Annals of Mathematics and Artificial Intelligence manuscript No.
(will be inserted by the editor)

The Skyline Algorithm for POMDP Value Function Pruning

Christopher Raphael · Guy Shani

Received: date / Accepted: date

Abstract We address the *pruning* or *filtering* problem, encountered in exact value iteration in POMDPs and elsewhere, in which a collection of linear functions is reduced to the minimal subset retaining the same maximal surface. We introduce the Skyline algorithm, which traces the graph corresponding to the maximal surface. The algorithm has both a complete and an iterative version, which we present, along with the classical Lark's algorithm, in terms of the basic dictionary-based simplex iteration from linear programming. We discuss computational complexity results, and present comparative experiments on both randomly-generated and well-known POMDP benchmarks.

Keywords POMDP · Linear Programming · Dynamic Programming

1 Introduction

Many autonomous agents operate in an environment where actions have stochastic effects. In many such cases, the agent perceives the environment through noisy and partial observations. Perhaps the most common example of this setting is a robot that receives input through an array of sensors [9, 8]. These sensors can provide only partial information about the environment. For example, robotic sensors such as cameras and lasers cannot see beyond walls, and the robot thus cannot directly observe the contents of the next room. Thus, many features of the problem, such as the existence of hazards or required resources beyond the range of the sensors are hidden from the robot. Other examples of applications where partial observability is prevalent are

Christopher Raphael
School of Informatics, Indiana University, Bloomington, IN
E-mail: craphael@indiana.edu

Guy Shani
Information Systems Engineering, Ben Gurion University, Beer Sheva, Israel
E-mail: shanigu@bgu.ac.il

dialog systems [20], preference elicitation tasks [4], automated fault recovery [15], medical diagnosis [7], recommender systems [14], and many more.

For such applications, the decision-theoretic model of choice is a partially observable Markov decision process (POMDP). POMDPs provide a principled mathematical framework to reason about the effects of actions and observations on the agent's perception of the environment, and to compute behaviors that optimize some aspect of the agent's interaction with the environment.

A POMDP can be solved by casting it into the belief-space MDP — a Markov Decision Process (MDP) defined over the belief states of the original POMDP. The value function for this MDP is convex [18, 16] and can be represented as a maximum of linear functions, often called α-vectors. The traditional MDP value iteration algorithm [1] can be translated to the belief space MDP through operations over sets of α-vectors. As the computation time of each iteration depends directly on the size of the set, it is crucial to maintain minimal sets of α-vectors. Indeed, as the value function is defined using the upper envelope of this set, many vectors may be completely dominated and hence redundant. The task of removing these dominated vectors is typically known as pruning [2].

The traditional method for removing redundant vectors is by using Lark's algorithm [19], that, given a set of α-vectors, attempts to find for each vector a witness point — a belief state where this α-vector dominates all others [2]. Finding the witness can be defined as a linear program (LP) and solved by any LP solver. Lark's algorithm treats LP as a black box, independently solving a collection of interconnected LP problems ignoring their relationship.

In this paper we propose, instead, an alternative to the popular approach that we call the Skyline algorithm. Skyline traverses the *skyline graph* formed by a set of vectors, as shown in Figure 1. The vertices of this graph are points, $x \in S$, where D of the linear functions are simultaneously equal and maximal, while the edges are line segments in S where $D-1$ of the linear function are equal and maximal. We provide first an intuitive, geometric description of the Skyline algorithm, followed by a more practical implementation drawing on ideas from LP.

A complete traversal of the skyline graph may by computationally infeasible, due to the potentially large number of vertices. We suggest an iterative variant of the Skyline algorithm that determines the fate of each vector by traversing only a part of the skyline graph. This modification has the benefit of uncovering additional dominating vectors as it explores the vector under consideration.

We present simple complexity analysis of the algorithms in two dimensions. We present experiments on some artificially constructed examples, as well as over a number of standard benchmarks from the POMDP literature. These experiments demonstrate both the merits and disadvantages of the Skyline algorithm and its iterative sibling. Finally we outline some ways in which Skyline can take advantage of the special structure of POMDP problems.

2 Background

We begin with an overview of MDPs and POMDPs, the belief space MDP, and value iteration for solving POMDPs. We then focus on Lark's algorithm for pruning dominated vectors.

2.1 MDPs, POMDPs and the belief-space MDP

Markov Decision Processes (MDPs) are designed to model autonomous agents, acting in a stochastic environment. Consider for example a robot traveling through a maze. The robot starts at some location and can either move forward, turn left, or turn right. As the robot moves its location may change, and thus, the environment, which includes the location of the robot, changes. The assumption is that the environment changes only as the result of the agent actions. The robot must reach some goal state, such as the exit door, or alternatively, collect rewards, such as items that are scattered through the maze.

Formally, an MDP is a tuple $\langle S, A, tr, R \rangle$ where:

- S is the set of all possible world states. In the example above the environment state is the location and orientation of the robot.
- A is a set of actions the agent can execute. Our robot can only turn left, right, or move forward.
- $tr(s, a, s')$ defines the probability of transitioning from state s to state s' using action a. The transition function models the stochastic nature of the environment, such as the robot attempting to move forward but failing due to engine malfunction or because the wheels were slipping.
- $R(s, a)$ defines a reward the agent receives for executing action a in state s. Action costs can be modeled as negative rewards. In our example the robot receives a reward for getting out of the maze or for collecting an item. The robot may pay a cost each time it moves, modeling the energy loss incurred by the move.

An MDP models an agent acting in an environment where it can directly observe the state it is at.

Realistically, a robot does not know where it is located within a maze. It has sensors that provide observations such as nearby walls. These sensors are imperfect, meaning that they sometimes detect a wall where none exist, and sometimes the sensors fail to detect an existing wall. Now, in order to find its way through the maze the robot must also gather information about the environment state — its own location within the maze.

A Partially Observable Markov Decision Process (POMDP) is designed to model such agents that do not have direct access to the current state, but rather observe it through noisy sensors. A POMDP is a tuple $\langle S, A, tr, R, \Omega, O, x_0 \rangle$ where:

- S, A, tr, R compose an MDP, known as the underlying MDP. This MDP models the behavior of the environment.
- Ω is a set of available observations — the possible outputs of the sensors. In the example above the set of observations consists of all possible wall configurations.
- $O(a, s, o)$ is the probability of observing o after executing a and reaching state s, modeling the sensor noise.

