Accelerated Vector Pruning for Optimal POMDP Solvers

Walraven, Erwin; Spaan, Matthijs

Publication date
2017

Document Version
Accepted author manuscript

Published in
Proceedings of the 31st Conference on Artificial Intelligence, AAAI 2017

Accelerated Vector Pruning for Optimal POMDP Solvers

Erwin Walraven and **Matthijs T. J. Spaan**
Delft University of Technology
Mekelweg 4, 2628 CD
Delft, The Netherlands

Abstract

Partially Observable Markov Decision Processes (POMDPs) are powerful models for planning under uncertainty in partially observable domains. However, computing optimal solutions for POMDPs is challenging because of the high computational requirements of POMDP solution algorithms. Several algorithms use a subroutine to prune dominated vectors in value functions, which requires a large number of linear programs (LPs) to be solved and it represents a large part of the total running time. In this paper we show how the LPs in POMDP pruning subroutines can be decomposed using a Benders decomposition. The resulting algorithm incrementally adds LP constraints and uses only a small fraction of the constraints. Our algorithm significantly improves the performance of existing pruning methods and the commonly used incremental pruning algorithm. Our new variant of incremental pruning is the fastest optimal pruning-based POMDP algorithm.

Introduction

Dealing with uncertainty and partial observability in planning problems is a challenging goal in the development of intelligent agents. Partially Observable Markov Decision Processes (POMDPs) have emerged as a successful framework for planning under uncertainty in partially observable domains (Kaelbling, Littman, and Cassandra 1998), and have been used in several real-world applications such as aircraft collision avoidance (Bai et al. 2012) and guidance of people with dementia (Boger et al. 2005). A significant body of research has focused on POMDPs in the past, but solving POMDPs to optimality remains difficult. Although several approximate methods for POMDPs exist (Pineau, Gordon, and Thrun 2003; Spaan and Vlassis 2005), optimal solutions are commonly used in recent literature (Karmokar, Senthuran, and Anpalagan 2012; Qian et al. 2016; Li and Jayaweera 2015; Blanco et al. 2015; Roijers, Whiteson, and Oliehoek 2013; Raphael and Shani 2012). Moreover, an advantage of optimal solutions is that they are independent of the initial belief.

Incremental pruning (Cassandra, Littman, and Zhang 1997) is a popular method for computing optimal POMDP solutions. It is based on a subroutine that removes dominated vectors from value functions, which is known as *pruning*. The subroutine solves a large number of LPs during its execution to check whether a vector is dominated by a set of vectors, which turns out to be a costly operation. For example, Cassandra, Littman, and Zhang (1997) have shown that linear programming represents a major part of the total running time. Existing research focusing on the scalability of the LP subroutine typically aims to solve fewer LPs and exploits the POMDP structure to create LPs with fewer constraints (Feng and Zilberstein 2004). However, existing work does not try to exploit the structure of the LPs to derive more efficient algorithms to solve the LPs. We demonstrate that such a structure can be used to derive faster algorithms.

In this paper we show that a more efficient vector pruning method can be obtained by selecting LP constraints in a smart way. We take the original LP and apply a Benders decomposition (Benders 1962) to derive an algorithm which incrementally adds constraints. LP solvers do not automatically apply such a decomposition, and therefore we show how it can be implemented manually in the context of pruning for POMDPs. The resulting algorithm only needs a small fraction of the constraints in the original LP. We show that the algorithm always finds the optimal LP solution, and we prove that some constraints will never be added to the LP formulation. Experiments show that our algorithm improves the performance of existing pruning methods, and our results show that the accelerated pruning algorithm creates the fastest variant of incremental pruning for POMDPs.

Background

In this section we introduce Partially Observable Markov Decision Processes and decomposition of linear programs.

Partially Observable Markov Decision Processes

A POMDP (Kaelbling, Littman, and Cassandra 1998) consists of a set of states S, a set of actions A and a set of observations O. If action $a \in A$ is executed in state $s \in S$, then the state changes to $s' \in S$ according to the probability distribution $P(s'|s,a)$ and a reward $R(s,a)$ is received. Rather than observing state s' directly, the agent receives an observation $o \in O$ according to probability distribution $P(o|a,s')$. The agent aims to maximize the expected discounted reward $E\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]$, where $0 \leq \gamma < 1$ is the discount rate and R_t is the reward at time t. The agent maintains a belief b over states, which can be updated using Bayes' rule.

input : vector set W
output : pruned set D

$D \leftarrow \emptyset$
while $W \neq \emptyset$ **do**
  $w \leftarrow$ arbitrary element in W
  if $w(s) \leq u(s), \exists u \in D, \forall s \in S$ **then**
    $W \leftarrow W \setminus \{w\}$
  else
    $b \leftarrow$ `FindBeliefStd`(D, w)
    if $b = \phi$ **then**
      $W \leftarrow W \setminus \{w\}$
    else
      $w \leftarrow$ `BestVector`(b, W)
      $D \leftarrow D \cup \{w\}, W \leftarrow W \setminus \{w\}$
    end
  end
end
return D

Algorithm 1: Vector pruning (White & Lark)

