In Proc. CPAIOR 2025, pages 119--136, Springer, 2025.

On the Efficiency of Algebraic Simplex Algorithms for Solving MDPs

Dibyangshu Mukherjee and Shivaram Kalyanakrishnan

Indian Institute of Technology Bombay, Mumbai 400076, India.
{dbnshu, shivaram}@cse.iitb.ac.in

Abstract. Markov Decision Problems (MDPs) are a widely-used abstraction of sequential decision-making. A *policy* specifies an action to take from each state of the MDP. Every MDP has an *optimal* policy, which maximises the expected long-term reward from each starting state. A well-known approach to compute an optimal policy for a given MDP is by solving an induced Linear Program (LP). This paper establishes the computational efficiency of a family of "Algebraic Simplex" (AS) algorithms for solving these induced LPs. Unlike geometric methods that rely on quantities such as "value", "gain", or "flux", AS algorithms query policies sequentially for the *discrete set* of locally-improving state-action pairs. We provide upper bounds on the complexity of AS algorithms in three settings: (1) when there are many more states than actions, (2) when there are many more actions than states, and (3) when transitions are deterministic. For all three cases, we furnish AS algorithms with running-time upper bounds that are within a polynomial factor of the tightest known yet across all algorithms. These results demonstrate that the use of geometric information and random access memory do not contribute substantively to the established efficiency of state-of-the-art algorithms.

Keywords: Markov Decision Process · Linear Programming · Simplex · Policy Iteration · Abstract Models

1 Introduction

Markov Decision Problems (MDPs) [32] are a well-studied and widely-applied formalism of long-term decision making under uncertainty. The most common approaches to solve MDPs—such as value iteration, policy iteration, and linear programming—were originally proposed as early as 7–8 decades back. The last 3–4 decades have also witnessed several results published on the computational complexity of solving MDPs, and of specific algorithms. Since MDPs can be solved through induced linear programs, the emergence of poly-time algorithms for linear programming [21, 22] has implied that the number of arithmetic operations required to solve any MDP is at most a polynomial in the number of bits used to encode the MDP. In the other direction, it has been shown that solving MDPs under common reward structures such as finite horizon, infinite horizon with discounted rewards, and infinite horizon with average reward, is P-complete [30].

An alternative measure of computational complexity arises from the "real RAM" or "infinite precision arithmetic" model, under which any arithmetic operation on real-valued operands can be performed in constant time, regardless of the bit-size [29]. This

perspective furnishes running-time bounds that depend only on the number of states and actions in the input MDP: in particular the bounds do not depend on the bit-size of real-valued parameters such as the transition probabilities, rewards, and discount factor. It remains unknown if general MDPs can be solved in *strongly polynomial* time: that is, by using at most $\text{poly}(n, k)$ real RAM operations, where $n \geq 2$ denotes the number of states, and $k \geq 2$ denotes the number of actions in the input MDP.

In this paper, we consider the complexity of solving MDPs under the real RAM computational model. Concretely, we take long-term rewards to be infinite discounted sums. We review three algorithms for this setting, chosen because they provide the tightest upper bounds currently known.

1. MDPs with $n \geq 2$ states and $k = 2$ actions induce acyclic unique sink orientations (AUSOs) of n-dimensional hypercubes [36], which the Random Facet algorithm [12] can solve using $\text{poly}(n) \cdot O(e^{2\sqrt{n}})$ operations in expectation. For $k \geq 2$, the upper bound can be generalised to $\text{poly}(n, k) \cdot e^{O\left(\sqrt{n \log nk}\right)}$ [27, 18]. For any fixed $k > 2$, this upper bound has the slowest (sub-exponential) growth rate in n. Let us denote the generalised version of the Random Facet algorithm $\mathcal{L}_1$.
2. Clarkson [7] proposed a randomised recursive algorithm to solve linear programs with a large number of constraints. Combined with the Random Facet algorithm it is possible to solve LPs induced by n-state, k-action MDPs with a guaranteed running-time of $O(n^3 k + e^{O(\sqrt{n \log n})})$ expected operations. This upper bound is only linear in k, and hence tighter than the one for $\mathcal{L}_1$ when $k \gg n$ (albeit looser when $n \gg k$). For our purposes, denote by $\mathcal{L}_2$ the algorithm of Clarkson.
3. An interesting sub-class of MDPs, called Deterministic MDPs (DMDPs) are those in which all transitions are deterministic. It is well known that DMDPs can be solved in strongly polynomial time—that is, using $\text{poly}(n, k)$ operations. Of the several algorithms that do enjoy polynomial running time on DMDPs [1, 25, 31], denote the one of Post and Ye [31] as $\mathcal{L}_3$. This algorithm implements the "max gain" simplex procedure on an LP induced by the input MDP.

From a distance, $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$ appear quite different in the way they consume and manipulate information from the input MDP. $\mathcal{L}_1$ and $\mathcal{L}_2$ operate on the set of constraints of an induced LP, eliminating constraints by random testing. Critical to the analysis of $\mathcal{L}_3$ is its choice of the single "max-gain" action for each policy update; other Simplex variants are known with even exponential lower bounds on their running time [2]. In this paper, we demonstrate that in spite of their apparent disparity, $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$ can all be implemented (albeit with minor modifications) as "Algebraic Simplex" (AS) algorithms. In the remainder of this section, we provide an informal introduction to this class of algorithms. Formal definitions follow in sections 2 and 3.

1.1 Algebraic Simplex Family of Algorithms

In a commonly-used formulation of an LP from an MDP, vertices of the feasible polytope of the LP are in 1-1 correspondence with (deterministic) policies of the MDP [31]. In this feasible polytope, edges connect any two policies that differ in their action for exactly one state. Simplex algorithms begin at some vertex of the LP, and repeatedly update to a

neighbouring vertex that improves the objective function. In general, Simplex algorithms can use any available information—such as the real-valued objective function (or "gain") along each edge, the geometric coordinates of the vertex, and so on—for selecting the next vertex at any iteration. We denote as *Algebraic Simplex* (AS) algorithms the subclass of Simplex algorithms that are only provided at each iteration the *set* of neighbouring vertices that improve the objective function (equivalently the set of improving state-action pairs). Algorithms from the AS family do not use any geometric information; for example, $\mathcal{L}_3$ [31], which implements the "max-gain" pivot rule, does not belong to the AS family. Note that standard policy iteration algorithms [16] that "jump" in the space of policies (equivalently, which switch multiple state-action pars at each iteration) are also not from the AS family.

The withholding of geometric information would appear rather restrictive. As we demonstrate in Section 2.2, the states values (that is, the expected long-term rewards) of only $\text{poly}(n, k)$ policies are needed to construct the LP induced by an MDP—and this LP, in turn, determines the set of optimal policies for the MDP. Hence, from an information-theoretic standpoint, only $\text{poly}(n, k)$ queries of the MDP are needed if each query—of a policy— can return the state values under that policy. On the other hand, if querying a policy only returns its discrete, locally-improving set of policies, how many queries are needed? And how much further limiting is the constraint of having to move only to some neighbouring vertex in the LP's feasible polytope?

