Improved Strong Worst-case Upper Bounds for MDP Planning

Anchit Gupta and **Shivaram Kalyanakrishnan**
Department of Computer Science and Engineering, Indian Institute of Technology Bombay
{anchit, shivaram}@cse.iitb.ac.in

Abstract

The Markov Decision Problem (MDP) plays a central role in AI as an abstraction of sequential decision making. We contribute to the theoretical analysis of MDP *planning*, which is the problem of computing an optimal policy for a given MDP. Specifically, we furnish improved *strong worst-case* upper bounds on the running time of MDP planning. Strong bounds are those that depend only on the number of states n and the number of actions k in the specified MDP; they have no dependence on affiliated variables such as the discount factor and the number of bits needed to represent the MDP. Worst-case bounds apply to *every* run of an algorithm; randomised algorithms can typically yield faster *expected* running times. While the special case of 2-action MDPs (that is, $k = 2$) has recently received some attention, bounds for general k have remained to be improved for several decades. Our contributions are to this general case. For $k \geq 3$, the tightest strong upper bound shown to date for MDP planning belongs to a family of algorithms called Policy Iteration. This bound is only a polynomial improvement over a trivial bound of $\text{poly}(n,k) \cdot k^n$ [Mansour and Singh, 1999]. In this paper, we generalise a contrasting algorithm called the Fibonacci Seesaw, and derive a bound of $\text{poly}(n,k) \cdot k^{0.6834n}$. The key construct that we use is a template to map algorithms for the 2-action setting to the general setting. Interestingly, this idea can also be used to design Policy Iteration algorithms with a running time upper bound of $\text{poly}(n,k) \cdot k^{0.7207n}$. Both our results improve upon bounds that have stood for several decades.

1 Introduction

The Markov Decision Problem (MDP) [Bellman, 1957; Puterman, 1994] is widely used as a model of sequential decision making under uncertainty. An MDP abstracts a setting in which an agent must decide which action to take from every possible state. The agent's current state and action determine (in general stochastically) the next state and an accompanying reward. In order to maximise its expected *long-term* reward, which action must the agent pick at every state? Presumably, this question would be among the foremost to answer about any given MDP. Formally, it constitutes the problem of MDP *planning*: that is, of finding an optimal *policy* (a mapping from states to actions) for a given MDP.

Even though MDPs have existed for over half a century, and several practically-efficient planning algorithms have been developed over the years, our *theoretical* understanding of the complexity of MDP planning has remained unsatisfactory. As a case in point, consider an MDP with $n \geq 2$ states and $k \geq 2$ actions. We assume a computational model in which arithmetic operations on real numbers can be performed in constant time (regardless of the number of bits used to represent the operands). Under this "infinite-precision arithmetic" model, any given policy for our MDP can be tested for optimality in $\text{poly}(n,k)$ time. Hence, by testing each of the possible k^n deterministic policies (of which one is guaranteed to be optimal), we can identify an optimal policy in $\text{poly}(n,k) \cdot k^n$ time. This bound is *strong*, in the sense that it has no dependence on affiliated parameters of the MDP, such as its discount factor and the precision of representing its transition probabilities and rewards. Bounds that depend on such parameters are termed *weak*. Ignoring polynomial factors, *no* deterministic algorithm has been shown to date to enjoy a tighter strong bound for MDP planning than the brute force procedure outlined above, except on 2-actions MDPs.[1]

In this paper, we propose a framework to bridge the special case of $k = 2$ to the general case ($k \geq 3$). Consequently, we are able to derive *exponentially* tighter strong bounds for the general case. We restrict our focus to deterministic algorithms, and derive bounds that apply to every run on every MDP instance with n states and k actions (that is, *worst case* bounds). In contrast, randomised algorithms can typically yield better *expected* running times for every n-state, k-action MDP.

We begin by highlighting our contribution against a backdrop of related work (in Section 2). The cornerstone of our approach is the problem of solving Unique Sink Orientations (USOs) [Szabó and Welzl, 2001], which generalises the problem of MDP planning. As we shall see, 2-action MDPs give rise to *Acyclic* USOs (abbreviated AUSOs). In Section 3,

[1] It is possible to derive weak bounds that are *polynomial* in n and k [Littman *et al.*, 1995].

we review existing algorithms for solving (A)USOs. In Section 4, we introduce Recursive (A)USOs, which generalise (A)USOs and facilitate improved bounds for general MDPs. We conclude with a discussion in Section 5.

2 Related Work and Contribution

The tightest strong bounds currently known for MDP planning are summarised in Table 1. The tightest bound for $k = 2$ is of the form $\text{poly}(n) \cdot 1.6059^n$. This bound is shown for the Fibonacci Seesaw algorithm, proposed by Szabó and Welzl [2001] for solving Unique Sink Orientations (USOs). A USO is a hypercube with directed edges, such that each face of the hypercube has exactly one sink. For an n-dimensional USO, the Fibonacci Seesaw algorithm *evaluates* at most 1.6059^n of the 2^n vertices in order to *solve* the USO: that is, find its (global) sink. Policies for 2-action MDPs can be interpreted as vertices of an Acyclic USO (AUSO), whose sink corresponds to an optimal policy. Details of this relationship are provided in Section 3.

