Improved Vector Pruning in Exact Algorithms for Solving POMDPs

Eric A. Hansen and **Thomas J. Bowman**
Dept. of Computer Science and Eng.
Mississippi State University
Mississippi State, MS 39762
hansen@cse.msstate.edu, tjb623@msstate.edu

Abstract

Exact dynamic programming algorithms for solving partially observable Markov decision processes (POMDPs) rely on a subroutine that removes, or "prunes," dominated vectors from vector sets that represent piecewise-linear and convex value functions. The subroutine solves many linear programs, where the size of the linear programs is proportional to both the number of undominated vectors in the set and their dimension, which severely limits scalability. Recent work improves the performance of this subroutine by limiting the number of constraints in the linear programs it solves by incrementally generating relevant constraints. In this paper, we show how to similarly limit the number of variables. By reducing the size of the linear programs in both ways, we further improve the performance of exact algorithms for POMDPs, especially in solving problems with larger state spaces.

1 INTRODUCTION

The model of a partially observable Markov decision process (POMDP) is widely used to formulate and solve single-agent planning problems in stochastic domains where decisions are made based on imperfect state information (Kaelbling et al., 1998; Spaan, 2012; Dutech and Scherrer, 2013). Because the optimization problem for POMDPs is intractable, approximation is often needed to solve real-world problems. But exact algorithms still have an important role to play. They provide a benchmark for approximation algorithms; they give optimal solutions to real-world problems that are simple enough to be solved exactly; and they offer a principled approach to approximation in which scalability can be improved in exchange for bounded suboptimality.

Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI), PMLR volume 124, 2020.

Both exact and approximation algorithms for POMDPs compute piecewise-linear and convex value functions that are represented by finite sets of vectors over the state space of a problem, a representation introduced by Smallwood and Sondik (1973). Among exact algorithms for POMDPs, the most efficient belong to a family of *incremental pruning* algorithms (Cassandra et al., 1997). These algorithms rely on a subroutine that removes, or "prunes," dominated vectors from a set of vectors that represents a value function. This subroutine solves one linear program for each vector. In the traditional approach, the number of variables in each linear program is equal to the size of the state space and the number of constraints is as large as the size of the vector set that represents the value function. As a result, use of this pruning subroutine severely limits scalability.

In recent work, Walraven and Spaan (2017) show how to limit the number of constraints in the linear programs solved by this pruning subroutine by incrementally generating relevant constraints. They show that the number of constraints in the resulting linear programs can be a small fraction of the number of vectors used to represent a value function. As a result, their approach substantially improves the performance of incremental pruning, and it is the current state-of-the-art exact solver. However, their approach does not reduce the number of variables in the linear programs, and it increases the number of linear programs that must be solved. As a result, the speedup it provides decreases as the size of the state space of a POMDP increases.

In this paper, we enhance the approach introduced by Walraven and Spaan by showing how to incrementally generate the variables as well as the constraints of the linear programs. By reducing even further the size of the linear programs solved by this subroutine, and especially by reducing the number of variables for problems with large state spaces, our enhancement of this approach improves performance even further, and leads to a more scalable algorithm for solving POMDPs exactly.

2 BACKGROUND

We begin with a review of relevant background.

2.1 POMDP MODEL AND VALUE ITERATION

A POMDP models a planning problem with a set of hidden states $\mathcal{S}$, a set of observations $\mathcal{Z}$, and a set of actions $\mathcal{A}$. The process unfolds over a sequence of stages. At each stage, an action $a \in \mathcal{A}$ taken in a state $s \in \mathcal{S}$ results in a reward with expected value $R(s,a)$, a transition to a state $s' \in \mathcal{S}$ with probability $P(s'|s,a)$, and an observation $z \in \mathcal{Z}$ with probability $P(z|s',a)$. A widely-used objective is to maximize the expected discounted sum of rewards over an infinite horizon, $E[\sum_{t=0}^{\infty} \gamma^t R_t]$, where R_t is the reward received at stage t, and $\gamma \in (0,1)$ is a discount factor that ensures this expected value is finite.

Although the state of the process is not directly observed, state probabilities can be computed using Bayes rule. Let b denote an $|\mathcal{S}|$-dimensional vector of state probabilities, called a *belief state*, where $b(s)$ denotes the probability that the process is in state s. If action a is taken and followed by observation z, the successor belief state, denoted b_z^a, is determined using Bayes' rule, as follows,

$$b_z^a(s') = P(z|s',a) \sum_{s \in \mathcal{S}} P(s'|s,a) b(s) / P(z|b,a), \quad (1)$$

for each successor state $s' \in \mathcal{S}$, where $P(z|b,a) = \sum_{s' \in \mathcal{S}} P(z|s',a) \sum_{s \in \mathcal{S}} P(s'|s,a) b(s)$. Because this belief state provides all information about the history of a process needed for optimal action selection, a POMDP can be recast as an equivalent completely observable MDP over belief states, called a *belief-state MDP*, where $\mathcal{B} = \{b \in \Re^{|\mathcal{S}|} | \sum_{s \in \mathcal{S}} b(s) = 1 \text{ and } b(s) \geq 0, \forall s \in \mathcal{S}\}$ is its continuous $|\mathcal{S}|$-dimensional state space.

Once a POMDP is recast as a belief-state MDP, it can be solved by dynamic programming. In the discounted infinite-horizon case, an optimal value function $V^* : \mathcal{B} \rightarrow \Re$ is the fixed point of a dynamic programming operator T that is defined for all belief states $b \in \mathcal{B}$ as

$$TV(b) = \max_{a \in A} \left\{ R(b,a) + \gamma \sum_{z \in \mathcal{Z}} P(z|b,a) V(b_z^a) \right\}. \quad (2)$$

Value iteration is a successive approximation algorithm that converges to this fixed point in the limit; that is, $V^* = \lim_{n \rightarrow \infty} T^n V_0$, where T^n denotes n applications of the operator T to any initial value function V_0. An optimal policy $\mu^* : \mathcal{B} \rightarrow \mathcal{A}$ is a greedy policy with respect to V^*. In the discounted infinite-horizon case, the dynamic programming operator T is a contraction operator with respect to the supremum norm, ensuring that value iteration computes an arbitrarily close approximation of V^* and μ^* after a finite number of iterations.