An agent uses a policy $\pi : \Delta(S) \rightarrow A$ to make decisions, where $\Delta(S)$ denotes the continuous set of probability distributions over S. A policy π can be defined using a value function $V^\pi : \Delta(S) \rightarrow \mathbb{R}$. The value $V^\pi(b)$ denotes the expected discounted reward when following policy π starting from b and is defined as:

$$E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(b_t, \pi(b_t)) \,\middle|\, b_0 = b \right], \tag{1}$$

where $R(b_t, \pi(b_t)) = \sum_{s \in S} R(s, \pi(b_t)) b_t(s)$ and belief b_t is the belief at time t. The optimal value function $V^*(b) = \max_\pi V^\pi(b)$ is the best value function that can be achieved. A maximizing policy $\pi^\star$ is an optimal policy.

Value functions are piecewise linear and convex in the finite-horizon setting, and can be defined using a set of vectors (Sondik 1971). If only immediate rewards are considered, then the optimal value function is $V_0(b) = \max_{a \in A} \sum_{s \in S} R(s, a) b(s) = \max_{a \in A} \; b \cdot \alpha_0^a$, where α_0^a is a vector $(R(1, a), \ldots, R(|S|, a))$ and $\cdot$ denotes the inner product. The value function V_n at stage n of value iteration is $V_n(b) = \max_{1 \leq k \leq |V_n|} b \cdot \alpha_n^k$, and V_{n+1} can be computed using the Bellman backup operator H:

$$HV_n = \bigcup_{a \in A} G_a, \;\; \text{with } G_a = \bigoplus\nolimits_{o \in O} G_a^o \;\text{ and} \tag{2}$$

$$G_a^o \;=\; \left\{ \frac{1}{|O|} \alpha_0^a + \gamma g_{ao}^k \,\middle|\, 1 \leq k \leq |V_n| \right\}.$$

Note that V_n denotes a set containing vectors and $V_n(b)$ denotes the value in belief b computed using the vectors in V_n. The operator $\oplus$ denotes the cross sum operator. For two sets Q and R the operator can be defined as $Q \oplus R = \{q + r \mid q \in Q, r \in R\}$. The vector g_{ao}^k can be obtained by back-projecting the vector α_n^k from value function V_n using action a and observation o using the equation $g_{ao}^k(s) = \sum_{s' \in S} P(o|a, s') P(s'|s, a) \alpha_n^k(s')$.

input : vector set U, vector w
output : belief state b or symbol ϕ

if $|U| = 0$ **then**
  return arbitrary belief b
end
max d
s.t. $(w - u) \cdot b \geq d \quad \forall u \in U$
  $\sum_{i=1}^{|S|} b_i = 1, \;\; b_i \geq 0 \;\; \forall i, \;\; d$ free
return b if $d > 0$ and ϕ otherwise

Algorithm 2: `FindBeliefStd` – computes the belief in which w improves U the most

When computing HV_n, it contains more vectors than necessary if there are vectors which are never the value-maximizing vector for a given belief b. A pruning subroutine `prune` can be executed after computing each cross sum. The resulting algorithm is known as incremental pruning (Cassandra, Littman, and Zhang 1997) and computes a Bellman backup as $HV_n = \texttt{prune}\left(\bigcup_{a \in A} G_a \right)$, where $G_a = \texttt{prune}\left(\texttt{prune}\left(\bar{G}_a^1 \oplus \bar{G}_a^2 \right) \ldots \oplus \bar{G}_a^{|O|} \right)$ and $\bar{G}_a^o = \texttt{prune}(G_a^o)$. The pruning operator can be implemented using a series of LPs. Algorithm 1 shows a pruning algorithm proposed by White and Lark (White 1991). The procedure `BestVector` returns the vector from W with the highest value in belief b (Littman 1996). The procedure `FindBeliefStd` uses an LP to find the belief in which the value function U improves the most when adding vector w. The procedure is shown in Algorithm 2.

Benders Decomposition

We use the Benders decomposition technique (Benders 1962), which can be applied to LPs of the following form:

$$\max \;\; px + hy \quad \text{s.t. } \; Cx + My \geq q \tag{3}$$

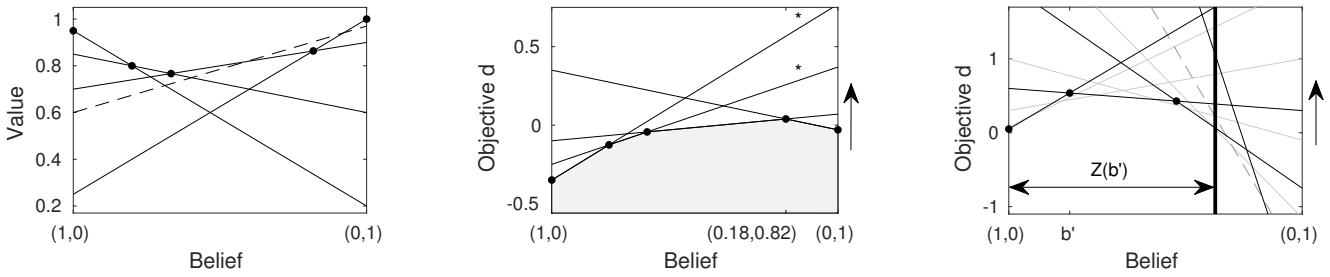
where p and h are row vectors containing coefficients and the column vectors x and y represent decision variables. The constraints are defined by the column vector q and matrices C and M, which contain constants. If the vector x is replaced by a vector $\bar{x}$ containing constants, then (3) reduces to:

$$\varphi(\bar{x}) = \max \; hy \quad \text{s.t. } \; My \geq q - C\bar{x}. \tag{4}$$