Policy Iteration (PI) is a contrasting—and arguably more popular—approach to MDP planning. A PI algorithm starts with some initial policy, and repeatedly updates it to a dominating one until an optimal policy is reached. The update is easy to effect: *policy evaluation* is a polynomial operation that can identify *improving* actions in *improvable* states; switching to any improving action in one or more improvable states necessarily yields a dominating policy. Although the most common variant of PI is Howard's PI [Howard, 1960] (under which *every* improvable state is switched), the tightest strong worst-case bound *proven* to date for 2-action MDPs belongs to Batch-Switching PI (BSPI) [Kalyanakrishnan *et al.*, 2016a]. Under BSPI, improvable states are switched in fixed-size batches. The best known bound on the number of iterations, achieved with a batch size of 7, is 1.6479^n.

The main technical gap between the cases of $k = 2$ and $k \geq 3$ is that in the former, an improvable state will have exactly one improving action: there is no choice of actions to which to switch. For $k \geq 3$, Mansour and Singh [1999] show that regardless of how actions are picked in improvable states, Howard's PI takes at most $O(k^n/n)$ iterations. In spite of a constant-factor improvement subsequently shown by Hollanders *et al.* [2014], this upper bound remains only a *linear* improvement over the trivial bound of k^n iterations. Thus, $\text{poly}(n, k) \cdot k^n$ is currently the tightest strong worst-case running time bound for the PI family. Interestingly, it is also the tightest across all planning algorithms.

As shown in Table 1, the tightest bound on the *expected* number of iterations of PI, for $k \geq 3$, has only a logarithmic dependence on k in the base of the exponent. The key to obtaining this improvement is the use of randomisation in picking improving actions [Kalyanakrishnan *et al.*, 2016b]. If we look beyond PI, we find even *subexponential* bounds on the expected running time of MDP planning. Bounds of the form $\text{poly}(n, k) \cdot \exp(O(\sqrt{n \log(n)}))$ [Matoušek *et al.*, 1996] follow directly from posing MDP planning as a linear program with n variables and nk constraints [Littman *et al.*, 1995]. The special structure that results when $k = 2$ admits an even tighter bound of $\text{poly}(n) \cdot \exp(2\sqrt{n})$ [Gärtner, 2002].

Table 1: Summary of tightest known strong upper bounds for MDP planning. Polynomial factors of n and k are omitted; improvements reported in this paper are shown with arrows.

k	Algorithms	Worst-case	Expected
2	All	1.6059^n	$\exp(2\sqrt{n})$
	PI	1.6479^n	1.6479^n
≥ 3	All	$k^n \rightarrow k^{0.6834n}$	$\exp(O(\sqrt{n \log(n)}))$
	PI	$k^n \rightarrow k^{0.7207n}$	$(2 + \ln(k-1))^n$

We must make clear that our focus here is on the dependence of the upper bound on n when k is fixed. Alternatively, if we fix n, the linear programming route can yield strong worst-case bounds that are *linear* in k: for example, $2^{O(2^n)} \cdot k$ [Megiddo, 1984] and $n^{O(n)} \cdot k$ [Chazelle and Matousek, 1996]. It must also be noted that for *deterministic* MDPs, strong worst-case bounds of the form $\text{poly}(n, k)$ are indeed possible [Madani *et al.*, 2010; Post and Ye, 2013].

We are unaware of any super-polynomial *lower* bounds for MDP planning (note that every algorithm must take at least $\Omega(n^2 k)$ time to read in the input MDP. A lower bound of $\Omega(n)$ iterations has been shown for Howard's PI on 2-action MDPs [Hansen and Zwick, 2010]. Exponential lower bounds have been shown for other variants of PI [Melekopoglou and Condon, 1994], and for Howard's PI when the number of actions depends linearly on the number of states [Fearnley, 2010; Hollanders *et al.*, 2012]. We do not know of any explicit lower bounds on the complexity of PI for fixed $k \geq 3$.

The scope of this paper is restricted to improving the $\text{poly}(n, k) \cdot k^n$ strong worst-case upper bound shown by Mansour and Singh [1999] and Hollanders *et al.* [2014]. We present *exponential* improvements over this bound, which we achieve by generalising (A)USOs, the Fibonacci Seesaw algorithm, and the BSPI algorithm all to work with $k \geq 3$. Thereby we obtain a running time bound of $\text{poly}(n, k) \cdot k^{0.7207n}$ for a PI algorithm, and a tighter bound of $\text{poly}(n, k) \cdot k^{0.6834n}$ for a non-PI algorithm. In both cases, our strategy is to solve a k-action problem by solving and combining the solutions of multiple $(k/2)$-sized problems. We present this recursive strategy in Section 4, but first we review existing approaches for planning in MDPs and (A)USOs.

3 MDPs and (A)USOs

This section provides a brief review of MDPs, USOs, and AUSOs; relationships between them; and corresponding algorithms—which serve as building blocks for our contributions in the next section. A detailed exposition on MDPs is available from Puterman [1994], and one on USOs and AUSOs from Szabó and Welzl [2001].

3.1 Definitions and Properties

MDP. An MDP $M = (S, A, R, T, \gamma)$ comprises a set of states S and a set of actions A, with $|S| = n$ and $|A| = k$. Taking action a from state s yields a bounded, real-valued reward of $R(s, a)$, and with probability $T(s, a, s')$, sets the next state to be $s' \in S$. The discount factor $\gamma \in [0, 1)$ encodes the importance of future rewards. For a (deterministic, stationary, and Markovian) *policy* $\pi : S \rightarrow A$, the value function $V^\pi : S \rightarrow \mathbb{R}$ gives the expected long-term reward accrued by

following π (starting from each given state). For $s \in S$,

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

Let Π be the set of all policies corresponding to MDP M. We are guaranteed [Bellman, 1957] that Π contains a policy $\pi^{\star}$ such that for $s \in S, \pi \in \Pi$, $V^{\pi^{\star}}(s) \geq V^{\pi}(s)$. The MDP planning problem is precisely that of finding an optimal policy for a given MDP $M = (S, A, R, T, \gamma)$.