Our contribution is to illustrate the surprising finding that algorithms from the AS family *match* the known upper bounds for $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$ up to a $\text{poly}(n, k)$ factor. In Section 4, we propose an implementation of $\mathcal{L}_1$ as a member of AS, denoting our variant $\mathcal{L}'_1$. Similarly, in Section 5, we implement $\mathcal{L}_2$ an AS algorithm $\mathcal{L}'_2$. For DMDPs, we propose an AS algorithm $\mathcal{L}'_3$, which is inspired by $\mathcal{L}_3$, but different. We show in Section 6 that $\mathcal{L}'_3$ is also strongly polynomial for DMDPs. Not surpisingly, a key element of AS algorithms $\mathcal{L}'_1$, $\mathcal{L}'_2$, and $\mathcal{L}'_3$ is the effective use of memory, which, nonetheless, remains polynomially-sized in the number of states and actions.

MDPs are at the heart of planning and reinforcement learning [28, 35]—topics pursued by the artificial intelligence community. The theoretical analysis of MDPs (and their induced LPs) has been of interest within operations research [31, 38] and combinatorial optimisation [33, 36]. Our results add to this literature. We begin with formal definitions of MDPs (Section 2) and the AS family (Section 3).

2 MDPs and Induced LPs

A **Markov Decision Problem** (**MDP**) models the interaction between an agent and a stochastic environment that the agent navigates sequentially. An MDP $M = (S, A, T, R, \gamma)$ consists of a set of states S, a set of actions A, a transition probability function T, a reward function R, and a discount factor $\gamma \in [0, 1)$. We assume S and A are finite, with $|S| = n \geq 2$ and $|A| = k \geq 2$. If an agent chooses action $a \in A$ from state $s \in S$, then the probability of transitioning to state s' is denoted by $T(s, a, s')$, and the associated reward is given by $R(s, a) \in \mathbb{R}$.

Formally, the agent's behaviour is encoded as a **policy**, which is a mapping from the set of states to the set of actions. For a policy $\pi : S \to A$, the **value function** V^π

characterises the expected long term reward while adhering to π. It can be recursively defined using the Bellman Equations for $s \in S$ as follows:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') \cdot V^{\pi}(s').$$

We define two natural partial orders, $\succeq$ and $\succ$, on the set of policies Π. For policies $\pi, \pi' \in \Pi$, we define $\pi \succeq \pi' \iff (V^{\pi}(s) \geq V^{\pi'}(s),\ \forall s \in S)$. Also, $\pi \succ \pi' \iff (\pi \succeq \pi'$ and $\exists s \in S$ such that $V^{\pi}(s) > V^{\pi'}(s))$. It is well known that there exists a policy $\pi^{\star} \in \Pi$ which maximises the values of all states: that is, $V^{\pi^{\star}}(s) \geq V^{\pi}(s),\ \forall s \in S, \forall \pi \in \Pi$ (equivalently, $\pi^{\star} \succeq \pi$ for all $\pi \in \Pi$) [3]. To simplify exposition, we assume that the optimum is unique. Thus, given an MDP M, the task is to compute the optimal policy for M.

In addition to value vectors, some other quantities are useful while designing and analysing algorithms for MDPs. The **action value function** Q^{π} of policy π gives for each $(s, a) \in S \times A$ the expected total discounted reward obtained by taking action a from state s and following policy π afterwards. We can write it as:

$$Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \cdot V^{\pi}(s').$$

The **flux** x^{π} of a policy π represents the total discounted number of times each $(s, a) \in S \times A$ is visited when the MDP starts simultaneously in all states and follows the Markov chain T^{π}, which is defined by taking action $a = \pi(s)$ at each state s, discounting by γ at each step. If an action a is unused by π, then $x^{\pi}(s, a) = 0$; otherwise, for $s \in S$,

$$x^{\pi}(s, \pi(s)) = 1 + \gamma \sum_{s' \in S} T(s', \pi(s'), s) \cdot x^{\pi}(s', \pi(s')).$$

The **gain** of $a \in A$ for state $s \in S$ with respect to a policy π is defined as: $G^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$.

2.1 LPs and Induced LPs

A linear programming problem is defined by a set of linear inequality constraints H and an objective vector $c \in \mathbb{R}^d$, where each constraint $h \in H$ has the form $a^T x \leq b$ with $a \in \mathbb{R}^d$, $b \in \mathbb{R}$, and $x \in \mathbb{R}^d$.

Let $\mathcal{P}(H)$ be the polyhedron defined by H. The objective is to find a point $x \in \mathcal{P}(H)$ that minimises $c^T x$. We define the **value** of H, denoted as v_H, as the lexicographically smallest point in $\mathcal{P}(H)$. If $\mathcal{P}(H) = \emptyset$, then $v_H = \infty$, and if $\mathcal{P}(H) \neq \emptyset$ but contains no minimum, then $v_H = -\infty$.

A set $B \subseteq H$ with $|B| = d$ is called a (feasible) **basis** if there exists a unique solution x_B. The point x_B corresponds to a vertex of the polyhedron defined by H. The value of a basis B is given by $v_B = c^T x_B$, and for any proper subset $B' \subset B$, it follows that $v_{B'} < v_B$. If $v_H > -\infty$, a basis of H is a minimal subset $B \subseteq H$ such that $v_B = v_H$. A constraint $h \in H$ is said to be **violated** by H if $v_H < v_{H \cup \{h\}}$.

The problem of finding an optimal policy of an MDP can be formulated as a linear program (LP). In this framework, the solution to the primal LP yields the optimal values

associated with the MDP, while the solutions to the dual LP encapsulate the flux of the optimal state-action pairs. The primal and dual LPs [24] are given below.

Primal: Minimise: $\sum_{s' \in S} V(s')$

Subject to: $V(s) \geq R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s'), \qquad \forall s \in S, \forall a \in A.$

Dual: Maximise: $\sum_{s \in S} \sum_{a \in A} R(s,a) \cdot x(s,a)$

Subject to: $\sum_{a \in A} x(s,a) = 1 + \gamma \sum_{s' \in S} \sum_{a \in A} T(s',a,s) \cdot x(s',a), \quad \forall s \in S.$

The constraints of the induced primal LP correspond bijectively to the state-action pairs of the MDP, resulting in a total of n variables and $m = nk$ constraints. Similarly, there exists a bijection between the policies of the MDP and the vertices of the induced dual LP [31]. Our work primarily consists of the induced primal LP, although the reader may find useful to conceptualise certain ideas through the dual.

The **LP digraph** of a Markov Decision Problem (MDP) is a directed graph where each vertex corresponds to a policy (feasible basis of the induced dual LP), and a directed edge exists between two vertices if one policy can be obtained from the other by improving a single state-action pair.

2.2 A Polynomial-time extraction of MDP information

Below, we present a procedure that uses a polynomial number of evaluations to extract all the necessary information to solve an MDP. This is achieved by utilising value functions, which serve as geometric coordinates. In contrast, the LP Digraph represents a much more restricted setting where such extraction is not feasible.