As the agent is unaware of its true world state, it must maintain a *belief* over its current state — a vector x of probabilities such that $x(s)$ is the probability that the agent is at state s. Such a vector is known as a belief state or *belief point*. x_0 defines the initial belief state — the belief of the agent over the state space before it has executed or observed anything.

Given a POMDP it is possible to define the belief-space MDP — an MDP over the belief states of the POMDP. The transition from belief state x to belief state x' using action a is deterministic given an observation o and defines the τ transition function. That is, we denote $x' = \tau(x, a, o)$ where:

$$x'(s') = \frac{O(a, s', o) \sum_s x(s) tr(s, a, s')}{pr(o|x, a)} \quad (1)$$

and

$$pr(o|x, a) = \sum_s x(s) \sum_{s'} tr(s, a, s') O(a, s', o). \quad (2)$$

As such, τ is computed in $O(|S|^2)$ and all the successors of a belief state are computed in $O(|\Omega||A||S|^2)$.

2.2 Value Functions for POMDPs

An agent that uses an MDP or POMDP attempts to optimize some function of its reward stream, such as the sum of rewards, the average reward or, more often, the discounted reward $\sum_i \gamma^i r_i$. The discount factor $0 < \gamma < 1$ models the higher value of present rewards compared to future rewards.

In most cases a solution to an MDP or POMDP is presented as a stationary policy $\pi : S \rightarrow A$ — a mapping from states (MDP) or belief states (POMDP) to actions. Policies can be computed directly [6] or through a value function that assigns a value to each state [1]. For the discounted, infinite horizon case that we discuss here there is a single optimal value function [1] that corresponds to the fixed point of the Bellman equation:

$$V(s) = \max_{a \in A} R(s, a) + \gamma \sum_{s' \in S} tr(s, a, s') V(s'). \quad (3)$$

The well known value iteration algorithm finds this fixed point by initializing a value function and then applying the Bellman update until the function

reaches its fixed point. An optimal policy, a policy that maximizes the stream of rewards, can be obtained using:

$$\pi_V(s) = argmax_{a \in A} R(s,a) + \gamma \sum_{s' \in S} tr(s,a,s')V(s'). \quad (4)$$

As $R(x,a)$, the immediate reward function for the belief space MDP, is linear and the Bellman equation preserve convexity, the value function V for the belief-space MDP can be approximated as a finite collection of $|S|$-dimensional vectors known as α vectors, and is both piecewise linear and convex [18]. A policy over the belief space can be defined by associating an action a to each vector α, such that $\alpha \cdot x = \sum_s \alpha(s)x(s)$ represents the value of taking a in belief state x and following the policy afterwards. It is therefore standard practice to compute a value function — a set V of α vectors, inducing a policy π_V by:

$$\pi_V(x) = argmax_{a:\alpha_a \in V} \alpha_a \cdot x. \quad (5)$$

2.3 Value Iteration in Vector Space

The value iteration algorithm over the belief-space MDP can be rewritten in terms of vector operations, and operations on sets of vectors [18, 2]:

$$V' = \bigcup_{a \in A} V^a \quad (6)$$

$$V^a = \bigoplus_{o \in \Omega} V^{a,o} \quad (7)$$

$$V^{a,o} = \{\frac{1}{|\Omega|} r_a + \alpha^{a,o} : \alpha \in V\} \quad (8)$$

$$\alpha^{a,o}(s) = \sum_{s' \in S} O(a,s',o)T(s,a,s')\alpha(s') \quad (9)$$

where $r_a(s) = R(s,a)$ is a vector view of the reward function, V is the vector set prior to the backup, V' is the new vector set after the backup, and $V_1 \oplus V_2 = \{\alpha_1 + \alpha_2 | \alpha_1 \in V_1, \alpha_2 \in V_2\}$.

This process is known as *exact value iteration*. In each iteration, the value function is updated across the entire belief space. There are $|V| \times |A| \times |\Omega|$ vectors generated at Equation 9, and computing each of these vectors takes $|S|^2$ operations. In Equation 7 we create $|V|^{|\Omega|}$ new vectors for each action, with a complexity of $|S|$ for each new vector. Hence, the overall complexity of a single iteration is $O(|V| \times |A| \times |\Omega| \times |S|^2 + |A| \times |S| \times |V|^{|\Omega|})$.

The set of α-vectors may grow exponentially with every iteration. As the computational cost of each iteration depends on the number of vectors in V, an exponential growth makes the algorithm prohibitively expensive. To some degree, the sets of α-vectors can be reduced to their minimal form after each stage, resulting in more manageable value functions [2].

2.4 Lark's Algorithm

The classic filtering approach, due to Lark [19], seeks, for each vector $\alpha_{j'}$, a point, $x' \in S$, such that $\alpha_{j'} \cdot x' > \alpha_j \cdot x'$ for $j \neq j'$. Such a point shows that $\alpha_{j'}$ is optimal at x', hence non-dominated. One can determine if such a point exists by executing the linear program, $\text{LP}(j', M)$, defined in terms of the variables $x_1, \ldots, x_D, \delta$, where $M = \{1 \ldots, N\} \setminus \{j'\}$. This LP is defined by minimizing δ subject to the constraints

$$(\alpha_{j'} - \alpha_j) \cdot x + \delta > 0 \text{ for } j \in M \tag{10}$$

with $\sum_i x_i = 1$ and $x_i \geq 0$. $\alpha_{j'}$ is dominated if and only if the optimizing value, δ^*, satisfies $\delta^* \geq 0$. The most straightforward use of the above linear program runs $\text{LP}(j', \{1, \ldots, N\} \setminus \{j'\})$ independently for each $j' \in \{1, \ldots, N\}$.