In the general case we can write (3) as $\max_x (px + \varphi(x))$. The dual of (4) can be written as:

$$\min \;\; (q - C\bar{x})^\top z \quad \text{s.t. } \; M^\top z = h^\top, \quad z \geq 0 \tag{5}$$

where z is a column vector containing the dual decision variables and $\top$ denotes the transpose operator. Any vector z satisfying the dual constraints remains feasible if $\bar{x}$ in the objective is replaced by another vector because the dual constraints do not depend on $\bar{x}$. If the dual in (5) is solved for a given $\bar{x}$ to obtain dual solution $\bar{z}$ then it holds that $\varphi(x) \leq (q - Cx)^\top \bar{z}$ for all vectors x. The Benders algorithm initializes the master problem $\max_x px + \varphi$ without constraints, where φ is a real-valued variable. It solves the master problem to obtain the solution $\bar{x}$, then it solves (5) to obtain the solution $\bar{z}$ and a new constraint $\varphi \leq (q - Cx)^\top \bar{z}$ is added to the master problem. This repeats until convergence.



(a) Vectors in U and vector w (dashed) (b) LP constraints of (U, w) (c) Region $Z(b')$ example

Figure 1: Value function U with vector w and the feasible region of the corresponding LP (a and b), region example (c)

Decomposing the Linear Program

As discussed, Algorithm 2 uses an LP to check whether there is a belief b in which the value function represented by U improves after adding vector w. The LP maximizes the improvement d and the algorithm returns the corresponding belief b. The LP in Algorithm 2 will be referred to as the standard LP, parameterized by the tuple (U, w). We start with an analysis of the standard LP, after which we use a decomposition to derive a new algorithm to replace Algorithm 2.

Analysis of the Standard LP

In Figure 1a we visualize a value function U containing 4 vectors and a vector w (dashed) for a POMDP with 2 states. We define a *corner belief* as a belief point in which the slope of the value function changes. In Figure 1a the corner beliefs of the value function U correspond to the dots on the upper surface of the value function. We call each extremum of the belief simplex an *extreme corner belief*. In Figure 1a there are 5 corner beliefs, 2 of which are extreme corner beliefs. The standard LP finds the belief point corresponding to the maximum improvement when adding w to U. It has been shown by Cheng that this belief point is one of the corner beliefs (Cheng 1988). The belief point corresponding to the maximum improvement will be referred to as the *witness corner belief*. In Figure 1a the witness corner belief is the corner belief $(0.18, 0.82)$.

The constraints are shown in Figure 1b, in which each line corresponds to a constraint $(w - u) \cdot b \geq d$, where $u \in U$. The vertical axis represents the objective that needs to be optimized. Therefore, the set of feasible LP solutions is represented by the shaded area under the concave surface and the arrow indicates the direction of optimization. We let $d(b)$ denote the optimal objective value in belief point b. The lemma below describes the correspondence between the constraints and the vectors in U. A proof can be found in the supplement that is available on the homepage of the authors.

Lemma 1. *Each corner of the feasible region of the standard LP (U, w) corresponds to a corner belief of value function U.*

Constraints intersecting at the witness corner belief are necessary to define the optimal LP solution. Other constraints can be removed without changing the optimal LP solution (e.g., constraints with a star). If there are multiple witness corner beliefs with the same optimal objective value, then the constraints intersecting at one of these corners must be kept.

Theorem 1. *Constraints that do not intersect at the witness corner belief are irrelevant and can be removed from the LP without affecting the optimal objective value d.*

Proof. We assume that the value function U has m corner beliefs $b_1, \ldots, b_m$ and w.l.o.g. we assume that b_m is the witness corner belief. From Lemma 1 we know that each corner belief b_l corresponds to an objective value $d(b_l)$. It holds that $d(b_l) \leq d(b_m)$ for $l = 1, \ldots, m-1$ because b_m is the witness corner belief and the objective is maximized. The LP returns the value $\max(d(b_1), \ldots, d(b_m)) = d(b_m)$. Only the constraints intersecting at witness corner belief b_m are required to impose constraints on this value. □

Ideally we would only add necessary constraints, but deciding which constraints are necessary is difficult since it requires knowledge about the unknown optimal LP solution. We will derive an algorithm which selects constraints in a smart way, such that some constraints are never used.