Input: vector set $\mathcal{W}$, tolerance $\epsilon \geq 0$
Output: pruned vector set $\mathcal{D}$

```
D ← ∅
while W ≠ ∅ do
    w ← arbitrary element in W
    if w(s) ≤ u(s), ∃u ∈ D, ∀s ∈ S then
        W ← W\{w}
    else if w(s) > u(s), ∃s ∈ S, ∀u ∈ D ∪ W then
        D ← D ∪ {w}, W ← W\{w}
    else
        b ← LP(w, D, ε)
        if (b = nil) then
            W ← W\{w}
        else
            ŵ ← argmax_{w∈W} Σ_{s∈S} b(s) · w(s)
            D ← D ∪ {ŵ}, W ← W\{ŵ}
return (D)
```

Algorithm 1: PR(W): Vector pruning subroutine.

2.2 PRUNING SUBROUTINE

A key property of the dynamic-programming operator for POMDPs is that it preserves the piecewise linearity and convexity of the value function. A value function $V : \mathcal{B} \rightarrow \Re$ that is piecewise linear and convex (PWLC) can be parameterized by a finite set of $|\mathcal{S}|$-dimensional vectors of real numbers, $\mathcal{V} = \{v^0, v^1, \ldots, v^k\}$, so that the value of any belief state b is given as follows:

$$V(b) = \max_{v^i \in \mathcal{V}} \sum_{s \in \mathcal{S}} b(s) v^i(s). \quad (3)$$

For any PWLC value function, there is a unique and minimal-size set of vectors that represents it. Algorithm 1, due to Lark (White, 1991), prunes a set of vectors to its minimum size by removing *dominated* vectors, which are vectors whose removal does not affect the value function that is represented. The following linear program tests whether a vector w is dominated by the vectors in a set $\mathcal{U}$:

$$\begin{aligned} \max d \text{ s.t. } & \sum_{s \in \mathcal{S}} b(s) \cdot \big(w(s) - u^i(s)\big) \geq d, \forall u^i \in \mathcal{U} \\ & \sum_{s \in \mathcal{S}} b(s) = 1, b(s) \geq 0, \forall s \in \mathcal{S}. \end{aligned} \quad (4)$$

This linear program finds the belief state b at which the value function represented by the vector set $\mathcal{U}$ is improved the most by adding the vector w to $\mathcal{U}$. If the value d maximized by the linear program is non-positive, the vector w is dominated. Otherwise, the vector w is not dominated and d is the amount by which it gives a better value for the belief state b than any vector in $\mathcal{U}$.

Input: vector w, vector set $\mathcal{U}$, tolerance ϵ
Output: belief state b or nil
1 Solve the linear program
2 maximize d subject to the constraints
3 $\sum_{s \in S} b(s) \left(w(s) - u^i(s) \right) \geq d, \forall u^i \in \mathcal{U}$
4 $\sum_{s \in S} b(s) = 1, b(s) \geq 0, \forall s \in \mathcal{S}$
5 **if** $(d \leq \epsilon)$ **then return** (nil)
6 **else return** (b)
Algorithm 2: *LP($w, \mathcal{U}, \epsilon$)* returns the belief state b that maximizes d, if $d > \epsilon$; otherwise, it returns nil.

Algorithm 2 is the subroutine invoked to perform this linear program test. For convenience, we refer to it as *LP($w, \mathcal{U}, \epsilon$)* from now on, where the parameter ϵ is explained below. If a vector w is dominated, it returns nil. Otherwise, it returns the belief state b that maximizes d. Line 13 of Algorithm 1 then selects the best vector for the belief state b returned by Algorithm 2. Littman et al. (1995) describe some subtle tie-breaking issues that are left out of our pseudocode. In some cases, it is possible to determine whether a vector is dominated or not without solving a linear program. A vector w is dominated by a vector $u \in \mathcal{U}$ if $w(s) \leq u(s)$, for all $s \in \mathcal{S}$, as tested by line 4 of Algorithm 1. It is *not* dominated if there is some state $s \in \mathcal{S}$ for which it gives a better value than any vector in $\mathcal{U}$, as tested by Line 6.

2.3 APPROXIMATION

When a vector set $\mathcal{U}$ that represents a PWLC value function becomes very large, a tolerance $\epsilon \geq 0$ can be used to reduce its size in exchange for bounded-error approximation. A vector w is said to be *ϵ-dominated* by the vectors in a set U if there is no belief state b for which $b \cdot w > b \cdot u + \epsilon$, for all $u \in U$, or, equivalently, if the scalar d maximized by the linear program of Equation (4) is less than or equal to ϵ. Algorithm 2 includes ϵ as a parameter because it can be used to test for ϵ-dominance. A subset $\mathcal{U}_\epsilon \subseteq \mathcal{U}$ is called an ϵ-approximate covering of a set $\mathcal{U}$ if every vector $v \in \mathcal{U} \backslash \mathcal{U}_\epsilon$ is ϵ-dominated by one or more vectors in $\mathcal{U}_\epsilon$. The value function represented by the vector set $\mathcal{U}_\epsilon$ gives a value for any belief state that is within ϵ of its exact value.

The pruning subroutine of Algorithm 1 allows ϵ to be specified as a parameter, and finds an ϵ-approximate covering of any vector set $\mathcal{U}$. Although there is a unique and minimal-size set of vectors $\mathcal{U}$ that represents a PWLC value function V, there is not a unique set of vectors that represents an ϵ-approximate covering of the set $\mathcal{U}$. Nevertheless, this approach to approximation offers a useful tradeoff between the size of the vector set that represents a value function and the accuracy of the approximation, where the tradeoff is adjusted by adjusting ϵ.