Consider the primal linear programming formulation of an MDP M, where each constraint corresponds to a state-action pair. For state s and action a, the hyperplane is:

$$H_{s,a} = \left\{ V \in \mathbb{R}^n \;\middle|\; V(s) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s') \right\}$$

With n states and k actions per state, there are n sets of k hyperplanes, resulting in a total of nk hyperplanes. Since a policy π prescribes exactly one action per state, its value can be represented in $\mathbb{R}^n$ as: $V^\pi = \bigcap_{s \in S} H_{s,\pi(s)}$, where the intersection captures the value function V^π, satisfying all constraints associated with π. For a given state-action pair (s,a), we define the set of policies: $\Pi_{s,a} = \{\pi \mid \pi(s) = a\}$. Clearly, $|\Pi_{s,a}| = k^{n-1}$ and for $\pi \in \Pi_{s,a}$, $V^\pi \in H_{s,a}$.

At each step, the algorithm selects an arbitrary subset $P_{s,a}$ of n policies from $\Pi_{s,a}$ and evaluates their value functions, yielding their coordinates in $\mathbb{R}^n$. Since n non-collinear points in $\mathbb{R}^n$ uniquely determine a hyperplane, these evaluations suffice to construct the hyperplane $H_{s,a}$. By repeating this process for all state-action pairs, the algorithm constructs all nk hyperplanes using at most n^2k policy evaluations.

With this information extracted from the MDP, the optimal value function $V^\star$ can be solved as: $V^\star = \max_{\pi \in \Pi} \left\{ \bigcap_{s \in S} H_{s,\pi(s)} \right\}$ without any further access to the MDP.

This highlights how value functions encode rich structural information about the MDP, enabling efficient computation of optimal policies. However, our approach circumvents the use for such continuos data, relying instead on discrete inputs—improving state-action pairs—to solve the MDP.

2.3 Simplex and Policy Iteration

The Simplex method operates through single-step transitions, referred to as pivots, shifting from one feasible vertex to an adjacent feasible vertex with a strictly higher objective on the LP-digraph. This iterative process persists until the algorithm converges to an optimum, which corresponds to a vertex with the highest objective.

The choice of pivot rule is key in differentiating Simplex methods and determining their complexity. While the original pivot rule proposed by Dantzig [9] remains the most commonly employed in practice, Klee and Minty [23] demonstrated that this rule can necessitate an exponential number of steps to converge. Numerous other pivot rules have subsequently been observed to exhibit similar behaviour [10, 14, 17].

On the other hand, Kalai [18] introduced a randomised algorithm akin to the Simplex method, which achieved a subexponential bound of $2^{O\left(\sqrt{n \log m}\right)}$ operations for solving an LP with n variables and m constraints. Matousek, Sharir, and Welzl [27] established an identical bound for the dual version algorithm. When combined with Clarkson's algorithm [7], their bound resolves to $O(n^2 m + e^{O(\sqrt{n \log n})})$. These bounds stand as the tightest known for solving LPs, without any derandomisations currently known.

Policy iteration (PI) [16] can be viewed as a generalised version of the Simplex method, where multiple simultaneous pivots or "jumps" are possible. It is a commonly used algorithm for solving Markov Decision Problems (MDPs). For a policy π, define the set of **improving state-actions pairs**, J^π, as follows:

$$J^\pi = \{(s, a) \in S \times A \mid Q^\pi(s, a) > V^\pi(s)\}.$$

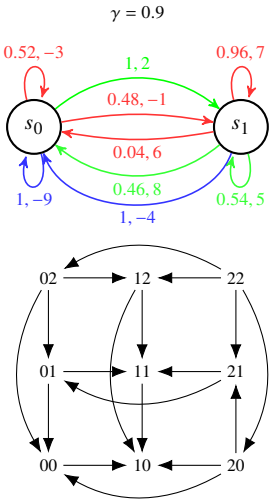
Policy Iteration algorithms rely on the following Policy Improvement Theorem, which states that if a policy has no improving state-action pair, it is already optimal. Otherwise, switching any improving pair yields a policy with a higher value. Formally,

Theorem 1. *For $\pi \in \Pi$: If $J^\pi = \emptyset$ then $\pi = \pi^\star$ otherwise for $(s_0, a_0) \in J^\pi$, define π' such that $\pi'(s_0) = a_0$ and $\pi'(s) = \pi(s)$ for all $s \in S,\ s \neq s_0$. Then $\pi' \succ \pi$.*

Starting from an initial policy, PI involves iteratively navigating through evaluation and improvement steps until reaching an optimal policy. Policy evaluation only needs $\text{poly}(n, k)$ arithmetic operations. For general MDPs the best known upper bounds on the number of iterations are exponential in the number of states n [19, 26]. In case of Deterministic MDPs, Post and Ye [31] obtained the tightest known bounds of $O(n^5 k^2 \log^2 n)$ iterations for the max-gain simplex method.

Figure 1 illustrates an MDP with 2 states and 3 actions. The table lists the nine possible policies, their corresponding value functions, and the improving state-action

pairs. The LP-digraph shows the possible simplex trajectories. Consider two simplex procedures starting with the initial policy $\pi_0 = 22$. One takes the following path: $22 \to 02 \to 12 \to 11 \to 10$, requiring 5 policy evaluations. The other takes a shorter trajectory: $22 \to 20 \to 10$, reducing the number of evaluations to 3.



π	V^π	J^π
0, 0	48.05, 63.89	{(1, 0)}
0, 1	18.05, 26.95	{(0, 0), (1, 1)}
0, 2	−26.31, −27.68	{(0, 0), (0, 1), (1, 2)}
1, 0	63.08, 67.87	∅
1, 1	47.87, 50.97	{(1, 0)}
1, 2	−8.42, −11.57	{(1, 0), (1, 1)}
2, 0	−90, 27.35	{(0, 0), (1, 0), (2, 1)}
2, 1	−90, 60.07	{(0, 1), (1, 1)}
2, 2	−90, −85	{(0, 2), (1, 2), (2, 0), (2, 1)}

Fig. 1: A 2-state, 3-action MDP and its LP digraph. The red, green, and blue edges correspond to actions 0, 1, and 2 respectively, and the tuples represent the transition probability and the rewards. The corresponding table contains values and improving state-action pairs of policies.

3 Algebraic Simplex Computation Model

We introduce a new coordinate-free combinatorial model of computation for solving MDPs, called Algebraic Simplex (**AS**). AS differs from standard PI algorithms in two key aspects: its algebraic property and its utilisation of a polynomial-sized memory.

Starting from an initial policy π_0, at each iteration i of a standard PI algorithm, a mapping is proposed from the action values of the policy to a subset of improving pairs. This mapping can be expressed as: $Q^{\pi_i} \mapsto U \in J^{\pi_i}$ where U is the set of state-action pairs the algorithm chooses to switch to get to π_{i+1}. In contrast an AS algorithm implements a mapping $(J^{\pi_i}, \mathcal{M}_i) \mapsto (j \in J^{\pi_i}, \mathcal{M}_{i+1})$ where j is a (single) state-action pair selected for switching, and $\mathcal{M}_i, \mathcal{M}_{i+1}$ are bit vectors of polynomial size. Additionally, the computational complexity of each step is polynomially bounded in n and k. For randomised PI/AS algorithms, the mapping is stochastic (we can equivalently assume that the input includes a random seed). A significant aspect of AS is that its memory set lacks access to any geometric information of the policies. The policy evaluation step is replaced by a query which returns the set of improving state-action pairs available from the current policy. Figure 2 illustrates the AS model. We review the literature on LPs along the key dimensions that distinguish AS.