When we set out to design MDP planning algorithms, we shall find use for one other function, $Q^{\pi} : S \times A \to \mathbb{R}$, which is the *action value function* of π. For $s \in S, a \in A$, $Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^{\pi}(s')$. $Q^{\pi}(s, a)$ is the expected long-term reward accrued by starting at state s, executing action a *once*, and *thereafter* following π.

The recursive definition of $V^{\pi}(\cdot)$ indicates that it can be obtained by solving a system of n linear equations in n unknowns. $Q^{\pi}(\cdot, \cdot)$ can thereafter be obtained by a linear operation for each state-action pair. We shall refer to the computation of $V^{\pi}(\cdot)$ and $Q^{\pi}(\cdot, \cdot)$ for a given policy π as *policy evaluation*. This step needs no more than $O(n^3 + n^2 k)$ time with infinite-precision arithmetic. In Section 3.3, we describe how policy evaluation can be used to ultimately find an optimal policy.

USO. An n-dimensional *hypercube* C is a graph with 2^n vertices. Each vertex is *labeled* by an n-length binary string. Let $v(i)$ refer to the i^{th} bit in the label of vertex v. In the hypercube, an edge exists between vertices u and v if and only if $u \oplus v = 2^r$ for some $0 \leq r < n$, where $\oplus$ is the bitwise XOR operation. We say in this case that u and v differ in the r^{th} dimension. The corresponding edge, which lies along the r^{th} dimension in the hypercube, is said to have *dimension* r. Figure 1(a) shows a 3-dimensional hypercube; the edge 110-111 has dimension 0, while the edge 001-101 has dimension 2.

A d-dimensional *face* of the hypercube C is a subgraph induced by C with 2^d vertices, which themselves form a d-dimensional hypercube. Any such "d-face" consists of vertices whose labels all have the same bits in some fixed $n - d$ dimensions, and vary over the the remaining d dimensions. In other words, a d-face can be characterised by a set of $n - d$ ordered pairs (p, b), wherein $p \in \{0, 1, \ldots, n-1\}$ represents a dimension, and $b \in \{0, 1\}$ a corresponding bit. Two d-faces characterised by the sets of ordered pairs F and G are called *antipodal* in C if for every $(p, b) \in F$, we have $(p, \neg b) \in G$. In the hypercube in Figure 1(a), the 0-face containing 100 is antipodal to the one containing 011; the 2-face containing 000, 100, 110, and 010 is antipodal to the one containing 001, 101, 111, and 011.

An *orientation* ψ specifies a direction for each edge of a hypercube C. The pair (C, ψ) is called a Unique Sink Orientation (USO) if *every* face of C has a unique *sink* with respect to the orientation ψ. A sink is a vertex with no outgoing edges. Among the four oriented hypercubes in Figure 1, the one in (a) is *not* a USO since both 001 and 111 are sinks, and nor is the one in (b), in which the face containing 000, 010, 011, and 001 has no sink. The oriented hypercubes in (c) and (d) are both USOs.

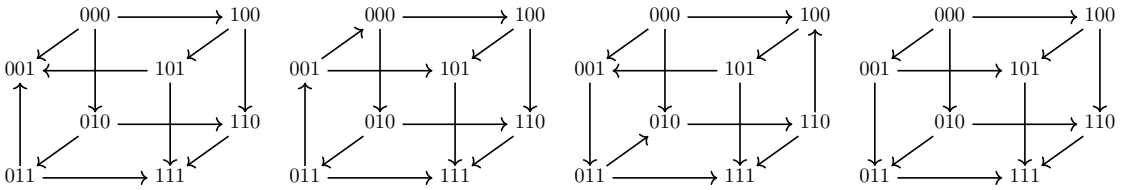
An *outmap* s_{ψ} of a USO (C, ψ) is a function from the set of vertices to the powerset of $\{0, 1, \ldots, n-1\}$. It is induced by ψ and maps each vertex of C to the set of the dimensions of the outgoing edges from that vertex. For example, in Figure 1(c), $s_{\psi}(011) = \{0, 2\}$, and $s_{\psi}(111) = \emptyset$. It is a property of USOs that their outmaps are *bijective*; in other words, no two vertices have the set of dimensions of their outgoing edges exactly the same. This result follows immediately from the following lemma, which we shall prove.

