates between actions and observations. Trees are generally infinite unless the planning horizon is finite. Two controllers that contain the same set of conditional plans are considered equivalent. Detecting equivalent controllers is related to the problem of detecting that two automata accept the same language. In fact, finite state controllers are deterministic finite automata (DFA) where the observations are input symbols and actions are output symbols. In automata theory, it is common to consider only two outputs (accept and reject) since the goal is to define a language that accepts a specific set of strings. Nevertheless, controllers can be viewed as a straightforward extension where strings of inputs/observations are partitioned into several classes labeled with different outputs/actions. Hence algorithms to detect equivalent DFA can be used to detect equivalent controllers.

Equivalent DFA/FSCs arise when some nodes are repeated or permuted. Several algorithms have been proposed to minimize automata by detecting repeated nodes [Huffman, 1954; Moore, 1956; Hopcroft, 1971]. The same can be done with controllers by detecting nodes that are the root of identical conditional plans. Alg. 2 describes how to do this by gradually updating a matrix that indicates which pairs of nodes are the roots of different trees. Let M be an $|N| \times |N|$ matrix where $|N|$ is the number of nodes in a FSC. The algorithm starts with all entries in M set to 1, indicating that all nodes of the FSC may be equivalent. Then the algorithm repeatedly inspects the entries and revises an entry $M_{i,j}$ to 0 when the conditional plans of nodes n_i and n_j are found to be different. This occurs when the actions of those nodes are different or their edges point to subtrees that are not equivalent. The algorithm terminates when there are no more entries to be revised. If there is a non-diagonal entry set to 1, then the FSC is not minimal since some of its nodes are repeated. While the procedure described in Alg. 2 is not the most efficient, it is conceptually simple and it can be easily modified to deal with partial controllers as we will see in Alg. 3.

Minimal FSCs (no repeated nodes) may still be equivalent if their nodes are permuted. In that case the FSCs are said to be isomorphic. Since our goal is to generate non-equivalent controllers, we can generate a single minimal FSC in each equivalence class by imposing a lexicographic ordering. This

Algorithm 2 Repeated Trees in Full FSC

REPEATEDTREESINFULLFSC(FSC)

```
1  Let M be an |N| × |N| matrix of ones
2  repeat
3      for each i, j ∈ {1, ..., |N|} such that i ≠ j
4          if φ(n_i) ≠ φ(n_j) then M_{i,j} ← 0
5          for each observation o
6              if M_{ψ(n_i,o),ψ(n_j,o)} = 0 then M_{i,j} ← 0
7  until M doesn't change
8  if ∃i ≠ j such that M_{ij} = 1 then return true
9  else return false
```

will ensure that among all node permutations, only one configuration is retained. Suppose that we number all nodes from 1 to $|N|$ and all observations from 1 to $|O|$, we can then number edges from 1 to $|N||O|$ such that the index of the edge coming out of node n and labeled with observation o is $(n-1)|O| + o$. We can retain a single permutation of the nodes by making sure that the edges point to nodes that satisfy the following conditions:

$$\psi(edge_1) \leq 2 \quad \text{and} \quad \psi(edge_i) \leq \max_{j<i} \psi(edge_j) + 1 \quad (2)$$

The first condition says that the first edge should point to node 1 or a new node that will take the next index. In this case the next index is 2. The second condition says that all other edges should point to a node that has already been pointed to by a previous edge or a new node that will take the next index. In this case the index is 1 plus the largest index of the nodes pointed to so far.

Theorem 1. *There is exactly one minimal controller in each equivalence class that satisfies the conditions in (2).*

Proof. We sketch an informal proof. There is at least one minimal controller in each equivalence class that satisfies the conditions in (2) since we can take any minimal controller and renumber its nodes according to the order in which the nodes would be visited by a breadth first search that expands each node at most once.

