The Smoothed Complexity of Policy Iteration for Markov Decision Processes

Miranda Christ
Columbia University
USA
mchrist@cs.columbia.edu

Mihalis Yannakakis
Columbia University
USA
mihalis@cs.columbia.edu

ABSTRACT

We show subexponential lower bounds (i.e., $2^{\Omega(n^c)}$) on the smoothed complexity of the classical Howard's Policy Iteration algorithm for Markov Decision Processes. The bounds hold for the total reward and the average reward criteria. The constructions are robust in the sense that the subexponential bound holds not only on the average for independent random perturbations of the MDP parameters (transition probabilities and rewards), but for all arbitrary perturbations within an inverse polynomial range. We show also an exponential lower bound on the worst-case complexity for the simple reachability objective.

CCS CONCEPTS

• **Mathematics of computing** → *Combinatorial optimization.*

KEYWORDS

Policy Iteration, Smoothed Analysis

ACM Reference Format:
Miranda Christ and Mihalis Yannakakis. 2023. The Smoothed Complexity of Policy Iteration for Markov Decision Processes. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC '23), June 20–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3564246.3585220

1 INTRODUCTION

Markov Decision Processes (MDP) are a fundamental model for dynamic optimization in a stochastic environment with applications in many areas, including operations research, artificial intelligence, game theory, robotics, control theory, and verification. They were originally introduced by Bellman [4] and have been studied extensively since then; see [10, 22, 25] for general expositions. We will define formally MDPs in Section 2, but we give here an informal brief description. MDPs are an extension of Markov chains with an agent, who can affect the evolution of the chain. An MDP consists of a set of states and a set of possible actions that the agent can take at each state, where each action yields a reward to the agent, and results in a probabilistic transition to a new state. Execution of the MDP starts at some state and then moves (probabilistically) in discrete steps from state to state according to the action selected by the agent in each step. The problem is to find an optimal *policy* for the agent, i.e. choice of action in each step, that maximizes a desired objective, such as the expected total reward collected during the execution. Although the agent is allowed in each step to use randomization in their choice of action and to base their decision on the complete past history, it is known that there is always a so-called *positional* optimal policy that is deterministic and memoryless, i.e. it depends only on the state and selects a unique action for each state.

In some applications (for example in verification and control theory), the objective is not based on rewards, but rather the goal is to maximize the probability that the execution that is generated satisfies a desirable property (expressed for example in a temporal logic); see e.g. [3, 7, 8, 32]. It is known that for a broad range of properties this problem reduces to the case of a simple reachability objective, where the goal is to hit a certain target state in a larger MDP that combines the desired property and the original MDP. The reachability objective can be viewed as a special case of the total reward objective (see Section 2), thus the solution methods for reward-based MDPs can be used also for the class of applications that seek to optimize the probability of a desirable execution.

MDPs can be solved in polynomial time using Linear Programming. From an MDP, one can construct a Linear Program (LP), whose basic feasible solutions (bfs) correspond to positional policies of the MDP, and the optimal bfs yields the optimal policy. The usual way however of solving MDPs in practice is using the *Policy Iteration* (PI) algorithm of Howard [22]. This is essentially a local search algorithm, an iterative algorithm which starts with an initial positional policy, and keeps improving it until it arrives at an optimal (positional) policy. In each iteration, the algorithm computes the value for each state according to the current policy and determines whether switching the selected action for a state would improve its value; if there are such switchable states, then their actions are switched to obtain the new policy, otherwise the policy is optimal, i.e., in this case local optimality guarantees global optimality. If at some point there are multiple switchable states, and/or multiple choices of a new action that improves the value for a state, then there is flexibility on which states the algorithm chooses to switch and to which actions, resulting in different versions of Policy Iteration. The most commonly used version, called *Howard's PI* (or *Greedy PI*), switches simultaneously all switchable states to their most "appealing" action (see Section 2 for the formal definition). At the other extreme one may choose to switch only one of the switchable states, where the choice of the state and the new action is based on some criterion. We refer to these choices as pivoting rules, in analogy with the Simplex algorithm. Indeed, there is a close correspondence between the variants of Policy Iteration

STOC '23, June 20–23, 2023, Orlando, FL, USA

ACM ISBN 978-1-4503-9913-5/23/06…$15.00
https://doi.org/10.1145/3564246.3585220

where only one state is switched in each iteration and Simplex applied to the LP for the MDP. Howard's PI corresponds to performing simultaneously many Simplex pivots.

The (worst-case) time complexity of Howard's PI was open for a long time, until it was finally resolved by Fearnley in [15], who showed an exponential lower bound under the total reward and the average reward objectives. This was extended to the discounted reward objective in [20] for discount factors that are exponentially close to 1 (in discounted reward MDPs, future rewards are discounted by a given discount factor $\gamma < 1$). For constant discount factor γ however, or even if $1 - \gamma > 1/poly$, Howard's PI runs in strongly polynomial time [34]; this holds more generally even in 2-player turned-based stochastic games for the analogous strategy improvement algorithm [18]. The complexity of PI where only one state is switched in each iteration was studied earlier by Melekopoglou and Condon [24], who gave exponential lower bounds for several pivoting rules. More recently, the close connection between single-switch Policy Iteration for MDPs and Simplex for LPs has been exploited to show exponential or subexponential lower bounds for Simplex under various open pivoting rules, by first showing the results for MDPs and then translating them to Simplex: this was shown for the Random-Facet and the Random-Edge rules in [17], for Cunningham's rule in [2], and for Zadeh's rule in [12]. There is ongoing extensive literature on the complexity of Policy Iteration, studying various variants (e.g. randomized PI, geometric PI etc.), special cases (e.g. deterministic MDP) and/or improving the bounds [21, 27, 31, 33].

Thus, although PI runs fast in practice, its worst-case complexity is exponential for Howard's PI, as well as other variants. This is similar to the behavior of the Simplex algorithm, and more generally a host of other local search algorithms for various optimization problems. To provide a more realistic explanation for the observed performance of Simplex, Spielman and Teng introduced the *smoothed analysis* framework [30], a hybrid between worst-case and average-case analysis. On one hand, average-case complexity is an algorithm's expected runtime given a probability distribution over inputs. On the other hand, we can think of worst-case complexity as the maximum of an algorithm's expected runtime over all input distributions, including those with all probability mass on a single input. The smoothed complexity of an algorithm is its maximum expected runtime over all input distributions *with some smoothness constraint*. For example, an input is picked arbitrarily by an adversary and then its parameters (for example the entries of the matrix in an LP, the rewards and transition probabilities in an MDP) are perturbed randomly according to a distribution with density function bounded by a parameter ϕ (for example, uniform, Gaussian or some other distribution). The smoothed complexity of the algorithm then is the expected running time as a function of the input size n and ϕ. Ideally we would like to have polynomial time in n and ϕ. Note this is useful if ϕ is polynomially bounded in n (or constant), because for exponentially large ϕ (i.e. perturbations that are sharply concentrated), polynomial time in n and ϕ is simply exponential time, which is not useful. Smoothed analysis may capture runtime in practice more effectively than worst-case analysis, especially when the numerical values in the input may have some natural variation, as problems formulated from the real world often do. Spielman and Teng showed that the Simplex algorithm under a certain pivoting rule has polynomial smoothed complexity [30] (and there is a series of subsequent papers simplifying the proof and improving on the bounds, eg. [9, 11]).

Smoothed analysis has since been applied to a range of problems in areas such as mathematical programming, machine learning, numerical analysis, etc. [29]. In the area of combinatorial optimization, it has been applied to local search algorithms for problems such as the Traveling Salesperson Problem (TSP), Max-Cut and others. It has been shown for example that the simple 2-Opt algorithm for TSP has polynomial smoothed complexity [13], in contrast to its worst-case exponential complexity [23]. For Max-Cut, the simple Flip algorithm has smoothed complexity that is at most quasi-polynomial for general graphs [6, 14] and polynomial for the complete graph [1, 5], again in contrast to its worst-case exponential complexity [26].