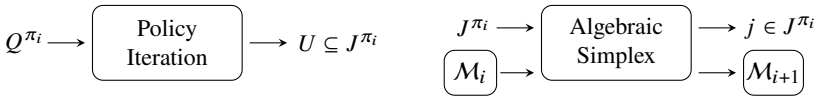


Fig. 2: An illustration of the AS model along with the standard PI model. The incoming/outgoing edges represent input/output to the algorithm.

3.1 Usage of geometry, randomness, memory, and jumping to solve LPs

There has been a longstanding pursuit for a strongly polynomial algorithm for linear programming. While polynomial-time algorithms such as the ellipsoid method [22] and Karmarkar's algorithm [21] exist, their running time depends on the bit representation of the input. Notably, strongly polynomial bounds for linear programming have been attainable when the geometry of the feasible region is favourable.

Consider the Geometric Random-Edge algorithm [11], a variant of the Random-Edge pivot rule [13] for the simplex method. This algorithm applies when the constraint matrix A satisfies a geometric property introduced by Brunsch and Röglin [5]: the sine of the angle of a row of A to a hyperplane spanned by $n-1$ other rows of A is at least δ. Under this condition, the expected running time is $\text{poly}(n, m, 1/\delta)$.

Policy iteration (PI) algorithms can leverage geometry, memory, and jump operations. An example is Geometric Policy Iteration (GPI), which follows its own distinct value function update scheme. The Line Theorem [8] states that policies differing in a single state correspond to the same line segment in the value function polytope. GPI exploits this structure by preventing updates along the same line segment, achieving performance that matches the best-known bounds for solving MDPs (see Table 1).

Another example that uses memory and jump operations is the Fibonacci-Seesaw algorithm [36], which recursively solves acyclic unique sink orientations (AUSOs) by exploring antipodal vertices. This approach extends to solving k-action MDPs using recursive AUSOs [15], a generalisation of AUSOs, in $\text{poly}(n, k) \cdot k^{0.68n}$ iterations.

Randomisation indeed plays a crucial role in complementing geometric approaches. For instance, consider Megiddo's multidimensional search algorithm for LPs [29], which uses geometry to achieve a deterministic algorithm with a bound linear in m but doubly exponential in n. Clarkson's algorithm can also be derandomised [6] to obtain a deterministic algorithm that uses epsilon nets, a geometric property. Without geometry, deterministic algorithms for MDPs currently only achieve bounds exponential in n.

We summarise the features and upper bounds of various algorithms in Table 1. In the following sections, we introduce algorithms based on our AS model. These algorithms begin with an arbitrary policy π_0 and a suitably initialised memory $\mathcal{M}$. At each iteration, they compute the next policy and memory state using the function *next* until the optimal policy is reached.

4 $\mathcal{L}_1$: Random-Facet

The **Random-Facet** (R-F) [27] algorithm for solving LPs proceeds as follows. Given a set H of m constraints, the algorithm selects a constraint h uniformly at random from

Algorithm	Running-time Upper Bound	Geometry	Randomness	Memory	Jump
Geometric RE [11]	$poly(n, m, 1/\delta)$	✓	✓	×	×
HPI [38]	$poly(n, k, 1/1-\gamma)$	✓	×	×	✓
Simplex [38]	$poly(n, k, 1/1-\gamma)$	✓	×	×	×
GPI [37]	$poly(n, k, 1/1-\gamma)$	✓	×	✓	✓
Megiddo [29]	$poly(m) \cdot exp(exp((n))$	✓	×	✓	✓
D-Clarkson [6]	$poly(m) \cdot exp(n)$	✓	×	✓	×
RSPI [20]	$poly(n, k) \cdot exp(n)$	✓	✓	×	×
FS [36] [15]	$poly(n, k) \cdot exp(n)$	×	×	✓	✓
$\mathcal{L}_1$ [27]	$poly(n, k) \cdot exp(\sqrt{n \log nk})$	×	✓	✓	×
$\mathcal{L}_2$ [27] [7]	$poly(n, k) \cdot exp(\sqrt{n \log n})$	×	✓	✓	×
$\mathcal{L}_3$ [31]	$poly(n, k)$	✓	×	×	×
$\mathcal{L}'_1$ (this paper)	$poly(n, k) \cdot exp(\sqrt{n \log nk})$	×	✓	✓	×
$\mathcal{L}'_2$ (this paper)	$poly(n, k) \cdot exp(\sqrt{n \log n})$	×	✓	✓	×
$\mathcal{L}'_3$ (this paper)	$poly(n, k)$	×	×	✓	×

Table 1: Algorithms and their key component features across four primary categories. It is not apparent from their original descriptions that $\mathcal{L}_1$ [27] and $\mathcal{L}_2$ [7] do not need geometry and jumping. By proposing $\mathcal{L}'_1$ and $\mathcal{L}'_2$ as replacements, we bring out this aspect explicitly.

$H \setminus C$, where C is a given input basis, initially arbitrary. Subsequently, it computes a basis B for $H \setminus h$ recursively. If h is *not* violated by B, then B serves as a basis for H, marking the completion of the algorithm. However, if h is violated by B, then h is added to B and the recursion continues. Pseudocode is given in Algorithm 1. The algorithm simplifies in the case of combinatorial cubes [12].

4.1 Algorithm: $\mathcal{L}'_1$

Recall that in MDPs, each policy corresponds to a basis in the dual LP. We implement R-F on an MDP M as follows. Starting from a policy π, the algorithm selects a state-action pair (s, a) uniformly at random that is not used by π. It then recursively solves the MDP without (s, a) (denoted M') to obtain an optimal policy π' for M'. If (s, a) is not an improving pair for π' in M, then π' is necessarily optimal for M. Otherwise, the algorithm switches state s with action a in π', yielding a new policy π'', and repeats the procedure from π''.

Algorithm 2 shows pseudocode for an implementation of R-F as an AS algorithm. Memory is implemented as a stack of state-action pairs, initialised as empty. Let P denote the set of all state-action pairs in the MDP, and let P^π represent the state-action pairs associated with a policy π, formally defined as $P^\pi = \{(s, \pi(s)) \mid s \in S\}$. Starting from an arbitrary policy, the algorithm iterates. At each step, the stack, of size at most $m - n$, is populated with state-action pairs uniformly sampled from a subset of P. These pairs are then removed from the stack sequentially until an improving pair is found. When such a pair is identified, the algorithm switches to a new policy using the improving pair and restarts with the updated policy and memory stack. The operation `switch`$(\pi, (s, a))$ modifies policy π by assigning action a to state s.