Lemma 1. *Let ψ be an orientation of a hypercube C and R a subset of the set of edge dimensions $\{0, 1, \ldots, n-1\}$. Let ψ^R denote the orientation that gives each edge with dimension in R the opposite direction to ψ, and each edge with dimension not in R the same direction as ψ. If (C, ψ) is a USO, then (C, ψ^R) is also a USO.*

Proof. It suffices to show that if (C, ψ) is a USO, then so is $(C, \psi^{\{a\}})$, for $a \in \{0, 1, \ldots, n-1\}$ (since the dimensions in R can be flipped in sequence, one at a time). Consider an arbitrary d-face F in C. (1) If F does not contain an edge with dimension a, clearly $\psi^{\{a\}}$ induces the same graph on F as ψ, implying that F continues to have a unique sink. (2) If F contains an edge with dimension a, consider F_1 and F_2, two antipodal $(d-1)$-faces of F that are connected only by edges with dimension a. Let u_1 be the unique sink of F_1, and u_2 be the unique sink of F_2—both under ψ and ψ^a. Clearly one of them—without loss of generality, assume u_1— must be the unique sink of F under ψ, implying u_1 has an *incoming* edge with dimension a, and u_2 has an *outgoing* edge with dimension a. Reversing the direction of all edges with dimension a would therefore make u_2 the unique sink of F. □

Lemma 2. *The outmap s_{ψ} of a unique sink orientation (C, ψ) is bijective.*

Proof. Let u and v be distinct vertices in C. If $s_{\psi}(u) = s_{\psi}(v) = R$, then u and v would both be sinks in the oriented hypercube (C, ψ^R), which contradicts Lemma 1. Also, the size of the vertex set of C (2^n) equals the number of possible sets of outgoing edges (2^n); hence s_{ψ} must be a bijection. □



(a) Not a USO (2 sinks). (b) Not a USO (0-sink face). (c) USO with directed cycle. (d) AUSO.

Figure 1: Hypercubes with oriented edges. Explanations are provided in the text.

AUSO. A USO (C, ψ) in which the graph induced by C and ψ does not contain any directed cycles is called an Acyclic USO (AUSO). The USO in Figure 1(c) is not an AUSO, since it contains the directed cycle 001-011-010-110-100-101-001. Figure 1(d) shows an AUSO.

3.2 2-action MDPs Induce AUSOs

We show that every 2-action MDP naturally gives rise to a corresponding AUSO, such that the sink of the AUSO immediately yields an optimal policy for the MDP. We begin by considering the Policy Improvement Theorem, which is a well-known result on MDPs [Howard, 1960]. Below we present a slight modification of the version given by Kalyanakrishnan *et al.* [2016a]; the proof is available in standard textbooks [Bertsekas, 2012].

Definition 3 ($\succeq, \succ$)**.** *For functions $X : S \to \mathbb{R}$, $Y : S \to \mathbb{R}$, (1) we define $X \succeq Y$ if $\forall s \in S : X(s) \geq Y(s)$, and (2) we define $X \succ Y$ if $X \succeq Y$ and $\exists s \in S : X(s) > Y(s)$. For policies $\pi_1, \pi_2 \in \Pi$, (1) we define $\pi_1 \succeq \pi_2$ if $V^{\pi_1} \succeq V^{\pi_2}$, and (2) we define $\pi_1 \succ \pi_2$ if $V^{\pi_1} \succ V^{\pi_2}$.*

Theorem 4. *Consider policies $\pi, \pi' \in \Pi$.*
(1) If $\forall s \in S : Q^{\pi}(s, \pi'(s)) \geq Q^{\pi}(s, \pi(s))$, then $\pi' \succeq \pi$.
(2) If $\pi' \succeq \pi$ and if $\exists s \in S : Q^{\pi}(s, \pi'(s)) > Q^{\pi}(s, \pi(s))$, then $\pi' \succ \pi$.

To go forward with our construction, we assume that a tie-breaking rule over policies is available in the form of a fixed total order $\gg$ on the set of policies Π.

Construction. Consider an MDP $M = (S, A, R, T, \gamma)$ with $S = \{s_1, s_2, \ldots, s_n\}$ and $A = \{0, 1\}$. We may represent policies for this 2-action MDP as n-bit strings, with the i^{th} bit specifying the action for s_i, $i \in \{1, 2, \ldots, n\}$. These bit strings are vertices in an n-dimensional hypercube C. Consider two adjacent vertices in the hypercube, u and v, which differ in exactly one dimension, i. Under ψ, we define the direction of the edge between u and v as follows: If (1) $Q^u(s_i, u(i)) < Q^u(s_i, v(i))$ or (2) $Q^u(s_i, u(i)) = Q^u(s_i, v(i))$ and $v \gg u$, then the edge direction is from u to v ($i \in s_\psi(u)$); otherwise the edge direction is from v to u ($i \in s_\psi(v)$).

Since u and v differ in exactly one dimension, an application of Theorem 4 shows that (1) $Q^u(s_i, u(i)) < Q^u(s_i, v(i))$ is equivalent to $Q^v(s_i, u(i)) < Q^v(s_i, v(i))$, and (2) $Q^u(s_i, u(i)) = Q^u(s_i, v(i))$ is equivalent to $Q^v(s_i, u(i)) = Q^v(s_i, v(i))$. Hence, the edge direction between u and v can be obtained by performing policy evaluation on *either* of these vertices. The outmap of a vertex will correspond to the set of improvable states of that policy. In fact, in order to use AUSO solution techniques for planning in M (coming up in Section 3.3), there is no need to explicitly construct (C, ψ). Rather, we will only have to compute the outmaps of—which amounts to performing policy evaluation on—a small subset of vertices in (C, ψ). Before proceeding to the planning algorithms, we establish the correctness of our construction.

Theorem 5. *(C, ψ), constructed from M, is an AUSO.*

Proof. For $d \in \{0, 1, \ldots, n\}$, consider an arbitrary d-face F in C, in which the vertices (policies) have some $n - d$ bits fixed. Without loss of generality, assume that actions at $s_1, s_2, \ldots, s_{n-d}$ are fixed: thus, policies in F are of the form xy, where x is a fixed $(n - d)$-bit string, and y is some d-bit string. We establish that F is an AUSO by showing that it cannot have more than one sink, and it cannot have a cycle. With no cycle, note that it must have at least one sink.