Applying a Benders Decomposition

We start with a high-level overview of our algorithm, shown in Algorithm 3. It initializes a master LP which initially only imposes constraints on the beliefs b_i. Then the algorithm iteratively selects vectors $\hat{u}$ and adds the corresponding constraints $d^* \leq (w - \hat{u})b$ to the master LP. In each iteration the master LP is solved to optimality, and it finds a new constraint which reduces the objective value for belief point b the most. If the belief points found in two successive iterations are identical, then the objective cannot be further reduced and the algorithm terminates. The optimization procedure on lines 4–15 will be referred to as the decomposed LP.

We show that the algorithm can be derived using a Benders decomposition. We define vector w and the vectors in $U = \{u_1, \ldots, u_k\}$ as row vectors and $b = [b_1, \ldots, b_{|S|}]^\top$ is a column vector. We rewrite the standard LP using matrix notation as follows:

$$
\begin{aligned}
d^* = \max\ & [1][d] \qquad (6)\\
\text{s.t.}\ & \begin{bmatrix} w - u_1 \\ \vdots \\ w - u_k \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_{|S|} \end{bmatrix} + \begin{bmatrix} -1 \\ \vdots \\ -1 \end{bmatrix} [d] \geq \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \\
& [1 \ldots 1] \left[b_1 \ldots b_{|S|} \right]^\top = 1, \quad d \text{ free}, \\
& b_i \geq 0 \quad i = 1, \ldots, |S|.
\end{aligned}
$$

Notice that there is a correspondence with the notation in Equation 3 (e.g., b corresponds to x and d corresponds to y). If the vector b is replaced by a fixed belief $\bar{b}$ for which $\sum_{i=1,\ldots,|S|} \bar{b}_i = 1$ and $\bar{b}_i \geq 0$ ($i = 1, \ldots, |S|$), then (6) reduces to the LP below.

$$d^*(\bar{b}) = \max\ [1][d] \quad \text{s.t.} \quad \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} [d] \leq C\bar{b} \quad \text{with } C = \begin{bmatrix} w - u_1 \\ \vdots \\ w - u_k \end{bmatrix} \quad d \text{ free} \tag{7}$$

The dual of (7) can be written as:

$$\min\ (C\bar{b})^\top z \quad \text{s.t.} \quad [1 \ldots 1]z = 1, \quad z_j \geq 0 \quad j = 1, \ldots, k, \tag{8}$$

where z is a column vector representing the dual solution. After solving (8) for a fixed belief $\bar{b}$, the dual solution $\bar{z}$ can be obtained, which we use to define an upper bound on d^*:

$$d^* \leq (Cb)^\top \bar{z} \tag{9}$$

for any b. A Benders algorithm initializes the following LP:

$$\max\ d^* \quad \text{s.t.} \quad [1 \ldots 1]b = 1, \quad d^* \text{ free}, \quad b_i \geq 0 \quad i = 1, \ldots, |S| \tag{10}$$

and solves this master LP to obtain $\bar{b}$. Then it solves (8) using $\bar{b}$ to obtain $\bar{z}$, after which constraint $d^* \leq (Cb)^\top \bar{z}$ is added to the master LP. This repeats until convergence.

The solution of (8) for a given $\bar{b}$ can be obtained without solving an LP. It holds that $\bar{z} = (\bar{z}_1, \ldots, \bar{z}_k)^\top$ where $\bar{z}_j$ is equal to 1 if j equals $\arg\min_{r=1,\ldots,k} \{(w - u_r)\bar{b}\}$ and 0 otherwise. Since $\bar{z}$ contains only one entry $\bar{z}_j$ that equals 1, the constraint in (9) can be written as $d^* \leq (w - u_j)b$ using row j of matrix C, where $j = \arg\min_{r=1,\ldots,k} \{(w - u_r)\bar{b}\}$. This constraint is equal to the constraint that is added on line 10 and 11 of Algorithm 3. The derivation shows that the decomposed LP (i.e., lines 4–15 of Algorithm 3) corresponds to the Benders decomposition of the LP in Algorithm 2.

Although we identified only one subproblem, we call this a decomposition because the full optimization problem has been decomposed into two smaller problems. The algorithm can be terminated early if the objective drops below 0, because it only returns a belief if the objective value is greater than 0. The supplement contains an example which illustrates the execution of our algorithm.

Analysis of the Decomposed LP

In this section we present a formal analysis of the characteristics of the decomposed LP, which makes clear why the algorithm can run faster compared to the standard LP. In our analysis we use the terms vector and constraint interchangeably, since each a vector corresponds to a constraint. The correctness of Algorithm 3 immediately follows from the fact that it corresponds to a Benders decomposition.