Given the good empirical performance of PI and its relationship to the Simplex algorithm, it is natural to hypothesize that the smoothed complexity of PI may well be also polynomial. Note that this does not follow from the result for Simplex, despite their strong connection, for various reasons. First, in the smoothed model for Linear Programming all the numerical parameters are randomly perturbed independently. In the MDP, we want to perturb similarly the rewards and transition probabilities, however we want the perturbed model to be also an MDP, in particular the transition probabilities for each action must sum to 1. Second, in the LP smoothed model, all entries of the constraint matrix are perturbed randomly, even those that are 0; if we apply such perturbation to the LP of an MDP, it will have the effect of introducing arbitrary new transitions that have no justification. In defining the smoothed model for an MDP, it is more natural to preserve the structure of the MDP (i.e. available actions at each state and possible transitions for each action), since there are usually constraints in the application that is modeled by the MDP that determine which transitions can or cannot occur from a state for each action. On the other hand, the rewards and transition probabilities may well be estimates, and thus for them it is reasonable to allow perturbations. Thus, in our smoothed model for MDP, we preserve the structure of the MDP, and allow perturbations of the (nonzero) transition probabilities and rewards.

In the literature on smoothed complexity, both models have been used, the *full perturbation* model, where all numerical parameters are perturbed, including those that are 0, and what we may call the *structured model*, where only the nonzero parameters are perturbed and the structure of the input is preserved. For example the analysis of Simplex uses the full perturbation model. Work on local search algorithms for combinatorial optimization have used both models. For example, in the case of the FLIP algorithm for Max Cut, [1, 5] use the full perturbation model and show that the smoothed complexity is polynomial. On the other hand, [6, 14] use the structured model and show that smoothed complexity is quasi-polynomial for every graph; note that the full model coincides with the structured model in the special case when the input graph is complete. Although the structured part of the input (the graph) is not perturbed, thus allowing for arbitrarily complex, "pathological" instances, the smoothening of the numerical parameters (the edge weights) brings the complexity down from exponential to

quasi-polynomial; the conjecture is in fact that the true smoothed complexity is polynomial.

Depending on the application, one or the other model may be more reasonable. In the case of MDPs, we believe that the structured model is more natural for the reasons discussed above. The MDP typically models an application at hand (for example, a probabilistic program that is analyzed, a control design problem, a game etc.), and the transitions have some meaning in the application. The precise values of probabilities and rewards may be fungible, but their existence is important. Changing the structure of the instance changes the problem, or may even render it meaningless.

For example, consider an MDP with a reachability objective. If perturbations are applied also to the zero-probability (i.e. nonexistent) transitions, then in the perturbed MDP every possible transition between any two states will be included with nonzero probability. This means that for any policy the graph of the MDP becomes strongly connected, every policy will reach the target with probability 1, and the problem has disappeared.

The issue of preserving the (zero-nonzero) structure is especially important when the model is used to formulate and solve other problems. For a simple example, consider the following: MDPs with rewards can be used to solve the simple reachability optimization problem for MDPs (without rewards), since the latter can be viewed as a special case (can be reduced to) the former: All transitions of the reachabilty MDP are given 0 reward, except for the transitions into the target state that are given reward 1; maximizing the expected total reward in the resulting MDP is equivalent to maximizing the probability of reaching the target state in the reachability MDP. If in the reward MDP we are allowed to perturb the zero rewards then the problem has changed, and optimization in the MDP with rewards no longer correctly captures the MDP reachability problem.

1.1 Our Results

In this paper we study the smoothed complexity of Policy Iteration. Given its similarity to the Simplex algorithm, one might hope to show polynomial smoothed upper bounds for PI. We show the contrary: for several prominent policy iteration variants, such a result is impossible; the smoothed complexity is subexponential or even exponential. We concentrate here mainly on the total reward objective.

Our main result concerns the classical Howard's (Greedy) PI which switches simultaneously all switchable states to their actions with greatest appeal. We show that Howard's PI has at least subexponential smoothed complexity under the total reward objective; a similar result holds for the average reward objective. Furthermore, the lower bound holds not only for the expected complexity under random independent perturbations of the parameters, but it holds in fact for *all (arbitrary)* perturbations within a certain inverse polynomial range. (The amount of perturbation corresponds to the $1/\phi$ parameter of the smoothed model, so to be meaningful, ϕ has to be at most polynomial.) Specifically, we construct an MDP with N states and bounded parameters (rewards and transition probabilities), such that in every MDP obtained by perturbing the parameters by any amount up to $1/N$, Howard's PI requires at least $2^{\Omega(N^{1/3})}$ iterations.

Our initial approach for this was to examine whether the construction of [15] for the worst-case complexity can be modified to prove a smoothed lower bound. However, we were not able to do this. Unfortunately, the construction seems to be brittle and does not hold up under perturbations. Thus, we started fresh and designed a new construction with robustness in mind. The construction and the proof are quite involved. This is to be expected, considering that the construction of [15] was also quite intricate. That construction involved positive and negative rewards, exponentially small probabilities, and exponentially large rewards. We show that the parameters do not need to be exponentially large or small, and furthermore they can tolerate arbitrary perturbations up to an inverse polynomial, without affecting the behavior of Howard's PI algorithm.

Furthermore, we use the robustness of our construction for MDP with rewards, to show that the worst-case complexity of Howard's PI for MDPs with the simple reachability objective is exponential. Note that these MDPs have no rewards (or as mentioned above they are a special case of MDPs with rewards 0 and 1). In some sense, this second construction is an approximate reduction from MDPs with rewards to the special case of reachability MDP. The robustness of the original reward MDP is essential to establish the correctness of the result for the weaker reachability MDP.

We also analyze three simple variants of PI from [24] that switch a single (switchable) state in each iteration, chosen according to some rule. In *Simple PI* the state is chosen according to an arbitrary initial priority order; in *Topological PI* it is chosen according to a topological order; and in *Difference PI* it is chosen according to the difference in value between the new and the old action of the state; see Section 2 for a formal definition of the variants. We make slight modifications to the constructions from [24] and prove that they are robust to perturbations. These constructions are reachability MDPs; thus the only numerical parameters are the transition probabilities, there are no rewards (or equivalently, all the rewards are 0 except for the transitions to the target state that have reward 1). Simple PI and Topological PI take exponential time, for very large (constant) perturbations of the transition probabilities. Difference PI takes at least subexponential time for inverse polynomial perturbations.

We finally discuss the relationship between our results for the Single switch PI variants and the Simplex algorithm, describing precisely how our perturbations of an MDP translate to the corresponding LP. We state the lower bounds implied by our Policy Iteration results for Bland's and Dantzig's pivot rules in the Simplex algorithm for LPs arising from MDPs, though these bounds are not new for Simplex.

1.2 Outline of Proof Techniques

The construction and proof of the main result on Howard's (Greedy) PI are quite complex and involved. We first design a new construction of an MDP M_1 for the exponential worst-case complexity of Greedy PI, which is more amenable to modifications to achieve the desired robustness. As is usual in exponential lower bounds for many problems, the MDP is constructed so that the iterations of Greedy PI will simulate a binary counter counting from 0 up to 2^n. There is a set of n states b_i of the MDP (among many others), each with two distinguished actions 0, 1, where the choices of states b_i

correspond to the bits of the counter. The MDP M_1 is constructed so that if in the initial policy all states b_i choose action 0, then Greedy PI will go through 2^n rounds until it arrives at the optimal policy where all states b_i choose action 1. Each round involves a number of steps. To manage the complexity of the construction, we build it in stages. We first design a simpler MDP M_0, which exhibits this exponential (worst-case) behavior for a slight variant of Greedy PI, call it Hybrid PI, which has an additional rule that a switchable state b_i can switch from action 0 to 1 only if all b_j with $j < i$ have chosen action 1, and no other non-b_i states are switchable. We then modify M_0 to an MDP M_1 by using a suitable gadget at the states b_i which serves the purpose of delaying the switches at states b_i when running Greedy PI on M_1 in such a way that it behaves like Hybrid PI on M_0. As a result, Greedy PI on M_1 simulates a binary counter and takes exponential time.