Lark's algorithm is a variation on this basic idea that may run faster. The algorithm manages two sets of vectors — a "clean" set, indexing the currently non-dominated vectors, initialized to index the maximizers at the "corners" of the simplex; and a "dirty" set, indexing the vectors whose fate is currently unknown, and is initialized to the remaining collection of vectors. This process is described in Algorithm 1.

Algorithm 1 Lark's filtering algorithm

```
F ← {1, ..., N}            // the dirty set
Q ← ∅                      // the clean set
for i ∈ 1, ..., D do
  j(i) ← arg max_{j=1,...,N} α_{ji}
  Add j(i) to Q and remove it from F.
end for
while F is not empty do
  choose j' ∈ F
  (δ, x) ← LP(j', Q)
  if δ < 0 then
    k ← arg max_{j∈F} α_j · x
    Add α_k to C and remove it from F
  else
    Remove j' from F
  end if
end while
```

The advantage of Lark is that, by comparing each undetermined vector to the clean set only, we reduce the size of the linear programs to be solved.

3 The Skyline Algorithm

We now describe an alternative to Lark's algorithm — the Skyline algorithm. Skyline traces the upper envelope (the skyline) formed by the set of vectors. All vectors "visited" during this traversal are *non-dominated*, while vectors that

are never visited can be pruned. Below we describe formally the skyline algorithm, starting first with an intuitive geometric description, then presenting a more practical implementation relying on concepts from LP.

3.1 A Geometric View

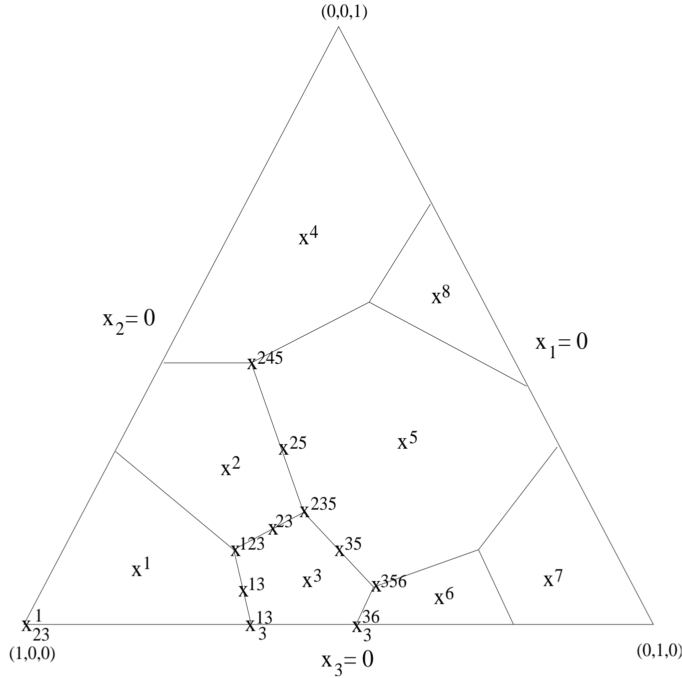

Fig. 1 The Skyline Graph. x^O is the set where the linear functions indexed by O are simultaneously maximal. For the point x_I^O, the functions of O are simultaneously maximal, while the coordinate variables of I are 0.

Our geometric description is easiest to visualize with $D = 3$. In this case the simplex domain, S, can be viewed as the interior of a triangle as indicated in Figure 1. One can imagine the graph of the function of Eqn. 5, describing the "upper envelope" over the simplex S. The Skyline algorithm identifies the minimal index set, Q, by *tracing out* the complete collection of edges lying on the maximal surface, as follows. We will ignore, for now, the many possible kinds of degeneracies that may exist, in favor of simple exposition.

Suppose we begin with a *maximal vertex* $x^O \in S$ where $O \subset \{1, \ldots, N\}$ with $|O| = D$, meaning that x^O satisfies

$$\alpha_j \cdot x^O = \alpha_k \cdot x^O \quad j, k \in O \tag{11}$$

$$\alpha_j \cdot x^O > \alpha_k \cdot x^O \quad j \in O, k \notin O \tag{12}$$

We will return to the construction of such a vertex later. For instance, in Figure 1, $x = x^{235}$ is the point where $\alpha_2 \cdot x = \alpha_3 \cdot x = \alpha_5 \cdot x > \alpha_j \cdot x$ for $j \notin \{2, 3, 5\}$.

Let us remove one of the indices from O, say j'. Equating the remaining $D-1$ linear functions (subject to $\sum_i x_i = 1$) we obtain an under-determined system of linear equations. The solution to this system will be a line containing x^O. There are two rays emanating from x^O that together compose this line, and along one of them the functions of $O - j'$ are simultaneously maximal:

$$\alpha_j \cdot x = \alpha_k \cdot x \quad j, k \in O - j' \tag{13}$$

$$\alpha_j \cdot x > \alpha_k \cdot x \quad j \in O - j', k \notin O - j' \tag{14}$$

For instance, if we drop 2 from $O = \{2, 3, 5\}$ in Figure 1, there is a line segment labeled x^{35} on which $\alpha_3 \cdot x = \alpha_5 \cdot x > \alpha_j \cdot x$ for $j \notin \{3, 5\}$.

Suppose we move along this ray until we find the first point where a new function, indexed by $k' \notin O$, is equal to the functions indexed by $O - j'$. Now if we define $\bar{O} = O + k' - j'$ and let $x^{\bar{O}}$ be the new point, we see that $x^{\bar{O}}$ is also a maximal vertex. In Figure 1 we have moved from the vertex x^{235} along the edge x^{35} until we arrive at vertex x^{356}. Since each of the D elements of O can be dropped, there are D edges connected to our initial vertex, x^O. The Skyline algorithm will explore all edges out of each vertex encountered through recursive application of this process.

As we explore an edge leading out of a vertex, x^O, by dropping the constraint indexed by $j' \in O$, it is possible that one of the variables, say $x_{i'}$, decreases to 0 before we encounter another linear function. In this case we let $\bar{O} = O - j'$ and $\bar{I} = \{i'\}$ and define the vertex $x_{\bar{I}}^{\bar{O}}$ as the point along this edge where $x_{i'} = 0$. This happens in Figure 1 as we explore the edge x^{13}, out of vertex x^{123} leading to the new vertex x_3^{13}.