Theorem 2. *The decomposed LP in Algorithm 3 computes the same optimal objective value d as the standard LP in Algorithm 2 and terminates after a finite number of iterations.*

input : vector set U, vector w
output : belief state b or symbol ϕ

1 **if** $|U| = 0$ **then**
2 **return** arbitrary belief b
3 **end**
4 define the following master LP:
5 max d^*
6 s.t. $\sum_{i=1}^{|S|} b_i = 1, \ b_i \geq 0 \quad i = 1, \ldots, |S|, \ d^*$ free
7 choose an arbitrary belief b'
8 $U' \leftarrow \emptyset$
9 **do**
10 $\bar{b} \leftarrow b', \ \hat{u} \leftarrow \arg\min_{u \in U} \{(w - u) \cdot \bar{b}\}$
11 add $d^* \leq (w - \hat{u}) \cdot b$ to master LP
12 $U' \leftarrow U' \cup \{\hat{u}\}$
13 solve master LP to obtain belief b'
14 **while** $b' \neq \bar{b}$;
15 $\bar{d} \leftarrow$ last objective d^* found
16 **return** $\bar{b}$ if $\bar{d} > 0$ and ϕ otherwise

Algorithm 3: `FindBeliefDec` – computes the belief in which w improves U the most

Before we proceed we introduce the notation corresponding to important concepts. The decomposed LP incrementally adds constraints, and each constraint corresponds to a vector $u \in U$. At any point in time during the execution of the decomposed LP, the constraints added to the master LP are defined using a set $U' \subseteq U$. This set is also defined on line 8 of Algorithm 3. For each $u \in U'$ there is a constraint $(w - u) \cdot b \geq d$. The constraints in $U' \subseteq U$ define an optimal solution b' and the corresponding objective value d'. If the algorithm selects a constraint u on lines 10–11 for a given $\bar{b}$, then we say that the algorithm *uses* belief $\bar{b}$ to add u. The region Z_u in which u restricts the LP solution space is:

$$Z_u = \{b \mid (w - u) \cdot b \leq (w - u') \cdot b \ \ \forall u' \in U'\}. \tag{11}$$

The belief b' has neighbors $b_1, \ldots, b_l$ which are also corners of the feasible region, with corresponding objective values $d(b_1), \ldots, d(b_l)$. In Figure 1b each corner of the feasible region has two neighbors, except the corners at the extrema of belief space. For state spaces with more than two states corner beliefs may have more than two neighbors. We define the neighbors of b' using a set $NB(b')$:

$$NB(b') = \{b \mid b \text{ is corner belief and } \exists c \in U' \text{ such that } b' \in Z_c \text{ and } b \in Z_c, b \neq b'\}. \tag{12}$$

This set contains the corners b of the feasible region that can be reached from b' in one step, because there is at least one constraint $c \in U'$ such that $b' \in Z_c$ and $b \in Z_c$. The lowest objective value of the neighbors is $d_{\min}(b') = \min_{b \in NB(b')} d(b)$. Since the feasible region of an LP is convex, it holds that $d_{\min}(b') \leq d'$. The region $Z(b')$ in which the objective value is at least $d_{\min}(b')$ is defined as:

$$Z(b') = \{b \mid \min_{u \in U'} \{(w - u) \cdot b\} \geq d_{\min}(b')\}. \tag{13}$$

In the example in Figure 1c the lines (except the bold vertical line) correspond to constraints in U. The black constraints

have been added so far and belong to the set $U' \subseteq U$. The belief b' is the current optimal solution of the master LP, and its two neighbors are represented by dots. In the example it holds that $d_{\min}(b')$ equals 0.05, and therefore the region $Z(b')$ contains the beliefs in which the objective is at least 0.05.

Now we will show that the optimal objective value of the standard LP, which corresponds to the objective value $\bar{d}$ on line 15 of Algorithm 3, is at least $d_{\min}(b')$. Since the feasible region of an LP is convex, this implies that the solution $\bar{b}$ returned by the decomposed LP is a belief point in $Z(b')$.

Theorem 3. *Given the current optimal solution b' and the corresponding objective value d' of the master LP, it holds that $d^* \geq d_{\min}(b')$, where d^* is the optimal objective value of the standard LP.*

Proof. By contradiction. We assume that $d^* < d_{\min}(b')$. For each $b \in Z(b')$ there must be a constraint $u \notin U'$ such that $(w - u) \cdot b \leq d^* < d_{\min}(b')$. We consider an arbitrary neighbor $b_l \in NB(b')$ of b' and a constraint $c \in U'$ such that $b' \in Z_c$ and $b_l \in Z_c$. All corner beliefs $b \in Z_c$ except b' are also neighbor of b' according to definition of NB, which implies that $d(b) \geq d_{\min}(b')$ for each $b \in Z_c$. Now we can conclude that $Z_c \subseteq Z(b')$. Consider the belief b^c that was used to add c. We know that $b^c \in Z_c$ because b^c is a belief in which c restricts the current LP solution space. It is impossible that $b^c \notin Z_c$ because outside the region Z_c there is already another constraint which is more restrictive than c in point b^c, which would have been selected in point b^c instead of c. It holds that $d(b^c) \geq d_{\min}(b')$ because $b^c \in Z_c \subseteq Z(b')$. For b^c there must be a constraint $u \notin U'$ for which $(w - u) \cdot b \leq d^* < d_{\min}(b')$. Constraint u must have been added before c on line 11, which leads to a contradiction. □

In the following theorem we define when a constraint $u \notin U'$ is never added during subsequent iterations, which shows why Algorithm 3 does not always use a constraint for each $u \in U$.