2.4 INCREMENTAL PRUNING

The pruning subroutine of Algorithm 1 plays a key role in the incremental pruning algorithm for POMDPs (Cassandra et al., 1997), which leverages the fact that the updated value function TV of Equation (2) can be defined as a combination of simpler value functions, as follows:

$$V_z^a(b) = R(b,a) + \gamma Pr(z|b,a) V(b_z^a), \quad (5)$$

$$V^a(b) = \sum_{z \in Z} V_z^a(b), \text{ and} \quad (6)$$

$$TV(b) = \max_{a \in A} V^a(b). \quad (7)$$

Each of these value functions is piecewise linear and convex, and thus can be represented by a set of vectors. We use the symbols $\mathcal{V}_z^a$, $\mathcal{V}^a$, and $\mathcal{V}'$ to denote these sets, and let $\mathcal{PR}(\mathcal{U})$ denote the set of vectors that remains after the subroutine of Algorithm 1 prunes the vector set $\mathcal{U}$.

The incremental pruning algorithm generates the vector sets $\mathcal{V}_z^a$, $\mathcal{V}^a$, and $\mathcal{V}'$ as follows. In the *projection* step that corresponds to Equation (5), a set of vectors is generated for each pair of action a and observation z, as follows,

$$\mathcal{V}_z^a = \mathcal{PR}\left(\{ v(i,a,z) | v^i \in \mathcal{V} \} \right), \quad (8)$$

where $v(i,a,z)$ is the $|\mathcal{S}|$-vector defined by

$$v(i,a,z)(s) = \frac{R(s,a)}{|\mathcal{Z}|} + \gamma \sum_{s' \in S} P(z,s'|s,a) v^i(s'), \quad (9)$$

and $P(z,s'|s,a) = P(z|a,s')P(s'|s,a)$. In the *cross-sum* step corresponding to Equation (6), a set of vectors is generated for each action a, as follows, $\mathcal{V}^a = \mathcal{PR}(\ldots(\mathcal{PR}(\mathcal{PR}(\mathcal{V}_1^a \oplus \mathcal{V}_2^a) \oplus \mathcal{V}_3^a) \ldots \oplus \mathcal{V}_{|\mathcal{Z}|}^a)$, where the observations are indexed from 1 through $|\mathcal{Z}|$. The *maximization* step corresponding to Equation (7) generates the vector set: $\mathcal{V}' = \mathcal{PR}\left(\cup_{a \in \mathcal{A}} \mathcal{V}^a \right)$.

3 CONSTRAINT GENERATION

The pruning subroutine of Algorithm 1 accounts for most of the computation time of the incremental pruning algorithm. Cassandra et al. (1997) report that it takes 95% of the running time of incremental pruning for their small test problems. For larger POMDPs, it may take a higher percentage because the linear programs are larger.

Recent work (Walraven and Spaan, 2017; Roijers et al., 2018; Walraven, 2019) shows how to use a row-generation technique called *Benders decomposition* to speed up the linear program test for dominance used by the pruning subroutine. We review this approach, and also introduce a subtle but important improvement.

Input: vector w,vector set $\mathcal{U}$, tolerance ϵ
Output: belief state b or nil

$b \leftarrow$ arbitrary belief state
$\mathcal{U}' \leftarrow \emptyset$
repeat
  $b' \leftarrow b$
  if $(\mathcal{U}' \neq \mathcal{U})$ **then**
    $\hat{u} \leftarrow \text{argmin}_{u \in \mathcal{U} \setminus \mathcal{U}'} \sum_{s \in \mathcal{S}} b(s)(w(s) - u(s))$
    $\mathcal{U}' \leftarrow \mathcal{U}' \cup \{\hat{u}\}$
    $(d, b) \leftarrow LP(w, \mathcal{U}', \epsilon)$
until $(d \leq \epsilon)$ **or** $(b = b')$
if $(d \leq \epsilon)$ **then return** (nil)
else return (b)

Algorithm 3: *BendersWS*$(w, \mathcal{U}, \epsilon)$ uses Benders decomposition to solve the linear program used to test for vector dominance.

3.1 BENDERS DECOMPOSITION

In Benders decomposition, a large linear program is solved more efficiently by decomposing it into a sequence of smaller linear programs. First, a *master linear program* is defined that has no constraints. Then relevant (or potentially relevant) constraints are added one at a time. After each constraint is added, the revised master linear program is solved, and information from the solution is used to add another constraint. The procedure repeats until the master linear program has all relevant constraints, and the procedure is terminated with a solution. This approach is effective when most of the constraints of the original linear program are superfluous, and can be removed without affecting the solution.

Algorithm 3 gives pseudocode for the algorithm described by Walraven and Spaan (2017), which uses Benders decomposition to solve the linear program of Equation (4). It is invoked by the pruning subroutine instead of invoking Algorithm 2, and it computes the same result. Let $\mathcal{U}$ denotes the set of all vectors, and let $\mathcal{U}'$ denote a subset of this set that defines the constraints of a master linear program. If $d \leq \epsilon$, where d is the scalar optimized by the linear program, the vector w is ϵ-dominated and the algorithm terminates. If $d > \epsilon$, the belief state b that solves the linear program is used to try to find an additional relevant constraint, as follows,

$$\hat{u} \leftarrow \underset{u^i \in \mathcal{U} \setminus \mathcal{U}'}{\text{argmin}} \sum_{s \in \mathcal{S}} b(s) \left(w(s) - u^i(s) \right), \qquad (10)$$

and the minimizing vector $\hat{u}$ is added to the vector set $\mathcal{U}'$ that defines the constraints of the master linear program, which is solved again. The process repeats until the scalar d returned by the linear program is less than or equal to ϵ, or the belief state in successive iterations is the same, which indicates that w is *not* dominated.

Input: vector w,vector set $\mathcal{U}$, tolerance ϵ
Output: belief state b or nil

$b \leftarrow$ arbitrary belief state
$\mathcal{U}' \leftarrow \emptyset$
repeat
  if $(\mathcal{U}' \neq \mathcal{U})$ **then**
    $\hat{u} \leftarrow \text{argmin}_{u \in \mathcal{U} \setminus \mathcal{U}'} \sum_{s \in \mathcal{S}} b(s)(w(s) - u(s))$
    $du \leftarrow \sum_{s \in \mathcal{S}} b(s) \left(w(s) - \hat{u}(s) \right)$
    if $(du \leq \epsilon)$ **then**
      $\mathcal{U}' \leftarrow \mathcal{U}' \cup \{\hat{u}\}$
      $(b, d) \leftarrow LP(w, \mathcal{U}', \epsilon)$
  else $du \leftarrow d$
until $(d \leq \epsilon)$ **or** $(du > \epsilon)$
if $(d \leq \epsilon)$ **then return** (nil)
else return (b)

Algorithm 4: Improved *BendersWS*$(w, \mathcal{U}, \epsilon)$.