More generally, a vertex of our algorithm, x_I^O, where $|I| + |O| = D$ is defined by the intersection of $|O|$ linear functions ($|O| - 1$ constraints), $|I|$ constraints $x_i = 0$ for $i \in I$, and the simplex constraint $\sum_i x_i = 1$, making a total of D constraints. Each iteration of our algorithm will drop an element from either O or I — if $|O| > 1$ there will be D such choices each giving an edge emanating from x_I^O. If $|O| = 1$ we only consider dropping the coordinate constraints. We follow the edge until a new linear function is encountered or a new coordinate decreases to 0. Our new vertex is given by $x_{\bar{I}}^{\bar{O}}$ where $\bar{I}$ and $\bar{O}$ account for both the constraint that was dropped as well as the one that was added.

We now return to the initialization of the algorithm. We begin at the i'th "corner" of the simplex having i'th coordinate equal to 1 while all others are 0, and suppose that $\alpha_{j'}$ is the maximizing function at the corner: $\alpha_{j'} \cdot x > \alpha_j \cdot x$ for $j \neq j'$. In this case, our initial configuration has $I = \{i : i \neq i'\}$ and $O = \{j'\}$.

The algorithm then proceeds to follow all edges out of each new vertex encountered, until no unexplored vertices exist. At this point the minimal index set, Q, will be given by $Q = \bigcup_v O_v$ over all vertices, v, discovered by

the algorithm, where O_v is the set of maximal linear functions associated with vertex v.

3.2 Linear Programming Implementation

Section 3.1 provides an intuitive view of the way that the Skyline algorithm operates, but this view is not well-suited for actual implementation. Here we provide a different description of the same algorithm using ideas from Linear Programming (LP) [3], better suited for computer implementation.

We begin by writing a system of equations subject to constraints in which certain solutions correspond to vertices of our Skyline graph. To do this we introduce *slack* variables, x_{D+j}, for $j = 1, \ldots, N$ which represent the difference between the jth function and the maximal function at $x = (x_1, \ldots, x_D)$:

$$\sum_{i=1}^{D} \alpha_{ji} x_i + x_{D+j} = \sum_{i=1}^{D} \alpha_{j'i} x_i + x_{D+j'} \tag{15}$$

for $j, j' = 1, \ldots, N$, subject to $\sum_{i=1}^{D} x_i = 1$ and $x_j \geq 0$ for $j = 1, \ldots, D + N$. We will refer to the variables $x_1, \ldots, x_D$ as *coordinate* variables.

Suppose we find a solution to all Eqns. 15, subject to our constraints, such that $x_{D+j'} = 0$ for some j'. Then the j'th linear function, $\alpha_{j'} x$, must be maximal at $x = (x_1, \ldots, x_D)$ since $x_{D+j} \geq 0$ for $j \neq j'$. That is, if $x_1, \ldots, x_{D+N}$ solves Eqns. 15 subject to the constraints, then

$$x_{D+j'} = 0 \Longleftrightarrow \sum_{i=1}^{D} \alpha_{j'i} x_i \geq \sum_{i=1}^{D} \alpha_{ji} x_i \tag{16}$$

for $j = 1 \ldots, N$.

Eqns. 15 are redundant if we consider all possible combinations of j and j' — at most $N - 1$ of these equations can be independent. Suppose we take $N - 1$ equations by fixing j' and letting j range, $j \neq j'$. If we augment these equations with the constraint $\sum_{i=1}^{D} x_i = 1$, we have a system with N equations and $N + D$ unknowns. Such a system must be under-determined. Barring degeneracies, we can solve for any N variables in terms of the remaining D. Our LP implementation will choose various partitions of our variable index set $\{1, \ldots, D + N\} = R \cup L$ where $R = \{r_1, \ldots, r_D\}$ (right-hand-side variables) and $L = \{l_1, \ldots, l_N\}$ (left-hand-side variables) and solve for the N variables indexed by L, x_L, in terms of the D variables indexed by R, x_R. The result is a system of the form

$$\begin{aligned} x_{l_1} &= c_1 + \textstyle\sum_{i=1}^{D} \beta_{1i} x_{r_i} \\ x_{l_2} &= c_2 + \textstyle\sum_{i=1}^{D} \beta_{2i} x_{r_i} \\ \vdots \quad & \vdots \ \vdots \\ x_{l_N} &= c_N + \textstyle\sum_{i=1}^{D} \beta_{Ni} x_{r_i} \end{aligned} \tag{17}$$

subject to $x_j \geq 0$ for $j = 1, \ldots, D + N$.

Such a system has an interesting connection to our geometric view of the Skyline algorithm. Suppose $x_R = 0$ leads to an admissible solution — that is, if we set the variables indexed by R to 0, then $x_{l_j} \geq 0$ for $j = 1, \ldots, N$. Consider then the point $x = (x_1, \ldots, x_D)$, noting that the coordinate variables $\{x_1, \ldots, x_D\}$ can appear on both sides of Eqn. 17. Let I be the set of indices of the coordinate variables in R, and O the indices of the linear functions corresponding to the slack variables in R, then $x = (x_1, \ldots, x_D) = b_I^O$ from the previous section. That is, x is the vertex in our Skyline graph where the linear functions of O are simultaneously optimal and the coordinate variables of I are 0.

The linear programming version of our Skyline algorithm will explore all systems of the form of Eqn. 17 in which $x_R = 0$ gives an admissible solution.

Algorithm 2 Init the matrix at $(1, 0, ..., 0)$.