4.2 Analysis

We revisit the analysis of $\mathcal{L}_1$ [27] to establish the complexity of $\mathcal{L}_1'$.

Theorem 2. *Starting from an arbitrary policy π_0, the expected number of calls to next-$\mathcal{L}_1'$ is upper-bounded by: $poly(n,m) \cdot e^{O\left(\sqrt{n \log m}\right)}$.*

Proof. Within a call to *next*, a state-action pair p is popped from the stack, and the process continues until p is improving for the current policy π. Let v_p denote the optimal value of the MDP without the state-action pair p, i.e., $v_p = v_{P \setminus p}$. Then, p is improving for π if and only if $v_p > v_{P^\pi}$. Clearly, if $p \in P^{\pi^\star}$, then $v_p \geq v_{P^\pi}$ for all $\pi \in \Pi$.

Consider an ordering of state-action pairs $P = (p_1, p_2, \ldots, p_m)$ such that

$$v_{p_1} \leq v_{p_2} \leq \cdots \leq v_{p_{n-\theta}} < v_{p_{n-\theta+1}} \leq \cdots \leq v_{p_m}.$$

Define a state-action pair p as *fixed* for π if $v_p < v_{P^\pi}$. Under this ordering, if $v_{p_{n-\theta}} < v_{P^\pi}$, then exactly $n-\theta$ pairs are fixed in π. Consequently, after transitioning from policy π to π', only θ state-action pairs from $\{p_{n-\theta+1}, \ldots, p_m\}$ can lead to further switches.

Since the algorithm selects pairs uniformly at random from $P \setminus P^\pi$, each has a probability of $\frac{1}{m-n}$ of being chosen. Let $T_\theta(m)$ denote the expected number of switches performed by algorithm $\mathcal{L}_1'$ with m state-action pairs, $n-\theta$ of which are fixed.

The recurrence relation for $T_\theta(m)$ is given by:

$$T_\theta(0) = 0, \quad 0 \leq \theta \leq n;$$

$$T_\theta(m) \leq T_\theta(m-1) + 1 + \frac{1}{m-n} \sum_{i=1}^{\min\{m-n,\theta\}} T_{\theta-i}(m).$$

As stated in the theorem, this recurrence satisfies [27]:

$$T_n(m) \leq \text{poly}(n,m) \cdot e^{O\left(\sqrt{n \log m}\right)}$$

Algorithm 1 Random Facet ($\mathcal{L}_1$) [27]

```
1:  procedure Φ_R(H, C)
2:      if H = C then
3:          return C
4:      else
5:          Pick h ∈ H \ C uniformly at
6:                                  random
7:          B ← Φ_R(H \ h, C)        ▷ 1st call
8:          if h is violated by B then
9:              return Φ_R(H, basis(B, h))
                                     ▷ 2nd call
10:         else
11:             return B
```

Algorithm 2 "Next" Function under $\mathcal{L}_1'$

```
1:  procedure next_L1'(π, M)
2:      U ← P \ M ∪ P^π
3:      while U ≠ ∅ do
4:          Pick u = (s, a) ∈ U uniformly
                                  at random
5:          Push (u, M)
6:          U ← U \ {u}
7:      repeat
8:          q ← Pop(M)
9:      until q ∈ J^π
10:     π' ← switch(π, q)
11:     return (π', M)
```

5 $\mathcal{L}_2$: Las Vegas

In this section, we study the scenario where the MDP possesses a large set of actions as compared to the set of states. We devise our AS algorithm, denoted $\mathcal{L}_2'$, drawing inspiration from a "Las Vegas" algorithm (denoted $\mathcal{L}_2$) proposed by Clarkson [7].

Consider an LP with many more constraints than variables. The key idea of $\mathcal{L}_2$ is to exclude redundant constraints from the LP and find the optimal basis quickly. The algorithm does this by building a set of constraints $W^\star$ that will eventually contain the optimal basis. It proceeds recursively in phases, adding a constraint set W to $W^\star$ at each phase. The set W satisfies two properties: its size remains below a threshold, and it includes at least one constraint from the optimal basis. The recursion base case is solved using the Simplex method. The pseudocode for $\mathcal{L}_2$ is given in Algorithm 3.

5.1 Algorithm: $\mathcal{L}_2'$

Our algorithm $\mathcal{L}_2'$ adapts the procedure $\mathcal{L}_2$ for solving MDPs, albeit with a few key modifications. At each stage, it solves a sub-MDP defined by a set of state-action pairs, storing them in memory until the size of the MDP drops below a threshold. It then solves the reduced MDP using $\mathcal{L}_1'$. Unlike $\mathcal{L}_2$, $\mathcal{L}_2'$ follows a simplex procedure. It initialises the set $W^\star$ with P^{π_0}, where π_0 is the initial policy. At every iteration, it adds an improving state-action pair to $W^\star$. This ensures that the algorithm follows a sequence of policies $\pi_0, \pi_1, \ldots, \pi_i$, such that $\pi_{i+1} \succ \pi_i$.

Pseudocode for $\mathcal{L}_2'$ is given in Algorithm 4. Note that the memory operates on sets, so the Push and Pop operations are performed with sets of pairs rather than individual elements. The memory has two components: $\mathcal{M}^0$, which stores the state-action pairs representing the sub-mdp and $\mathcal{M}^1$, which contains the set $W^\star$. Initially, $\mathcal{M}^0$ holds all state-action pairs of the MDP, while $\mathcal{M}^1$ starts with pairs representing the initial policy.

5.2 Analysis

We extend Clarkson's analysis to $\mathcal{L}_2'$ as follows.

Theorem 3. *Starting from an arbitrary policy π_0, the expected number of calls to next-$\mathcal{L}_2'$ is upper-bounded by: $O\left(n^2 m + e^{O\left(\sqrt{n \log n}\right)}\right)$.*

Proof. The algorithm involves populating the memory $\mathcal{M} = \left(\mathcal{M}^0, \mathcal{M}^1\right)$ with sets of state-action pairs. It can be shown that if the set W is non-empty, it must contain a state-action pair belonging to the optimal policy $\pi^\star$ [7]. Consequently, the size of $\mathcal{M}^1$ is augmented at most $n + 1$ times.

A key result is that the expected size of W cannot exceed $\frac{mn}{|R|}$ [7]. Given that $|R| = n\sqrt{m}$, we have $E[|W|] \leq \sqrt{m}$. By Markov's inequality, $P(|W| > 2\sqrt{m}) \leq 0.5$. We define a success at an iteration as the event $|W| \leq 2\sqrt{m}$ and a failure otherwise. During a success, $2\sqrt{m}$ state-action pairs are added to $W^\star$, while only one pair is added during a failure. Hence, the size of $\mathcal{M}^1$ increases by at most $\sqrt{m} + 1$ in expectation. Given that there are at most $n + 1$ increments, the total size of $\mathcal{M}^1$ is bounded by $2n\sqrt{m}$. Since

each element of $\mathcal{M}^0$ (beyond initialisation) is a union of R and $\mathcal{M}^1$, the size of $\mathcal{M}^0$ is bounded by $3n\sqrt{m}$.

