Point-Based POMDP Solving with Factored Value Function Approximation

Tiago S. Veiga
Institute for Systems and Robotics
Instituto Superior Técnico
Universidade de Lisboa
Lisbon, Portugal
tsveiga@isr.ist.utl.pt

Matthijs T. J. Spaan
Delft University of Technology
Delft, The Netherlands
m.t.j.spaan@tudelft.nl

Pedro U. Lima
Institute for Systems and Robotics
Instituto Superior Técnico
Universidade de Lisboa
Lisbon, Portugal
pal@isr.ist.utl.pt

Abstract

Partially observable Markov decision processes (POMDPs) provide a principled mathematical framework for modeling autonomous decision-making problems. A POMDP solution is often represented by a value function comprised of a set of vectors. In the case of factored models, the size of these vectors grows exponentially with the number of state factors, leading to scalability issues. We consider an approximate value function representation based on a linear combination of basis functions. In particular, we present a backup operator that can be used in any point-based POMDP solver. Furthermore, we show how under certain conditions independence between observation factors can be exploited for large computational gains. We experimentally verify our contributions and show that they have the potential to improve point-based methods in policy quality and solution size.

Introduction

Partially observable Markov decision processes (POMDPs) provide a powerful framework for planning under sensing and actuation uncertainty, but suffer from scalability issues. When modeling real-world problems, *factored* POMDP models are often used (Boger et al. 2005; Williams and Young 2007; Spaan, Veiga, and Lima 2010). They allow for capturing problem structure by representing models in a two-stage dynamic Bayesian network (Boutilier and Poole 1996; Hansen and Feng 2000). Further reductions in model size can be obtained by representing the conditional probability tables as algebraic decision diagrams (ADDs) (Poupart 2005; Shani et al. 2008).

Structured model representations, however, do not guarantee any type of structure in the solution (Koller and Parr 1999). POMDP solutions are often represented as a set of so-called α-vectors that form a piecewise linear and convex value function. As each vector needs to span the state space, its size grows exponentially with the number of state factors. An ADD representation provides only partial relief, since when each entry in an α-vector has a different value the corresponding ADD will consist of a full tree. Instead, in our work we focus on value function approximation to deal with large state spaces. This allows us to keep each component of the value function small.

Our contributions in this paper are three-fold. First, we extend a linear value function approximation scheme (Guestrin, Koller, and Parr 2001b) for use in point-based algorithms, a popular family of approximate POMDP algorithms. An important point is that our work focuses on optimizing a basic operation—the backup of a single belief point—in POMDP algorithms. Hence, our advances can be used to speed up many POMDP solvers.

Second, we consider the case of factored observations and show how under certain conditions we can gain large computational advantages. The number of observations is a major source of computational complexity and has been largely neglected in the literature on solving factored POMDPs.

Third, we provide the first experimental evaluation of linear value function approximation in a POMDP context. When compared to the fully observable setting, linear value function approximation in POMDPs faces challenges regarding computation time due to the additional complexity introduced by the observation variables. However, we experimentally test each algorithmic contribution and show that they have the potential to improve point-based methods in policy quality and solution size.

POMDP Background

A POMDP (Kaelbling, Littman, and Cassandra 1998) can be represented by a tuple $\langle S, A, O, T, \Omega, R, h, \gamma \rangle$. At any time step the environment is in a state $s \in S$, the agent takes an action $a \in A$ and receives a reward $R(s, a)$ from the environment as a result of this action, while the environment switches to a new state s' according to a known stochastic transition model $T : p(s'|s, a)$. After transitioning to a new state, the agent perceives an observation $o \in O$ that may be conditional on its action, which provides information about the state s' through a known stochastic observation model $\Omega : p(o|s', a)$. The agent's task is defined by the reward it receives at each timestep t and its goal is to maximize its expected long-term reward $E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$, where γ is a discount rate, $0 \leq \gamma < 1$.

A POMDP can be transformed to a belief-state MDP in which the agent summarizes all information about its past using a belief vector $b(s)$. The initial state of the system is drawn from the initial belief b_0. Every time the agent takes

an action a and observes o, b is updated by Bayes' rule:

$$b_a^o(s') = \frac{p(o|s',a)}{p(o|a,b)} \sum_{s \in S} p(s'|s,a)b(s), \quad (1)$$

where $p(o|a,b)$ is a normalizing constant. In a POMDP, a policy π can be characterized by a value function $V^\pi : \Delta(S) \rightarrow \mathbb{R}$, which is defined as the expected future discounted reward $V^\pi(b)$ the agent can gather by following π starting from belief b. The value of an optimal policy π^* is defined by the optimal value function V^*:

$$V^*(b) = \max_{a \in A} \Big[\sum_{s \in S} R(s,a)b(s) + \gamma \sum_{o \in O} p(o|b,a)V^*(b_a^o) \Big], \quad (2)$$

with b_a^o given by (1).