The MDP M_1 has exponentially large and small rewards (both positive and negative), and exponentially small probabilities. The next stage in the proof transforms M_1 to another MDP M_2 that has bounded rewards and probabilities, and which is robust in the sense that Greedy PI has the same behavior for any perturbation of the rewards and probabilities up to an inverse polynomial amount. This transformation is done using appropriate gadgets. We design gadgets to simulate exponentially large and exponentially small rewards and transition probabilities, and ensure that the gadgets are robust, i.e., they perform correctly (approximately) even under perturbation of their rewards and probabilities. Finally, we ensure that the analysis for M_1 is robust enough, so that the behavior of Greedy PI on it is simulated by M_2 even under perturbation of its parameters.

The proof for the worst-case exponential complexity of Greedy PI under the reachability objective uses the constructed MDP M_1 for the total reward objective (with somewhat modified parameters). The MDP M_1 has positive and negative rewards, whereas there are no rewards in the reachability objective. We design suitable gadgets to eliminate positive and negative rewards using random actions, and apply them to transform M_1 to a new MDP M_3, without rewards, for the reachability problem. The effective rewards simulated by these gadgets change slightly for different policies; thus they are similar to additive rewards with perturbations. In order to argue that these perturbations are small, we must know bounds on the minimum and maximum values of the nodes where the gadgets are plugged in, which we have from the analysis of M_1. The robustness of the MDP M_1 is critical for the correctness of the transformation, i.e. so that the behavior of Greedy PI under the reachability objective in M_3 simulates the behavior of Greedy PI in M_1 under the total reward objective.

The proofs for the results on the variants of PI with a single switch use the constructions of [24], sometimes with some small modifications. The proofs are relatively simple and offer a gentle introduction to the issues, and the unfamiliar reader might like to read this section first. In the case of Simple PI and Topological PI, the analysis follows closely that of [24], except that it is carried out for general values of the transition probabilities, rather than specific values. In the case of Difference PI, we use a parameterized gadget with suitable choice of parameters to modify the construction in such a way that it can tolerate perturbations of the transition probabilities within an inverse polynomial range.

Most proofs of lemmas and propositions are omitted in this extended abstract and can be found in the full version.

Organization of the paper. The rest of the paper is organized as follows. Section 2 gives basic definitions and notation. Section 3, which is the heart of the paper, shows that Greedy (Howard's) PI has at least subexponential smoothed complexity under the total reward objective. Section 4 builds on our construction to show the worst-case exponential complexity of Greedy PI under the simple reachability objective. Section 5 presents the results for three PI variants with single state switch: Simple PI, Topological PI and Difference PI, and Section 6 notes the connection to Simplex pivoting rules.

2 PRELIMINARIES

A Markov Decision Process consists of a (finite) set of states S, and a (finite) set A_s of available actions for each state $s \in S$. Let $A = \cup_{s \in S} A_s$ denote the set of all actions. For each action $a \in A_s$ there is a probability distribution of the state(s) resulting when taking action a at state s that is described by a function $p : S \times S \times A \to \mathbb{R}^+$, where $p(s'|s, a)$ denotes the probability of ending up at state s' when taking action a from s. The action a is *deterministic* if $p(s'|s, a) = 1$ for some s' and $p(s"|s, a) = 0$ for all other $s"$. Each action yields some (possibly zero) reward, represented by a function $r : S \times A \to \mathbb{R}$ where $r(s, a)$ denotes the reward obtained by taking action $a \in A_s$ from s. A (positional) *policy* is a function $\pi : S \to A$, where for each state $s \in S$, $\pi(s) \in A_s$ is the action selected at that state. A policy π for an MDP M induces a Markov chain M_π on the same state set S, where the transition probabilities out of each state s are given by $p(s'|s, \pi(s))$.

A *criterion* (or *objective*) is a function that, given a policy, associates a value with each state. We consider primarily the *total reward* criterion, which yields the following notions of value and appeal. The value of a state s captures the expectation of the sum of rewards accrued by starting at s and taking the actions given in the policy as time goes to infinity; it is well-defined for MDPs where one must eventually reach a sink state, one that has no actions and no outgoing transitions. Under the total reward criterion, the *value* of a state s under a policy π satisfies the equation

$$\mathsf{Val}^\pi(s) = r(s, \pi(s)) + \sum_{s' \in S} p(s'|s, \pi(s)) \cdot \mathsf{Val}^\pi(s')$$

Given policy π, the *appeal* of an action a at s is

$$\mathsf{Appeal}^\pi(s, a) = r(s, a) + \sum_{s' \in S} p(s'|s, a) \cdot \mathsf{Val}^\pi(s')$$

An *optimal policy* is a policy that maximizes the value of every state (there is always such a policy).

We later consider the *reachability* criterion, where the goal is to maximize the probability of reaching a given target sink state s^*. In this case, the value of a state s under a policy π is the probability of reaching the target s^* following the actions selected in π. The reachability criterion can be viewed as a special case of the total reward criterion, by assigning reward zero to all transitions except for those going from other states into the target state s^*, which are assigned reward 1.

Policy Iteration or *Policy Improvement* (PI) is a family of local search algorithms used to find an optimal policy of an MDP. We

say a state s is *switchable* under a current policy π if there is an action $a \in A_s$ such that $\text{Appeal}^\pi(s,a) > \text{Val}^\pi(s)$. We also say any such value-improving action a is switchable. In each iteration, PI switches some number of switchable states to their value-improving actions. An optimal policy is reached when no states are switchable. There are several PI variants, which involve various switching rules for choosing the state(s) and actions(s) to switch in each iteration.

The most widely used variant, *Howard's PI* (or *Greedy PI*) involves switching *all* switchable states in each iteration. A switchable state with multiple switchable actions is switched to the action with greatest appeal. More precisely, given that the current policy is π, Greedy PI switches each switchable state s to an action in $\arg\max_{a \in A_s} \text{Appeal}^\pi(s,a)$.

We discuss also several variants of PI that switch only one state in each iteration, *Simple PI*, *Topological PI*, and *Difference PI*. Simple policy iteration fixes an ordering over the states and switches the highest-numbered switchable state. If there are multiple improving actions at a state, one of them is chosen according to some rule; in the constructions we discuss, every state has only two actions, so there is no choice of improving action. Topological policy iteration considers a topological ordering over the states, where if there is a path (sequence of actions with nonzero probability) from a state i to a state j, the order of i is at least the order of j; that is, the graph is partitioned into strongly connected components and each state is assigned the index of its component in a topological order. Topological PI switches the highest-numbered switchable state of the component with lowest topological order that contains switchable states. Difference policy iteration switches the switchable state with the greatest difference between the appeal of the action it switches to its current value.

We represent MDPs graphically, where states are vertices and actions are directed edges. We sometimes use this terminology in our discussion. Each deterministic action is shown as a directed edge between two nodes. Each probabilistic action is shown as starting as a single line at the origin state and branching into multiple lines to reach the various possible resulting states. Given a policy π, the set of directed edges corresponding to the selected actions form a subgraph of the MDP (this is the graph of the Markov chain M_π). That is, a node s selecting action a under π has a directed edge to every node s' that a takes s to with nonzero probability. We say a node s' is *reachable* from s if there exists a path from s to s' in this subgraph.