3.2 IMPROVED TERMINATION CONDITION

Algorithm 4 shows pseudocode for a modified version of Algorithm 3 that we have found to be more efficient. The two algorithms behave in exactly the same way when they detect that a vector w is dominated. But when w is not dominated, Algorithm 3 returns the same belief state as the linear program of Equation (4), which is the belief state that maximizes the objective value d. Algorithm 4 simply returns the first belief state it finds that shows w is not dominated.

When Algorithm 4 adds a constraint to the master linear program by selecting a vector $\hat{u} \in \mathcal{U} \setminus \mathcal{U}'$, as follows,

$$\hat{u} \leftarrow \underset{u^i \in \mathcal{U} \setminus \mathcal{U}'}{\text{argmin}} \sum_{s \in \mathcal{S}} b(s) \left(w(s) - u^i(s) \right), \qquad (11)$$

it also computes the value:

$$du \leftarrow \sum_{s \in \mathcal{S}} b(s) \left(w(s) - \hat{u}(s) \right). \qquad (12)$$

If $du < \epsilon$, the minimizing vector $\hat{u}$ is added to the set $\mathcal{U}'$ used to define the constraints of the master linear program, which is then solved again. But if $du \geq \epsilon$, no other vector in $\mathcal{U}$ gives a better value (within ϵ) for the belief state b than the vector w, and so the algorithm concludes that w is *not* dominated and terminates.

It follows that Algorithm 4 does not always return the same belief state as Algorithms 2 and 3. But that does not affect its correctness. Algorithm 4 always correctly determines whether or not w is dominated (or ϵ-dominated), and it usually does so by solving fewer and smaller linear programs. As a result, it usually runs faster, and often *much* faster than Algorithm 3, as our experimental results will show.

4 VARIABLE GENERATION

We next describe the primary contribution of our paper: an approach to improving the scalability of vector pruning by incrementally generating the variables as well as the constraints of the linear program dominance test.

4.1 DUAL LINEAR PROGRAM

The linear program dominance test given by Equation (4) has a dual linear program that can also be used to test for dominance: a vector w is ϵ-dominated if the following linear program has a solution d where $d \leq \epsilon$:

$$\min d \text{ s.t. } \sum_{i=1}^{|\mathcal{U}|} c(i) \cdot \big(w(s) - u^i(s)\big) \leq d, \forall s \in \mathcal{S}$$
$$\sum_{i=1}^{|\mathcal{U}|} c(i) = 1, c(i) \geq 0, i = 1 \ldots |\mathcal{U}|. \quad (13)$$

In this case, the vector w is ϵ-dominated if there is some probability distribution c over the constraints of the linear program such that $\sum_{i=1}^{|\mathcal{U}|} c(i) \cdot u^i(s) + \epsilon \geq w(s), \forall s \in \mathcal{S}$. We refer to c from now on as a *convex combination*.

As noted by Poupart and Boutilier (2004), use of this dual linear program to test whether a vector w is dominated generalizes the test for pointwise dominance used in line 4 of Algorithm 1. That is, the convex combination c can be used to define an $|\mathcal{S}|$-dimensional vector u^c, where for each $s \in \mathcal{S}$:

$$u^c(s) = \sum_{i=1}^{|\mathcal{U}|} c(i) \cdot u^i(s). \quad (14)$$

If $u^c(s) \geq w(s), \forall s \in \mathcal{S}$, then the vector w is dominated. (From this perspective, the test for pointwise dominance in line 4 of Algorithm 1 is a degenerate convex combination where a single constraint has a probability of 1.0 and the other constraints have a probability of zero.)

Since most linear program solvers return the solution of the dual linear program in addition to the solution of the primal linear program, Algorithm 5 shows a revision of Algorithm 2 that returns the solutions of both. We refer to it as *LP(*$w, \mathcal{U}, \mathcal{S}, \epsilon$*)*. Note that the scalar d that is minimized by the dual linear program is the same as the scalar d that is maximized by the primal linear program. There is an interesting relationship between these two cases. In solving the primal linear program, the value d is maximized in an attempt to find a belief state b for which the vector w is *not* dominated. In solving the dual linear program, the value d is minimized in an attempt to find a convex combination of the vectors that shows that w *is* dominated. Thus these two cases complement each other in establishing whether a vector w is dominated or not.

Input: vector w, vector set U, state set S, tolerance ϵ
Output: belief state b, convex combination c, value d
1 Solve the primal linear program
2 maximize d subject to the constraints
3 $\sum_{s \in S} b(s) \big(w(s) - u^i(s)\big) \geq d, \forall u^i \in \mathcal{U}$
4 $\sum_{s \in S} b(s) = 1, b(s) \geq 0, \forall s \in S$
5 and simultaneously solve the dual linear program
6 minimize d subject to the constraints
7 $\sum_{i=1}^{|\mathcal{U}|} c(i) \cdot \big(w(s) - u^i(s)\big) \leq d, \forall s \in \mathcal{S}$
8 $\sum_{i=1}^{|\mathcal{U}|} c(i) = 1, c(i) \geq 0, i = 1 \ldots |\mathcal{U}|$
9 **if** $(d \leq \epsilon)$ **then return** (nil, c, d)
10 **else return** (b, c, d)

Algorithm 5: *LP(*$w, \mathcal{U}, \mathcal{S}, \epsilon$*)* returns the belief state b that maximizes d, if $d > \epsilon$; the convex combination c that minimizes d; and the scalar objective d.

4.2 DIMENSION OF SOLUTION

We next note a useful relationship between the belief state b that solves the primal linear program and the convex combination c that solves its dual: the number of states for which the belief state b has positive probability is (almost always) equal to the number of vectors for which the convex combination c has positive probability.

