Efficient Planning in Large POMDPs through Policy Graph Based Factorized Approximations

Joni Pajarinen[1], Jaakko Peltonen[1], Ari Hottinen[2], and Mikko A. Uusitalo[2]

[1] Aalto University School of Science and Technology,
Department of Information and Computer Science,
P.O. Box 15400, FI-00076 Aalto, Finland
{Joni.Pajarinen,Jaakko.Peltonen}@tkk.fi
[2] Nokia Research Center, P.O. Box 407, FI-00045 NOKIA GROUP, Finland
{Ari.Hottinen,Mikko.A.Uusitalo}@nokia.com

Abstract. Partially observable Markov decision processes (POMDPs) are widely used for planning under uncertainty. In many applications, the huge size of the POMDP state space makes straightforward optimization of plans (policies) computationally intractable. To solve this, we introduce an efficient POMDP planning algorithm. Many current methods store the policy partly through a set of "value vectors" which is updated at each iteration by planning one step further; the size of such vectors follows the size of the state space, making computation intractable for large POMDPs. We store the policy as a graph only, which allows tractable approximations in each policy update step: for a state space described by several variables, we approximate beliefs over future states with factorized forms, minimizing Kullback-Leibler divergence to the non-factorized distributions. Our other speedup approximations include bounding potential rewards. We demonstrate the advantage of our method in several reinforcement learning problems, compared to four previous methods.

1 Introduction

Planning under uncertainty is a central task in many applications, such as control of various robots and machines, medical diagnosis, dynamic spectrum access for cognitive radio, and many others. Such planning can often be described as a reinforcement learning problem where an agent must decide a behavior (action policy), and the quality of any policy can be evaluated in terms of a reward function. *Partially observable Markov decision processes* (POMDPs) [1] are a widely used class of models for planning (choosing good action policies) in such scenarios. In brief, in a POMDP the latent state of the world evolves according to a Markov model given each action chosen; the state is not directly observable, and end results of potential actions are not known, but the agent receives observations that depend on the state, and can plan ahead based on probabilistic beliefs about current and future states. For a survey of POMDP applications see, e.g., [2].

Policies are optimized in POMDPs by iterative algorithms. A central problem is that the optimization becomes computationally intractable when the size of the

J.L. Balcázar et al. (Eds.): ECML PKDD 2010, Part III, LNAI 6323, pp. 1–16, 2010.

underlying state space is large. We consider POMDPs with discrete state spaces; if the state is described by N variables, the number of states is at worst exponential in N and the number of state transitions is at worst exponential in $2N$. To combat computational intractability, many planning algorithms have been introduced with various approaches for improving efficiency [3–5]; we describe several approaches in Section 2.1. Overall, however, computational intractability remains a large problem which limits current applicability of POMDPs.

We present a novel method for efficient planning with POMDPs. In POMDPs the state can often be described by several variables whose individual transition probabilities do not depend on the whole state but only on a subset of variables. However, this does not yet ensure tractability: POMDP planning requires posterior probabilities of current and future states integrated over a distribution (belief) about previous states. Such integration is done over values of the whole previous state, and does not reduce to a computationally tractable form. However, the result can be *approximated* by a tractable form: in each such computation we use a factorized approximation optimized to minimize Kullback-Leibler divergence to the non-factorized intractable belief. We apply such factorized approximation in several central parts of the computation, which is organized based on a policy graph. The approximate computations ensure that beliefs over states remain in a factorized form, which crucially reduces complexity of evaluating and optimizing plans. We use a speedup based on computing bounds for potential policy rewards, to avoid evaluating policy alternatives that cannot compete with best existing policies; effectiveness of such pruning can be further increased with suitable ordering of evaluations. We describe our method in Section 3.

We compare the performance of our method to four existing POMDP solutions: two traditional approaches (Perseus [3] and HSVI [4]) and two methods designed for large problems (Symbolic Perseus [5] and Truncated Krylov Iteration combined with Perseus [5]). We compare the methods on four POMDP benchmark problems of scalable size, including two new benchmarks introduced here: the *Uncertain RockSample* problem, which is a more difficult variant of the traditional RockSample benchmark, and *Spectrum Access* which is adapted from a cognitive radio application and is described further below. Our method gets better policies than others in the same running time, and can handle large problems where other methods run out of memory, disk space, or time.

One increasingly important application area of reinforcement learning is *opportunistic spectrum access*, where devices such as cognitive radios detect available unused radio channels and exploit them for communication, avoiding collisions with existing users of the channels. This task can be formulated as a POMDP problem, and various POMDP solutions with different levels of model detail exist [6, 7]. Computational intractability is a problem for POMDP solutions: if the status of each channel (describing ongoing packet trains) is modeled with a detailed model having several states, like 15 in [6], state space grows exponentially as 15^N with respect to the number of channels N used by the model; this makes POMDP computation challenging. The simple solution, restricting policies to few channels only, is not desirable: the more channels one can take

into account, the more efficient will be the use of spectrum and the more benefit the users will get. We present a new benchmark problem for POMDPs called *Spectrum Access* which is directly adapted from our proposal for a cognitive radio solution [6]. We use spectrum access as one of the benchmark problems in the experiments, and show that our method yields the best results for it.

In the following, we first describe the basic concepts of POMDPs and review existing methods for planning in POMDP problems in Section 2; in Section 3 we present our solution; and in Section 4 we describe the comparison experiments including the two new benchmark problems. In Section 5 results are discussed. We give conclusions in Section 6.