Let $j' = \arg\max_j \alpha_{j1}$
for $j \neq j'$ **do**
Add to M:

$$x_{D+j} = (\alpha_{j'1} - \alpha_{j1}) + x_{D+j'} + \sum_{i=2}^{D} ((\alpha_{j'i} - \alpha_{ji}) - (\alpha_{j'1} - \alpha_{j1}))x_i$$

end for
Add to M: $x_1 = 1 - \sum_{i=2}^{D} x_i$
Let L(M) $= \{x_1, x_{D+1}, x_{D+j'-1}, x_{D+j'+1}, \ldots, x_N\}$
Let R(M) $= \{x_2, \ldots, x_D, x_{D+j'}\}$

First, we must construct an initial admissible system of the form of Eqn. 17 (Algorithm 2). Suppose we begin with Eqn. 15 and consider the i'th "corner" point where $x_i = 1$, and suppose that $\alpha_{j'} x$ is maximal at this point. As a consequence we must have $x_{D+j'} = 0$ from Eqn. 16. Thus there are D variables that are equal to 0, indexed by

$$R = \{r_1, \ldots, r_D\} = \{1 \ldots, i' - 1, i' + 1, \ldots, D, D + j'\}. \tag{18}$$

To write the variables of $L = R^c$ in terms of the variables of R, we first note that

$$x_{i'} = 1 - \sum_{i=1, i \neq i'}^{D} x_i. \tag{19}$$

For the remaining variables of L, we solve Eqn. 15 for x_{D+j}, $j \neq j'$, and substitute Eqn. 19 to remove the dependence on $x_{i'}$. The result expresses x_L in terms of x_R as in Eqn. 17. At $x_R = 0$ we see from Eqn. 15 that, since the j'th equation is maximal, we must have $x_{D+j} \geq 0$ for $j \neq j'$. Finally, Eqn. 19 shows that the remaining variable of L, $x_{i'}$, is also nonnegative at $x_R = 0$, so we have an admissible initial system.

From our initial representation of the form of Eqn. 17 with admissible solution given by $x_R = 0$, we can easily find others using the basic iteration of the simplex method [3]. To do this, we choose an R-variable, $x_{i'}$, and consider increasing its value from 0. As we do this some of the L-variables will decrease[1]. Suppose that $x_{j'}$ is the first L-variable to decrease to 0. From Eqns. 17 the equation for $x_{j'}$ can be written

$$x_{j'} = c_{j'} + \sum_{i=1, r_i \neq i'}^{D} \beta_{j'i} x_{r_i} + \beta_{j'i'} x_{i'}. \tag{20}$$

We solve this equation for $x_{i'}$ giving

$$x_{i'} = \frac{c_{j'}}{-\beta_{j'i'}} + \sum_{i=1, r_i \neq i'}^{D} \frac{\beta_{j'i}}{-\beta_{j'i'}} x_{r_i} + \frac{1}{\beta_{j'i'}} x_{j'} \tag{21}$$

thus writing $x_{i'}$ in terms of the variables of $R' = R \cup \{j'\} \setminus \{i'\}$. We drop Eqn. 20 from the original system of Eqns. 17 replacing it with Eqn. 21. Then we substitute Eqn. 21 for $x_{i'}$ in the original system to get an equivalent system written in terms of the variables of R'. The way the system was constructed — by increasing an R variable until the first L variable decreases to 0 — guarantees the new system is admissible at $x_{R'} = 0$. Thus we have moved to a new vertex of our Skyline graph (Algorithm 3).

Algorithm 3 Moving $x_{i'}$ from $\mathrm{R}(M)$ to $\mathrm{L}(M)$: $\mathrm{MOVE}(M, x_{i'})$

$M' \leftarrow \phi$

$j' = \arg\min_{j: x_j \in \mathrm{L}(M)} -\frac{c_j^M}{\beta_{ji'}^M}$

Add to M':

$$x_{i'} = -\frac{c_{j'}^M}{\beta_{j'i'}^M} + \frac{1}{\beta_{j'i'}^M} x_{j'} + \sum_{i: x_i \in \mathrm{R}(M) - \{x_{i'}\}} \frac{-\beta_{j'i}^M}{\beta_{j'i'}^M} x_i$$

for $j : x_j \in L(M) - \{x_{j'}\}$ **do**
 Rewrite equation for x_j substituting for $x_{i'}$ as above and add to M'
end for
Let $\mathrm{L}(M') = \mathrm{L}(M) \cup \{x_{i'}\} - \{x_{j'}\}$
Let $\mathrm{R}(M') = \mathrm{R}(M) \cup \{x_{j'}\} - \{x_{i'}\}$

The iteration described above is the basic iteration of the simplex algorithm. However, the simplex algorithm tries to optimize an objective function, and, in doing so, swaps a left-hand side variable with a right-hand side variable for each system of the form of Eqn. 17 where the swap variables are chosen to increase the objective function at $x_{R'} = 0$. In contrast, the Skyline algorithm seeks to explore *all* edges on the maximal surface, thus, in principle, it traces

[1] unless we are at a corner point and $x_{i'}$ is the slack variable for the maximal function at the corner

all edges out of each vertex. The Skyline algorithm has no objective function. Thus, given a system of the form of Eqn. 17, admissible for $x_R = 0$ (a vertex), we allow each right-hand side variable to increase, thus following each edge out of the vertex.

Our Skyline algorithm is described in pseudo-code in Algorithm 4. Upon termination, V' contains the minimal set of α-vectors.

Algorithm 4 Traversing the skyline.

```
Init M_0 to the feasible solution at (1, 0, ..., 0)
Init L to the empty list
Init V' ← φ
Add M_0 to L
while L is not empty do
  Let M be the first element of L
  for x_i ∈ R(M) : i > n do
    If α_{i−D} ∉ V' add α_{i−D} to V'
  end for
  for each variable x_i ∈ R(M) do
    M' ←MOVE(M, x_i) — move x_i to L, creating M'
    if M' was not observed then
      Add M' to L
    end if
  end for
end while
```

4 Complexity Analysis

The version of Skyline presented in Algorithm 4 traverses each edge of the skyline graph. A more efficient implementation would "hash" all of the outgoing edges of an already-visited vertex. This way we can avoid traversing an edge (performing a simplex iteration) that leads to an already-visited vertex. Thus, the number of vertices, V, in the skyline graph is the number of simplex iterations that must be performed. Since the time complexity of the simplex iteration (Algorithm 3) is $O(ND)$, the total time complexity of the Skyline algorithm is $O(VND)$.