(1) Let policy $\pi_1 = xy_1$ be a sink in F with $y_1 \in \{0, 1\}^d$. From our construction and from Theorem 4, we get that for every policy π in F, $\pi_1 \succ \pi$ or ($V^{\pi_1} = V^{\pi}$ and $\pi_1 \gg \pi$). Now, if there is also a policy $\pi_2 = xy_2$ that is a sink, with $y_2 \in \{0, 1\}^d$ and $y_1 \neq y_2$, then we get "$\pi_2 \succ \pi_1$ or ($V^{\pi_1} = V^{\pi_2}$ and $\pi_2 \gg \pi_1$)" and "$\pi_1 \succ \pi_2$ or ($V^{\pi_1} = V^{\pi_2}$ and $\pi_1 \gg \pi_2$)". Clearly, both statements cannot be true. (2) If F contains a cycle, it would mean that there exist vertices $\pi_1 \neq \pi_2$ in the cycle such that ($\pi_1 \succ \pi_2$ and $\pi_2 \succ \pi_1$) or ($\pi_1 \gg \pi_2$ and $\pi_2 \gg \pi_1$). This is also not possible. □

It is also clear that the unique sink of (C, ψ) is an optimal policy for M.

3.3 Algorithms for AUSOs and USOs

Policy Iteration. PI [Howard, 1960] is usually discussed in the context of MDP planning, but it is not hard to see that it can be generalised to AUSOs. At every iteration, a PI algorithm for an AUSO (C, ψ) only keeps track of a single vertex and its outmap. If u is the current vertex, PI computes its outmap $s_\psi(u)$. If $s_\psi(u) = \emptyset$, then clearly u is the unique sink of C. If not, consider the $|s_\psi(u)|$-dimensional face F of C containing u and vertices that differ from u only in the dimensions present in $s_\psi(u)$. Since F is an AUSO, and u is an "anti-sink" (all edges outgoing) in F, by Lemma 1, there must exist a directed path from u to every other vertex in F. Thus, PI can change its current vertex from u to any other vertex v in F. The exact rule to choose v, when given u and $s_\psi(u)$, is what distinguishes one PI algorithm from another. Regardless, Since there is always a directed path from the current vertex to the next, and since (C, ψ) is acyclic, PI is guaranteed to eventually reach the sink of C.

Existing analyses of the complexity of PI on MDPs [Mansour and Singh, 1999; Kalyanakrishnan *et al.*, 2016a] carry over quite easily to AUSOs. Although the most common approach is to pick v to be the antipode of u in F [Howard, 1960], the best upper bounds were shown recently for BSPI [Kalyanakrishnan *et al.*, 2016a], in which u and v vary in at most b dimensions, where b is a parameter to BSPI. For $b = 7$, BSPI is shown to take at most 1.6479^n vertex evaluations to find a sink in an n-dimensional AUSO.

Fibonacci Seesaw. Unlike PI, which traverses a contiguous sequence of improving faces, the Fibonacci Seesaw (FS) algorithm [Szabó and Welzl, 2001] evaluates vertices from *antipodal* faces. In fact, FS can find the unique sink of a USO (C, ψ), whereas PI is only meant to work on AUSOs.

FS maintains the following invariant for i, incremented from 0 to $n - 1$: there are two antipodal i-faces F and G, with their sinks u and v, respectively, *already evaluated*. This invariant can be obtained for $i = 0$ by 2 vertex evaluations. If we have reached $i = n - 1$, then we are done, since either u or v must be the sink of the whole cube.

In order to proceed from i to $i+1$, the algorithm chooses some dimension $b \in s_\psi(u) \triangle s_\psi(v)$ ($s_\psi(u)$ symmetric difference $s_\psi(v)$), which exists because s_ψ is bijective. Without loss of generality, assume that $b \in s_\psi(v)$. We can then extend F along b to get an $(i+1)$-face F' of which u remains the sink, since $b \notin s_\psi(u)$. The $(i+1)$-face G' antipodal to F' clearly has G as a face. To find the sink of G', we only need find the sink of the i-face antipodal to G in G'. We do so recursively. Observe that if $t(d)$ is the number of vertex evaluations performed to find the sink of a d-face, we can extend the invariant from i to $i+1$ with $t(i)$ vertex evaluations. The overall recurrence becomes: $t(0) = 1$, and for $d \geq 1$, $t(d) \leq 2 + t(0) + t(1) + \cdots + t(d-2)$. The recurrence is upper bounded by the one for Fibonacci numbers, and solves to $t(n) = O(1.6181^n)$. A slight change to the FS algorithm, by which a special procedure is used to solve 4-dimensional hypercubes in 7 evaluations, improves the bound to $O(1.6059^n)$ [Szabó and Welzl, 2001].

4 Recursive (A)USOs for $k \geq 3$

In the previous section, we reviewed existing literature showing that 2-action MDPs can be solved by applying methods to solve AUSOs and USOs. Although 2-action MDPs encapsulate the very essence of sequential decision making, several real-world problems inherently present an agent with more than 2 choices of action. The focus of this paper is on deriving improved strong worst-case bounds for the general case ($k \geq 3$). In order to do so, we introduce two new constructs: Recursive USOs (RUSOs) and Recursive AUSOs (RAUSOs). As shown in Figure 2 , the RAUSO problem generalises both k-action MDP planning (for general k) and the AUSO problem. The RUSO problem generalises all the others we have discussed.