2 Partially Observable Markov Decision Processes

A partially observable Markov decision process (POMDP) is defined completely by (1) a *Markov model* describing the possible state transitions and observation probabilities given each action of the agent, and (2) a *reward model* defining how much reward is given when performing an action in a certain state. Formally a POMDP consists of the finite sets of states S, actions A, observations O, and rewards $R : S \times A \to \mathcal{R}$. At each time step, the agent performs action a, the world transitions from its current state s to a new state s' chosen according to the transition probabilities $P(s'|s, a)$, and the agent receives observation o according to the observation probability $P(o|s', a)$. The reward at each time step is $R(s, a)$.

The goal of the agent is to choose her actions to maximize a cumulative reward over time. We discuss the typical infinite-horizon discounted objective [1] $E\left(\sum_{t=0}^{\infty} \gamma^t R_t\right)$, where γ is the discount factor, $0 < \gamma < 1$, and R_t is the reward at time step t. The exact state of the world is not known to the agent, but a probability distribution, which tells the probability for being in state s can be maintained. This distribution is the so-called belief b: we denote the whole distribution by b, and the belief (probability) of being in state s is $b(s)$. The Bayes formula for updating the belief, after doing action a and getting observation o is

$$b'(s'|b, a, o) = P(o|s', a) \sum_s P(s'|s, a)b(s)/P(o|b, a) \ , \tag{1}$$

where $b'(s'|b, a, o)$ is the posterior belief, given o, that after action a the world is in state s'. The normalization term is the overall observation probability given a and the belief about the starting state, $P(o|b, a) = \sum_{s'} P(o|s', a) \sum_s P(s'|s, a)b(s)$. An optimal action a maximizes expected total discounted reward over possible futures; a precise definition is given later. Choosing a, given belief b over current states s, entails considering all possible action–observation sequences into the future. A function choosing an action for each b is called a *plan* or *policy*.

A brute force approach to choosing optimal actions would yield exponential complexity for planning with respect to how many steps ahead it looks. Many state-of-the-art algorithms exploit the Bellman equation for planning:

$$V^*(b) = \max_{a \in A} \left[\sum_{s \in S} R(s, a)b(s) + \gamma \sum_{o \in O} P(o|b, a)V^*(b'(s'|b, a, o)) \right] \ , \tag{2}$$

where $V^*(b)$ is the value (total expected discounted reward) that can be attained when acting optimally if the current state is distributed according to belief b; we call this the "value of b" for short. Above, differently from our usual notation, $b'(s'|b,a,o)$ denotes the whole posterior distribution over states s', rather than a single value. The action a giving the maximum at right in (2) is optimal for belief b. State-of-the-art methods use Equation 2 iteratively to optimize policies.

Factored definition of POMDPs. POMDP problems can have millions of states. It would be computationally intractable to define such problems in a "flat" format with transitions as $|S| \times |S|$ probability tables for each action. Luckily, many POMDP problems can be defined in factored form [8]. In a factored POMDP the state s is described as a combination of several variables s_i; an observation o is described by several variables o_i. There can be millions of states s, and a large number of observations o, but the POMDP can be defined in terms of the individual s_i and o_i, which have only few elements each. The transition probabilities of each s_i and observation probabilities of each o_i depend only on a subset of the state variables: Parents(s_i) denotes the set of state variables affecting transitions of s_i, and Parents(o_i) is the set of state variables affecting the observation probabilities of o_i. The transition probabilities are then written as $P(s'|s,a) = \prod_i P(s'_i|\text{Parents}(s'_i),a)$, and observation probabilities are $P(o|\text{Parents}(o),a) = \prod_i P(o_i|\text{Parents}(o_i,a))$. The reward functions are defined as functions over subsets S_i of state variables: $R(s,a) = \sum_i R_i(S_i,a)$, where R_i is a reward function operating on subset S_i.

2.1 POMDP Methods

Several methods exist for planning in POMDPs. In principle, *exact optimal planning* in POMDPs can be done using incremental algorithms based on linear programming [9] to cover all possible beliefs. However, such algorithms can handle only problems with few states. *Point based value iteration* algorithms do not compute a solution for all beliefs, but either sample a set of beliefs (as in Perseus [3]) before the main algorithm or select beliefs during planning (as in HSVI [4]). 'Value iteration' refers to updating the value for a belief using a form of the Bellman equation (2). Point based value iteration algorithms scale to problems with thousands of states [4], which can still be insufficient. To cope with large state spaces, *POMDP compression methods* [10, 11] reduce the size of the state space. A linear static compression is computed and used to compress the transition and observation probabilities and the reward function. These compressions can then be fed to modified versions of POMDP algorithms like Perseus, to compute policies; this can give good performance [5] despite the (lossy) compression. In all these methods the value function is stored explicitly in vector form or in algebraic decision diagram (ADD) form; even for factored problems, its storage size grows exponentially with the number of state variables.

Some algorithms [5] store the policy as a finite state controller (FSC), a graph with actions as nodes and observations as edges. Any two nodes can be connected. 'Policy iteration' can improve the FSC by repeated policy improvement

and evaluation. Even if the FSC can be kept compact, the computed value functions have the same size as in the value iteration algorithms above. We maintain the value function as a policy graph using value iteration one layer at a time; nodes in each layer are connected only to the next layer.

Some methods *approximate belief updates*. In [12] it is shown that approximation errors in belief updates are bounded; a bound is given when minimizing Kullback-Leibler divergence between approximated and true belief. In [13] theoretical bound analysis of POMDP planning with approximated beliefs is done. Dynamic Bayesian network inference is performed in [14] using approximations similar to two of our approximations in Section 3.1.

The online POMDP method in [15] uses a factorized belief similar to ours; their planning involves searching different action-observation paths using a pruning heuristic; the worst case complexity is exponential in the search depth.