This observation follows from the elementary theory of linear programming: a belief state that solves the primal linear program (or, equivalently, a convex combination that solves its dual) must be a *basic feasible solution* of the linear program. That means it is the solution of a system of linear equations where the equations correspond to a subset of the constraints of the linear program with the inequalities changed to equalities, and the variables in this system of linear equations are the variables of the linear program that have non-zero values in a solution, that is, they are *basic variables*. (Geometrically, each basic feasible solution corresponds to a corner point of the polyhedron of feasible solutions defined by the constraints. A corner point is a belief state, and the number of states with non-zero probabilities in the belief state is equal to the number of hyperplanes that intersect at this corner point.) From this observation, it follows that the number of useful constraints in the solution of the linear program test for dominance is bounded by the size of the state set.

If the linear program of Equation (4) has a *degenerate* solution, in which case there are multiple solutions, the number of non-zero probabilities in the belief state b that solves the linear program may not be equal to the number of non-zero probabilities in the convex combination c that solves its dual. But in our experiments with POMDP pruning, this occurs infrequently, and when it does, the difference is small, and usually just a difference of one.

4.3 COMBINED BENDERS DECOMPOSITIONS

Given that the number of non-zero variables in the solution of the linear program test for dominance is equal to the number of non-zero constraints, we next consider whether the number of variables in the linear program can be limited in the same way as the number of constraints. If so, that could be useful in pruning vector sets for POMDPs with large state spaces.

Obviously, a similar Benders decomposition can be used to solve the dual linear program of Equation (13). Let $\mathcal{S}'$ denote a subset of the state set $\mathcal{S}$ that is used to define the constraints of a master linear program. If the scalar d minimized by this linear program has a value greater than 0 (or greater than some $\epsilon > 0$, in the case of approximation), the vector w is *not* dominated. Since no convex combination of vectors in $\mathcal{U}$ dominates w for this subset of states $\mathcal{S}'$, then no convex combination of vectors can dominate w for the full state space $\mathcal{S}$. If $d \leq \epsilon$, however, it is not yet clear whether w is dominated or not because only a subset of the state set has been considered so far. In this case, the convex distribution c over vectors given by the solution of the linear program is used to select a state $\hat{s}$ to add to the master linear program, as follows

$$\hat{s} \leftarrow \arg\max_{s \in \mathcal{S} \setminus \mathcal{S}'} \sum_{i=1}^{|\mathcal{U}|} c(i) \cdot \big(w(s) - u^i(s)\big), \qquad (15)$$

where

$$ds \leftarrow w(\hat{s}) - \sum_{i=1}^{|\mathcal{U}|} c(i) \cdot u^i(\hat{s}). \qquad (16)$$

If $ds \leq d$, the vector w must be dominated because for every state $s \in \mathcal{S}$, its value is less than or equal to the value of the vector created by a convex combination c of vectors from $\mathcal{U}$. If $ds > d$, the minimizing state $\hat{s}$ is added to the master linear program, which is re-solved.

However, this use of Benders decomposition only limits the number of states used to define the dual linear program test for dominance. We want to combine it with the Benders decomposition described by Walraven and Spaan (2017), which limits the number of vectors used to define the primal linear program test for dominance. That is, we want to use both Benders decompositions in order to limit both the number of states and the number of vectors used to define a master linear program that is used to test whether a vector w is dominated. How we do so is summarized by the following theorem, which has a straightforward proof.

Theorem 1. *Consider the linear program of Equation* (4) *and its dual, given by Equation* (13)*, which test whether an $|\mathcal{S}|$-dimensional vector w is ϵ-dominated by the value function represented by a vector set $\mathcal{U}$. If defined for only a subset $\mathcal{S}' \subseteq \mathcal{S}$ of states and a subset $\mathcal{U}' \subseteq \mathcal{U}$ of vectors, the smaller linear program and its dual, denoted $LP(w, \mathcal{S}', \mathcal{U}', \epsilon)$, can still detect whether a vector w is dominated or not, as follows.*

(a) A vector w is not *ϵ-dominated if the scalar d optimized by the linear program $LP(w, \mathcal{U}', \mathcal{S}', \epsilon)$ is greater than ϵ, and for the belief state b in the solution of the linear program,*

$$\min_{u^i \in \mathcal{U} \setminus \mathcal{U}'} \sum_{s \in \mathcal{S}'} b(s) \big(w(s) - u^i(s)\big) > \epsilon, \qquad (17)$$

where the latter condition is tested without linear programming.

(b) A vector w is ϵ-dominated if the scalar d optimized by the linear program $LP(w, \mathcal{U}', \mathcal{S}', \epsilon)$ is less than or equal to ϵ, and for the convex combination c given by the solution of the linear program,

$$\max_{s \in \mathcal{S} \setminus \mathcal{S}'} \sum_{i=1}^{|\mathcal{U}'|} c(i) \big(w(s) - u^i(s)\big) \leq \epsilon, \qquad (18)$$

where the latter condition is tested without linear programming.

Input: vector w, vector set $\mathcal{U}$, tolerance ϵ
Output: belief state b or nil