Smoothed model. The smoothed analysis framework lies between average-case analysis and worst-case analysis. It considers input instances with each parameter drawn independently from some probability distribution (e.g., Gaussian, uniform or any other distribution) with an upper bound ϕ on its density function. An algorithm A has polynomial smoothed complexity if the maximum expected runtime of A over all such distributions is polynomial in both ϕ and in the size of the input. Alternatively, before A is given an arbitrary (worst-case) input x, the values of x (the numerical parameters) are perturbed according to some distribution still of bounded density at most ϕ. The perturbed input x' is then given to A. The smoothed runtime of A is its worst-case (over all inputs x) expected runtime (expected over the perturbation distribution).

As discussed in the Introduction, we consider the structured perturbation model for MDPs, where we perturb only the nonzero transition probabilities and rewards. Our constructions are robust not only to random perturbations, but moreover to *all* perturbations within a certain wide range. When proving our lower bounds, we model the nonzero rewards and the probabilities associated with probabilistic actions as adversarially chosen within some perturbation radius $\sigma = 1/\phi$, where the adversary aims to minimize the expected runtime (i.e. to defeat the lower bound construction). That is, any reward $r \neq 0$ of x can take on any value r' in $[r-\sigma, r+\sigma]$ in x'. We do not perturb rewards of zero. For any probabilistic action with nonzero transition probabilities $p_1, \ldots, p_k$, the corresponding perturbed probabilities p_i' in x' are any (non-negative) values in $[p_i - \sigma, p_i + \sigma]$ that sum to 1. Note that we cannot perturb the probabilities independently, since they must sum to 1 for the resulting MDP to be valid. (An alternative, equivalent model, is to perturb independently all the p_i within the allowed range and normalize them so they sum to 1.) We say such an x' is within *perturbation radius* σ of x.

Each lower bound in this paper gives an MDP x where *every* MDP within perturbation radius σ of x yields superpolynomial runtime. Thus, our results are stronger than the usual smoothed analysis: the superpolynomial lower bounds hold not only for random perturbations according to a specific probability distribution of (inverse polynomial) bounded density, but they moreover hold for *all* perturbations within the specified ranges, i.e., even when an adversary who wants to defeat the construction and minimize the running time chooses any perturbations they want within the specified range.

3 A SMOOTHED LOWER BOUND FOR GREEDY PI UNDER THE TOTAL REWARD AND AVERAGE REWARD CRITERIA

In this section, we prove a subexponential lower bound on the smoothed complexity of Greedy PI. More specifically, we construct an MDP with N states and bounded parameters (rewards and transition probabilities), such that in every MDP obtained by perturbing the parameters by any amount up to $1/N$, Greedy PI requires at least $2^{\Omega(N^{1/3})}$ iterations. We prove this for the total reward criterion. The same result applies to the average reward criterion.

The construction is quite involved and is presented in several stages. We present first in Section 3.1 a simplified construction which forces exponential worst-case runtime for a variation of Greedy PI (we call it hybrid Greedy PI), in which in certain cases some switchable states are not switched until some conditions are satisfied. In Section 3.2, we add suitable gadgets to this MDP so that Greedy PI in the new MDP simulates the hybrid variant in the simplified construction; thus, Greedy PI has exponential worst-case runtime in this full construction. This MDP includes rewards that are exponentially large and small, and some probabilities that are exponentially small. In Section 3.3 we transform this MDP to our final robust MDP by using gadgets that allow us to eliminate the exponentially large and small rewards and probabilities and simulate them by parameters that lie in a bounded range in a robust way; that is, the behavior of Greedy PI is not affected by perturbation of the parameters up to an inverse polynomial amount.

3.1 Simple Construction

In this section, we present the simplified construction shown in Figure 1, which is designed to simulate a binary counter over the "bit" nodes b_i. We argue that a variant of greedy policy iteration takes exponentially many iterations for the total reward criterion. This variant, which we call *hybrid policy iteration* and which we define for this construction only, is nearly greedy policy iteration, except at the nodes b_i. We define hybrid PI and the simple construction for ease of presentation, and later in Section 3.2 we present our full construction, where we use gadgets to ensure that greedy PI behaves similarly to hybrid PI.

Definition 1 (hybrid policy iteration)**.** Given a policy π, hybrid policy iteration chooses the next policy π', where:

- Every switchable non-b_i vertex is switched to its appeal-maximizing action, as in greedy PI.
- Every switchable b_i vertex with $\pi(b_i) = 1$ is switched so that $\pi'(b_i) = 0$.
- Every switchable vertex b_i with $\pi(b_i) = 0$ switches if and only if no non-b vertices are switchable and $\pi(b_j) = 1$ for all $j < i$.

The simple construction is shown in Figure 1, with parameters as follows. Let $r : [n] \to \mathbb{Z}$ be a function where $r(i)$ is the reward on the edge associated with taking action 1 from b_i. r need only satisfy that for all i, $r(i) > \sum_{j<i} r(j)$. We can achieve this by letting $r(i) = 2^{2i}$. Let $-\epsilon$ denote a very small negative reward. Assume that $\epsilon \ll r(i)$ for all i.

We describe the available actions at each state, for each $i \le n$:

- b_i: b_i has a deterministic action 0 to w_{i+1} with reward 0 and a deterministic action 1 to d_{i+1} with reward $r(i)$.
- c_i: c_i has a deterministic action 0 to w_{i+1} with reward 0 and a deterministic action 1 to b_i with reward $-\epsilon$.
- d_i: d_i has a deterministic action 0 to w_{i+1} with reward 0 and a deterministic action 1 to c_i with reward $-\epsilon$.
- w_i: w_i has a deterministic action 1 to b_i with reward 0. It also has a deterministic action to b_j for every $j > i$ with reward 0, and a deterministic action to the sink with reward 0.

w_{n+1} has only a deterministic action to the sink node with reward 0. c_{n+1} has a deterministic action to the sink with reward $-\epsilon$. d_{n+1} has a deterministic action to c_{n+1} with reward $-\epsilon$.

In the following we will often simply write $b_i = 0$ or $b_i = 1$ for the action of a policy at a node, instead of $\pi(b_i) = 0$ or 1.

At a high level, our construction simulates a binary counter. Each state b_i represents a bit. When $b_i = 1$, a large reward is incurred when leaving b_i. When $b_i = 0$, no reward is incurred. The starting policy for our lower bound will have all bits b_i set to 0, and the optimal policy is when all bits b_i are set to 1. The following two properties ensure that the bits behave as a binary counter, iterating through all binary strings of length n before reaching the optimal policy. We state them here and prove them later. We achieve Property 1 by construction, and we achieve Property 2 by definition of hybrid PI.

Property 1. When a bit b_i is set to 1, all lower bits b_j for $j < i$ are reset to 0 within two iterations.

Property 2. For each i, b_i switches to 1 only after $b_j = 1$ for all $j < i$.

We will show using these properties that hybrid policy iteration proceeds in three phases. We will show that between every set of three phases (i.e., before the first phase and after the third phase), the following invariant always holds. Let $B = \{i | b_i = 1\}$.

Invariant. For all $i \in B$, $w_i = b_i = c_i = d_i = 1$ for all $i \in B$. For all $i \notin B$, we have $b_i = c_i = d_i = 0$. For all $i \notin B$ and $i > \max(B \cup \{0\})$, w_i chooses the deterministic action to the sink. Otherwise, if $i \notin B$ and $i \le \max B$, w_i chooses the deterministic action to b_ℓ where ℓ is the smallest index such that $\ell \in B$ and $\ell \ge i$.

Phases. We now describe the phases. Each set of 3 phases involves adding the minimum index $i = \min([n] \setminus B)$ to B and resetting all lower indices, so that i is the minimum index in B.

(1) b_i switches from 0 to 1.
(2) w_j switches to b_i for all $j \le i$. c_i switches to 1.
(3) b_j switches to 0 for all $j < i$. d_i switches to 1. c_j, d_j switch to 0 for all $j < i$.