We lastly note that we use a policy graph based approach; a similar approach has been discussed in [16] for optimizing POMDP policy for unknown stationary transition probabilities but the setting is very different and approximation is not considered. We next describe our approach.

3 Our Algorithm: Factorized Belief Value Projection

We present our novel POMDP planning algorithm called Factorized Belief Value Projection (FBVP). Similarly to Perseus [3], FBVP starts by sampling a set of beliefs B before the actual planning. The main planning algorithm (Algorithm 1) takes the belief set B as input and produces a *policy graph* that can be used for decision making during online operation.

On a high level, Algorithm 1 works as follows. Initially, a simple policy α_0 is created, and all beliefs in B are associated with it. Then, at each iteration, beliefs are picked in random order, and for each belief b, a new policy α is optimized that looks one step further than previously; this optimization is called the *backup* operation. The belief b is associated with this new policy α. The policy α may have been optimal also for other beliefs earlier during this iteration; if not (α is new at this iteration), we check if any beliefs that haven't been processed yet during this iteration could get improved value from policy α. If they do get improved value, we simply associate those beliefs with α instead of separately picking them for optimization during this iteration. A speedup is to run *backup* for a small set of beliefs concurrently (randomly chosen set with heuristically chosen size) and check if any returned policy improves the values of the unprocessed beliefs; in the following description we omit this speedup for clarity. When all beliefs have been associated to some improved policy, the iteration ends and the new policies found during the iteration become the nodes of a new layer in the policy graph. The main algorithm is similar to the main algorithm of Perseus [3].

The key operations are the *backup* operation and evaluation (called *eval*) of the value of a policy for a belief. To perform them tractably, we exploit the policy graph from previous iterations, and use approximations; we describe the equations and approximations in Section 3.1 after the algorithms.

Algorithm 1. The main algorithm of our planning method FBVP

Input: $P(s_i'|Parents(s_i'), a)$, $P(o_i|Parents(o_i), a)$, $R(S_i, a)$, V_0, B
Output: policy graph G
Initialize $n = 0$, $\alpha_0(b) = V_0$, $G_0 = \alpha_0$
repeat
  Initialize $\tilde{B} = B$ and $H = \emptyset$.
  repeat
    Sample belief b from $\tilde{B}$ and compute $\alpha = \textbf{backup}(b, G, P, R)$.
    if $\alpha \notin H$ **then**
      $H = (H, \alpha)$
      if $\alpha \notin G$ **then**
        for $b \in \tilde{B}$ **do**
          Compute $V_{n+1}(b) = \textbf{eval}(\alpha, b, G, P, R)$
        end for
      end if
      $\tilde{B} = (b \in \tilde{B} : V_{n+1}(b) < V_n(b))$
    end if
  until $\tilde{B}$ is $\emptyset$
  $n = n + 1$
  $G_n = H$
until convergence

The *eval* algorithm (Algorithm 2) evaluates the value of a belief at a graph node α, as follows: we proceed through the graph from the newest layer where α is to the oldest. At each layer, we process all nodes: for each node α_k, we know which younger nodes link to it (we call them "caller nodes"), and which observation is associated to each such link. At each caller node α_c we have previously computed a projected belief $b_c^{a_c}$ with values $b_c^{a_c}(s') = b_c'(s'|b_c, a_c)$; here b_c is the belief projected from the starting node α all the way to α_c, and a_c is the action given by node α_c. The observations when leaving α_c are taken into account next: at each caller node α_c we have previously computed the *path probability*, p_c, of arriving at that node. The belief at α_k is a sum of caller beliefs, weighted by their path probabilities and the probabilities $p(\alpha_c \rightarrow \alpha_k)$ of continuing their paths to α_k. We have $p(\alpha_c \rightarrow \alpha_k) = p(o_{c,k}|a_c, b_c)$ where $o_{c,k}$ is the observation in the link $(\alpha_c \rightarrow \alpha_k)$. Similarly, the path probability p_k at α_k is a sum of incoming path probabilities; p_k is used to normalize the beliefs at α_k. As a computational trick, we multiply the path probability by the discount factor. The newest layer is a special case since it has no caller nodes: there, we just set the belief at the starting node and set the path probability to one for the starting node and zero for others. Having the belief at α_k, we compute expected direct reward over the belief, given the action a_k chosen by α_k. We multiply the expected reward by the path probability at α_k, and add it to total reward for the original node α; since we incorporated the discount factor into path probabilities, the added rewards get properly discounted. Lastly, we compute the distribution $b_k^{a_k}$ with values $b_k^{a_k}(s') = b_k'(s'|b_k, a_k)$. Now the node α_k can be used as a "caller node" for further layers. We process all nodes in the layer, then proceed to the next layer, and so on through the graph.

Algorithm 2. Evaluate belief value at an α node: **eval**$(\alpha_{n,j}, b, G, P, R)$