The POMDP value function has been proven to be piecewise linear and convex (PWLC) for finite horizons, allowing for compact representation. For the infinite-horizon case, it can be approximated to any desired precision using a PWLC value function. We parameterize a value function V_n at stage n by a finite set of $|S|$-dimensional vectors or hyperplanes $\{\alpha_n^k\}$, indexed by k. Given a set of vectors, the value of a belief b is given by $V_n(b) = \max_{\{\alpha_n^k\}_k} b \cdot \alpha_n^k$.

Many of the exact and approximate algorithms exploit the fact that the POMDP value function is PWLC. Point-based POMDP methods (Pineau, Gordon, and Thrun 2003; Spaan and Vlassis 2005; Kurniawati, Hsu, and Lee 2008) in particular compute an α-vector at a belief point b with the operator $\texttt{backup}(b)$:

$$\texttt{backup}(b) = \operatorname*{argmax}_{\{g_a^b\}_{a \in A}} b \cdot g_a^b, \quad \text{with} \quad (3)$$

$$g_a^b = R(s,a) + \gamma \sum_o \operatorname*{argmax}_{\{g_{ao}^k\}} b \cdot g_{ao}^k, \text{ and} \quad (4)$$

$$g_{ao}^k(s) = \sum_{s'} p(o|s',a)p(s'|s,a)\alpha_n^k(s'). \quad (5)$$

In our work, we considered factored POMDP models that use a two-stage dynamic Bayesian network (DBN) representation (Boutilier and Poole 1996; Hansen and Feng 2000). Figure 1 shows an example DBN for the fire fighting problem (which we will use in our experiments). In this case the state and observation spaces are factored as follows:

$$S = X_1 \times X_2 \times \ldots \times X_i \times \ldots \times X_{n_S}, \quad (6)$$

$$O = O_1 \times O_2 \times \ldots \times O_i \times \ldots \times O_{n_O}, \quad (7)$$

where n_S are the number of state factors and n_O the number of observation factors in the model. Subsequently, rewards can be defined over subsets of state factors:

$$R(S,A) = \sum_j R(Y_j, A), \quad (8)$$

where Y_j is a subset of the state factors that make up S. We call the scope of a variable the set of all its parents in a DBN, i.e., all the variables on which it is conditionally dependent. We denote the scope of a variable X_i by $\Gamma(X_i)$.

Maintaining a factorized but exact belief state is typically not desirable for reasons of tractability, but bounded approximations are possible (Boyen and Koller 1998) in which the exact belief b is approximated by the product of marginals over individual state factors b_i. This is a common approach in factorized POMDP solving (Poupart 2005).

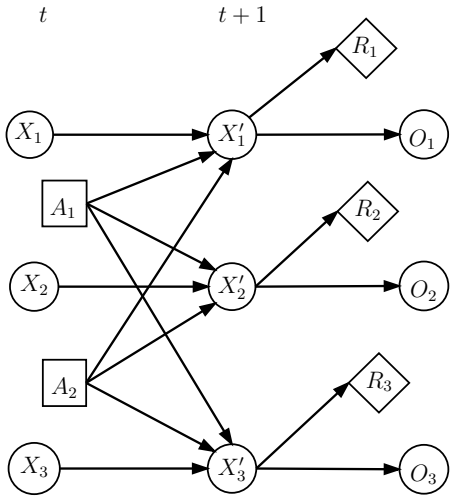


Figure 1: Two-stage DBN for the adapted fire fighting problem, where $X_i = \{0, 1, \ldots, k-1\}$; $O_i = \{0, 1, \ldots, l-1\}$.

Linear value function approximation

As discussed in the introduction, approximating value functions by linear regression has been proposed as a technique to induce a compact representation of the value function for factored fully observable Markov decision process (MDP) and factored POMDP models.