Theorem 4. *Consider the current optimal solution b' and a constraint $c \notin U'$. If $Z_c \cap Z(b') = \emptyset$, then constraint c will never be added to the master LP in subsequent iterations.*

Proof. For each $b \in Z_c$ it holds that $(w - c) \cdot b < d_{\min}(b')$ because $Z_c \cap Z(b') = \emptyset$. During subsequent iterations Algorithm 3 will never find a belief b in which $d(b) < d_{\min}(b')$, because it terminates after finding the optimal solution, which is at least $d_{\min}(b')$ according to Theorem 3. This implies that Algorithm 3 never finds a belief $b \in Z_c$ during subsequent iterations. Hence, constraint c is never added to the master LP during subsequent iterations. □

Figure 1c visualizes the ideas behind Theorems 3 and 4. The optimal solution of the standard LP belongs to the region $Z(b')$ and is at least $d_{\min}(b')$. The dashed constraint restricts the solution space in a region that is not part of $Z(b)$, and therefore it is never added in remaining iterations. Below we show that the decomposed LP only finds beliefs $b \in Z(b')$ during subsequent iterations.

Theorem 5. *Consider the current optimal solution b'. The decomposed LP only finds belief points $b \in Z(b')$ during subsequent iterations.*

Domain	Std (s)	Dec (s)	Speedup	Constr. (%)
Hallway2	88.75	13.35	6.65	18.4 ± 27.6
4x5x2	70.55	12.49	5.65	12.5 ± 17.1
AircraftID	36.66	7.39	4.96	8.8 ± 14.6
4x3	34.38	8.69	3.96	17.8 ± 19.4
Shuttle	28.13	8.04	3.50	18.5 ± 20.3
Tiger-grid	43.86	12.86	3.41	12.9 ± 18.7
Hallway	20.83	7.97	2.61	26.9 ± 28.1
RockS4x4	0.54	0.25	2.16	36.8 ± 22.9
Cheese	0.08	0.04	2.00	87.7 ± 22.2
Network	5.77	4.65	1.24	26.7 ± 19.8
4x4	0.16	0.15	1.07	74.9 ± 22.9
1D	0.003	0.003	1.00	84.0 ± 21.8
Partpaint	0.62	0.62	1.00	39.0 ± 31.9

Table 1: Performance of the standard and decomposed LP

Proof. By contradiction. Suppose that a belief $b \notin Z(b')$ is found during a subsequent iteration, then it holds that $d(b) < d_{\min}(b')$. For each $b \in Z(b')$ there must be a constraint $u \notin U'$ such that $(w - u) \cdot b < d_{\min}(b')$. There exists a constraint $c \in U'$ for which $Z_c \subseteq Z(b')$, and we consider the belief b^c that was used to add c. It holds that $b^c \in Z_c$ because b^c is a belief point in which c restricts the current LP solution space. Moreover, it holds that $d(b^c) \geq d_{\min}(b')$ because $b^c \in Z_c \subseteq Z(b')$. In belief b^c there must be a constraint $u \notin U'$ for which $(w - u) \cdot b < d_{\min}(b')$. Hence, constraint u must have been added before c, which leads to a contradiction. □

Experiments

In this section we evaluate our algorithm at the level of individual LPs, vector pruning and POMDP solving.

Performance of the Decomposed LP

First we compare the performance of the decomposed LP and the standard LP. We selected all POMDP domains used by Cassandra, Littman, and Zhang (1997), Feng and Zilberstein (2004) and Raphael and Shani (2012). For each domain we consider the first 30000 LPs that are solved during the execution of incremental pruning (or until the problem is solved or memory limits are exceeded, details in the supplement). For each LP we execute the standard LP and the decomposed LP, during which we measure the running times and the number of constraints added by the decomposed LP. In the paper we use the LP solver GLPK. Results for Gurobi and lpsolve can be found in the supplement.

The results are shown in Table 1. The columns Std (Standard, Algorithm 2) and Dec (Decomposed, Algorithm 3) represent the total running time of solving 30000 LPs, and the column Speedup shows the corresponding speedup. From the results we conclude that our algorithm improves the performance in each POMDP domain. The column Constr shows the average fraction of the constraints that is used by the decomposed LP. In many cases the decomposed LP uses only a small fraction of the constraints. The relatively large standard deviation can be explained by observing that in small LPs a