Input: node $\alpha_{n,j}$ (where n is the index of the newest graph layer and j is the index of the node within the layer), belief b at the node, graph G, probability tables P, reward tables R
Output: value V of using $\alpha_{n,j}$ as policy for belief b
At the newest layer initialize beliefs : $b_{n,j} = b$, compute $b^a_{n,j}$ (Equation 3) using action $a_{n,j}$, and initialize path probabilities: $p_{n,j} = 1$ and $p_{n,i} = 0$ for all $i \neq j$.
Compute immediate reward at start: $V = R(b_{n,j}, a_{n,j})$ (by Equation 7).
for layers $i = n - 1$ to 1 **do**
  for each node k in layer i **do**
    For this node $\alpha_{i,k}$ (here denoted α_k for short) do the following:
    1. For each caller node α_c linked to α_k from the previous layer, compute $b_c^{a_c,o_{c,k}} p_c p(\alpha_c \to \alpha_k)$, where $p(\alpha_c \to \alpha_k) = p(o_{c,k}|a_c, b_c)$, $o_{c,k}$ is the observation associated with the link $(\alpha_c \to \alpha_k)$, a_c and b_c are the action and belief at α_c, and $b^{a_c,o_{c,k}}$ is the belief conditioned on a_c and $o_{c,k}$ using Equation 4.
    2. The path probability $p_{i,k}$ at this node is a weighted sum over incoming path probabilities from caller nodes: $p_{i,k} = \sum_c p_c p(o_{c,k}|a_c, b_c)$.
    3. The belief at this node α_k is a weighted sum of incoming beliefs from caller nodes: $b_{i,k} = \sum_c b_c^{a_c,o_{c,k}} p_c p(o_{c,k}|a_c, b_c)/p_{i,k}$. Approximate the sum by a single factorized belief by minimizing KL-divergence (Equation 6).
    4. Calculate expected immediate reward $R_{i,k}$ for the belief $b_{i,k}$ and the action $a_{i,k}$ at this node (Equation 7).
    5. Add the path-weighted reward $R_{i,k} p_{i,k}$ to the total value V.
    6. Project the belief $b_{i,k}$ using the action $a_{i,k}$ but not conditioning on observations, to obtain $b^a_{i,k}$ with values $b^a_{i,k} = b'_{i,k}(s'|b_{i,k}, a_{i,k})$ (Equation 3).
    7. Multiply the path probability $p_{i,k}$ by the discount factor γ.
  end for
end for

Figure 1 illustrates *eval*, when computing value of the node α (marked with green color and bold outline) for a given belief. The algorithm is currently processing node α_k (marked with yellow color and an asterisk) in Layer 2. Nodes α_c in Layer 3 are caller nodes for α_k. Bold red arrows show where the belief at α_k originates. The a indicate actions chosen in each state (many indices omitted for clarity) and o are observations attached to each link.

The *backup* (Algorithm 3) finds an improved policy for belief b as follows. The candidate improved value V is first $-\infty$. We plan to create a new node into the graph, and we must decide which action to choose there, and which further nodes it should link to for each observation. A posterior belief b' with values $b'(s'|a, o)$ is computed for all actions a and observations o. For each action, we go through all observations, and for each observation we try nodes from the next graph layer, as candidates the new node could link to. We use *eval* to get the value given by each candidate for the belief b' we computed above. We choose the candidate with the highest value as the link for this action-observation pair. Having processed all observations for action a, we compute the value given by a for the original belief b: it is the sum of immediate rewards for this belief-action pair, and values given by the links chosen above for observation of this

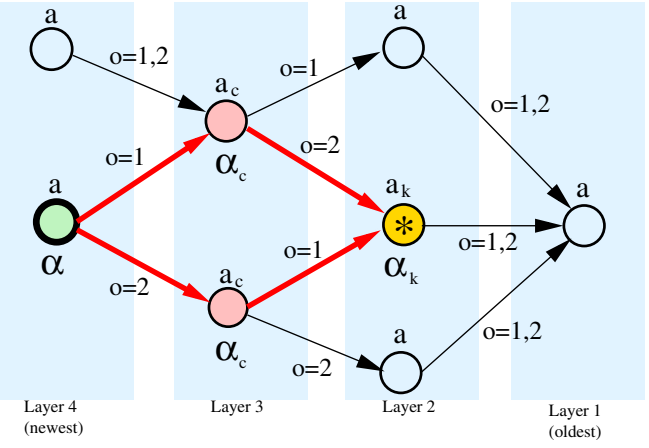

Fig. 1. Illustration of the *eval* algorithm, which proceeds through a fixed policy graph from the left to the right, evaluating the reward acquired by starting from a certain node. The graph that *eval* operates on is constructed by Algorithm 1, adding one layer to the left end in each iteration and creating nodes into it using the *backup* algorithm.

a, which are weighted by observation probabilities and by the discount factor. If this value is the best found so far, action a (along with the associated links for each observation from a) replaces the previous best action.

3.1 Equations and Approximations

We keep the belief values always in a fully factored form $b(s) = \prod_i b(s_i)$. We maintain an exact lower bound for the value function in the form of the policy graph. However, when using the graph for belief value evaluation, there are three operations that can potentially break the fully factored form of beliefs. For each of these operations we find a fully factorized approximation (sometimes called a mean-field approximation).

We approximate the transition from the current belief to the next without observation conditioning with Equation 3. Let state variable s_i have K parents out of all N state variables; denote them $Parents(s'_i) = s_1, \ldots, s_K$. It is easy to show that when we want to approximate the posterior distribution of several state variables by a fully factored form, the best approximation minimizing KL divergence from the real posterior is simply the product of marginals of the real posterior; for our model each marginal depends on the parents of the corresponding variable. The approximation is

$$b^a(s'_i) = b'(s'_i|b,a) = \sum_{s_1} b(s_1) \cdots \sum_{s_K} b(s_K) P(s'_i|s_1, \ldots, s_K, a) \,. \tag{3}$$

This approximation minimizes KL divergence from the real posterior belief $b^a(s')$ to the factorized approximation $\prod_i b^a(s'_i)$, where $b^a(s') = b'(s'|b,a) =$