We prove by contradiction that there is at most one minimal controller in each equivalence class that satisfies the conditions in (2). Suppose that we have two syntactically different minimal controllers in the same equivalence class. By definition, these controllers encode the same policy, which means that they have the same set of conditional plans. Alg. 2 ensures that none of their conditional plans are repeated. So they must have the same number of nodes and there exists a one-to-one mapping between the conditional plans of the two controllers. The conditions in (2) also ensure that the nodes are ordered in the same way, which means that there is no syntactic difference between the controllers. This contradicts our assumption. □

Alg. 2 and the conditions in (2) give us the tools to identify a unique minimal controller in each equivalence class that we will call the *canonical* controller. We can use these tools in a branch and bound procedure to generate only canonical controllers corresponding to different policies. That being said,

branch and bound operates by gradually completing partial controllers. Hence, we would like to detect as early as possible that all completions of a partial controller will yield non-canonical controllers in order to cut the search early. This can be done by modifying Alg. 2 to detect repeated trees in partial controllers. When an action or an edge is unassigned, we will assume that it can take any desirable assignment to show that two trees are different. Hence, Alg. 3 will detect identical trees only when two trees will necessarily be identical, regardless of how unassigned variables are assigned. We denote by '*' the value of an unassigned variable.

Algorithm 3 Repeated Trees in Partial FSC

REPEATEDTREESINPARTIALFSC(FSC)

```
Let M be an |N| × |N| matrix of ones
repeat
    for each i, j ∈ {1, ..., |N|} such that i ≠ j
        if φ(n_i) = * or φ(n_j) = * or φ(n_i) ≠ φ(n_j)
            M_{i,j} ← 0
        for each observation o
            if ψ(n_i, o) = * or ψ(n_j, o) = *
            or M_{ψ(n_i,o),ψ(n_j,o)} = 0 then M_{i,j} ← 0
until M doesn't change
if ∃i ≠ j such that M_{ij} = 1 then return true
else return false
```

5 Improved Branch and Bound

There are several factors that influence the effectiveness of branch and bound. In addition to pruning non-canonical controllers as in the previous section, the quality of the upper bounds, the order in which variables and values are selected and the number of edges will influence the amount of search.

5.1 Upper Bound

Tighter upper bounds can significantly reduce the amount of search. We propose to adapt the fast informed bound (FIB) to controllers since this bound is tighter than QMDP. While QMDP provides an upper bound by assuming that the *current* state is observed, FIB assumes that the *previous* state is observed. In practice, neither the current nor the previous states are observable, however the previous state provides less additional information than the current state, which is why FIB is tighter. When ϕ and ψ are completely unspecified, the upper bound $\bar{V}$ based on FIB is obtained as follows:

$$\bar{V}(s,n) = max_a \bar{Q}(s,n,a) \text{ where}$$
$$\bar{Q}(s,n,a) = R(s,a) + \gamma \sum_o \max_{n',a'} \sum_{s'} \Pr(o,s'|s,a)\bar{Q}(s',n',a') \quad \forall s,n,a$$

The above equation can easily be adapted to partially specified controllers by replacing the maximization over actions by the action selected for each specified action mapping and by replacing the maximization over next nodes by the selected next node for each specified next node mapping.

Upper bounds such as QMDP and FIB can be further improved by creating an augmented POMDP [Hauskrecht, 2000; Poupart *et al.*, 2011a]. The idea is to augment the state space with belief points to create a larger POMDP. If we treat all reachable beliefs as states, the augmented POMDP becomes a belief state MDP that can be solved exactly by MDP techniques. QMDP and FIB then return tight upper bounds that correspond exactly to the optimal value function. Since it is not practical to treat all reachable beliefs as states, we only consider a subset of reachable beliefs, which yield an augmented POMDP somewhere between the original POMDP and the belief state MDP. Although the QMDP and FIB upper bounds won't be tight, they will be tighter than for the original POMDP. The combination of FIB with an augmented POMDP currently yields the lowest known upper bounds for many POMDPs [Poupart *et al.*, 2011a]. We use the same approach as in the gapMin solver [Poupart *et al.*, 2011a] to select the belief points and create the augmented POMDP.

5.2 Variable and Value Ranking

The order in which variables and values are selected for assignment can greatly influence the amount of search. If we make good assignments initially then we will quickly obtain a good controller with a high lower bound, allowing us to prune more aggressively in the rest of the search. We propose to rank variables according to their occupancy frequencies. Intuitively, the more often a node or an edge is visited, then the more critical it will be to set it properly. Hence, we rank the unassigned variables in decreasing order of occupancy frequency and select the one with highest frequency. Occupancy frequencies are estimated by performing a simulation of the controller and recording how often each node and each edge is visited. When the simulation reaches an unassigned node or edge in a partial controller, the best action or best next node according to the FIB upper bound is assumed. This allows us to simulate the specified and unspecified parts of a controller.