Recursive USO. A Recursive USO (RUSO) of *level* 1 (a 1-RUSO) is a USO. For $l \geq 2$, an RUSO (C, ψ) of level l (an l-RUSO) is a graph over an n-dimensional hypercube C, wherein each vertex of C is an $(l-1)$-RUSO. Each edge of the hypercube is oriented according to ψ, and follows the constraint that every face of C has exactly one sink.

Recursive AUSO. A Recursive AUSO (RAUSO) of *level* 1 (a 1-RAUSO) is an AUSO. For $l \geq 2$, an RAUSO (C, ψ) of level l (an l-RAUSO) is a graph over an n-dimensional hypercube C, wherein each vertex of C is an $(l-1)$-RAUSO. Each edge of the hypercube is oriented according to ψ, and follows the constraints that every face of C has exactly one sink, and ψ does not induce any directed cycles.

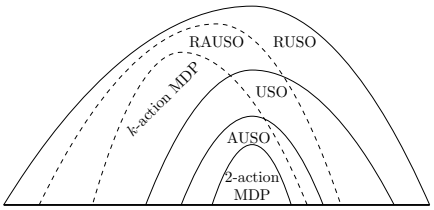


Figure 2: Containment relationship between problem classes discussed. RAUSO and RUSO are introduced in this paper.

In general, we do not expect the outmaps of vertices in an R(A)USO to be available explicitly. Rather, to compute the outmap of a vertex in an l-R(A)USO, it is necessary to solve the corresponding $(l-1)$-R(A)USO. This is so in the case of MDPs, which we see next.

4.1 k-action MDPs Induce RAUSOs

Consider an MDP M with n states $s_1, s_2, \ldots, s_n$, and $k \geq 2$ actions. We find it convenient to assume that k is a power of 2: that is, $k = 2^l$ from some $l \geq 1$. We may therefore take actions in M to be l-bit strings (which we place inside angle brackets): hence, $A = \{\langle\{0,1\}^l\rangle\}$. Relaxing the assumption that k is a power of 2 does not significantly change our bounds, as we discuss at the end of this section.

We build a complete binary tree with the k actions of M as the leaves. The depth of the tree is l: the leaves are at level 0, and the root node at level l. For $j \in \{1, 2, \ldots, l\}$, the j^{th} level contains 2^{l-j} internal nodes, each containing an $(l-j)$-bit string that prefixes those of its two children. Figure 3 shows such an "action tree" for $k = 8$ (that is, $l = 3$).

For $j \in \{0, 1, \ldots, l\}$, a level-$j$ *composite policy* is an n-length string of the form $L_1 L_2 \ldots L_n$, where for $i \in \{1, 2, \ldots, n\}$, L_i is the label of a node in the j^{th} level of the l-action tree. Thus, for $n = 3$, $\langle 0\rangle\langle 0\rangle\langle 1\rangle$ is an example of a level-2 composite policy, and $\langle 010\rangle\langle 011\rangle\langle 110\rangle$ is an example of a level-0 composite policy. We refer to composite policies $L_1 L_2 \ldots L_n$ and $L'_1 L'_2 \ldots L'_n$ as being ancestors, descendants, parents, or children of each other if the corresponding relation holds for *every* pair of nodes labeled L_i and L'_i, for $i \in \{1, 2, \ldots, n\}$. We also refer to level-0 composite policies as *policies*, since they are equivalent.

In essence, a composite policy p restricts the set of actions available from each state. It follows that there must be a dominant policy among the level-0 descendants of p. To see why, consider a new MDP M' that has the same set of states and discount factor as our original MDP M, but which only has the corresponding level-0 descendants of p as actions at each state. The reward and transition functions for these actions in M' are the same as those in M. By Theorem 4, M' must have an optimal policy. In other words, p must have a level-0 descendant π such that for every level-0 descendant π' of p, $\pi \succeq \pi'$ and $(V^\pi = V^{\pi'} \implies \pi \gg \pi')$. We refer to π as the *optimiser* of p. To find an optimal policy for M, we exploit the fact that the optimiser of a composite policy can be obtained by comparing the optimisers of its children. Although each composite policy of level 1 through l has 2^n children, we can use the structure underlying them and evaluate only a small subset of children, each evaluated recursively. Indeed

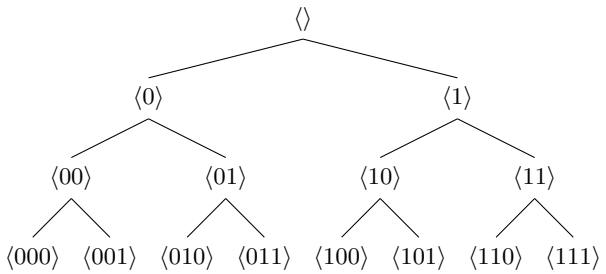


Figure 3: 3-level action tree (explained in text).

the children naturally become the vertices of an RAUSO, as we describe next.