Algorithm 3. Backup operation $\alpha = \mathbf{backup}(b, G, P, R)$

Input: belief b for which an improved policy is wanted, policy graph G, Markov model P, reward function R
Output: policy graph node α corresponding to an improved policy for b
Initialize the value of the improved policy to $V_{max} = -\infty$.
Compute posterior belief $b^{a,o}$ (having values $b(s'|a,o,b)$) for all actions a and observations o
for all a **do**
  for all o **do**
    for candidate link target nodes $\tilde{\alpha} \in G_{n-1}$ **do**
      compute value $V(b^{a,o}, \tilde{\alpha}) = \mathbf{eval}(\tilde{\alpha}, b^{a,o}, G_{n-1,\ldots,1}, P, R)$
    end for
    choose the link target as the best candidate, $\hat{\alpha}^{a,o} = \arg\max_{\tilde{\alpha}} V(b^{a,o}, \tilde{\alpha})$
  end for
  Compute value of this action a, $V = R(b) + \gamma \sum_o P(o|b,a) V(b^{a,o}, \hat{\alpha}^{a,o})$
  if $V > V_{max}$ **then**
    This action becomes the best candidate so far, $V_{max} = V$
    Set the new node to use this action and its associated link targets, $\alpha = (a, (\hat{\alpha}^{a,o} : \forall o \in O))$
  end if
end for

$\sum_{s_1,\ldots,s_N} (\prod_i P(s'_i|\text{Parents}(s'_i), a))\, b(s_1) \ldots b(s_N)$. We next approximate the conditioning of the above approximated belief on an observation, with Equation 4. Let observation o have L parents out of N state variables, call them $Parents(o) = s'_1, \ldots, s'_L$. We minimize KL divergence to a factored approximation $\prod_i b^{a,o}(s'_i)$; we set the gradient of the divergence with respect to each factor (distribution) to zero, which yields the distribution

$$b^{a,o}(s'_i) = \frac{1}{p(o|b,a)} \sum_{s'_1} b^a(s'_1) \cdots \sum_{s'_{i-1}} b^a(s'_{i-1}) \sum_{s'_{i+1}} b^a(s'_{i+1}) \cdots \sum_{s'_L} b^a(s'_L) P(o|s'_1, \ldots, s'_L, a) \;, \tag{4}$$

where $p(o|b,a) = \sum_{s'_1} b^a(s'_1) \cdots \sum_{s'_L} b^a(s'_L) P(o|s'_1, \ldots, s'_L, a)$. The approximations in Equations 3 and 4 are used in [14] for inference in Bayesian networks.

We also update the path probability (used in Algorithm 2) for each path arriving at a node α_k from a node α_c, associated with a belief b_c, an action a_c and an observation $o_{c,k}$:

$$p(c, c \to k) = p_c p(o_{c,k}|a_c, b_c) \tag{5}$$

When multiple beliefs "arrive" at a node α_k from previous nodes α_c (see Algorithm 2 for details), the belief at α_k is a sum $\sum_c (p(c, c \to k)/p_k) \prod_{i=1}^N b_c^{a_c, o_{c,k}}(s_i)$ where p_k is the path probability at α_k; this belief is approximated as a factored

form $\prod_{i=1}^{N} b_k(s_i)$ using Equation 6; minimizing KL divergence (by again setting the gradient with respect to each discrete distribution to zero) we have

$$b_k(s_i) = \sum_c (p(c, c \to k)/p_k) b_c^{a_c, o_{c,k}}(s_i) \ , \tag{6}$$

where $p_k = \sum_c p(c, c \to k)$.

The reward at the α nodes can be calculated exactly using Equation 7:

$$R(b, a) = \sum_{s_1} b(s_1) \cdots \sum_{s_N} b(s_N) R(s_1, \ldots, s_N, a) \ . \tag{7}$$

Note that the approximation error in FBVP can reduce further into the policy graph, with a rate depending on several factors; see [12] for analysis. The value function is a convex piecewise linear function [1] corresponding to the policy graph; beliefs that share the same optimal linear function, i.e. same optimal policy graph node, are more likely to be "near" each other in the belief simplex. Thus the error in the approximation in Equation 6 is usually small, because beliefs in the approximated sum are usually similar.

3.2 Pruning

In the worst case, in iteration t each belief is evaluated $\mathcal{O}(|A||O||G_{t-1}|)$ times, where $|G_{t-1}|$ is the number of policy graph nodes in layer $t-1$; for each belief, the maximum number of calls to Equations 3, 6, and 7 is $\mathcal{O}(\sum_{i=1}^{t-1} |G_i|)$ and $\mathcal{O}(|O| \sum_{i=1}^{t-1} |G_i|)$ for Equation 4. The algorithm has polynomial complexity with respect to the number of state variables and to the horizon (maximum plan-ahead depth) and scales well to large state spaces, but evaluating the whole belief tree yields significant computational overhead. When the number of α nodes is large, the *backup* algorithm (see Algorithm 3) dominates. To eliminate a part of policy graph evaluation we compute an approximate upper bound for the value at each graph node. This bound is used to compute maximum values during policy graph evaluation. Evaluation can be stopped if the value accumulated so far, added to the maximum value possible, is smaller than the best found solution's value.

The requirements for a *policy graph node upper bound* are: it should not underestimate the value of a node, should be tight, fast to evaluate, and to a lesser extent fast to compute.

Each of our policy graph nodes would, in a traditional approach [3], correspond to a vector, whose dot product with a belief, would yield the value for the belief. Because of the state space size, we use a sum of linear functions of individual state variables as an approximation. Then $B\boldsymbol{v} = V$, where B is the matrix of beliefs in factored form for which we have computed values V and $\boldsymbol{v}$ is a vector of concatenated linear state variable functions. Each row of the matrix B has the probabilities of the single state variables concatenated. The approximation is not exponential in the number of state variables and is tractable. To guarantee $\boldsymbol{v}$ does not underestimate the value, all extreme points of the beliefs would have to be added to B. But as the number of extreme points equals the size of the

state space, we calculate additional values for a randomly selected set of beliefs, with individual belief component values set to unity.