At the end of the 3 phases, the invariant is again satisfied.

All-zero policy. Let π_0 denote the policy with each w_i choosing the action to the sink and all other nodes' actions equal to 0.

We show that Hybrid PI indeed follows the above phases when starting with π_0. We first prove several useful facts. The proofs of the following propositions and lemmas are deferred to the full version of this paper.

Proposition 1. When the invariant is satisfied, for every $i \notin B$ and $i < \max B$ we have $\mathrm{Val}(b_i) = \mathrm{Val}(c_i) = \mathrm{Val}(d_i) = \mathrm{Val}(b_\ell)$ where ℓ is the smallest index such that $\ell \in B$ and $\ell \ge i$.

Proposition 2. When the invariant is satisfied, for every $i \in [n]$ we have $(\sum_{\substack{j \ge i \\ j \in B}} r(j)) - 2(n - i + 1)\epsilon \le \mathrm{Val}(b_i) \le \sum_{\substack{j \ge i \\ j \in B}} r(j)$.

Proposition 3. Let $b_i = 1$. Then when the invariant is satisfied, $\mathrm{Val}(b_i) \ge \mathrm{Val}(b_\ell) + r(i) - 2\epsilon n$ for all $\ell > i$.

Lemma 1. When the invariant is satisfied, the set of switchable nodes is exactly the set of bits not in B.

The proof of this lemma is by verifying using the propositions that the values of the nodes are such that the bits in B are switchable, and all other nodes are not switchable.

Proposition 4. When the invariant is satisfied, $\mathrm{Val}(b_i) - 2\epsilon \le \mathrm{Val}(d_i) \le \mathrm{Val}(b_i)$ for every $i \le n$.

Theorem 1. Given the simple construction and starting policy π_0, hybrid policy iteration requires at least 2^n iterations to reach the optimal policy under the total reward criterion.

Proof. We show that if the invariant is satisfied, the algorithm proceeds in the three phases. For each phase, we argue that the switches made follow hybrid PI.

Phase 1. By Lemma 1, the set of switchable nodes is exactly the bits not in B. By definition of hybrid PI, the only node that switches is b_i where i is the minimum index such that $b_i = 0$. b_i thus switches to 1, and its value is now $\mathrm{Val}(b_i) = \mathrm{Val}(d_{i+1}) + r(i)$.

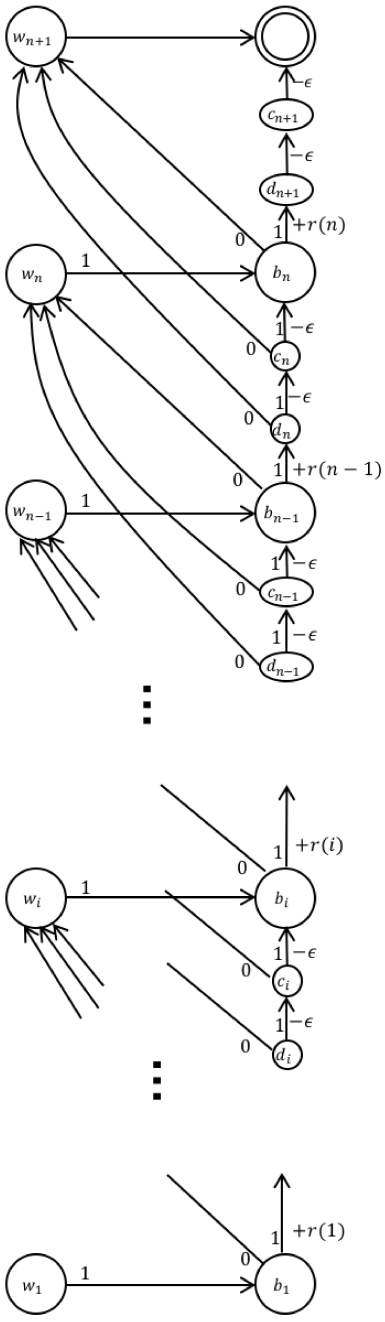

Figure 1: The base MDP for our greedy PI construction. The sink is shown with a double border. Each w_i has a deterministic edge with reward 0 to every b_j for $j \in [i, n]$, and a deterministic edge to the sink. These edges are omitted in the figure for clarity.

Phase 2. Switching b_i in Phase 1 affected the value of only b_i, since no node selects an action to b_i according to the invariant. The only nodes that may become switchable in Phase 2 are those with actions to b_i; these nodes are exactly c_i and w_j for $j \le i$. Now, applying Proposition 4 for the first inequality, $\mathrm{Val}(b_i) \ge \mathrm{Val}(b_{i+1}) + r(i) - 2\epsilon = r(i) - 2\epsilon + \sum_{\substack{j' \ge i+1 \\ j' \in B}} r(j')$. Thus, for any $j > i$, $\mathrm{Val}(b_i) > \mathrm{Val}(b_j)$. For any $j < i$, by Proposition 2 and the fact that switching b_i affected the value of only b_i, we have

$$\mathrm{Val}(b_j) \le \sum_{\substack{j' \ge j \\ j' \in B}} r(j') = \sum_{\substack{j' \ge i+1 \\ j' \in B}} r(j') + \sum_{\substack{j \le j' < i \\ j' \in B}} r(j')$$
$$< \left(\sum_{\substack{j' \ge i+1 \\ j' \in B}} r(j') \right) + r(i) - 2\epsilon$$

since $r(i) > \sum_{j'<i} r(j')$ and ϵ is sufficiently small. Thus, b_i has the highest value of any bit, and all w_j for $j \le i$ switch to b_i. When $c_i = 0$, it takes on the value of some higher bit (or zero), so c_i switches to 1 and takes on the greater value of b_i.

Phase 3. By choice of b_i, we have $b_j = c_j = d_j = 1$ for all $j < i$. After the switches in Phase 2, we have $w_j = b_i$. Thus, for any fixed b_j, b_j follows the right column up to d_i, which follows its action 0 to w_{i+1} and does not collect the reward $r(i)$. Since $r(i) > \sum_{\ell<i} r(\ell)$, we have $\mathrm{Val}(b_j) \le \mathrm{Val}(w_{i+1}) + \sum_{\ell<i} r(\ell) < \mathrm{Val}(b_i) = \mathrm{Val}(w_{j+1})$. Thus each b_j for $j < i$ switches to action 0 going to w_{j+1}, and Property 1 holds. Similarly, c_j and b_j switch to 0, since $\mathrm{Val}(w_{j+1}) = \mathrm{Val}(b_i) > \mathrm{Val}(b_j)$. d_i switches to 1, since $\mathrm{Val}(w_{i+1}) < \mathrm{Val}(b_i) - 2\epsilon$.

Observe that the invariant is again satisfied. Thus, if we start with all actions equal to 0 and each w_i choosing the action taking it to the sink, the invariant is satisfied after every set of three phases.

Furthermore, since the only bit that switches to 1 is b_i where i is the minimum index such that $b_i = 0$, Property 2 holds. □

We have thus shown that hybrid policy iteration on the simple construction simulates a binary counter on the n bits, taking at least 2^n iterations to reach the optimal policy.

3.2 Full Construction

We now show that we can insert a gadget at each b_i node for $i \neq 1$ in order to satisfy Property 2. This gadget is depicted in Figure 2. b_1 remains as in the simple construction, with no added gadget. For the other nodes b_i, we add $f(i) + 1$ actions $a_0^i, a_1^i, \ldots, a_{f(i)}^i$ to b_i, where $f(i) = 3 + 6i$. a_0^i is deterministic and goes to w_1 with zero reward. Each other action a_j^i for $j \ge 1$ returns to b_i with high probability, and goes to b_1 and incurs a small reward with the remaining probability. The 0 actions of b_i from the simple constructions are deleted; they are replaced by these new actions a_0^i and $a_{j\ge1}^i$. These actions a_j^i for $j \ge 0$ have increasing rewards (relative to index j) but decreasing appeals. Thus, policy iteration starts by selecting the action a_0^i with the largest appeal but smallest reward and cycles through $a_1^i, a_2^i, \ldots$ in order. We also amend the action 1 from the simple construction to loop back to b_i with overwhelming probability (this trick was also used by [15]). Without changing any values, this lowers the appeal of action 1 so that all of the actions a_j^i are

more appealing. Consequently, these extra actions delay b_i from switching to action 1.