Let $T(m)$ denote the expected time to solve an MDP with n states and m state-action pairs. As the algorithm requires solving at most $2(n+1)$ MDPs with at most $3n\sqrt{m}$ pairs, we obtain [7]:

$$T(m) \leq 2(n+1)T(3n\sqrt{m}) + O(n^2 m).$$

For $m \leq 9n^2$, substituting $T(3n\sqrt{m})$ with our $\mathcal{L}_1'$ bound yields the desired result:

$$T(m) \leq O\left(n^2 m + e^{O(\sqrt{n \log n})}\right).$$

6 $\mathcal{L}_3$: Max-Gain

The Max-gain Simplex algorithm (here denoted $\mathcal{L}_3$) operates as follows: starting from an initial policy, it iteratively switches to a new policy by selecting a state-action pair with the highest gain. Specifically, at each iteration, it updates the policy π by replacing the action at state s_m with a_m, where: $(s_m, a_m) \in \arg\max_{(s,a)} G^\pi(s,a)$. This process transitions to the next policy in the sequence. Post and Ye [31] showed that $\mathcal{L}_3$ is strongly polynomial on Deterministic MDPs (DMDPs). Below, we summarise their analysis.

Algorithm 3 Las Vegas ($\mathcal{L}_2$) [7]

procedure $\Psi_r(H$: set of constraints)
  $W^\star \leftarrow \varnothing$; $C_n \leftarrow 9n^2$
  if $|H| \leq C_n$ **then**
    return $\Psi_s^\star(H)$ ▷ Simplex
  repeat
    Choose $R \subset H \setminus W^\star$ uniformly at random, $|R| = n\sqrt{m}$
    $x^\star \leftarrow \Psi_r^\star(R \cup W^\star)$
    $W \leftarrow \{h \in H \mid x^\star \text{ violates } h\}$
    if $|W| \leq 2\sqrt{m}$ **then**
      $W^\star \leftarrow W^\star \cup W$
  until $W = \emptyset$
  return $x^\star$

Algorithm 4 "Next" Function under $\mathcal{L}_2'$

procedure $next\_\mathcal{L}_2'(\pi, \mathcal{M})$
  $U \leftarrow$ **Pop**$(\mathcal{M}^0)$; $W^\star \leftarrow$ **Pop**$(\mathcal{M}^1)$
  Push$(U, \mathcal{M}^0)$
  while $|U| > 9n^2$ **do**
    Pick $R \subset U \setminus W^\star$ uniformly at random, $|R| = n\sqrt{|U|}$
    $U \leftarrow R \cup W^\star$
    Push $\left(U, \mathcal{M}^0\right)$
  Curr $\leftarrow$ **Pop**$(\mathcal{M}^0)$
  $\pi' \leftarrow \mathcal{L}_1'(\pi, \text{Curr})$
  repeat
    Prv $\leftarrow$ **Pop**$(\mathcal{M}^0)$
    $W \leftarrow J^{\pi'} \cap$ Prv
  until $W \neq \emptyset$
  Push(Prv, $\mathcal{M}^0$)
  if $|W| < 2\sqrt{|\text{Prv}|}$ **then**
    $W^\star \leftarrow W \cup W^\star$
  else
    Pick a $w \in W$
    $W^\star \leftarrow w \cup W^\star$
  Push $\left(W^\star, \mathcal{M}^1\right)$
  return $(\pi', \mathcal{M})$

6.1 Analysis of $\mathcal{L}_3$

A DMDP can be represented as a weighted directed multigraph, where the vertices correspond to states and the edges represent actions. A policy corresponds to a subgraph in which each vertex has exactly one outgoing edge. As a result, every policy can be visualised as a collection of disjoint paths that lead to corresponding disjoint cycles. The analysis leverages the concept of flux within these paths and cycles.

Fix some policy $\pi \in \Pi$. If an action of π resides on a path, it can only be utilised once, contributing a unit of flux per state. In contrast, if an action is part of a cycle, each state within the cycle contributes a total flux of $1/(1-\gamma)$ to the cycle. Therefore, the flux bounds can be expressed for $s \in S$ as: $1 \leq x^{\pi}(s, a_p) \leq n$ and $\frac{1}{1-\gamma} \leq x^{\pi}(s, a_c) \leq \frac{n}{1-\gamma}$, where a_p and a_c denote actions on a path and in a cycle, respectively. A key aspect of the analysis is that an action update on a path contributes to optimising the paths leading to their respective cycles, whereas an update on a cycle results in a significant improvement in the overall objective. The progress is quantified as follows.

Suppose the algorithm pivots with a path update from policy π to π' and subsequently to π'', where π'' represents a policy in which no further path updates are possible. Post and Ye show that:

$$\sum_s \{U(\pi'', s) - U(\pi', s)\} \leq \left(1 - \frac{1}{n^2}\right) \sum_s \{U(\pi', s) - U(\pi, s)\}. \tag{1}$$

On the other hand if π and π' were policies where π' creates a new cycle we have:

$$\sum_s \left\{U(\pi^\star, s) - U(\pi', s)\right\} \leq \left(1 - \frac{1}{n}\right) \sum_s \left\{U(\pi^\star, s) - U(\pi, s)\right\} \tag{2}$$

where $U(\pi, s) = R(s, \pi(s)) \cdot x^{\pi}(s, \pi(s))$. We obtain the two lemmas below using (1) and (2), respectively [31].

Lemma 1. *Let π be a policy. After $O(n^2 \log n)$ iterations starting from π, either $\mathcal{L}_3$ finishes, a new cycle is created, a cycle is broken, or some action in π never appears in a policy again until a new cycle is created.*

Lemma 2. *Let π be a policy. Starting from π, after $O(n \log n)$ iterations in which a new cycle is created, some action in π is either eliminated from cycles for the remainder of $\mathcal{L}_3$ or entirely eliminated from policies for the remainder of the algorithm.*

Lemmas 1 and 2 combine to provide the required result:

Theorem 4. *$\mathcal{L}_3$ converges in at most $O(n^5 k^2 \log^2 n)$ iterations on DMDPs.*

6.2 Algorithm: $\mathcal{L}_3'$

Since gain is a real-valued geometric quantity, $\mathcal{L}_3$ alone does not satisfy the AS property. We design a strongly polynomial-time AS algorithm, $\mathcal{L}_3'$, using the analysis of $\mathcal{L}_3$.

Let π and $\overline{\pi}$ be two policies such that $\overline{\pi} \succ \pi$. Starting from π, we define a procedure $\Phi(\pi, \overline{\pi})$ to identify a policy π' satisfying $\pi' \succeq \overline{\pi}$. Define $D = J^{\pi} \cap P^{\overline{\pi}}$. At each step,

an element p is arbitrarily selected from D, and π is updated by switching p, yielding a new policy π'. The procedure then continues with π' replacing π.

Since this follows the simplex procedure, any selected p cannot reappear in a subsequent D, ensuring that $|D|$ decreases by one at each step. Given that $|D| \leq n$, the process terminates in at most n steps.