Linear Value Functions in MDPs

To approximate value functions by linear regression we define the space of allowable value functions $\mathcal{V} \in \mathcal{H} \subseteq \mathbb{R}^{|S|}$ via a set of n_h basis functions $H = \{h_1, \ldots, h_{n_h}\}$ (Tsitsiklis and van Roy 1996). A linear value function is a function defined over a set of basis functions that can be written as

$$\mathcal{V}(s) = \sum_{i=1}^{n_h} \omega_i h_i(s) \quad (9)$$

for some coefficients $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_{n_h})$. $\mathcal{H}$ is the linear subspace of $\mathbb{R}^{|S|}$ spanned by the basis functions H.

A *factored* linear value function is defined as a linear function over the basis set $\{h_1, \ldots, h_{n_h}\}$, where the scope of each h_i is restricted to some subset of variables $\mathbf{C}_i$. Guestrin, Koller, and Parr (2001a) and Guestrin et al. (2003) present solutions in which value iteration can be implemented more efficiently using factored representations. If we see value functions as sums of functions which depend on a restricted set of variables, then it is possible to perform value iteration in a compact way. The Bellman recursion using a factored representation becomes:

$$V^{n+1}(\mathbf{x}) = R(\mathbf{x},a) + \gamma \sum_i \omega_i^n \sum_{\mathbf{x}' \in \mathbf{X}} p(\mathbf{x}'|\mathbf{x},a)h_i(\mathbf{x}). \quad (10)$$

The new weight vector $\boldsymbol{\omega}^{n+1}$ is found by projecting V^{n+1} back onto $\mathcal{H}$, which is equivalent to finding the solution of $\text{argmin}_{\boldsymbol{\omega}^{n+1}} \| \sum_{i=1}^{n_h} \omega_i^{n+1} h_i(s) - V^{n+1} \|_p$ for any given p-norm.

Extension to POMDPs

Linear value function approximation has been extended to exact POMDP solving (Guestrin, Koller, and Parr 2001b). Recall that a value function in an infinite-horizon POMDP can be approximated arbitrarily well by a finite set of α-vectors. The key idea is that we can approximate each vector as a linear combination of basis functions.

In the POMDP case, we have a formulation in which the state is represented by a factored representation $\mathbf{x}$ and α-vectors are approximated by $\widetilde{\alpha}$-vectors, written as a linear combination of basis functions h_i:

$$\widetilde{\alpha}(\mathbf{x}) = \sum_{i=1}^{n_h} \omega_{\alpha,i} h_i(\mathbf{c}_i). \tag{11}$$

At each value iteration step we can compute all new α-vectors which make up the new value function V^{n+1} and then project those onto $\mathcal{H}$.

To our knowledge, the POMDP extension has not been evaluated experimentally. Furthermore, Guestrin, Koller, and Parr (2001b) focused on exact POMDP algorithms such as incremental pruning, which scale poorly due to the blowup in the number of vectors.

A point-based POMDP method with linear value function approximation

We introduce a point-based algorithm for approximate POMDP solving that uses linear value function approximation. Recall that point-based methods compute the vector at each point b with the operator $\texttt{backup}(b)$ (3).

First, we rewrite (5) using (11):

$$g_{ao}^k(\mathbf{x}) = \sum_{i=1}^{n_h} \omega_{k,i} \sum_{\mathbf{x}'} p(o|\mathbf{x}', a) p(\mathbf{x}'|\mathbf{x}, a) h_i(\mathbf{x}'). \tag{12}$$

The newly computed g_{ao}^k vectors will generally not be in the space spanned by the basis functions. That is, we are interested in finding the set of weights which gives the solution of $\boldsymbol{\omega}_{ao}^k = \text{argmin}_{\boldsymbol{\omega}} \| \sum_{i=1}^{n_h} \omega_{k,i} h_i(c_i) - g_{ao}^k \|_p$ for any given p-norm. If we consider the infinity norm, we are able to use factored maxnorm projection methods (Guestrin, Koller, and Parr 2001a) to project those vectors back onto this space. In the end, for each vector we only need to save an array of n_h values (the weights in (11)).