The gadget is reset (b_i chooses a_0^i) whenever any bit other than b_1 is set to 1. When this bit is switched to 1, w_1 will have greater value than b_1 in Phase 3 of that switch. Thus, in Phase 3, b_i will switch to the action a_0^i that goes to w_1.

We achieve this using the following probabilities and rewards for each action a_j^i:

Probabilities. a_j^i goes from b_i to b_1 with probability $\frac{1}{2^{2j}}$ and from b_i back to itself with the remaining probability $1 - \frac{1}{2^{2j}}$.

Reward. a_j^i gives a reward of $\delta_j = 3j \cdot \delta$, where $\delta = 2^{-100n}$.

We also slightly modify the rewards $r(i)$, introduce costs $c(i)$, and change the actions at the nodes w_i to incur these costs. When b_i is set to 0 in the full construction, its value is a bit higher than that of w_1 or b_1, which is greater than the value b_i would have when set to 0 in the simple construction. Since we do not want w_i to set its action to 1 until b_i switches to 1, we add the cost $c(i)$ from b_i^- to b_i to counteract this extra value, and direct action 1 from w_i to b_i^-. We redefine $r(i) = 2^{2i+1}$. We set $c(i) = \frac{r(i)}{2} = 2^{2i}$ for $i > 1$, and let $c(1) = 0$. Setting $c(1) = 0$ is for ease of notation; as there is no gadget for b_1, there is no real cost attached to b_1. The effective reward (as seen by w_i) associated with $b_i = 1$ is $r(i) - c(i) = 2^{2i}$, as in the simple construction. We show later that this preserves the property that w_i does not switch to 1 until b_i has switched to 1.

We also add an action a_0^i from b_i to w_1 with no reward and with probability 1. We add a probabilistic action 1 from b_i to b_i^+ that loops back to b_i with extremely high probability $1 - \frac{1}{2^{1000n}}$. This ensures that the appeal of action 1 is smaller than the appeal of any other action a_j^i.

We refer to the rewards, costs, and probabilities as *parameter values*. When analyzing the behavior of policy iteration on our constructions, we point out which of our arguments depend on these exact parameter values and which do not.

Invariant. At the beginning of the phases, we have $w_i = b_i = c_i = d_i = 1$ for all $i \in B$. For all $i \notin B$, we have $c_i = d_i = 0$. For all $i \notin B$ and $i > \max(B \cup \{0\})$, w_i chooses its deterministic action to the sink. Otherwise, if $i \notin B$ and $i \leq \max B$, w_i chooses the deterministic action to b_ℓ where ℓ is the smallest index such that $\ell \in B$ and $\ell \geq i$. If the last bit added to B was not b_1, $b_i = a_0^i$ for all $i \notin B$. If the last bit added to B was b_1, we have $b_i = a_0^i$ or $b_i = a_3^i$ for all $i \notin B$.

Phases. Each set of 3 phases involves adding an index i to B and resetting all lower indices, so that i is now the minimum index in B. Suppose first that $i \neq 1$. For each phase, we show the modified behavior compared to in the simple construction. We also introduce a new phase 0, where the bits not in B cycle through their actions a_j.

(0) If b_1 is switchable to 1, proceed to Phase 1. Otherwise, for each $j \notin B$, b_j increments its current action a_m^j to the next action a_{m+1}^j. This repeats until $m+1 = f(i)$ for some i; when this happens, the next iteration begins Phase 1. We will show later that this i is unique, and in fact $i = \min \bar{B}$.

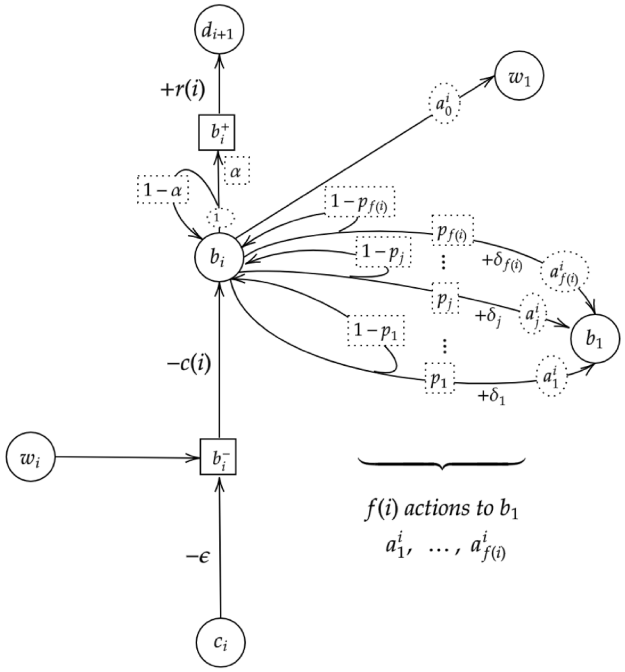

Figure 2: The structure of the full construction at node b_i. Probabilities are shown with dotted boxes around them; action names are shown with dotted ovals around them. Rewards are free-floating with plus or minus signs. The nodes b_i^- and b_i^+ have only one action and are shown as squares.

(1) b_i where $i = \min \bar{B}$ switches to 1. For all $i' \neq i$, $i' \notin B$, $b_{i'}$ switches from its current action $a_\ell^{i'}$ to $a_{\ell+1}^{i'}$.

(2) w_j switches to b_i for all $j \leq i$. c_i switches to 1. For all $i' \neq i$, $i \notin B$, $b_{i'}$ switches from $a_{\ell+1}^{i'}$ to $a_{\ell+2}^{i'}$.

(3) b_j switches to a_0^j for all $j < i$ and $j > 1$. d_i switches to 1. c_j, d_j switch to 0 for all $j < i$. If $i > 1$, for all $i' > i$ where $i' \notin B$, $b_{i'}$ switches to $a_0^{i'}$, resetting the actions. Otherwise, if $i = 1$, for all $i' > i$, $b_{i'}$ switches from $a_{\ell+2}^{i'}$ to $a_{\ell+3}^{i'}$.

Cycling, and defining $f(\cdot)$. We let $f(i) := 3 + 6i$. We first note that this choice of f is large enough that each bit $b_{i'}$ for $i' \notin B$ can indeed increment up to $a_{\ell+3}^{i'}$ when specified. More precisely, we want to show that when b_i selects action a_{m+1}^i in Phase 0, $b_{i'}$ is selecting an action $a_\ell^{i'}$ where $\ell \leq f(i') - 3$ (so there are enough actions for $b_{i'}$ to make its remaining increments). For each i, i', $f(i)$ and $f(i')$ differ by at least 6. In the worst case, Phase 0 starts with $b_i = a_3^i$ and $b_{i'} = a_0^{i'}$ by the invariant. Thus, when $b_i = a_{m+1}^i$ in Phase 0, we have $b_{i'} = a_\ell^{i'}$ for $\ell \leq f(i') - 3$.

We now show the properties that we claimed in Phase 0: if the invariant is satisfied at the start of Phase 0 and the bits increment their actions as described, then the unique first index i to reach $m + 1 = f(i)$ is $i = \min \bar{B}$. See the full version for a proof.