Since LP forms the "guts" of Lark's algorithm, Lark can be similarly cast in terms of simplex iterations. The LP(j', M) program of Lark would be implemented by introducing slack variables, x_{D+j}, for $j \in M$ and writing Eqn. 10 as: Minimize δ subject to

$$\sum_{i=1}^{D} \alpha_{j'i} x_i + \delta = \sum_{i=1}^{D} \alpha_{ji} x_i + x_{D+j} \text{ for } j \in M. \tag{22}$$

$\sum_i x_i = 1$ and $\delta, x_i, x_{D+j} \geq 0$ for $j \in M$, $1 \leq i \leq D$. Note the close similarity with Eqn. 15 which was the basis for Skyline. The most straightforward simplex approach would proceed exactly as we have done with Skyline. That is, we

rewrite Eqn. 22 as a system of $M+1$ equations in which the D right-hand-side variables are 0 by evaluating at a corner point, as in Section 3.2. From there we run simplex iterations that decrease the left-hand-side variable, δ, until δ becomes negative or can be no longer decreased. Thus, Lark's algorithm can be seen to trace a portion of the skyline graph defined by the vectors of M, until either vector $\alpha_{j'}$ touches the sub-skyline (δ decreases to 0), or is shown to be dominated (all right-hand-side variables of the equation for δ have positive coefficients). While a generic use of LP would find an optimizing configuration for δ, there is no reason to continue the iterations after δ decreases to 0. This view of Lark's algorithm facilitates comparisons with Skyline. Thus, we implement Lark as described above for the experiments in Section 6.

For the two-variable case, the computational complexity of *both* algorithms is straightforward. Here, taking the simplex constraint into account, our domain, S, is one-dimensional. We consider first the case in which all of our N vectors are non-dominated — pictorially, this case could be visualized as a collection of lines whose upper envelope forms a bowl shape. In this case there are $N+1$ vertices in the skyline graph, so, regarding D as a constant, the total time complexity of Skyline ($O(VND)$) reduces to $O(N^2)$. On the other hand, Lark's algorithm requires that, after the initialization, we solve the LP problem from *scratch* for each of the remaining $N-2$ vectors. For the nth of these LP problems the "clean" set has $|M| = n+1$. Thus, assuming we choose randomly from the "dirty" set, the LP problem requires $\frac{n+2}{2}$ simplex iterations on average (because there are $n+2$ vertices of the current skyline graph, all equally likely to determine the fate of the nth vector). Thus the expected time complexity for Lark is $\sum_{n=1}^{N-2} \frac{n+2}{2}(n+1) \in O(N^3)$. Here the disadvantage of Lark is that it performs nearly the same calculation repeatedly.

The complexity is similar for the case in which we have N vectors and assume that some proportion of these, p, are dominated. For Skyline, we must traverse $N(1-p)$ skyline vertices, each requiring an $O(ND)$ simplex iteration, thus giving total time complexity again of $O(N^2)$. For Lark's algorithm, still assuming we choose randomly from the "dirty" set, we must solve $N-2$ LPs, in which the nth of these has $E(|M(n)|) = (1-p)(n+1)$. If we choose a non-dominated vector the expected number of simplex iterations is $|M(n)|/2$; however, if we choose a dominated vector it is unclear how many simplex iterations will be needed to determine this fact. Even if we *disregard* the contribution from the dominated vectors, we still get time of $\sum |M(n)|^2/2$ where the sum is over the non-dominated vectors. The result has expected time complexity of $O(N^3)$, as before.

The analysis above shows that for $D = 2$ Skyline beats Lark by an order of magnitude. However, as the dimension increases, the number of vertices can grow rather rapidly, bounded sharply [10] by:

$$\binom{N+1-\frac{D+2}{2}}{N+1-D} - D \tag{23}$$

which is exponential in the dimension, D. This analysis is consistent with our experiments in Section 6 which show better performance for Skyline in dimen-

sions 2 and 3, though showing exponential time complexity in the dimension, thus leading to worse performance for higher dimensions.

5 The Iterative Skyline

Algorithm 5 Iterative Skyline

$F \leftarrow \{1, \ldots, N\}$
$Q \leftarrow \emptyset$
while F is not empty **do**
  Let M be the system at $(1, 0, ..., 0)$ (Algo. 2)
  choose $j \in F$
  while $\exists x_i \in \mathrm{R}(M)$ s. t. $\mathrm{MOVE}(M, x_i)$ decreases x_{D+j} **do**
    $M \leftarrow \mathrm{MOVE}(M, x_i)$
    for $x_i \in \mathrm{R}(M) : i > D$ **do**
      Add $i - D$ to Q, remove $i - D$ from F
    end for
  end while
  Remove j from F.
end while

Lark's algorithm scales well to higher dimensions because the number of simplex iterations involved in the $\mathrm{LP}(j', M)$ program does not seem to increase much with dimension. As the complexity analysis shows, one suboptimal aspect of Lark is that it tends to repeat the same calculation over and over in the exploration of new vectors. Here we develop a hybrid algorithm called Iterative Skyline (IS) (Algorithm 5), designed to incorporate the best of both worlds.

IS iteratively considers each vector, $\alpha_{j'}$, exploring a portion of the complete skyline graph in an attempt to "bring $\alpha_{j'}$ to the surface." The simplex iterations that embody this exploration are very close to those performed by Lark, except that they are taken with reference to the *complete* set of vectors — Lark considers only the current set of "clean" vectors. As with Lark, IS traverses only a single edge out of each skyline vertex. Since the IS graph traversal for a single vector takes place on the complete skyline graph (using all vectors), we continue to visit and revisit the same skyline vertices. To avoid duplicating computations, we retain the linear systems associated with each skyline vertex. Thus, when a new vector is considered, we traverse the "already visited" edges for free, reserving the computation for the unexplored portion of the skyline graph. IS has an additional benefit: as we traverse the skyline graph seeking to learn the fate of the j'th vector, we may visit skyline vertices that uncover other non-dominated vectors.

For example, consider Figure 1 and suppose we begin by determining if vector 9 is somewhere-maximal. In performing our Iterative Skyline iterations, suppose we trace out a sequence of vertices that progressively decrease the slack associated with function 9: $x^1_{23}, x^{13}_3, x^{123}, x^{235}, x^{245}$. Suppose that

we find that the slack for function 9 is still positive at x^{245} and cannot be further decreased at a neighboring vertex. In this case we have determined that function 9 is dominated. However, we also have discovered the functions 1,2,3,4,5 are non-dominated. None of the functions need be investigated further. Pseudo-code for the Iterative Skyline algorithm is given as Algorithm 5.