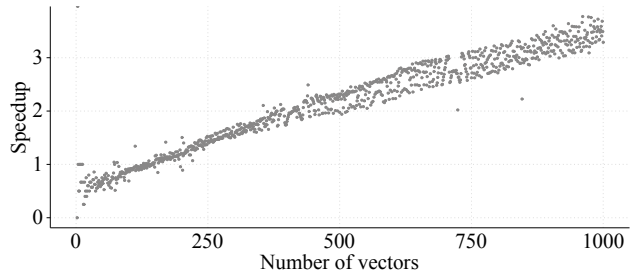


Figure 2: Vector pruning speedup

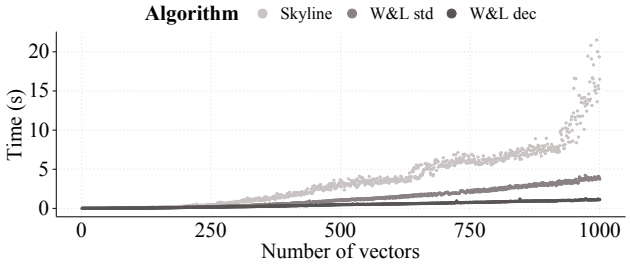


Figure 3: Pruning method comparison

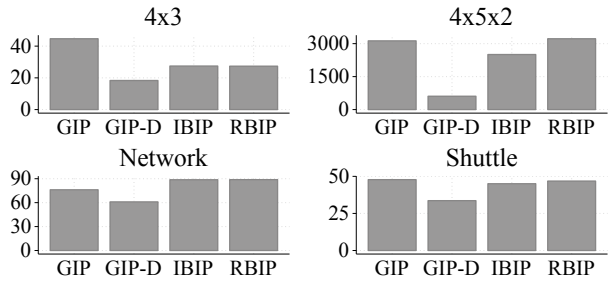


Figure 4: Incremental pruning performance in seconds

relatively large fraction of the constraints is needed, which affects the standard deviation. The supplement contains graphs confirming this explanation.

Performance of Pruning Algorithms

Now we will show that Algorithm 3 improves the performance of Algorithm 1. We use a value function of 4x5x2 to create value functions $V_1, \ldots, V_{1000}$ for which $|V_q| = q$ and $\texttt{prune}(V_q) = V_q$. The pruning algorithm solves q LPs to prune V_q. More details about the value functions V_q can be found in the supplement. For each V_q we measure the speedup that is obtained when using Algorithm 3. Figure 2 shows the speedup for value functions of increasing size. On small value functions our algorithm runs slightly slower because then the gain is small compared to the overhead introduced by solving multiple LPs. On larger instances our algorithm performs consistently better, which confirms that Algorithm 3 improves the performance of Algorithm 1.

Figure 3 shows the pruning time of three methods: the standard variant of White & Lark's method (Algorithm 1+2, W&L std), the decomposed variant of White & Lark's method (Algorithm 1+3, W&L dec) and Skyline (Raphael and Shani 2012). We do not consider Cheng's pruning algorithm (Cheng 1988) because it enumerates corners of the belief space, which scales exponentially in the number of states. In the figure each dot represents an instance. For Skyline we use the iterative variant, which is the fastest variant available. It makes transitions in the so-called Skyline graph using an algorithm inspired by simplex for LPs, and in our tests it runs slower than White & Lark's algorithm. Our algorithm improves the performance of White & Lark's pruning algorithm, and it outperforms all other pruning algorithms.

Performance of Incremental Pruning

Now we show that integrating our algorithm in incremental pruning creates the fastest incremental pruning algorithm. We do not consider other value iteration algorithms, because incremental pruning delivers superior performance compared to other exact POMDP algorithms (Cassandra, Littman, and Zhang 1997). We implemented generalized incremental pruning (Cassandra, Littman, and Zhang 1997), abbreviated GIP, which is the fastest variant available, and we enhanced it with our decomposition method (GIP-D). We also compare with region-based incremental pruning algorithms (Feng and Zilberstein 2004), abbreviated IBIP and RBIP, which exploit information about regions of the belief space when pruning vectors after computing the cross sum.

We provide the results for 4x3, 4x5x2, Network and Shuttle, which we solved to optimality in 10, 18, 98 and 59 iterations, respectively. Figure 4 shows the total running time of the dynamic programming stages in seconds. We conclude that Algorithm 3 improves the performance of generalized incremental pruning, and the resulting algorithm outperforms IBIP and RBIP. Other large domains from Table 1 could not be solved to optimality due to the large number of vectors remaining after pruning. However, GIP-D only replaces the LP in GIP by a faster alternative, as shown in the first experiment, and therefore it can be expected that GIP-D also provides improved performance in these larger domains.

Related Work

Region-based pruning (Feng and Zilberstein 2004) and Skyline (Raphael and Shani 2012) are recent pruning methods, and we have shown that our algorithm outperforms both methods. Among the exact value iteration methods which seek the minimum number of beliefs to construct the next value function (Sondik 1971; Cheng 1988; Kaelbling, Littman, and Cassandra 1998), the witness algorithm is the fastest (Littman, Cassandra, and Kaelbling 1996). However, incremental pruning delivers superior performance compared to the witness algorithm (Cassandra, Littman, and Zhang 1997).

Benders decompositions have been used to create a distributed algorithm for factored MDPs (Guestrin and Gordon 2002). Our subproblems cannot be distributed since there is only one subproblem. Methods have been developed to identify LP constraints that do not constrain the feasible region (Mattheiss 1973). Such constraints never occur in our LPs, because they only contain constraints corresponding to

dominating vectors. The approximate POMDP algorithm α-min adds constraints to a MILP when expanding the set of belief points (Dujardin, Dietterich, and Chadès 2015). Since we consider given and finite sets of constraints, we can rely on a constraint selection rule that is conceptually simpler.