3.2.1 Propositions. As in our proof of the lower bound for the simple construction, we first present propositions and lemmas that

we'll use to show that policy iteration follows the outlined phases. We again defer their proofs to the full version. For the sake of these propositions, we introduce a *weak invariant*. The weak invariant is the same condition as the strong invariant, except that any bit b_i not in B may select any action a^i_j.

These propositions establish relationships between the values and appeals of the vertices. We prove them here in more generality than is necessary for proving the main theorem of this subsection, Theorem 2, since this generality will be useful when we reuse them later for the robust construction. We let $\epsilon_{\max}$ denote the maximum value of any small cost ϵ; here, $\epsilon_{\max} = \epsilon$ since all small costs are the same. We let $\delta_{\max}$ denote the maximum value of any small reward δ; here, $\delta_{\max} = 3(6n+3)\delta$. We let $\epsilon(d_j)$ and $\epsilon(c_j)$ denote the values of the small ϵ costs on the 1 actions from d_j and c_j respectively. In the full construction, $\epsilon(d_j) = \epsilon(c_j) = \epsilon$.

In proving the propositions, we use a notion of reachability in the MDP. At any point t in time, we consider the subgraph $G(t)$ induced by the actions selected at the nodes. A node v is reachable from a node u if there is a path from u to v in $G(t)$.

Propositions 5 and 6 below relate the large rewards $r(i)$ and large costs $c(i)$. Proposition 5 shows that the reward $r(i)$ is substantially larger than the sum of all smaller rewards. As in the simple construction, we use this to show that when a bit b_i is set to 1, all lower bits b_j are enticed by its large reward and reset to 0. Proposition 6 will be used to show that w_i does not switch to any b_j until b_j has switched to 1.

Recall that $r(i) = 2^{2i+1}$, and $c(i) = \frac{r(i)}{2} = 2^{2i}$. Thus, the effective reward associated with $b_i = 1$ is $r(i) - c(i) = 2^{2i}$, as in the simple construction.

Proposition 5. $r(i) - c(i) - 2n\epsilon_{\max} - \delta_{\max} > \sum_{j<i} r(j)$ for all i.

Proposition 6. $c(i) > \delta_{\max} + 2n\epsilon_{\max} + \sum_{j<i} r(j)$ for all i.

Proposition 7 establishes an upper bound on the value of b_1, helping us later upper bound the appeal of switching any bit to 1.

Proposition 7. When the weak invariant is satisfied, $\mathrm{Val}(b_1) \leq r(1) + \sum_{\substack{i\in B \\ i>1}} r(i) - c(i)$.

Proposition 8 gives an upper and a lower bound for every bit b_i for $i \in B$. .

Proposition 8. When the weak invariant is satisfied, for every $i \in B$ we have

$$r(i)+\left(\sum_{\substack{j>i \\ j\in B}} r(j) - c(j)\right) - 2(n-i+1)\epsilon_{\max} \leq \mathrm{Val}(b_i) \leq r(i)+\sum_{\substack{j>i \\ j\in B}} r(j)-c(j)$$

Proposition 9 shows that the actions a^i_j for a bit b_i have decreasing appeal. This property ensures that b_i cycles through all of its actions a^i_j before switching to 1. In its proof, we will make use of the following fact, which we state as a lemma to use it again in proving the main theorem.

Lemma 2. When the weak invariant is satisfied, regardless of the parameter values, $\mathrm{Val}(b_1) = \mathrm{Val}(w_1)$.

Lemma 3. Let the weak invariant be satisfied, where $b_i = a^i_{f(i)}$. Then for any parameters values for which Propositions 5-8 hold, the action of b_i with greatest appeal is 1.

Proposition 9. If the weak invariant is satisfied, and $b_i = a^i_j$, b_i is switchable and the action with greatest appeal is a^i_{j+1} if $j \neq f(i)$, and action 1 if $j = f(i)$.

3.2.2 Main theorem. Our main result of this subsection is that on the full construction, Greedy PI takes subexponentially many iterations. The proof follows the same structure as the proof of the analogous result for the simple construction. We first state Lemma 4, which is analogous to Lemma 1. We then state Theorem 2, arguing as in the simple construction that Greedy PI follows our prescribed phases. This shows that with the all-zero starting policy, Greedy PI behaves like a binary counter, iterating through all 2^n bit strings to reach the optimal policy.

All-zero policy. The all-zero policy π_0 for the full construction is the same as that of the simple construction for all nodes other than the b_i nodes. Each node b_i selects its action a^i_0.

Lemma 4. Let Propositions 5 through 9 hold given the parameter values. When the weak invariant is satisfied, the set of switchable nodes is exactly the set of bits not in B.

Theorem 2. Let the parameter values be such that propositions 5 through 9 hold. If we start at the all-zero policy, Greedy PI on the full construction takes at least 2^n iterations to arrive at the optimal policy under the total reward criterion.

The outline of this proof is similar to that of the proof of Theorem 1, the equivalent theorem for the simple construction. We start with the all-zero policy, which satisfies the invariant. We first show that when the invariant is satisfied, Greedy PI proceeds in the four previously described phases, ending Phase 4 with the invariant again satisfied. Since each set of four phases involves adding the lowest zero bit to B and resetting all lower bits to zero, Greedy PI behaves exactly as a binary counter and iterates through all binary strings for the bits b_i before reaching the optimal all-one policy, where all bits b_i are set to 1. Thus, it remains only to show that Greedy PI follows the four phases, preserving the invariant at the end of Phase 4. The complete proof is provided in the full version.

We showed previously that for our parameter values ($r(i) = 2^{2i+1}, c(i) = 2^{2i}, \epsilon = 2^{-100n}$, and $\delta = 2^{-100n}$), the propositions hold. Therefore, Greedy PI takes at least 2^n iterations to arrive at the optimal policy on the full construction MDP with these parameter values.

This result also holds for the average reward criterion. Policy iteration under the average reward criterion determines which actions to switch based first on a *gain function*, then based on a *bias function* in the case that multiple actions yield equal gain. Fearnley [15] notes that for MDPs that are guaranteed to reach a 0-reward sink state, like our full construction, this gain function is always zero. Here, the bias function also becomes equivalent to our appeal function. Thus, on the full construction, the choices that Greedy PI makes under the average reward criterion are the same as those made under the total reward criterion. This gives us the following corollary:

Corollary 2. Let the parameter values be such that propositions 5 through 9 hold. If we start at the all-zero policy, Greedy PI on the full construction takes at least 2^n iterations to arrive at the optimal policy under the average reward criterion.

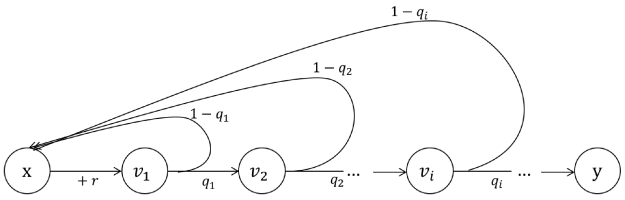

Figure 3: Gadget $g_2(k)$ with k intermediate vertices allows us to manufacture an exponentially large reward between x and y.

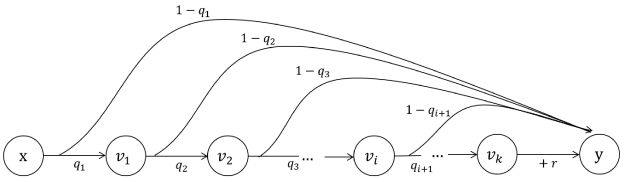

Figure 4: Gadget $g_3(k)$ with k intermediate vertices allows us to manufacture an exponentially small reward between x and y.

3.3 Robust Construction

We show that our construction can be made robust to perturbations of the rewards and probabilities by replacing certain edges with gadgets shown in Figures 3 and 4. We first present a gadget g_2 that allows us to manufacture exponentially large positive and negative rewards, as well as exponentially small probabilities.