Values are ranked based on their expected utility. For instance, when instantiating a node with an action, we select actions in decreasing order of expected utility measured by the upper bound at the belief corresponding to the normalized state occupancy frequency of the node. Let $b(s)$ be the belief proportional to the state occupancy frequency at node n. We then rank actions according to $\sum_s b(s)\bar{Q}(n,s,a)$. Similarly, when assigning an edge to a node, we select nodes in decreasing order of expected utility measured by the upper bound. Again, let $b(s)$ be the belief proportional to the state occupancy at node n. We then compute the reachable belief $b(s') \propto \sum_s b(s) \Pr(s'|s,\phi(n)) \Pr(o|s,\phi(n))$ and rank next nodes n' according to $\sum_{s'} b(s')\bar{Q}(n',s',\phi(n'))$.

5.3 Observation Aggregation

The size of the controller greatly influences the amount of search because it determines the number of variables. While we bound the number of nodes, most of the variables in the search correspond to edges instead of nodes since there are $|O|$ edges for each node. As a result, problems with many observations tend to have a search space that is intractable, even when the number of nodes is small. We propose to bound the number of edges. More precisely, we propose to merge several edges together in a way that only a bounded number of edge groups need to be instantiated. Grouping edges really corresponds to merging observations. We show that for

several benchmarks, merging observations by grouping edges does not hurt the performance in the sense that a good controller can still be found while cutting the search space.

Hoey et al. [2005] demonstrated that observations can be merged without any loss as long as they do not lead to different decisions. In a controller, observations label edges which are used to decide which conditional plan will be executed next. If a problem has more observations than the number of nodes, then observations will be partitioned into at most $|N|$ groups. In practice, the optimal controller will often have fewer than $|N|$ observation groups per node since the graph is often sparse (i.e., each node links to only a few nodes).

We propose to bound the number of edge groups (and therefore observation groups). Let $|E|$ be the maximum number of edge groups. We propose to instantiate $|E| - |N|$ individual edges as usual. Then $|N|$ edge groups are formed by grouping together all unassigned edges for each node. The edges in a group will be instantiated to the same value simultaneously, which reduces the amount of search. Since edges are instantiated in decreasing order of occupancy frequency, the remaining edges that are grouped together at each node are the ones with the lowest occupancy frequency and therefore the loss of accuracy due to grouping should be small.

6 Experiments

We compare Meuleau's search technique with our improved branch and bound with and without pruning equivalent controllers as well as three optimization techniques for stochastic controllers: bounded policy iteration (BPI) with escape [Poupart and Boutilier, 2003], quadratically constrained linear programming (QCLP) [Amato *et al.*, 2009], expectation maximization (EM) with forward search [Poupart *et al.*, 2011b]. The experiments are conducted with some benchmark problems and a real-world wayfinding POMDP that runs on a mobile phone.

6.1 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" [Hoey *et al.*, 2012] 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. 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—initially designed with our technique [Grześ *et al.*, 2013] for engineering POMDPs.

6.2 Results

Table 1 summarizes the results. The branch and bound (B&B) techniques find FSCs with highest value $V(b_0)$ for a fixed number of nodes and close to the upper bound computed by GapMin [Poupart *et al.*, 2011a]. For our improved B&B with and without pruning, the number of edge groups is bounded as indicated in the column #edges. This did not diminish the value of the controllers found except for lacasa3-ext. #evals indicates the number of (partial) controllers that were evaluated to complete the search. Although the controllers are small, the search space is huge. Meuleau's B&B was not able to complete the search within 72 hours for most problems. Our improved B&B with tighter upper bounds and prioritized variable/value selection significantly reduced the search and an additional reduction was achieved by pruning equivalent controllers. BPI, QCLP and EM search in the space of stochastic controllers and therefore have an advantage since the best stochastic controller of a given size may have higher value than the best deterministic controller of the same size. Nevertheless, they are prone to local optima and as a result they found controllers with lower average value than B&B.

While our approach clearly improves the state of the art for B&B, the time taken to find a controller is still significantly higher than the time taken by search techniques for stochastic controllers, point-based techniques and online search techniques. However, as illustrated with the LaCasa domain, there is a need for techniques that can reliably find good controllers that can be executed with minimal online computation. In these cases, the additional offline cost to find the controller is acceptable and worthwhile.

7 Conclusion

This work shows how to improve branch and bound by using tighter bounds, prioritizing the selection of variables/values, bounding the number of edge groups and pruning equivalent controllers. The controllers that we considered in this work are Moore automata. In the future, it would be interesting to extend this work to Mealy automata [Amato *et al.*, 2010] which can encode the same policies as Moore automata with fewer nodes, but more parameters (since the action mapping depends on nodes and observations instead of nodes only).