With this, we may easily compute g_a^b vectors, using a linear representation for g_{ao}^k and $R(s,a)$ vectors. The operations needed to compute (4) are the inner product with a belief point using an approximate representation, which can be computed as follows:

$$b \cdot \widetilde{\alpha} = \sum_{\mathbf{x} \in \mathbf{X}} \sum_{i=1}^{n_h} \omega_{\alpha,i} h_i(\mathbf{x}) b(\mathbf{x}) \tag{13}$$

$$= \sum_{i=1}^{n_h} \omega_{\alpha,i} \sum_{\mathbf{c}_i \in \mathbf{C}_i} h_i(\mathbf{c}_i) b_i(\mathbf{c}_i), \tag{14}$$

Algorithm 1 Point-based backup with linear value function approximation

```
for all a ∈ A do {independent of b, can be cached}
    ω_R ← Project R(s,a) vectors to H.
    for all o ∈ O do
        Compute g^k_ao vectors using (12).
        ω^k_ao ← Project g^k_ao vectors to H.
    end for
end for
for all a ∈ A do
    Compute ω^b_a vectors using (16).
end for
Find the maximizing vector backup(b) using (14).
```

and the sum of two vectors represented by basis functions:

$$\sum_{i=1}^{n_h} \omega_{1,i} h_i(\mathbf{x}) + \sum_{j=1}^{n_h} \omega_{2,j} h_j(\mathbf{x}) = \sum_{i=1}^{n_h} (\omega_{1,i} + \omega_{2,i}) h_i(\mathbf{x}). \tag{15}$$

Therefore, computing the g_a^b vectors (4) can be performed using solely a reduced representation with weight vectors:

$$\boldsymbol{\omega}_a^b = \boldsymbol{\omega}_R + \gamma \sum_o \underset{\boldsymbol{\omega}_{ao}^k}{\text{argmax}} \left(\sum_{i=1}^{n_h} \omega_{ao,i}^k \sum_{\mathbf{c}_i \in \mathbf{C}_i} h_i(\mathbf{c}_i) b_i(\mathbf{c}_i) \right). \tag{16}$$

Here, $\boldsymbol{\omega}_R$ and $\boldsymbol{\omega}_{ao}^k$ represent the projections onto the space spanned by the basis functions of, respectively, the immediate reward $R(s,a)$ and g_{ao}^k vectors which result from (12). The inner product is performed using an approximated representation as in (14), and summation of weight vectors as in (15).

Algorithm 1 summarizes the steps needed to perform the backup step for each belief point using point-based methods with linear value function approximations. In our experiments, we implemented this backup operator in a straightforward point-based method called Perseus (Spaan and Vlassis 2005), but it can be used in any method that backs up beliefs.

Exploiting factored observation spaces

Let us now consider that we have an observation space represented in a factored way. Also, assume that we are able to write the factored state space S as the cross product of subsets of variables U_i. Each subset U_i contains the parents of observation i in the DBN (that is, $\mathbf{u}_i' \in \Gamma(O_i)$). We will also assume that all subsets are disjoint ($\forall\, i, j \;:\; U_i' \cap U_j' = \emptyset$). The key idea is that since basis functions are defined over a subset of the state space we may exploit similarities between the scope of basis functions and the structure of the DBN, although imposing some restrictions on the choice of basis functions. Those limitations restrict the set of problem domains to which this method can be applied.

Looking at Equation 12 we can replace both observation and transition functions by their factored, thus independent, representation. Additionally, we will assume that for each problem we define n_O basis functions, each one with domain U_i, that is, the domain of the basis functions is the

same as of the observation variables. From this, we get the following:

$$g^k_{ao}(\mathbf{x}) = \sum_{i=1}^{n_O} \omega_{k,i} \sum_{\mathbf{u}'_1 \in \Gamma(O_1)} \cdots \sum_{\mathbf{u}'_{n_O} \in \Gamma(O_{n_O})} p(o_1|\mathbf{u}'_1, a) \times \ldots \times p(o_{n_O}|\mathbf{u}'_{n_O}, a) \times \times p(\mathbf{u}'_1|\Gamma(U'_1), a) \times \ldots \times p(\mathbf{u}'_{n_O}|\Gamma(U'_{n_O}), a) \times h_i(\mathbf{u}'_i). \quad (17)$$

Due to the independence between variables we can rewrite as follows:

$$g^k_{ao}(\mathbf{x}) = \sum_{i=1}^{n_O} \omega_{k,i} \left(\sum_{\mathbf{u}'_1 \in \Gamma(O_1)} p(o_1|\mathbf{u}'_1, a) p(\mathbf{u}'_1|\Gamma(U'_1), a) \right) \times \ldots \times \left(\sum_{\mathbf{u}'_{n_O} \in \Gamma(O_{n_O})} p(o_{n_O}|\mathbf{u}'_{n_O}, a) p(\mathbf{u}'_{n_O}|\Gamma(U'_{n_O}), a) \right) \times h_i(\mathbf{u}'_i). \quad (18)$$