$\|V\|$	Lark	Skyline	IS
Cheese $\|S\| = 11$			
21	6	72	5
Shuttle $\|S\| = 8$			
92	57	554	41
118	63	469	66
155	168	1583	139
Maze 4 $\|S\| = 11$			
100	204	29072	93
152	374	46077	244
195	639	98011	543
Tiger grid $\|S\| = 36$			
100	1639	-	863
162	4191	-	3051
262	11803	-	42278
Hallway $\|S\| = 62$			
55	1126	-	445
110	6031	-	2275
156	12080	-	3768
229	25786	-	10370
Hallway2 $\|S\| = 96$			
69	3577	-	3086
150	20644	-	17393
245	49035	-	52412
RockSample 4×4 $\|S\| = 256$			
49	3655	-	2077
106	21296	-	16577

Table 1 Pruning results for value functions of different sizes over POMDP benchmarks. Time is measured in milliseconds. Operations is the number of generated equations $\times 1000$. For each domain we report results over the largest value function and a few other value functions.

6 Empirical Results

We performed experiments with both randomly generated and standard benchmarks problems using the three algorithms: Lark, Skyline, and IS. In these experiments, we used our own implementation of Lark's algorithm, as described in Section 4, in order to put the algorithms on equal footing. There are well-known and more sophisticated implementations of the simplex algorithm than the dictionary-based implementation we have employed, though they would apply equally well to all three algorithms discussed here.

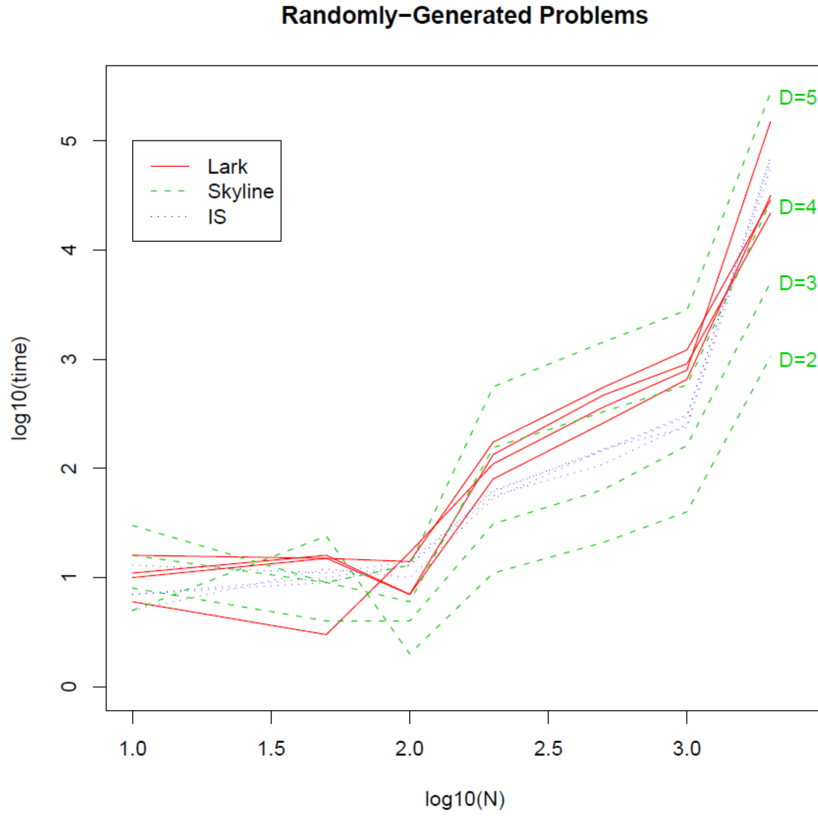

Fig. 2 Time of pruning calculation using Lark, Skyline, and IS for various values of N and D. The dimension is labeled for the various Skyline experiments.

For the randomly generated experiments we synthesized different sized collections of vectors with random coefficients before applying our pruning algorithms, with results, presented in Figure 2, showing log(msecs) plotted against $\log(N)$. The plot demonstrates a clear advantage for Skyline in dimensions 2 (as is consistent with our complexity analysis) and 3, though also suggests that, for fixed N, the running time of Skyline increases exponentially in the dimension. Thus the full-fledged Skyline algorithm is only practical for problems with low dimension. The comparison between Lark and IS here is less clear, perhaps showing less variability to IS than Lark with changing dimension, though not demonstrating a clear winner.

In an attempt to examine more realistic pruning scenarios, we also performed experiments on a collection of standard POMDP benchmarks, displayed in Table 1. In these examples we created representations of value func-

tions using the FSVI [12] point-based algorithm, stopping it after each iteration to prune the dominated vectors. We used the rapid point-wise dominance method after each vector insertion, as done by most POMDP solvers, still leaving us with a collection of vectors in which many may be dominated. We then ran our three algorithms on this resulting collection. This table shows slightly better results for IS than Lark on the majority of the benchmarks, though, perhaps no definitive advantage for either algorithm.

7 Conclusions and Future Research

In this paper we presented the Skyline algorithm which traverses the skyline graph formed by a set of linear functions in an attempt to find the subset of functions that touch the skyline. Through the notion of the skyline graph, our discussion sheds light on the close relations between Lark's algorithm and the two other skyline-related algorithms we propose. Our approach shows how to integrate simplex iterations directly into the larger computation, without treating LP like a "black box." While we have only demonstrated a computational advantage to our approach in problems of the lowest dimensions, we believe the foundation we have presented may lead to computational improvements with further effort. We outline here what we believe to be the most promising ideas and how they may be further exploited.