// Initialize $\mathcal{S}'$ with 2 states and $\mathcal{U}'$ with 2 vectors
$(\hat{s}, \hat{u}) \leftarrow \text{argmin}_{s \in \mathcal{S}, u \in \mathcal{U}} (w(s) - u(s))$
$\mathcal{S}' \leftarrow \{\hat{s}\}$, $\mathcal{U}' \leftarrow \{\hat{u}\}$
$\hat{s} \leftarrow \text{argmax}_{s \in \mathcal{S} \setminus \mathcal{S}'} (w(s) - \hat{u}(s))$
$\mathcal{S}' \leftarrow \mathcal{S}' \cup \{\hat{s}\}$
$\hat{u} \leftarrow \text{argmin}_{u \in \mathcal{U} \setminus \mathcal{U}'} (w(\hat{s}) - u(\hat{s}))$
$\mathcal{U}' \leftarrow \mathcal{U}' \cup \{\hat{u}\}$
repeat
  $(b, c, d) \leftarrow LP(w, \mathcal{U}', \mathcal{S}', \epsilon)$
  if $(d > \epsilon)$ **and** $(\mathcal{U}' \neq \mathcal{U})$ **then** // add vector
    $\hat{u} \leftarrow \text{argmin}_{u^i \in \mathcal{U} \setminus \mathcal{U}'} \sum_{s \in \mathcal{S}'} b(s) \big(w(s) - u^i(s)\big)$
    $du \leftarrow \sum_{s \in \mathcal{S}'} b(s) (w(s) - \hat{u}(s))$
    if $(du \leq \epsilon)$ **then** $\mathcal{U}' \leftarrow \mathcal{U}' \cup \{\hat{u}\}$
  else $du \leftarrow d$
  if $(d \leq \epsilon)$ **and** $(\mathcal{S}' \neq \mathcal{S})$ **then** // add state
    $\hat{s} \leftarrow \arg\max_{s \in \mathcal{S} \setminus \mathcal{S}'} \Big(w(s) - \sum_{i=1}^{|\mathcal{U}'|} c(i) \cdot u^i(s)\Big)$
    $ds \leftarrow w(\hat{s}) - \sum_{i=1}^{|\mathcal{U}'|} c(i) \cdot u^i(\hat{s})$
    if $(ds > \epsilon)$ **then** $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{\hat{s}\}$
  else $ds \leftarrow d$
until $((d \leq \epsilon)$ **and** $(ds \leq \epsilon))$ // w is dominated
  or $((d > \epsilon)$ **and** $(du > \epsilon))$ // w is not dominated
if $(d \leq \epsilon)$ **then return** (nil)
else return (b)

Algorithm 6: *BendersNew*$(w, \mathcal{U}, \mathcal{S}, \epsilon)$ uses Benders decomposition of both variables and constraints to perform linear program dominance test.

Proof. (a) In the first case, the vector w gives a better value (by more than ϵ) for the belief state b over $\mathcal{S}'$ than any vector in $\mathcal{U}'$. If w also gives a better value (by more than ϵ) for belief state b than any vector in $\mathcal{U}\backslash\mathcal{U}'$, it follows that

$$\min_{u^i \in \mathcal{U}} \sum_{s \in \mathcal{S}'} b(s) \Big(w(s) - u^i(s) \Big) \Big) > \epsilon, \tag{19}$$

which means that w is not ϵ-dominated.

(b) In the second case, the vector w is ϵ-dominated by the convex combination c of vectors from $\mathcal{U}'$ when just the subset of states $\mathcal{S}'$ is considered. If it is also ϵ-dominated by this convex combination c when all states in $\mathcal{S}$ are considered, we have

$$\max_{s \in \mathcal{S}} \left(w(s) - \sum_{i=1}^{|\mathcal{U}'|} c(i) \cdot u^i(s) \right) \leq \epsilon, \tag{20}$$

which means that w is ϵ-dominated. □

Algorithm 6 shows pseudocode for an algorithm that solves the linear program test for dominance using both Benders decompositions, that is, by incremental generating both variables and constraints. It starts with an initial subset $\mathcal{S}'$ of two states and an initial subset $\mathcal{U}'$ of two vectors, selected in lines 2 through 7 by the same heuristic used in the rest of the algorithm to add states and vectors. In each iteration of the algorithm, the linear program $LP(w, \mathcal{S}', \mathcal{U}', \epsilon)$ is solved, which is defined by the subsets $\mathcal{S}'$ and $\mathcal{U}'$. If $d \leq \epsilon$, where ϵ is the approximation parameter and $\epsilon = 0$ means no approximation, the algorithm uses the reasoning of case (a) of Theorem 1 to try to show w is not dominated. But if it finds a vector $\hat{u} \in \mathcal{U}\backslash\mathcal{U}'$ that gives as good or better a value for the belief state b than w, it adds $\hat{u}$ to $\mathcal{U}'$ and returns to the top of the loop to solve the linear program again. If $d > \epsilon$, the algorithm uses the reasoning of case (b) of Theorem 1 to try to show w *is* dominated. But if it finds a state $\hat{s} \in \mathcal{S}\backslash\mathcal{S}'$ for which w is not dominated, it adds $\hat{s}$ to $\mathcal{S}'$ and returns to the top of the loop to solve the linear program again.

5 EXPERIMENTS AND ANALYSIS

Our experiments compare the performance of three pruning subroutines. For convenience, we let "WS" denote the Walraven and Spaan algorithm; "IWS" denotes our revision of their algorithm, which uses an "improved" termination condition; and "New" denotes the new algorithm. All algorithms are implemented in a Java open-source POMDP solver made available by Walraven (`www.erwinwalraven.nl/solvepomdp`), using the Gurobi 8 linear program solver, on an Intel 4.2GHz processor with 16GB RAM.

5.1 RANDOMLY-GENERATED VECTOR SETS

Table 1 compares the performance of the three algorithms in pruning randomly-generated vector sets of varying size and dimension, where roughly 30% of the vectors in each set are undominated. Pruning is exact, that is, $\epsilon = 0$, and the table compares running times in CPU seconds. The table also compares the size of the largest (that is, the final) linear program solved when testing a given vector for dominance, for each method. It shows the average and maximum number of *non-zero* variables and constraints in the final linear program for each method, and the average and maximum number of *total* variables and constraints in the final linear program. For *WS* and *IWS*, the number of variables is always equal to the size of the state set. Finally, the table shows the average and maximum number of linear programs solved for each vector tested.

It is clear from the results that *WS* is slower than the other two methods. Note also the different sizes of the largest linear programs solved by each method. Both *IWS* and *New* solve linear programs with many fewer constraints than the linear programs solved by *WS*. That is because they terminate as soon as they find a belief state for which w is better than any other vector, whereas *WS* continues to try to find a belief state for which d is maximized. When we don't care about maximizing d, and just want to determine whether w is dominated or not, the final linear program is just the first linear program that provides enough information to settle that question. It follows that there is not a single final linear program that every method finds. That is why the sizes of the final linear programs solved by each method are different. Importantly, *New* solves linear programs with even fewer constraints than those solved by *IWS*. That is because incremental variable generation biases the search towards solutions with fewer variables, and a bound on the number of non-zero variables in the solution also bounds the number of non-zero constraints.