In this representation, at iteration i all terms are constant except for term i, which includes the corresponding value of the basis function. Therefore, we can consider a set of auxiliary vectors defined as:

$$d_{ao_i}(\mathbf{x}) = \sum_{\mathbf{u}'_i \in \Gamma(O_i)} p(o_i|\mathbf{u}'_i, a) p(\mathbf{u}'_i|\Gamma(U'_i), a), \quad (19)$$

$$f_{ao_i}(\mathbf{x}) = \sum_{\mathbf{u}'_i \in \Gamma(O_i)} p(o_i|\mathbf{u}'_i, a) p(\mathbf{u}'_i|\Gamma(U'_i), a) h_i(\mathbf{u}'_i), \quad (20)$$

and replace them in (18):

$$g^k_{ao}(\mathbf{x}) = \sum_{i=1}^{n_O} \omega_{k,i} \prod_{\substack{j=1 \\ j \neq i}}^{n_O} f_{ao_i}(\mathbf{x}) d_{ao_j}(\mathbf{x}) \quad (21)$$

$$= \sum_{i=1}^{n_O} \omega_{k,i} f_{ao_i}(\mathbf{x}) \prod_{\substack{j=1 \\ j \neq i}}^{n_O} d_{ao_j}(\mathbf{x}). \quad (22)$$

Given that we can precompute these auxiliary vectors prior to performing value iteration, it is possible to reduce the number of numerical operations needed during computation of g^k_{ao} vectors, thereby reducing the complexity of this step.

Experiments

To test the viability of the ideas presented in this paper, we performed a series of experiments.

Experimental setup

Our experiments have been performed with a Matlab implementation of Perseus (Spaan and Vlassis 2005) using basis functions to approximate value functions. All tests used a belief set of 500 belief points, and results are averaged over 10 times the number of start positions runs of the algorithm (we test all possible states as start positions). The benchmark problem domains for our tests are a variation of the fire fighting problem (Oliehoek et al. 2008) and the network management problem (Poupart 2005).

Fire fighting In this problem a number of agents must ensure there are no fires burning. A model of the dynamics of the problem, represented as a dynamic Bayesian network is shown in Figure 1. We consider a variation of the problem with 2 agents and 3 houses. Each house can be in one of k fire levels, $X_i = \{0, 1, \ldots, k-1\}$, and there are l observation levels at each house. In our adaptation it is possible to observe all houses independent of the agents' positions. Therefore, we include one observation variable corresponding to the observation of each house. Like in the original problem the reward function is additive, that is, there is a reward for each house, according to its fire level. Agents are rewarded with $-x_i$ for each house i.

If a fire fighter is present at a house, its fire level will lower one level with probability 0.8, while if two fire fighters go to the same house, its fire level will certainly decrease to the lowest level. If no fire fighter is present at a house, its fire level will increase with probability 0.6. The observation model for 3 fire levels per house and 2 observation levels is shown in Figure 2a.

Network management In this domain, we consider a model with 3 machines in a *cycle* configuration (Poupart 2005). At any stage, each machine may be running or down, and a system administrator receives an observation for each machine. If he performs an action (ping or reboot) in a particular machine he can observe if it is running with a probability 0.95, otherwise he receives a null observation. We keep the rest of the problem parameters as in the original problem.

Comparing basis functions

Our first test consists of running the algorithm with several different basis functions. We present the set of basis functions tested in Table 2b. The number of basis functions is equal to the number of state factors, and vectors in table indicate how much the basis valuates each value of a state factor. For instance, in the fire fighting problem with n houses there are n basis functions, where each function is a vector that gives a value for each possible fire level.

In Figure 2c we present the average sum of discounted rewards using each basis function, and in Figure 2d the total solution size needed to represent the value function, for each iteration step. Our results show that the choice of basis influences the method's performance. If we remember that the reward function for this problem (with 2 agents and 3 houses) is $R_i(X_i) = [0\ {-1}\ {-2}]$, we can compare the performance of each basis function with its similarities to the reward function. As Guestrin et al. (2003) stress for the MDP case, the success of the technique depends on the ability to capture the most important structure in the value function with a good choice of basis functions.