Acknowledgments

This work was supported by the Ontario Ministry of Research and Innovation, NSERC, Toronto Rehabilitation Institute and the Alzheimer's Association grant ETAC-10-173237.

Table 1: Initial lower bound, LB-init, indicates the lower bound used to initialize the search in all branch and bound algorithms. If not specified, LB was set to the maximum value of a random controller and a one-node controller. "Upper bound" is the upper bound on the optimal value for arbitrarily large controllers as computed by GapMin with a 10,000-second time limit. Meuleau's B&B is our implementation of Meuleau's branch and bound. Improved B&B is our branch and bound technique without any pruning as described in Section 5. Improved B&B with pruning is our branch and bound (Section 5) with pruning (Section 4). '-' means that the algorithm did not finish the search within 72 hours. '$*$' means that QCLP did not finish on the NEOS server due to insufficient memory (3GB limit). When # of edges is provided, it indicates a reduced number of edges in the optimized controller.

problem (# of nodes)	algorithm	$V(b_0)$	SEM	time [s]	# of evaluations	# of edges	LB-init
chainOfChains3 (10)	Meuleau's B&B	-		-	-		
$\|S\| = 10$	improved B&B	**157**	$\pm$ 0	2.86	236		
$\|A\| = 4, \|O\| = 1$	improved B&B with pruning	**157**	$\pm$ 0	1.67	81		
Upper bound = 157	QCLP	0	$\pm$ 0	0.16			
	BPI	25.7	$\pm$ 0.77	4.25			
	EM	62.6	$\pm$ 9.46	21.18			
hhepisobs_woNoise (8)	Meuleau's B&B	-		-	-		8.6
$\|S\| = 20$	improved B&B	**8.64**	$\pm$ 0	6.78	505		8.6
$\|A\| = 4, \|O\| = 6$	improved B&B with pruning	**8.64**	$\pm$ 0	4.48	405		8.6
Upper bound = 8.64	QCLP	0	$\pm$ 0	5.20			
	BPI	0	$\pm$ 0	0.41			
	EM	0	$\pm$ 0	0.79			
lacasa1 (6)	Meuleau's B&B	**294.0**	$\pm$ 0	209204.47	3096207114		
$\|S\| = 16$	improved B&B	**294.0**	$\pm$ 0	1121.17	4156430		
$\|A\| = 2, \|O\| = 3$	improved B&B with pruning	**294.0**	$\pm$ 0	39.57	143943		
Upper bound = 294.3	QCLP	293.8	$\pm$ 0.1	1.76			
	BPI	290.8	$\pm$ 0.15	0.30			
	EM	293.5	$\pm$ 0	0.24			
lacasa3 (3)	Meuleau's B&B	-		-	-		
$\|S\| = 640$	improved B&B	**292.0**	$\pm$ 0	514.44	1586	6	
$\|A\| = 5, \|O\| = 12$	improved B&B with pruning	**292.0**	$\pm$ 0	347.88	788	6	
Upper bound = 294.9	QCLP	$*$		$*$	$*$		
	BPI	288.2	$\pm$ 0.57	20.38			
	EM	290.5	$\pm$ 0.02	40.71			
lacasa3-ext (3)	Meuleau's B&B	**292.0**	$\pm$ 0	1146.73	18162.14		
$\|S\| = 1920$	improved B&B	291.0	$\pm$ 0	2385.37	5958.62	6	
$\|A\| = 5, \|O\| = 3$	improved B&B with pruning	291.0	$\pm$ 0	1364.03	2964.52	6	
Upper bound = 295.6	QCLP	$*$		$*$	$*$		
	BPI	283.8	$\pm$ 0.30	4.01			
	EM	283.6	$\pm$ 0	261.15			
machine (6)	Meuleau's B&B	-		-	-		62.2
$\|S\| = 256$	improved B&B	-		-	-	10	62.2
$\|A\| = 4, \|O\| = 16$	improved B&B with pruning	**62.6**	$\pm$ 0	52100	338486	10	62.2
Upper bound = 63.8	QCLP	62.47	$\pm$ 0.16	2640			
	BPI	26.6	$\pm$ 0.77	0.74			
	EM	62.43	$\pm$ 0.07	101.1			
tiger.95 (5)	Meuleau's B&B	**19.3**	$\pm$ 0	15.07	911940		
$\|S\| = 2$	improved B&B	**19.3**	$\pm$ 0	15.49	83359		
$\|A\| = 3, \|O\| = 2$	improved B&B with pruning	**19.3**	$\pm$ 0	1.42	4418		
Upper bound = 19.3	QCLP	-6.3	$\pm$ 3.79	0.70			
	BPI	-20.2	$\pm$ 0.12	0.06			
	EM	6.91	$\pm$ 2.48	0.15			
4x5x2.95 (5)	Meuleau's B&B	-		-	-		
$\|S\| = 39$	improved B&B	**2.02**	$\pm$ 0	1738.92	409980	10	
$\|A\| = 4, \|O\| = 4$	improved B&B with pruning	**2.02**	$\pm$ 0	639.99	206317	10	
Upper bound = 2.08	QCLP	1.43	$\pm$ 0.07	0.75			
	BPI	0.55	$\pm$ 0.09	0.22			
	EM	0.85	$\pm$ 0.04	1.12			