In order to find a tight bound we can reformulate the problem as $\boldsymbol{v}^T B^T B\boldsymbol{v} - V^T B^T \boldsymbol{v} = \epsilon$, $B\boldsymbol{v} \geq V$ and find the $\boldsymbol{v}$ that minimizes ϵ using quadratic programming or fit $\boldsymbol{v}$ using least squares. The quadratic programming approach is possible in many problems, because the individual state variables can be very small. For example in the computer network problem (see Section 4 for more details) there are 16 computers each having a binary state variable.

We either compute a fit with least squares or a bound with quadratic programming (with a time limit), and then use cross-validation to estimate the maximum error of the bound and add that to the bound to ensure optimal actions for the current belief set are not pruned. The procedure does not guarantee an exact bound for future belief sets but performed well in the experiments; we used 5-fold cross-validation and quadratic programming to compute the bounds.

A further optimization can be done by *observation ordering*. The value for an action is the sum of the immediate reward for the belief and the values for the beliefs projected for each observation. When we have accumulated the sum of the immediate reward and the rewards for part of the observations, the remaining observations must produce at least a "minimum" value that is greater than the best found action value so far, minus the accumulated value, in order for the current action to exceed previous actions. As we can compute maximum values for policy graph nodes, we can use the probability of the observation under consideration, the required "minimum" value, and the maximum values for the remaining observations to compute the smallest possible value for an observation that can make the current action exceed previous actions. If the observation does not reach this smallest possible value during value evaluation, the action under consideration can be pruned. By ordering projected beliefs according to their observation probability (see line 6 of Algorithm 3) this early stopping can be made more likely and the effectiveness of the upper bound pruning increased.

4 Experiments

We compare our method against four others on several benchmark problems. We next describe the comparison methods, benchmark problems, and results.

4.1 Comparison Methods

We use the following POMDP algorithms as comparison methods:

1. Perseus[1] iteratively improves a value function lower bound processing a fixed set of belief samples in random order. The proposed method FBVP resembles Perseus in that it also samples a fixed set of beliefs and then improves the value lower bound for each belief.

[1] Code available at `http://staff.science.uva.nl/~mtjspaan/software/approx/`

2. HSVI2[2] maintains both an upper and lower bound on the value function. In contrast to Perseus it generates beliefs during training and applies a depth-first type of search, while Perseus uses a breadth-first type of search.
3. Symbolic Perseus[3] is a version of Perseus that uses abstract decision diagrams (ADDs) for representing the POMDP data structures and is configured in factored form. It uses a mean-field approximation on the beliefs in Bellman backups and cuts of non-significant ADD leafs. Symbolic Perseus has been applied on POMDP problems with millions of states [17].
4. In truncated Krylov iteration [5] the POMDP problem is compressed into a linear approximation with smaller dimension than the original problem and a policy is found by using a (slightly modified) standard POMDP solver with the compressed problem. We use Perseus [3] as the POMDP solver. For factored problems the problem structure can potentially be exploited to perform Krylov iteration efficiently. We have a generic truncated Krylov iteration implementation using ADD data structures as suggested in [5] to provide a fair comparison. In our implementation the basis vectors are stored as sums of products of state variable functions. With this implementation we were able to compute compressions even for the largest problems.

4.2 POMDP Problems

We use two traditional benchmarks: *RockSample* [4] (a rover moves on a grid and tries to sample rocks that are "good"; e.g. RS(5,15) denotes a 5×5 field with 15 rocks) and *Computer Network* (computers are up or down and administrators can ping or reboot them; included with Symbolic Perseus software; e.g. CN(16) denotes 16 computers). In RockSample, the rover's position is deterministic; we introduce a new *Uncertain RockSample* problem, where the rover stays at the same location (except when moving to a terminal state) with 0.05 probability, when it should move. The uncertain location makes the problem harder. E.g. URS(5,15) again denotes a 5×5 field with 15 rocks.

We also present a new benchmark problem for POMDPs which we call *Spectrum Access* (SA).[4] It is motivated by real-life needs of growing wireless communication: the number of devices communicating over wireless connections is growing, and ever more data is transmitted due to e.g. increasing video communication. To avoid congestion over the limited amount of available spectrum, it is important to allocate resources over the spectrum efficiently; here a cognitive radio [18] must predict when a radio channel will be free of traffic, and use such "time slots" for communication but avoid access conflicts with existing (primary) users of the channels. Each channel evolves according to a 15-state Markov model estimated from simulated data; it describes packet burst lengths, pauses etc. The cognitive radio device can only sense three channels at a time and transmit on one at each step. Observations tell if a channel is busy/idle but

[2] Software available from `http://www.cs.cmu.edu/~trey/zmdp/`
[3] Software available from `http://www.cs.uwaterloo.ca/~ppoupart/software.html`
[4] See SA and URS specifications at `www.cis.hut.fi/jpajarin/pomdp/problems/`

not the exact channel state (burst type etc.). Rewards are given for successful transmissions penalties for using energy for listening (−0.01 per channel) and strong penalties for access conflicts. E.g. SA(4) denotes four radio channels.

5 Results and Discussion

Table 1 shows discounted expected rewards for all the problems and algorithms tested. The discount factor was 0.95. In the classic RockSample (RS) problem, because of time constaints, algorithms were run for two days and in the other problems for three days. Part of the algorithms converged on some of the problems before maximum running time. The algorithms were initialized using their default initialization and FBVP was initialized with zero valued initial values for all problems. In spectrum access 3000 beliefs were used and 10000 in the other problems for Perseus, FBVP, and Symbolic Perseus. If an algorithm ran out of memory, then the intermediate policy output (if any) was used for evaluation. Evaluation was run for 500 runs of 60 steps. The methods were evaluated using their default evaluation method. Symbolic Perseus and FBVP were evaluated by following their policy graphs after belief evaluation, and Perseus, HSVI, and truncated Krylov iteration with Perseus were evaluated by using the computed value vectors.