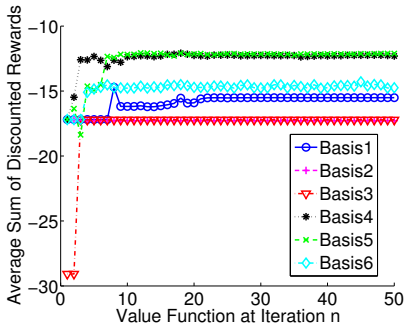
In our experiments, we distinguish between those bases which are related to the reward function, and those which are not. For instance, all bases except for basis 1 assign a higher value to lower fire levels, and decreasing values for increasing fire levels. Indeed, we notice that the performance of basis 1 is one of the worst in the test. It is also important

x_i	$p(o_1\|x_i)$	$p(o_2\|x_i)$
1	0.775	0.225
2	0.5	0.5
3	0.225	0.775

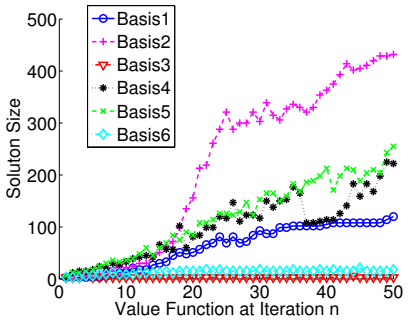
(a) Observation model

#	$h_i(X_i)$
1	$[-1 \quad 1 \quad -1]$
2	$[0 \quad -1 \quad -2]$
3	$[2 \quad 1 \quad 0]$
4	$[2 \quad 0 \quad -1]$
5	$[5 \quad -2 \quad -5]$
6	$[3 \quad 1 \quad -1]$

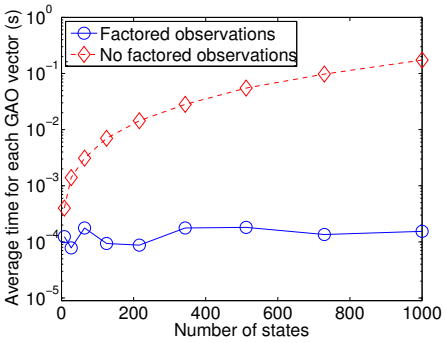
(b) Basis functions



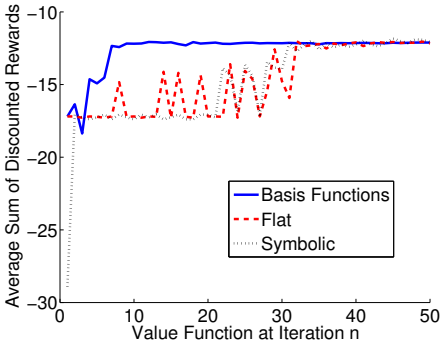
(c) Average sum of discounted rewards



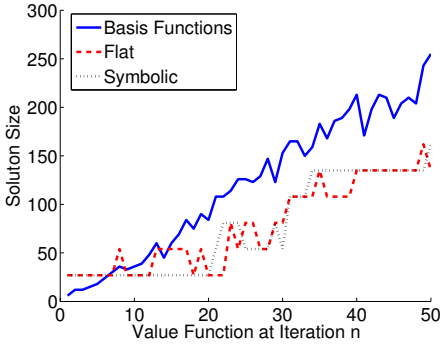
(d) Total # of values stored



(e) Factored observations (log-scale y-axis)



(f) Average sum of discounted rewards



(g) Total # of values stored

Figure 2: Fire fighting problem. (a) Observation model for $k = 3, l = 2$. (b) Basis functions tested. (c), (d) Comparison of different basis functions. (e) Exploiting factored observations, with increasing k but constant l. (f), (g) For $k = 3$ and $l = 2$, comparison of different representations for V^n: basis functions, flat vectors, and a symbolic (ADD) representation.

to be aware of numerical issues resulting from the choice of basis functions. In contrast to expectation, bases 2 and 3 do not perform well, when comparing with others in the test. In particular, basis 2 is a direct translation of the reward function to a basis function. We note that having a value of 0 in a basis function results in numerical issues when projecting vectors onto the space spanned by the basis functions. Our results indicate that it can be beneficial to develop a fully automated way of choosing basis functions, which will need to capture the most important structure of each problem.