Vector pruning is also used in multi-objective decision making (Roijers, Whiteson, and Oliehoek 2013), finite state controllers for POMDPs (Poupart and Boutilier 2003), approximate point-based POMDP algorithms (Smith and Simmons 2005) and Decentralized POMDP algorithms (Spaan, Oliehoek, and Amato 2011). We expect that our work has the potential to improve algorithms in these areas.

Conclusions

We presented a new algorithm to replace the LP that is used in several exact POMDP solution methods to check for dominating vectors. Our algorithm is based on a Benders decomposition and uses only a small fraction of the constraints in the original LP. We proved the correctness of our algorithm and we analyzed its characteristics. Experiments have shown that our method outperforms commonly used vector pruning algorithms for POMDPs and it reduces the running time of the generalized incremental pruning algorithm. The resulting variant of incremental pruning runs faster than any existing pruning-based algorithm to solve POMDPs optimally.

In future work we will study whether other decomposition methods for LPs can be applied in this domain. For instance, the Dantzig-Wolfe decomposition (Dantzig and Wolfe 1960) generates columns rather than rows and may enable the construction of pruning algorithms with different characteristics.

Acknowledgments

This research is funded by the Netherlands Organisation for Scientific Research (NWO), as part of the Uncertainty Reduction in Smart Energy Systems program. We would like to thank Guy Shani for sharing the source code of Skyline.

References

Bai, H.; Hsu, D.; Kochenderfer, M. J.; and Lee, W. S. 2012. Unmanned Aircraft Collision Avoidance Using Continuous-State POMDPs. In *Robotics: Science and Systems*, 1–8.

Benders, J. F. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4(1):238–252.

Blanco, N. J.; Love, B. C.; Cooper, J. A.; McGeary, J. E.; Knopik, V. S.; and Maddox, W. T. 2015. A frontal dopamine system for reflective exploratory behavior. *Neurobiology of Learning and Memory* 123:84–91.

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 *IJCAI*, 1293–1299.

Cassandra, A. R.; Littman, M. L.; and Zhang, N. L. 1997. Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes. In *UAI*.

Cheng, H. 1988. *Algorithms for Partially Observable Markov Decision Processes*. Ph.D. Dissertation, Univ. of British Columbia.

Dantzig, G. B., and Wolfe, P. 1960. Decomposition principle for linear programs. *Operations Research* 8(1):101–111.

Dujardin, Y.; Dietterich, T.; and Chadès, I. 2015. α-min: A Compact Approximate Solver For Finite-Horizon POMDPs. In *IJCAI*, 2582–2588.

Feng, Z., and Zilberstein, S. 2004. Region-Based Incremental Pruning for POMDPs. In *UAI*, 146–153.

Guestrin, C., and Gordon, G. 2002. Distributed Planning in Hierarchical Factored MDPs. In *UAI*, 197–206.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1):99–134.

Karmokar, A. K.; Senthuran, S.; and Anpalagan, A. 2012. POMDP-based cross-layer power adaptation techniques in cognitive radio networks. In *IEEE Global Communications Conference*, 1380–1385.

Li, D., and Jayaweera, S. K. 2015. Machine-Learning Aided Optimal Customer Decisions for an Interactive Smart Grid. *IEEE Systems Journal* 9(4):1529–1540.

Littman, M. L.; Cassandra, A. R.; and Kaelbling, L. P. 1996. Efficient dynamic-programming updates in partially observable Markov decision processes. Technical report, Brown Univ.

Littman, M. L. 1996. *Algorithms for Sequential Decision Making*. Ph.D. Dissertation, Brown Univ.

Mattheiss, T. H. 1973. An Algorithm for Determining Irrelevant Constraints and all Vertices in Systems of Linear Inequalities. *Operations Research* 21(1):247–260.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, 1025–1030.

Poupart, P., and Boutilier, C. 2003. Bounded Finite State Controllers. In *NIPS*.

Qian, Y.; Zhang, C.; Krishnamachari, B.; and Tambe, M. 2016. Restless Poachers: Handling Exploration-Exploitation Tradeoffs in Security Domains. In *AAMAS*.

Raphael, C., and Shani, G. 2012. The Skyline algorithm for POMDP value function pruning. *Annals of Mathematics and Artificial Intelligence* 65(1):61–77.

Roijers, D. M.; Whiteson, S.; and Oliehoek, F. A. 2013. Computing Convex Coverage Sets for Multi-Objective Coordination Graphs. In *Algorithmic Decision Theory*, 309–323.

Smith, T., and Simmons, R. 2005. Point-Based POMDP Algorithms: Improved Analysis and Implementation. In *UAI*.

Sondik, E. J. 1971. *The optimal control of partially observable Markov processes.* Ph.D. Dissertation, Stanford Univ.

Spaan, M. T. J., and Vlassis, N. 2005. Perseus: Randomized Point-based Value Iteration for POMDPs. *JAIR* 24:195–220.

Spaan, M. T. J.; Oliehoek, F. A.; and Amato, C. 2011. Scaling Up Optimal Heuristic Search in Dec-POMDPs via Incremental Expansion. In *IJCAI*, 2027–2032.

White, C. C. 1991. A survey of solution techniques for the partially observed Markov decision process. *Annals of Operations Research* 32(1):215–230.