Given M, we construct an l-RAUSO (C, ψ), wherein each vertex of C corresponds to a level-$(l-1)$ composite policy. Edges exist between level-$(l-1)$ composite policies that differ in exactly one dimension. Let p_1 and p_2 be neighbours that differ only in the i^{th} dimension (corresponding to state s_i), and let optimiser$(p_1) = \pi_1$. Now, consider the (l-bit) actions prefixed by the string in the i^{th} dimension of p_2. Among such actions a, let $a_{\max}$ be one that maximises $Q^{\pi_1}(s_i, a)$. If $Q^{\pi_1}(s_i, a_{\max}) > Q^{\pi_1}(s_i, \pi_1(s_i))$, or if $(Q^{\pi_1}(s_i, a_{\max}) = Q^{\pi_1}(s_i, \pi_1(s_i))$ and a fixed tie-breaking rule picks p_2 over p_1), then ψ orients the edge from p_1 to p_2. Otherwise ψ orients the edge from p_2 to p_1. As before, we are guaranteed to arrive at the same orientation if consider optimiser(p_2) to begin with.

Observe that to determine the outmap of p_1, we need to compute $\pi_1 = $ optimiser(p_1). We may do so recursively, since each vertex p of C is itself a similarly-constructed $(l-1)$-RAUSO, whose vertices are children of p. 1-RAUSOs are AUSOs, and have no children.

Lemma 6. *(C, ψ), constructed from M, is an RAUSO.*

Proof. If the edge between p_1 and p_2 is directed from p_1 to p_2, it follows from our construction and Theorem 4 that optimiser$(p_2) \succ$ optimiser(p_1) or ($V^{\text{optimiser}(p_2)} = V^{\text{optimiser}(p_1)}$ and p_2 wins a tie over p_1). If we apply the proof of Theorem 5 to the *optimisers* of the vertices at each level, we obtain that (C, ψ) is an RAUSO. □

4.2 Algorithms for RAUSOs and RUSOs

Fibonacci Seesaw. By definition, the vertices at each level of an RUSO induce a USO, and so we may continue to apply the FS algorithm to RUSOs. The only change to account for is that to evaluate a vertex in an l-RUSO, we have to solve the corresponding $(l-1)$-RUSO first. We may do so recursively.

Policy Iteration. The PI algorithm for AUSOs can also be naturally extended to the setting of Recursive AUSOs. After evaluating a vertex v at level j of the RAUSO, a PI algorithm, as before, will only evaluate a vertex in the face formed by extending v along all its outgoing dimensions. Actual policy evaluations (solving for $V(\cdot)$ and $Q(\cdot, \cdot)$) are only performed to evaluate the vertices of 1-RAUSOs. For j-RAUSOs, $j \geq 2$, vertex-evaluation proceeds by picking a $(j-1)$-RAUSO to evaluate (recursively) next.

Interestingly, this recursive algorithm can be made to conform with the definition of PI for MDPs: that is, if the algorithm proceeds from evaluating policy π to policy π' (through a chain of book-keeping involving RAUSOs of level 1 and higher), it can be ensured that (1) π and π' differ only on states in the improvable set of π, and (2) any different actions taken by π' are improving actions. Let $S^{\text{improvable}}$ be the set of improvable states for π, and let $S^{\text{switch}} \subseteq S^{\text{improvable}}$ be the set of states that a particular PI algorithm would pick to improve. For example, Howard's PI would set $S^{\text{switch}} = S^{\text{improvable}}$; BSPI would ensure $|S^{\text{switch}}|$ is at most the batch size. It can be verified that our recursive algorithm implies the following switching rule: *Over all $s \in S^{\text{switch}}$, find the minimum distance in the l-level action tree between $\pi(s)$ and an improving action. Switch all (and only) the states in S^{switch} attaining this minimum; in each case switch to the closest improving action (breaking ties using a fixed tie-breaking rule).*

4.3 Complexity

Let $t(n, l)$ denote the number of vertex evaluations performed to solve an l-R(A)USO with n vertices at each level. For $l \geq 2$, both the FS and PI algorithms specified in this section give rise to the recurrence $t(n, l) = t(n, 1)t(n, l-1)$, which evaluates to $t(n, l) = t(n, 1)^l$. Thus, if we use FS to solve an n-state, k-action MDP ($k = 2^l$), the number of policy evaluations is upper-bounded by $(O(1.6059^n))^{\log_2(k)} < \text{poly}(n, k) \cdot k^{0.6834n}$. If we pick BSPI [Kalyanakrishnan *et al.*, 2016a] as an underlying PI algorithm (to decide which states to switch) and apply our rule for picking actions, the number of iterations is at most $(1.6479^n)^{\log_2(k)} < k^{0.7207n}$.

If k is not a power of 2, we can implement our action tree as an incomplete binary tree (some nodes would have only one child). This could potentially yield a faster running time than the easier approach of "rounding up" the underlying MDP to have $k^\star > k$ actions, where $k^\star$ is a power of 2. Our bounds would then depend on $k^\star$ in place of k—but still improve exponentially upon existing bounds for $k \geq 3$.

5 Conclusion

We present the first exponential improvements over the trivial strong worst-case upper bound on the running time of planning in n-state, k-action MDPs (under the infinite-precision arithmetic computational model). Our analysis makes use of existing results for n-state, 2-action MDPs, which also apply to AUSOs and USOs.

Although it is straightforward to "flatten" an n-state, k-action MDP into an $n(k-1)$-state, 2-action MDP, such a reduction does not yield better complexity bounds. In some settings—for example, constraint satisfaction problems—such transformations can indeed speed up the computation [Scheder, 2011]. For MDPs, the primary tool we devise is a method to recursively halve the set of actions at every state such that eventually, known methods can be applied. We generalise the Fibonacci Seesaw algorithm [Szabó and Welzl, 2001] to obtain a running-time bound of $\text{poly}(n, k) \cdot k^{0.6834n}$, and the BSPI algorithm [Kalyanakrishnan *et al.*, 2016a] to obtain a PI algorithm that needs at most $k^{0.7207n}$ iterations.