Factored observations

We introduced a method to speed up computation of g_{ao}^k by exploiting a factored representation of the observation model. In Figure 2e we compare the average computation time for each g_{ao}^k vector when using Equation 12 and Equation 22. The implementation of this test is based on the adapted fire fighting problem. We compare different instances with a constant number of observations (2 observation levels per house), while we increase the number of states (by increasing one fire level at the time).

We conclude that there is an excellent speedup in this step with an increasing number of states, confirming our hypothesis. Since we reduce the number of operations over the state space, its size is the most significant parameter that affects the efficiency of speeding up this step. Therefore, the average time while exploiting factored observations using (22) does not increase in contrast to computing the g_{ao}^k vectors in the traditional way using (12).

Comparison with other methods

We tested our method against two other point-based methods, regular Perseus (Spaan and Vlassis 2005) and Symbolic Perseus (Poupart 2005). Perseus is a point-based POMDP solver which uses a flat representation for its models and solutions, while Symbolic Perseus uses a factored ADD representation. We report the average sum of discounted rewards (Figure 2f and 3a) and the solution sizes (Figure 2g and 3b) with each method. The latter is computed as the total number of values needed to store the value function which provides a fair measure when comparing different representations. For this test we used the best basis function found in the tests from Figure 2b (Basis 5, $h_i(X_i) = [5 \ -2 \ -5]$) for the fire fighting problem domain, and a basis function $h_i(X_i) = [-5 \ 2]$ for the network management problem.

We observe that in these problem domains we can obtain good results by using value function approximation. After convergence, our average sum of discounted rewards is similar to those attained by other methods, which shows that it is possible to solve POMDPs with a value function based

on basis functions. Also, we observe that our method converges faster to a good solution: in both cases, before 10 value iteration steps, when other methods take 30 or more iterations to converge. In terms of solution size our representation presents a similar size as other methods when compared at the same points. However, if we compare the sizes of solutions at convergence, our method returns a smaller size. Also, our value function representation size grows slower than other representations when comparing in the long run. This may be explained by noting that we are projecting vectors onto smaller spaces (defined by the basis functions), therefore different vectors might be projected onto the same weights.

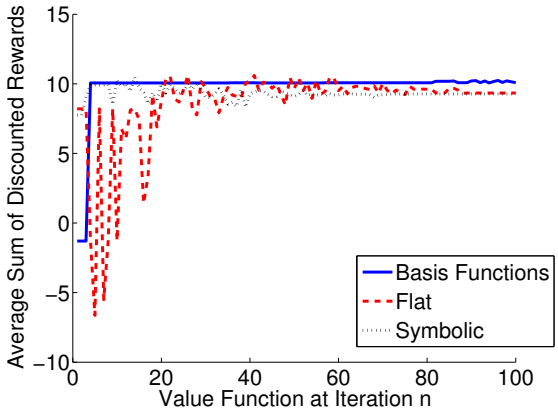
Our algorithm implements an extra operation, when compared to others, which is the projection of vectors onto the space spanned by the basis functions. This operation requires much of the computational effort in our algorithm, even if we use methods to take advantage of factored representations (Guestrin, Koller, and Parr 2001a). There are two steps inside this method: building the set of constraints and solving the resulting linear program. The former step is the most time consuming, scaling poorly with the number of states in each vector's scope. When compared to MDPs, the introduction of observation variables in POMDPs represents an additional layer of dependencies in the DBN, thus, additional complexity. This increases the scope of vectors used in point-based methods, in particular of the g_{ao}^k vectors. In the end, this results in worse computation times compared to flat and Symbolic Perseus on these problem domains.

Conclusions

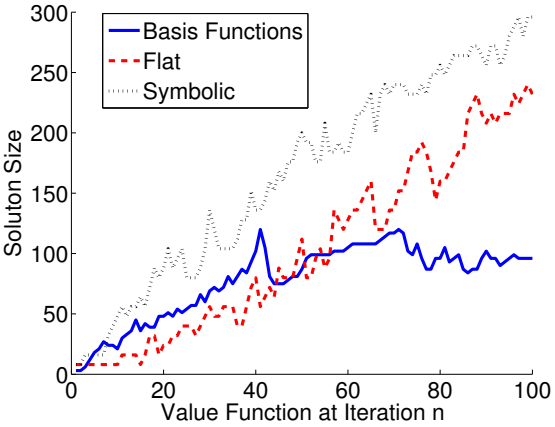
Planning under sensing and actuation uncertainty is a hard problem that has been formalized in the partially observable Markov decision process (POMDP) framework. In this paper, we explored the use of linear value function approximation for factored POMDPs, which had been proposed but not tested experimentally. We extended the use of linear value function approximation for factored POMDPs to point-based methods.