Although *New* solves smaller linear programs than *IWS*, it also solves more linear programs because it incremental generates variables as well as constraints. Unfortunately, this offsets some of the advantage of solving smaller linear programs. But as the size of the state space increases, *New* gains more of an advantage.

The presence of more zero-probability constraints in the final linear program as the size of the vector set increases is explained by Walraven and Spaan (2017). Constraints are first added for the extrema of the belief simplex, and then additional constraints are added that more narrowly define the region of belief space for which the vector w optimizes the value function. In the process, the initial constraints often become zero-valued.

Table 1: For randomly generated vector sets of varying size and dimension, the table compares the performance of the three pruning subroutines. It shows the average (avg) and maximum (max) number of nonzero and total variables and constraints in the solution of the final linear program used to test a vector for dominance, the average and maximum number of linear programs solved in testing a vector, and the runtime in CPU seconds for pruning the entire set.

			Variables				**Constraints**					
		Time	**Non-0**		**Total**		**Non-0**		**Total**		**LPs**	
Problem	**Alg.**	*sec*	*avg*	*max*	*avg*	*max*	*avg*	*max*	*avg*	*max*	*avg*	*max*
$\|\mathcal{S}\| = 500$	WS	134	42	64	500	500	42	64	58	106	58	106
$\|\mathcal{V}\| = 5,000$	IWS	24	11	24	500	500	11	24	11	26	11	26
	New	13	6	11	7	20	6	11	15	34	19	41
$\|\mathcal{S}\| = 500$	WS	454	47	74	500	500	47	74	68	125	68	125
$\|\mathcal{V}\| = 10,000$	IWS	69	13	25	500	500	13	25	13	25	13	25
	New	36	7	11	8	23	7	11	19	37	24	45
$\|\mathcal{S}\| = 1,000$	WS	375	58	92	1,000	1,000	58	92	78	137	78	137
$\|\mathcal{V}\| = 5,000$	IWS	45	10	21	1,000	1,000	10	21	11	22	11	22
	New	19	6	14	7	31	6	14	14	27	18	43
$\|\mathcal{S}\| = 1,000$	WS	1,261	66	107	1,000	1,000	66	107	94	175	94	175
$\|\mathcal{V}\| = 10,000$	IWS	126	12	25	1,000	1,000	12	25	12	25	12	25
	New	49	6	16	7	28	6	16	18	36	22	43

5.2 POMDP BENCHMARKS

Table 2 shows how the pruning subroutines perform when used as part of the incremental pruning algorithm in solving two robot navigation POMDPs, *CIT* and *Hallway2*, described in Cassandra's (1998) dissertation and available on his website (`https://www.pomdp.org`). It also shows results for a benchmark POMDP described by Smith and Simmons (2004), called *RockSample[4,4]*, available at the website (`http://longhorizon.org/trey/zmdp/index.html`).

Table 2 shows similar statistics as Table 1, but for the first six iterations of value iteration, and for all steps of the incremental pruning algorithm. That means the vector sets that are pruned in each iteration have different sizes, and can grow from one iteration to the next. To prevent the vector sets from growing too large too quickly, an ϵ-approximation threshold was used and tuned to each problem, with the value of ϵ shown in the table. The size of the state sets, and the average and maximum size of the vector sets, is also shown in the table, since both have a significant influence on the efficiency of pruning.

Similar to the results for randomly-generated vector sets, the table shows that the improved pruning subroutine *IWS* is often more efficient than the original subroutine *WS*. In particular, note that no results are shown for *WS* in solving *RockSample[4,4]*. For this problem, *WS* runs so slowly compared to the other methods that it does not finish in a reasonable amount of time. It is still struggling to finish iteration two of value iteration after *IWS* and *New* have finished six iterations.

In fact, a large part of the reason for the improved performance of *New* relative to *WS* is the same as the reason for the improved performance of *IWS* relative to *WS*. Both *New* and *IWS* terminate as soon as they find a belief state b for which the vector w ϵ-dominates every vector u in $\mathcal{U}$, which proves that w is not ϵ-dominated. By contrast, *WS* continues to search for a belief state b for which the vector w maximizes d. As a result, it takes longer to converge, and solves larger linear programs.

It is worth noting that increasing the tolerance ϵ used for approximation further reduces the number of iterations required by *New* and *IWS* to converge, in addition to further reducing the size of the linear programs solved. By contrast, previous approaches to vector pruning do not allow the number of variables and constraints in linear programs to be decreased by increasing the tolerance ϵ.

Although *New* solves smaller linear programs than *IWS*, it has the drawback that it requires more iterations to converge, as is clear from the results in the table. It would help to have some way to reduce the number of extra iterations. With that in mind, we added a simple heuristic to the *New* pruning subroutine, and used it in solving these benchmark POMDPs. When a state $\hat{s}$ is added to $\mathcal{S}'$ in line 18 of Algorithm 6, a vector $\hat{u}$ is also added to $\mathcal{U}'$, which is selected as follows:

$$\hat{u} \leftarrow \operatorname*{argmin}_{u \in \mathcal{U} \setminus \mathcal{U}'} \left(w(\hat{s}) - u(\hat{s}) \right). \tag{21}$$

The intent of this heuristic is to further reduce the number of iterations of the algorithm, and it often has this effect, although the effect is modest and problem-dependent.

Table 2: For each POMDP and pruning algorithm, the table shows the average and maximum number of nonzero and total variables and constraints in the solution of the final linear program solved to test a vector for dominance, the average and maximum number of linear programs solved to test a vector for dominance, and the running time in CPU seconds, for the first six iterations of value iteration using incremental pruning. For each POMDP, it also gives the ϵ tolerance, and the size of the state and vector sets, denoted $|\mathcal{S}|$ and $|\mathcal{V}|$, respectively.