At present, we are not aware of applications of RUSOs and RAUSOs—which are objects we introduce in this paper—outside of MDP planning. It would be worth considering their relevance to other applications of USOs and AUSOs, of which there are several [Jaggi, 2006; Rüst, 2007]. Preliminary experiments have yet to demonstrate any significant *practical* benefits for the algorithms we have presented in this paper, suggesting that our improved running-time bounds are still extremely loose for typical MDPs.

Acknowledgments

Supratik Chakraborty, Sundar Vishwanathan, and anonymous reviewers provided the authors several useful comments.

References

[Bellman, 1957] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1^{st} edition, 1957.

[Bertsekas, 2012] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 4^{th} edition, 2012.

[Chazelle and Matousek, 1996] Bernard Chazelle and Jirí Matousek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *Journal of Algorithms*, 21(3):579–597, 1996.

[Fearnley, 2010] John Fearnley. Exponential lower bounds for policy iteration. In *Proceedings of the Thirty-seventh International Colloquium on Automata, Languages and Programming (ICALP 2010)*, pages 551–562. Springer, 2010.

[Gärtner, 2002] Bernd Gärtner. The random-facet simplex algorithm on combinatorial cubes. *Random Structures and Algorithms*, 20(3):353–381, 2002.

[Hansen and Zwick, 2010] Thomas Dueholm Hansen and Uri Zwick. Lower bounds for Howard's algorithm for finding minimum mean-cost cycles. In *Proceedings of the Twenty-second International Symposium on Algorithms and Computation (ISAAC 2011)*, pages 425–426. Springer, 2010.

[Hollanders *et al.*, 2012] Romain Hollanders, Balázs Gerencsér, and Jean-Charles Delvenne. The complexity of policy iteration is exponential for discounted Markov decision processes. In *Proceedings of the Fifty-first IEEE Conference on Decision and Control (CDC 2012)*, pages 5997–6002. IEEE, 2012.

[Hollanders *et al.*, 2014] Romain Hollanders, Balázs Gerencsér, Jean-Charles Delvenne, and Raphaël M. Jungers. Improved bound on the worst case complexity of policy iteration, 2014. URL: http://arxiv.org/pdf/1410.7583v1.pdf.

[Howard, 1960] Ronald A. Howard. *Dynamic programming and Markov processes*. MIT Press, 1960.

[Jaggi, 2006] Martin Jaggi. Linear and quadratic programming by unique sink orientations, 2006. Diploma thesis in Mathematics, ETH Zürich. URL: http://www.m8j.net/data/List/Files-106/DA.pdf.

[Kalyanakrishnan *et al.*, 2016a] Shivaram Kalyanakrishnan, Utkarsh Mall, and Ritish Goyal. Batch-switching policy iteration. In *Proceedings of the Twenty-fifth International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pages 3147–3153. AAAI Press, 2016.

[Kalyanakrishnan *et al.*, 2016b] Shivaram Kalyanakrishnan, Neeldhara Misra, and Aditya Gopalan. Randomised procedures for initialising and switching actions in policy iteration. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016)*, pages 3145–3151. AAAI Press, 2016.

[Littman *et al.*, 1995] Michael L. Littman, Thomas L. Dean, and Leslie Pack Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI 1995)*, pages 394–402. Morgan Kaufmann, 1995.

[Madani *et al.*, 2010] Omid Madani, Mikkel Thorup, and Uri Zwick. Discounted deterministic Markov decision processes and discounted all-pairs shortest paths. *ACM Transactions on Algorithms*, 6(2):33:1–33:25, 2010.

[Mansour and Singh, 1999] Yishay Mansour and Satinder Singh. On the complexity of policy iteration. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI 1999)*, pages 401–408. Morgan Kaufmann, 1999.

[Matoušek *et al.*, 1996] Jiří Matoušek, Micha Sharir, and Emo Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, 1996.

[Megiddo, 1984] Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, 1984.

[Melekopoglou and Condon, 1994] Mary Melekopoglou and Anne Condon. On the complexity of the policy improvement algorithm for Markov decision processes. *INFORMS Journal on Computing*, 6(2):188–192, 1994.

[Post and Ye, 2013] Ian Post and Yinyu Ye. 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)*, pages 1465–1473. Morgan Kaufmann, 2013.

[Puterman, 1994] Martin L. Puterman. *Markov Decision Processes*. Wiley, 1994.

[Rüst, 2007] Leonard Yves Rüst. The P-Matrix Complementarity Problem: Generalizations and specializations, 2007. Dissertation for the degree of Doctor of Sciences, Swiss Federal Institute of Technology. URL: https://www.inf.ethz.ch/personal/emo/DoctThesisFiles/ruest07.pdf.

[Scheder, 2011] Dominik Alban Scheder. *Algorithms and Extremal Properties of SAT and CSP*. PhD thesis, Swiss Federal Institute of Technology Zurich, 2011. URL: http://basics.sjtu.edu.cn/ dominik/publications/Dissertation-Dominik-Scheder.pdf.

[Szabó and Welzl, 2001] Tibor Szabó and Emo Welzl. Unique sink orientations of cubes. In *Proceedings of the Forty-second Annual Symposium on Foundations of Computer Science (FOCS 2001)*, pages 547–555. IEEE, 2001.