Each maximum time experiment was run on one core of a "AMD Opteron 2435" processor with 8GB allocated memory. In only few cases such as Symbolic Perseus in the 16-machine computer network problem, all 8GB was needed.

Perseus and HSVI performed best in the smallest RS problem, but FBVP was very close to them. In the other RS problems Perseus and HSVI could not be configured. Truncated Krylov iteration together with Perseus did not perform well in any of the RS problems. In the 15-rock RS problem Symbolic Perseus achieved best results. Symbolic Perseus seems to be able to exploit the deterministic location of the rover using its ADD data structures. In the largest RS problems only FBVP had good results.

In the Computer Network (CN) problems we were not able to reproduce the results of Poupart et al. reported in [5] with our truncated Krylov iteration with Perseus implementation. This can be due to truncated Krylov iteration selecting basis vectors using an Euclidean metric. At what point the exact L1-normalization of basis vectors suggested in [5] is done in the truncated Krylov iteration algorithm may change the order of basis vectors added to the compression matrix. Also, even if the required adding of a constant to rewards does not change the optimal policy for the objective function, it changes the Euclidean distance between vectors. We used the results from [5] as an additional comparison and ran additional evaluation for Symbolic Perseus and FBVP for the reported running times. Note that the running times are not directly comparable. For the full training time Symbolic Perseus gave a policy that was very slow to evaluate and thus was limited to 276 evaluation runs.

In the CN problems Perseus and HSVI could not be configured due to size of the POMDP configurations. For the 16-machine problem FBVP and Symbolic Perseus gave better results than truncated Krylov iteration with Perseus. For the shorter training times Symbolic Perseus was better than FBVP and for the

Table 1. Performance of selected POMDP algorithms

Problem (states /actions/obs.)	Reward	Time
RS(5,5) (801s 10a 2o)		
Perseus	19.05	2days
HSVI2	19.05	converged
SymbolicPerseus	17.78	2days
Tr.Kry.+Perseus	7.02	2days
FBVP	18.67	2days
RS(5,15) (819201s 20a 2o)		
Perseus	-	config. fail
HSVI2	-	config. fail
SymbolicPerseus	31.66	2days
Tr.Kry.+Perseus	−14.81	2days
FBVP	24.29	2days
RS(5,20) (26214401s 25a 2o)		
Perseus	-	config. fail
HSVI2	-	config. fail
SymbolicPerseus	-	out of mem
Tr.Kry.+Perseus.	−20.70	2days
FBVP	23.89	2days
RS(6,25) ($\sim 1.2G$s 30a 2o)		
Perseus	-	config. fail
HSVI2	-	config. fail
SymbolicPerseus	-	out of mem
Tr.Kry.+Perseus	0.00	2days
FBVP	18.05	2days
CN(16) (2^{16}s 33a 2o)		
Perseus	-	config. fail
HSVI2	-	config. fail
SymbolicPerseus	107.98	12658sec
SymbolicPerseus	108.14	3days
Tr.Kry.+Perseus	103.6	[Poupart05]
Tr.Kry.+Perseus	88.84	3days
FBVP	105.97	12658sec
FBVP	109.05	3days
CN(25) (2^{25}s 51a 2o)		
Perseus	-	config. fail
HSVI2	-	config. fail
SymbolicPerseus	-	out of mem
Tr.Kry.+Perseus	148	[Poupart05]
Tr.Kry.+Perseus	136.69	3days
FBVP	152.01	8574sec
FBVP	154.66	3days

Problem (states /actions/obs.)	Reward	Time
URS(5,5) (801s 10a 2o)		
Perseus	16.25	3days
HSVI2	16.43	out of mem
SymbolicPerseus	15.47	3days
Tr.Kry.+Perseus	16.73	3days
FBVP	15.64	3days
URS(5,15) (819201s 20a 2o)		
Perseus	-	config. fail
HSVI2	-	config. fail
SymbolicPerseus	13.21	3days
Tr.Kry.+Perseus	−8.37	3days
FBVP	21.15	3days
URS(5,20) (26214401s 25a 2o)		
Perseus	-	config. fail
HSVI2	-	config. fail
SymbolicPerseus	-	out of mem
Tr.Kry.+Perseus	2.25	3days
FBVP	18.55	3days
URS(6,25) ($\sim 1.2G$s 30a 2o)		
Perseus	-	config. fail
HSVI2	-	config. fail
SymbolicPerseus	-	out of mem
Tr.Kry.+Perseus	0.00	3days
FBVP	19.22	3days
SA(4) (15^4s 9a 8o)		
Perseus	-	out of mem
HSVI2	13.71	out of mem
SymbolicPerseus	14.19	3days
Tr.Kry.+Perseus	13.14	3days
FBVP	14.52	3days
SA(8) (15^8s 25a 8o)		
Perseus	-	config. fail
HSVI2	-	config. fail
SymbolicPerseus	-	out of mem
Tr.Kry.+Perseus	−43.07	3days
FBVP	13.86	3days

longer training times at the same level. In the 25-machine problem Symbolic Perseus ran out of memory and FBVP performed best.

In the smallest Uncertain RockSample (URS) problem Perseus and HSVI could be configured and trained successfully. All five methods got discounted rewards that were roughly at the same level. The second smallest URS problem had hundreds of thousands of states and FBVP performed best. Most likely the uncertain location of the rover makes the abstract decision diagram presentation in Symbolic Perseus less efficient than in the original RS problem of same size. For the two largest URS problems only FBVP had acceptable results. Symbolic Perseus ran out of memory in these problems. Truncated Krylov iteration with Perseus did not produce good results.