We define our AS algorithm based on the procedure Φ as follows: Let Θ^π be the set of policies which result after switching a state-action pair from J^π. At each step, we select $\theta \in \Theta^\pi$ and apply Φ to obtain a policy π' that dominates θ. Iterating over all elements of Θ^π yields a policy π'' that dominates every element in Θ^π.

Since the policy θ_m, obtained via a max-gain switch from π, is guaranteed to be in Θ^π, it follows that $\pi'' \succeq \theta_m$. The process then restarts from π'' and repeats until optimality is reached. The pseudocode for the *next* procedure is provided in Algorithm 5. We use memory $\mathcal{M}$ to store Θ^π, initialising it as an empty set.

Algorithm 5 "Next" Function under $\mathcal{L}'_3$

procedure $next\_\mathcal{L}'_3(\pi, \mathcal{M})$
  repeat
    if $\mathcal{M} = \emptyset$ **then**
      for $p \in J^\pi$ **do**
        Push $(\texttt{switch}(\pi, p), \mathcal{M})$
    $\overline{\pi} \leftarrow$ **Pop** $(\mathcal{M})$
  until $J^\pi \cap P^{\overline{\pi}} \neq \emptyset$
  return $\Phi(\pi, \overline{\pi}, \mathcal{M})$

Algorithm Helper Function: Φ

procedure $\Phi(\pi, \overline{\pi}, \mathcal{M})$
  Pick $p \in J^\pi \cap P^{\overline{\pi}}$
  $\pi' \leftarrow \texttt{switch}(\pi, p)$
  if $J^{\pi'} \cap P^{\overline{\pi}} \neq \emptyset$ **then**
    Push$(\overline{\pi}, \mathcal{M})$
  return $(\pi', \mathcal{M})$

6.3 Analysis

Starting from a policy π, we define the operator η, which returns a policy dominating every element of Θ^π. Consequently, the policies in the trajectory of $\mathcal{L}'_3$ can be structured into two layers: i) Those forming the sequence $\mathcal{X} = \big(\pi, \eta(\pi), \eta^2(\pi), \ldots\big)$, and ii) those appearing between successive elements of $\mathcal{X}$. As established earlier, $\eta^{i+1}(\pi) \succeq \theta_m$, where θ_m is the policy obtained by selecting the action with the highest gain from $\eta^i(\pi)$. The following two lemmas help establish a correspondence between the trajectory of policies in $\mathcal{X}$ and those visited by $\mathcal{L}_3$.

Lemma 3. *Let π, π', and π'' be three adjacent elements in the sequence $\mathcal{X}$. Suppose the switches from π to π' and from π' to π'' do not introduce any new cycles, and no further path updates are possible from π''. Then, π, π', and π'' satisfy equation* (1)*.*

Proof. Let $\Delta = \max_{(s,a)} G^\pi(s, a)$ be the highest gain and let A_c denote the set of actions belonging to cycles in π. Since π'' does not create a new cycle, $G^\pi(s, a) = 0$ for all $a \in A_c$. Define the operator $\Psi(\pi', \pi) = \sum_s (U(\pi', s) - U(\pi, s))$. Now we have:

$$\Psi(\pi'', \pi') = \Psi(\pi'', \pi) - \Psi(\pi', \pi) \tag{3}$$

Let $\bar{\pi}'$ be the policy which results by switching the action with the highest gain from π. Since $\pi' \succeq \bar{\pi}'$, we have: $\Psi(\pi', \pi) \geq \Psi(\bar{\pi}', \pi)$. Since policy $\bar{\pi}'$ must have at least 1 unit

of flux on the action with gain Δ, $\Psi(\pi', \pi) \geq \Psi(\bar{\pi}', \pi) \geq \Delta$. Replacing $\Psi(\pi', \pi)$ with Δ in equation (3) we obtain: $\Psi(\pi'', \pi') \leq \Psi(\pi'', \pi) - \Delta$. Recall $x^{\pi}(s, a_P) \leq n$ for all π and a_P therefore the total flux on paths is bounded by n^2 which gives $\Psi(\pi'', \pi) \leq n^2\Delta$. Thus we have: $\Psi(\pi'', \pi') \leq \Psi(\pi'', \pi) - \Delta$ and $\Delta \geq \Psi(\pi'', \pi)/n^2$. Therefore: $\Psi(\pi'', \pi') \leq (1 - 1/n^2)\Psi(\pi'', \pi)$.

Lemma 4. *Suppose π and π' be adjacent policies of a sequence $\mathcal{X}$ such that π' creates a new cycle. Then π and π' satisfy equation* (2)*.*

Proof. We have:

$$\Psi(\pi^\star, \pi') = \Psi(\pi^\star, \pi) - \Psi(\pi', \pi) \tag{4}$$

Again let $\bar{\pi}'$ be the policy which results by switching the action with the highest gain from π and let a be an action which forms a cycle. Since $x^{\pi}(s, a_C) \geq 1/(1 - \gamma)$, switching a will result in at least $1/(1 - \gamma)$ units of flux through a. Since $\pi' \succeq \bar{\pi}'$, we have: $\Psi(\pi', \pi) \geq \Delta/(1 - \gamma)$. Replacing in equation (4), we obtain: $\Psi(\pi^\star, \pi') \leq \Psi(\pi^\star, \pi) - \Delta/(1 - \gamma)$. Since the total flux on an MDP is bounded by $\frac{n}{1-\gamma}$, we have: $\Psi(\pi^\star, \pi) \leq n\Delta/(1 - \gamma)$. Therefore we get: $\Psi(\pi^\star, \pi') \leq (1 - 1/n)\Psi(\pi^\star, \pi)$.

Lemmas 1 and 2 hold for the policies in the sequence $\mathcal{X}$ as a direct consequence of Lemmas 3 and 4, leading analogously to the following result:

Theorem 5. *For a given policy π, the length of the sequence $\mathcal{X}$ and the number of iterations in $\mathcal{L}_3$ have the same asymptotic order.*

Consider a policy $\pi \in \mathcal{X}$. It is clear that: $|\Theta^{\pi}| \leq n(k - 1)$. Now, let θ_1 and θ_2 be two consecutive policies in the trajectory of $\mathcal{L}_3'$ such that both $\theta_1, \theta_2 \in \Theta^{\pi}$. Between θ_1 and θ_2, there can be at most n policies in $\mathcal{L}_3'$. Consequently, the total number of policies in a trajectory of $\mathcal{L}_3'$ is bounded by: $n^2(k - 1) \cdot O(n^5k^2 \log^2 n) = O(n^7k^3 \log^2 n)$.

7 Summary

We have introduced a coordinate-free computation model to explore the role of geometry in MDP planning. By analysing three algorithms renowned for their efficiency in solving LPs and MDPs and adapting them to our algebraic and jump-free model we have uncovered intriguing insights. These findings are striking given the substantial advantages geometry often confers on algorithms. To the best of our knowledge, this is a novel investigation in the MDP literature. Our results prompt an important question: do geometry and jumping provide only a polynomial advantage in solving MDPs, or can they lead to significantly tighter upper bounds than currently established?