Methods based on linear value function approximation with factored models focus on exploiting independence between state factors and on reducing complexity by using the fact that most functions involved have restricted domains. When comparing to implementations of linear function approximation in MDPs, however, we face challenges due to the increased complexity that a POMDP introduces over an MDP due to observation variables. Hence, we also proposed a technique to exploit independence between observations, another major source of complexity when computing POMDP solutions.

We presented the first experimental tests with linear value function approximation in a POMDP context and we showed that it is possible to successfully approximate α-vectors through a set of basis functions. We experimentally verified our contributions by showing that using this method has the potential to improve point-based methods in policy quality and solution size. Our results show that we can get faster convergence with our approach, while having a



(a) Average sum of discounted rewards



(b) Total # of values stored

Figure 3: Network management problem with 3 machines. Comparison of different representations for V^n: basis functions, flat vectors, and a symbolic (ADD) representation.

smaller value function representation at the point of convergence. Projecting vectors onto the space spanned by the basis functions forms a bottleneck, however, resulting in worse computation times. On the other hand, under certain assumptions, exploiting independence between observation factors results in speedups of several orders of magnitudes.

Future improvements might consider to scale our algorithm and improve its efficiency and computation times compared to methods which use a full representation for the value function. We might further exploit factored observation spaces not only to speed up computation of g_{ao}^k vectors, but also to reduce the number of vectors evaluated at each step. Note that the number of vectors computed and evaluated is still directly proportional to the full observation space size and that factored representations might also be exploited in this step. Finally, the choice of basis functions is not straightforward but is crucial to the success of the methods discussed in this paper. For now it is the task of the system designer to choose the basis functions, but further developments to automate the choice of basis may improve performance.

Acknowledgments

This work was partially supported by Fundação para a Ciência e a Tecnologia (FCT) through grant SFRH/BD/70559/2010 (T.V.), as well as by strategic project PEst-OE/EEI/LA0009/2013.

References

Boger, J.; Poupart, P.; Hoey, J.; Boutilier, C.; Fernie, G.; and Mihailidis, A. 2005. A decision-theoretic approach to task assistance for persons with dementia. In *Proc. Int. Joint Conf. on Artificial Intelligence*.

Boutilier, C., and Poole, D. 1996. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*.

Boyen, X., and Koller, D. 1998. Tractable inference for complex stochastic processes. In *Proc. Uncertainty in Artificial Intelligence*, 33–42.

Guestrin, C.; Koller, D.; Parr, R.; and Venkataraman, S. 2003. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research* 19(1):399–468.

Guestrin, C.; Koller, D.; and Parr, R. 2001a. Max-norm projections for factored MDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 673–682.

Guestrin, C.; Koller, D.; and Parr, R. 2001b. Solving factored POMDPs with linear value functions. In *Proceedings IJCAI-01 Workshop on Planning under Uncertainty and Incomplete Information*.

Hansen, E. A., and Feng, Z. 2000. Dynamic programming for POMDPs using a factored state representation. In *Int. Conf. on Artificial Intelligence Planning and Scheduling*.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101:99–134.

Koller, D., and Parr, R. 1999. Computing factored value functions for policies in structured MDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 1332–1339.

Kurniawati, H.; Hsu, D.; and Lee, W. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. Robotics: Science and Systems*.

Oliehoek, F. A.; Spaan, M. T. J.; Whiteson, S.; and Vlassis, N. 2008. Exploiting locality of interaction in factored Dec-POMDPs. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*.

Poupart, P. 2005. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. Ph.D. Dissertation, University of Toronto.

Shani, G.; Poupart, P.; Brafman, R. I.; and Shimony, S. E. 2008. Efficient ADD operations for point-based algorithms. In *Proc. Int. Conf. on Automated Planning and Scheduling*.

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

Spaan, M. T. J.; Veiga, T. S.; and Lima, P. U. 2010. Active cooperative perception in network robot systems using POMDPs. In *Proc. International Conference on Intelligent Robots and Systems*, 4800–4805.

Tsitsiklis, J. N., and van Roy, B. 1996. Feature-based methods for large scale dynamic programming. *Machine Learning* 22(1-3):59–94.

Williams, J. D., and Young, S. 2007. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech and Language* 21(2):393–422.