In the four channel Spectrum Access (SA) problem Perseus ran out of memory before actual training, but HSVI could be configured and trained for a while before memory ran out. FBVP and Symbolic Perseus had the best results. Truncated Krylov iteration with Perseus and HSVI had results that were not bad for such a big problem. In this problem truncated Krylov iteration was able to produce a compression close to a lossless compression in 1000 dimensions, when the original state space had 50625 dimensions. Only FBVP got good performance in the eight channel SA problem. Perseus, HSVI, and Symbolic Perseus did not yield any policy, and the reward obtained by truncated Krylov iteration with Perseus was much lower than that of FBVP.

6 Conclusions

We have presented a novel efficient POMDP algorithm policy graph based computation with factorized approximations and bounding. The approximations and policy graph approach ensure polynomial complexity with respect to number of state variables and look ahead depth. The results show that our algorithm, called Factorized Belief Value Projection (FBVP), scales well to very large problems and produces adequate rewards for smaller problems compared to algorithms that do not employ similar approximations. FBVP does not require a domain expert for specific tasks such as grouping state variables.

In the future it may be interesting to extend FBVP to problems where using the policy graph as the only value function representation can be inherently advantageous such as for POMDPs with unknown transition probabilities [16]. The effect of pruning and observation ordering also needs further study.

Acknowledgments. JoP belongs to AIRC; JaP to AIRC and HIIT. The work was supported by TEKES, PASCAL2, and Academy of Finland decision 123983.

References

1. Sondik, E.J.: The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. Operations Research 26(2), 282–304 (1978)
2. Cassandra, A.R.: A Survey of POMDP Applications. Technical report, Austin, USA (1998) Presented at the AAAI Fall Symposium (1998)

3. Spaan, M., Vlassis, N.: Perseus: Randomized point-based value iteration for POMDPs. Journal of Artificial Intelligence Research 24, 195–220 (2005)
4. Smith, T., Simmons, R.: Point-Based POMDP Algorithms: Improved Analysis and Implementation. In: Twenty-First Annual Conf. on Uncertainty in Artif. Int., Arlington, Virginia, pp. 542–549. AUAI Press (2005)
5. Poupart, P.: Exploiting structure to efficiently solve large scale partially observable Markov decision processes. PhD thesis, Univ. of Toronto, Toronto, Canada (2005)
6. Pajarinen, J., Peltonen, J., Uusitalo, M.A., Hottinen, A.: Latent state models of primary user behavior for opportunistic spectrum access. In: 20th Intl. Symposium on Personal, Indoor and Mobile Radio Communications, pp. 1267–1271. IEEE, Los Alamitos (2009)
7. Zhao, Q., Tong, L., Swami, A., Chen, Y.: Decentralized cognitive MAC for opportunistic spectrum access in ad hoc networks: A POMDP framework. IEEE J. Sel. Areas Commun. 25(3), 589–600 (2007)
8. Boutilier, C., Poole, D.: Computing optimal policies for partially observable decision processes using compact representations. In: Thirteenth National Conf. on Artif. Int., pp. 1168–1175. The AAAI Press, Menlo Park (1996)
9. Cassandra, A., Littman, M., Zhang, N.: Incremental pruning: a simple, fast, exact method for partially observable Markov decision processes. In: 13th Annual Conf. on Uncertainty in Artif. Int., pp. 54–61. Morgan Kaufmann, San Francisco (1997)
10. Poupart, P., Boutilier, C.: Value-directed compression of POMDPs. In: Becker, S., Obermayer, K. (eds.) Advances in Neural Information Processing Systems, vol. 15, pp. 1547–1554. MIT Press, Cambridge (2003)
11. Li, X., Cheung, W., Liu, J., Wu, Z.: A novel orthogonal NMF-based belief compression for POMDPs. In: Ghahramani, Z. (ed.) 24th Annual International Conference on Machine Learning, pp. 537–544. Omnipress (2007)
12. Boyen, X., Koller, D.: Tractable inference for complex stochastic processes. In: Fourteenth Annual Conf. on Uncertainty in Artif. Int., pp. 33–42. Morgan Kaufmann, San Francisco (1998)
13. McAllester, D., Singh, S.: Approximate planning for factored POMDPs using belief state simplification. In: Fifteenth Annual Conf. on Uncertainty in Artif. Int., pp. 409–417. Morgan Kaufmann, San Francisco (1999)
14. Murphy, K., Weiss, Y.: The factored frontier algorithm for approximate inference in DBNs. In: Seventeenth Annual Conf. on Uncertainty in Artif. Int., pp. 378–385. Morgan Kaufmann, San Francisco (2001)
15. Paquet, S., Tobin, L., Chaib-draa, B.: An online POMDP algorithm for complex multiagent environments. In: Fourth International Joint Conference on Autonomous Agents and Multiagent systems, pp. 970–977. ACM, New York (2005)
16. Poupart, P., Vlassis, N.: Model-based Bayesian reinforcement learning in partially observable domains. In: Tenth Intl. Symp. on Artif. Intelligence and Math. (2008)
17. Boger, J., Poupart, P., Hoey, J., Boutilier, C., Fernie, G., Mihailidis, A.: A decision-theoretic approach to task assistance for persons with dementia. In: Nineteenth Intl. Joint Conf. on Artif. Int., vol. 19, pp. 1293–1299 (2005)
18. Haykin, S.: Cognitive radio: brain-empowered wireless communications. IEEE J. Sel. Areas Commun. 23, 201–220 (2005)