Our coordinate-free AS framework has promising applications. It provides a stable and efficient method for computing Blackwell optimal policies [4], mitigating numerical instabilities arising from dependence on geometric properties. Furthermore, our study suggests that the polytope structure of MDPs remains underutilised in algorithm design. By harnessing this structure—much like LP algorithms exploit polytopes—it may be possible to develop fundamentally more efficient MDP solvers. Additionally, extending AS to generalisations like Stochastic Games [34] could yield new insights into computing solutions such as Nash Equilibria.

Acknowledgements

The authors thank Sundar Vishwanathan and Supratik Chakraborty for providing useful comments.

References

1. Andersson, D., Vorobyov, S.: Fast Algorithms for Monotonic Discounted Linear Programs with Two Variables per Inequality. Preprint NI06019-LAA, Isaac Newton Institute for Mathematical Sciences, Cambridge, UK (2006)
2. Ashutosh, K., Consul, S., Dedhia, B., Khirwadkar, P., Shah, S., Kalyanakrishnan, S.: Lower Bounds for Policy Iteration on Multi-action MDPs. In: 59th IEEE Conference on Decision and Control, CDC 2020. pp. 1744–1749. IEEE (2020)
3. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton, NJ, USA (1957)
4. Blackwell, D.: Discrete Dynamic Programming. The Annals of Mathematical Statistics pp. 719–726 (1962)
5. Brunsch, T., Röglin, H.: Finding Short Paths on Polytopes by the Shadow Vertex Algorithm. In: Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013. pp. 279–290. Springer (2013)
6. Chan, T.M.: Improved Deterministic Algorithms for Linear Programming in Low Dimensions. ACM Trans. Algorithms **14**(3), 30:1–30:10 (2018)
7. Clarkson, K.L.: Las Vegas Algorithms for Linear and Integer Programming When the Dimension is Small. J. ACM **42**(2), 488–499 (1995)
8. Dadashi, R., Bellemare, M.G., Taïga, A.A., Roux, N.L., Schuurmans, D.: The Value Function Polytope in Reinforcement Learning. In: Proceedings of the 36th International Conference on Machine Learning, ICML 2019, USA. pp. 1486–1495. PMLR (2019)
9. Dantzig, G.B.: Linear Programming and Extensions. Princeton university press (1963)
10. Disser, Y., Friedmann, O., Hopp, A.V.: An Exponential Lower Bound for Zadeh's pivot rule. Math. Program. **199**(1), 865–936 (2023)
11. Eisenbrand, F., Vempala, S.S.: Geometric Random Edge. Math. Program. **164**(1-2), 325–339 (2017)
12. Gärtner, B.: The Random-Facet Simplex Algorithm on Combinatorial Cubes. Random Struct. Algorithms **20**(3), 353–381 (2002)
13. Gärtner, B., Kaibel, V.: Two New Bounds on the Random-Edge Simplex Algorithm. SIAM J. Discret. Math. **21**(1), 178–190 (2007)
14. Goldfarb, D., Sit, W.Y.: Worst Case Behavior of the Steepest Edge Simplex Method. Discret. Appl. Math. **1**(4), 277–285 (1979)
15. Gupta, A., Kalyanakrishnan, S.: Improved Strong Worst-case Upper Bounds for MDP Planning. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017. pp. 1788–1794. ijcai.org (2017)
16. Howard, R.A.: Dynamic programming and Markov processes. MIT Press (1960)
17. Jeroslow, R.: The Simplex Algorithm with the Pivot Rule of Maximizing Criterion Improvement. Discrete Mathematics **4**(4), 367–377 (1973)
18. Kalai, G.: A Subexponential Randomized Simplex Algorithm (Extended Abstract). In: Proceedings of the 24th Annual ACM Symposium on Theory of Computing,. pp. 475–482. ACM (1992)
19. Kalyanakrishnan, S., Mall, U., Goyal, R.: Batch-Switching Policy Iteration. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016. pp. 3147–3153. IJCAI/AAAI Press (2016)

20. Kalyanakrishnan, S., Misra, N., Gopalan, A.: Randomised Procedures for Initialising and Switching Actions in Policy Iteration. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence,. pp. 3145–3151. AAAI Press (2016)
21. Karmarkar, N.: A New Polynomial-Time Algorithm for Linear Programming. In: Proceedings of the 16th Annual ACM Symposium on Theory of Computing,. pp. 302–311. ACM (1984)
22. Khachiyan, L.G.: Polynomial Algorithms in Linear Programming. USSR Computational Mathematics and Mathematical Physics **20**(1), 53–72 (1980)
23. Klee, V., Minty, G.J.: How Good Is the Simplex Algorithm. Inequalities **3**(3), 159–175 (1972)
24. Littman, M.L., Dean, T.L., Kaelbling, L.P.: On the Complexity of Solving Markov Decision Problems. In: UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence. pp. 394–402. Morgan Kaufmann (1995)
25. Madani, O., Thorup, M., Zwick, U.: Discounted Deterministic Markov Decision Processes and Discounted All-Pairs Shortest Paths. ACM Trans. Algorithms **6**(2), 33:1–33:25 (2010)
26. Mansour, Y., Singh, S.: On the Complexity of Policy Iteration. In: UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, 1999. pp. 401–408. Morgan Kaufmann (1999)
27. Matousek, J., Sharir, M., Welzl, E.: A Subexponential Bound for Linear Programming. Algorithmica **16**(4/5), 498–516 (1996)
28. Mausam, Kolobov, A.: Planning with Markov Decision Processes: An AI Perspective. Morgan & Claypool (2012)
29. Megiddo, N.: Linear Programming in Linear Time When the Dimension Is Fixed. J. ACM **31**(1), 114–127 (1984)
30. Papadimitriou, C.H., Tsitsiklis, J.N.: The Complexity of Markov Decision Processes. Math. of Oper. Res. **12**(3), 441–450 (1987)
31. Post, I., Ye, Y.: The Simplex Method is Strongly Polynomial for Deterministic Markov Decision Processes. In: Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013. pp. 1465–1473. SIAM (2013)
32. Puterman, M.L.: Markov Decision Processes. Wiley (1994)
33. Schurr, I., Szabó, T.: Jumping Doesn't Help in Abstract Cubes. In: Integer Programming and Combinatorial Optimization, 11th International IPCO Conference. Lecture Notes in Computer Science, vol. 3509, pp. 225–235. Springer (2005)
34. Shapley, L.S.: Stochastic games. Proceedings of the national academy of sciences **39**(10), 1095–1100 (1953)
35. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
36. Szabó, T., Welzl, E.: Unique Sink Orientations of Cubes. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001,. pp. 547–555. IEEE Computer Society (2001)
37. Wu, Y., Loera, J.A.D.: Geometric Policy Iteration for Markov Decision Processes. In: KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 2070–2078. ACM (2022)
38. Ye, Y.: The Simplex and Policy-Iteration Methods are Strongly Polynomial for the Markov Decision Problem with a Fixed Discount Rate. Math. Oper. Res. **36**(4), 593–603 (2011)