References

[Amato *et al.*, 2009] C. Amato, D. Bernstein, and S. Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Journal of Autonomous Agents and Multi-Agent Systems*, 2009.

[Amato *et al.*, 2010] C. Amato, B. Bonet, and S. Zilberstein. Finite-state controllers based on mealy machines for centralized and decentralized POMDPs. In *AAAI*, 2010.

[Braziunas and Boutilier, 2004] D. Braziunas and C. Boutilier. Stochastic local search for POMDP controllers. In *AAAI*, pages 690–696, 2004.

[Grześ *et al.*, 2013] Marek Grześ, Jesse Hoey, Shehroz S. Khan, Alex Mihailidis, Stephen Czarnuch, Daniel Jackson, and Andrew Monk. Relational approach to knowledge engineering for POMDP-based assistance systems as a translation of a psychological model. *International Journal of Approximate Reasoning*, 2013. http://dx.doi.org/10.1016/j.ijar.2013.03.006.

[Hansen, 1998] E. Hansen. An improved policy iteration algorithm for partially observable MDPs. In *NIPS*, 1998.

[Hauskrecht, 2000] Milos Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[Hoey and Poupart, 2005] Jesse Hoey and Pascal Poupart. Solving POMDPs with continuous or large discrete observation spaces. In *Proc. of the International Joint Conference on Artificial Intelligence*, pages 1332–1338, July 2005.

[Hoey *et al.*, 2012] Jesse Hoey, Xiao Yang, Eduardo Quintana, and Jesús Favela. Lacasa: Location and context-aware safety assistant. In *Proceedings of the International Conference on Pervasive Computing Technologies for Healthcare*, pages 171–174, 2012.

[Hopcroft, 1971] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical Report StaN-CS-70-190, Stanford University, Department of Computer Science, 1971.

[Huffman, 1954] David A Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257(3):161–190, 1954.

[Meuleau *et al.*, 1999] Nicolas Meuleau, Kee-Eung Kim, Leslie Pack Kaelbling, and Anthony R. Cassandra. Solving POMDPs by searching the space of finite policies. In Kathryn B. Laskey and Henri Prade, editors, *UAI*, pages 417–426. Morgan Kaufmann, 1999.

[Moore, 1956] Edward F Moore. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956.

[Poupart and Boutilier, 2003] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *NIPS*. MIT Press, 2003.

[Poupart *et al.*, 2011a] Pascal Poupart, Kee-Eung Kim, and Dongho Kim. Closing the gap: Improved bounds on optimal POMDP solutions. In Fahiem Bacchus, Carmel Domshlak, Stefan Edelkamp, and Malte Helmert, editors, *ICAPS*. AAAI, 2011.

[Poupart *et al.*, 2011b] Pascal Poupart, Tobias Lang, and Marc Toussaint. Analyzing and escaping local optima in planning as inference for partially observable domains. In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, editors, *ECML/PKDD (2)*, volume 6912 of *Lecture Notes in Computer Science*, pages 613–628. Springer, 2011.

[Toussaint *et al.*, 2006] M. Toussaint, S. Harmeling, and A. Storkey. Probabilistic inference for solving (PO)MDPs. Technical Report EDI-INF-RR-0934, School of Informatics, University of Edinburgh, 2006.

[Vlassis *et al.*, 2012] Nikos Vlassis, Michael L. Littman, and David Barber. On the computational complexity of stochastic controller optimization in POMDPs. *TOCT*, 4(4):12, 2012.