			Variables				**Constraints**					
		Time	**Non-0**		**Total**		**Non-0**		**Total**		**LPs**	
Problem	**Alg.**	*sec*	*avg*	*max*	*avg*	*max*	*avg*	*max*	*avg*	*max*	*avg*	*max*
Hallway2, $\epsilon = 0.012$	WS	9,948	9	16	92	92	9	17	24	57	24	57
$\|\mathcal{V}\|_{avg} = 598, \|\mathcal{S}\| = 92$	IWS	5,135	8	15	92	92	8	15	21	54	21	54
$\|\mathcal{V}\|_{max} = 37,211$	New	2,898	7	14	9	17	7	14	28	80	27	79
RockSample[4,4], $\epsilon = 1.5$	WS	-	-	-	-	-	-	-	-	-	-	-
$\|\mathcal{V}\|_{avg} = 5,583, \|S\| = 257$	IWS	5,581	4	13	257	257	4	13	8	31	8	31
$\|\mathcal{V}\|_{max} = 218,448$	New	3,548	4	12	6	21	4	12	12	53	11	52
CIT, $\epsilon = 0.05$	WS	37,414	12	25	284	284	12	25	27	65	27	65
$\|\mathcal{V}\|_{avg} = 1,476, \|\mathcal{S}\| = 284$	IWS	29,664	12	25	284	284	12	25	24	66	24	66
$\|\mathcal{V}\|_{max} = 22,352$	New	12,119	9	22	11	27	9	22	34	92	33	91

5.3 POTENTIAL IMPROVEMENTS

Given how much smaller the linear programs solved by *New* are compared the the size of the linear programs solved by the other methods, the timing results reported above still seem to leave room for further improvement.

Roijers et al. (2018) describe an optimization of the Walraven and Spaan (*WS*) algorithm for vector pruning in which constraint sets from previous iterations of value iteration are used to *bootstrap* constraint generation in the current iteration. It seems likely that this approach could be generalized to allow bootstrapping of variable generation as well.

For pruning small vector sets, incremental constraint generation is not needed or helpful, and Walraven's implementation of *WS* includes a test for the size of the vector set, and only uses incremental constraint generation to prune large vector sets. Similarly, incremental variable elimination is not needed or helpful for small-dimensional vector sets, and should only be used for problems with a relatively large state set.

Perhaps the most significant drawback of the additional iterations required by incremental variable and constraint generation is the time overhead of repeated calls to an external linear program solver, especially for very small linear programs. It seems likely that performance could be improved substantially by implementing a special-purpose linear program solver that leverages the structure of this problem – the fact that a solution often has small dimension, with the same number of non-zero variables as constraints – to accelerate the search for a solution, and limit the overhead of calls to an external solver.

6 CONCLUSION

Our experimental results show that the approach introduced in this paper outperforms the previous state-of-the-art pruning subroutine for vector sets, especially for problems with larger state sets. By incrementally generating variables and constraints, the size of the linear programs that must be solved in order to prune a vector set can be substantially reduced, which in turn improves the performance of exact value iteration for POMDPs.

The pruning subroutine considered in this paper is also used by exact algorithms for decentralized POMDPs (Hansen et al., 2004), by algorithms for multi-objective MDPs (Roijers et al., 2018), and by algorithms for related high-dimensional convex hull problems (Zhang, 2010; Dula and Helgason, 1996), and so the technique is very general. In addition, other exact algorithms for POMDPs solve similar linear programs that limit scalability, including the Witness algorithm (Kaelbling et al., 1998) and bounded policy iteration (Poupart and Boutilier, 2004; Bernstein et al., 2005). Hansen (2008) shows how to improve the performance of bounded policy iteration by incrementally generating the variables in the linear programs it solves, and incrementally generating both variables and constraints, as described in this paper, may improve performance further.

Acknowledgements

Partial support for this research was provided by National Science Foundation Award IIS:RI #1718384.

References

Bernstein, D., Hansen, E., and Zilberstein, S. (2005). Bounded policy iteration for decentralized POMDPs. In *Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 52–57.

Cassandra, A. (1998). *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University.

Cassandra, A., Littman, M., and Zhang, N. (1997). Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proc. of the 13th Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 54–61.

Dula, J. and Helgason, R. (1996). A new procedure for identifying the frame of the convex hull of a finite collection of points in multidimensional space. *European Journal of Operational Research*, 92:352–367.

Dutech, A. and Scherrer, B. (2013). Partially observable Markov decision processes. In *Markov Decision Processes in Artificial Intelligence*, chapter 7, pages 185–228. John Wiley & Sons, Ltd.

Hansen, E. (2008). Sparse stochastic finite-state controllers for POMDPs. In *Proc. of the 24th Conference on Uncertainty in Artificial Intelligence (UAI-08)*, pages 256–263. AUAI Press.

Hansen, E., Bernstein, D., and Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *Proc. of the 19th National Conference on Artificial Intelligence (AAAI-04)*, pages 709–715.

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

Littman, M., Cassandra, A., and Kaelbling, L. (1995). Efficient dynamic-programming updates in partially observable Markov decision processes. Technical Report CS-95-19, Brown University.

Poupart, P. and Boutilier, C. (2004). Bounded finite state controllers. In *Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference*, Vancouver, Canada. MIT Press.

Roijers, D., Walraven, E., and Spaan, M. (2018). Bootstrapping LPs in value iteration for multi-objective and partially observable MDPs. In *Proc. of the 28th International Conference on Automated Planning and Scheduling*, pages 218–226. The AAAI Press.

Smallwood, R. and Sondik, E. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088.

Smith, T. and Simmons, R. (2004). Heuristic search value iteration for POMDPs. In *Proc. of the 20th Conference on Uncertainty in Artificial Intelligence (UAI-04)*, pages 520–527.

Spaan, M. (2012). Partially observable Markov decision processes. In Wiering, M. and van Otterlo, M., editors, *Reinforcement Learning: State of the Art*, pages 387–414. Springer Verlag.

Walraven, E. (2019). *Planning under Uncertainty in Constrained and Partially Observable Environments*. PhD thesis, Delft University of Technology, Netherlands.

Walraven, E. and Spaan, M. (2017). Accelerated vector pruning for optimal POMDP solvers. In *Proc. of the 31st Conference on Artificial Intelligence (AAAI 2017)*, pages 3672–3678.

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

Zhang, H. (2010). Partially observable Markov decision processes: A geometric technique and analysis. *Operations Research*, 58(1):214–228.