First, the Skyline algorithm provides a context, the skyline graph, in which one can "remember" past calculations that can streamline future calculations. For instance, if we have an explicit representation of a Skyline graph for a collection of linear functions, the process of adding a new linear function is simple. One would trace a path through the skyline graph following only edges that decrease the slack of the new function, as in Lark or the IS algorithm. The cost of tracing this path would be greatly reduced if we explicitly store the systems of equations at the graph vertices, since this is the computation performed at the vertex. This allows an iterative construction of the skyline graph. While it may be prohibitively expensive to iteratively construct the complete skyline graph, this approach also applies to the subsets of the skyline graph explored by the Lark's algorithm.

Second, the skyline graph gives information about which functions are maximal where, and how they intersect. This information may be exploited, for example, in rapidly merging two or more collections of linear functions, each with skyline graph representations, into a skyline graph for the union of functions. As this latter operation is central to exact value iteration in POMDPs [5], and the focus of the Incremental Pruning algorithm [21], [2], we hope Skyline may make a significant contribution here. The essential idea of the skyline "merging" algorithm is analogous to the merging of several sorted lists into a larger sorted list. In merging the sorted lists it is wasteful to disregard the sort of each collection; rather, one can simply merge the sorted lists into a single list using one pass over the lists. Similarly, the individual skyline graphs can be merged into a single graph through a merging operation. Again, we

face the computational intractability of dealing with the entire skyline graph, though these ideas may be extended to the partial graph representation of our IS algorithm.

The problem of acquiring many redundant α-vectors is still an issue with modern point-based algorithms [17,13], which incrementally add new vectors into the value function. These algorithms also benefit from a reduction in the value function. Still, to date, the cost of running Lark or Skyline does not pay off in terms of overall computation cost. As such, further exploration of efficient pruning techniques may potentially help even approximate algorithms to solve larger domains faster. We hope that the Skyline algorithm may trigger more ideas on maintaining small value functions.

The filtering operation is also relevant outside the POMDP context. Consider the familiar problem of finding the maximal scoring path through weighted trellis graph, where the score of the path is the sum of arc scores traversed by the path. It is well known that the optimal path can be computed with dynamic programming (DP) [1]. Now consider the variation in which the score of each arc is a linear function of some D-dimensional parameter, θ, while we wish to solve the DP problem *simultaneously* for all θ. The familiar DP recursion computes the cost of the best path to each intermediate trellis node. Since the score of each individual path to a node is linear in θ, the optimal score over all possible paths is also of the form of Eqn. 5, where each α_j corresponds to a particular path to the node under consideration and θ plays the role of x. Again, here, we encounter the rapid increase in the number of terms in the summation, quickly rendering the calculations infeasible if we cannot reduce the sum to a manageable size. [11] contains a discussion of this problem in the context of supervised training a DP-based sequence estimator. There are, likely, many other scenarios in which this same filtering problem emerges.

References

1. Bellman, R.E.: Dynamic Programming. Princeton University Press (1962)
2. Cassandra, A.R., Littman, M.L., Zhang, N.: Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In: Proceedings of the Conference in Uncertainty in Artificial Intelligence (UAI'97), pp. 54–61 (1997)
3. Chvatal, V.: Linear Programming (Series of Books in the Mathematical Sciences). W. H. Freeman (1983)
4. Doshi, F., Roy, N.: The permutable POMDP: fast solutions to POMDPs for preference elicitation. In: Proceddings of the International Conference on autonomous Agents and Multiagent Systems (AAMAS), pp. 493–500 (2008)
5. Feng, Z., Zilberstein, S.: Region-based incremental pruning for POMDPs. In: Proceedings of the 20th conference on Uncertainty in artificial intelligence, UAI '04, pp. 146–153 (2004)
6. Hansen, E.A.: Solving POMDPs by searching in policy space. In: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, pp. 211–219 (1998)
7. Hauskrecht, M., Fraser, H.S.F.: Planning treatment of ischemic heart disease with partially observable markov decision processes. Artificial Intelligence in Medicine **18**(3), 221–244 (2000)
8. Hsiao, K., Kaelbling, L.P., Lozano-Pérez, T.: Grasping POMDPs. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 4685–4692 (2007)

9. Huynh, V.A., Roy, N.: icLQG: Combining local and global optimization for control in information space. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 2851–2858 (2009)
10. Kalai, G.: Linear programming, the simplex algorithm and simple polytopes. Math. Prog. (Ser. B) **79**, 217–234 (1997)
11. Raphael, C., Nichols, E.: Linear dynamic programming and the training of sequence estimators. In: J.W. Chinneck, B. Kristjansson, M. Saltzman (eds.) Operations Research and Cyber-Infrastructure, vol. 47, pp. 219–231. Springer, US (2009)
12. Shani, G., Brafman, R.I., Shimony, S.E.: Forward search value iteration for POMDPs. In: Proceedings of the Interantional Joint Conference on Artificial Intelligence (IJCAI), pp. 2619–2624 (2007)
13. Shani, G., Brafman, R.I., Shimony, S.E.: Prioritizing point-based POMDP solvers. IEEE Transactions on Systems, Man, and Cybernetics, Part B **38**(6), 1592–1605 (2008)
14. Shani, G., Heckerman, D., Brafman, R.I.: An MDP-based recommender system. Journal of Machine Learning Research **6**, 1265–1295 (2005)
15. Shani, G., Meek, C.: Improving existing fault recovery policies. In: Proceddings of the Neural Information Processing Systems (NIPS), pp. 1642–1650 (2009)
16. Smallwood, R., Sondik, E.: The optimal control of partially observable processes over a finite horizon. OR **21** (1973)
17. Smith, T., Simmons, R.: Point-based POMDP algorithms: Improved analysis and implementation. In: Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence (UAI) (2005)
18. Sondik, E.J.: The optimal control of partially observable Markov processes. Ph.D. thesis (1971)
19. White, C.C.: Partially observed Markov decision processes: A survey. Annals of Operations Research **32** (1991)
20. Williams, J.D., Young, S.: Partially observable Markov decision processes for spoken dialog systems. Computer Speech & Language **21**(2), 393–422 (2007)
21. Zhang, N., Liu, W.: Planning in stochastic domains: Problem characteristics and approximation. Tech. Rep. HKUST-CS96-31, Dept. of Comp. Sci., Hong Kong Univ. of Sci. and Tech. (1997)