Lemma 5. If the probabilities and rewards are perturbed by at most $\frac{1}{4k'}$ for any $k' \geq k$, the gadget $g_2(k)$ with reward $r = 1 - \frac{1}{2k}$ and probabilities $q_i = \frac{1}{2} + \frac{1}{4k'}$ yields $\mathrm{Val}(y) + 2^{k-2} \leq \mathrm{Val}(x) \leq \mathrm{Val}(y) + 2^k$ after perturbation. If $r = -(1 - \frac{1}{2k})$, the resulting negative reward is between -2^{k+2} and -2^k.

Lemma 6. Let $\mathrm{Appeal}(x)$ be the appeal of taking the action leading to the gadget g_2 from x. If the probabilities and rewards are perturbed by at most $\frac{1}{3000n^2}$, then for any $k \leq 1000n + 2$, the gadget $g_2(k)$ with reward $r = 0$ and probabilities $q_i = \frac{1}{n} + \frac{1}{1000n^2}$ yields $\mathrm{Appeal}(x) = (1-p)\mathrm{Val}(x) + p\mathrm{Val}(y)$ after perturbation, for $\frac{1}{n^k} \leq p \leq \frac{1}{n^{k-2}}$.

We also present a gadget g_3 allowing us to make exponentially small rewards. This is used to make the ϵ costs in the full construction.

Lemma 7. If the probabilities and rewards are perturbed by at most $\frac{1}{4k'}$ where $k' \geq k$, the gadget $g_3(k)$ with reward $r = 1 + \frac{1}{4k}$ and probabilities $q_i = \frac{1}{2} + \frac{1}{4k'}$ yields $\mathrm{Val}(y) + 2^{-k} \leq \mathrm{Val}(x) \leq \mathrm{Val}(y) + 2^{-k+2}$ after perturbation. If $r = -(1 + \frac{1}{4k})$, the resulting negative reward is between -2^{-k+2} and -2^{-k}.

3.3.1 Constructing the parameters $r(i), c(i), \epsilon, \delta_j, p_j, \alpha$. We use the gadgets to construct each of these parameters. The *raw* values of all the rewards and probabilities used in the resulting MDP lie in $[-2, 2]$ and the allowed perturbations are inverse polynomial.

Reward $r(i)$. We construct $r(i)$ using $g_2(7(i-1)+6)$, with $k' = 10n$. Each probability q_j is set to $\frac{1}{2} + \frac{1}{40n}$, with allowable perturbation up to $\frac{1}{40n}$. The reward r used in the gadget is set to $1 - \frac{1}{2(7(i-1)+6)}$, with perturbation up to $\frac{1}{40n}$.

Cost $c(i)$. We construct $c(i)$ using $g_2(7(i-1)+3)$, with $k' = 10n$. Each probability q_j is set to to $\frac{1}{2} + \frac{1}{40n}$, with allowable perturbation up to $\frac{1}{40n}$. The cost r used in the gadget is set to $-(1 - \frac{1}{2(7(i-1)+3)})$, with perturbation up to $\frac{1}{40n}$.

Cost ϵ. We construct ϵ using $g_3(100n)$, with $k' = 100n$. Each probability q_j is set to $\frac{1}{2} + \frac{1}{400n}$, with allowable perturbation up to $\frac{1}{400n}$. The cost r used in the gadget is set to $-(1 + \frac{1}{200n})$, with perturbation up to $\frac{1}{400n}$.

Reward δ_j. We set $\delta_j = \frac{2j}{4n^2} + \frac{1}{8n^2}$ and allow perturbation up to $\frac{1}{8n^2}$.

Probability p_j. We construct p_j using $g_2(4j+3)$ as described in Lemma 6, with reward $r = 0$, probabilities q_ℓ is set to $\frac{1}{n} + \frac{1}{1000n^2}$, with allowable perturbation up to $\frac{1}{2000n^2}$. Lemma 6 applies because $4j + 3 \leq 1000n + 2$.

Probability α. Recall that α is the probability associated with action 1 from each b_i. We construct α using $g_2(1000n+2)$ with reward $r = 0$, as described in Lemma 6. Each probability q_j within the gadget is set to $\frac{1}{n} + \frac{1}{1000n^2}$, with allowable perturbation up to $\frac{1}{2000n^2}$.

The allowed perturbations for the raw values in the gadgets induce for each parameter of the full construction ($r(i), c(i)$, etc.) an *effective* value that lies in a certain interval of uncertainty:

- $r(i) \in [2^{7(i-1)+4}, 2^{7(i-1)+6}]$
- $c(i) \in [2^{7(i-1)+1}, 2^{7(i-1)+3}]$
- $\epsilon \in [2^{-100n}, 2^{-100n+2}]$
- $\delta_j \in [\frac{2j}{4n^2}, \frac{2j+1}{4n^2}]$. Note that $0 < \delta_1 < \delta_2 < \ldots < \delta_{\max}$, and $\delta_{\max} \leq \frac{4}{n}$, and $|\delta_j - \delta_{j'}| \geq \frac{1}{4n^2}$
- $p_j \in \left[\frac{1}{n^{4j+3}}, \frac{1}{n^{4j+1}}\right]$
- $\alpha \in [n^{-1000n-2}, n^{-1000n}]$

3.3.2 Reproving the propositions. We reprove Propositions 5-9 and Lemmas 2-3 from Section 3.2.1. The propositions are sufficient to prove Theorem 2 for the full construction, giving us the analogous result for the robust construction: Greedy PI again requires at least 2^n iterations to arrive at the optimal policy.

While the original propositions were in terms of single global parameters ϵ and δ, we now have varying values of ϵ and δ for different actions due to the perturbations. We instead define upper bounds $\epsilon_{\max}$ and $\delta_{\max}$ and redefine the propositions in terms of these values; the propositions are otherwise unchanged from Section 3.2.1. More details are given in the full version.

3.3.3 Main theorem. Because the propositions imply the main theorem from the greedy construction section, Theorem 2, we have the analogous result for the robust construction given the same all-zero policy as in the full construction, which we state as the following lemma:

Lemma 8. Let the robust construction have parameters lying in $[-2, 2]$ as in Section 3.3.1, with perturbations of up to $\frac{1}{8n^2}$. When

started at the all-zero policy, Greedy PI takes at least 2^n iterations to arrive at the optimal policy under the total reward criterion.

We previously showed that Greedy PI takes at least 2^n iterations. Because of the gadgets, our MDP now has far more than n nodes (recalling that n is the number of bits). Letting N be the number of nodes, one can compute that $N \leq (n+1)(5n^2+1229n+8) \leq 6n^3$ for n sufficiently large. Thus, in terms of N, Greedy PI takes at least $2^{\sqrt[3]{\frac{N}{6}}}$ iterations. Recall that our perturbations were up to $\frac{1}{8n^2}$. Since $n \leq \sqrt[3]{\frac{N}{6}}$, perturbations of $\frac{1}{N}$ are at most $\frac{1}{8n^2}$ for sufficiently large n and N.

Theorem 3. There is an MDP with N nodes and all rewards in $[-2, 2]$, such that for any perturbations of the rewards and the transition probabilities up to $1/N$, Greedy PI takes at least $2^{\sqrt[3]{\frac{N}{6}}}$ iterations to arrive at the optimal policy under the total reward criterion.

By the same argument as in Section 3.2, since the MDP always terminates at a 0-reward sink state, we have as a corollary the same result for the average reward criterion:

Corollary 3. There is an MDP with N nodes and all rewards in $[-2, 2]$, such that for any perturbations of the rewards and the transition probabilities up to $1/N$, Greedy PI takes at least $2^{\sqrt[3]{\frac{N}{6}}}$ iterations to arrive at the optimal policy under the average reward criterion.

4 A LOWER BOUND FOR GREEDY PI UNDER THE REACHABILITY CRITERION

We prove an exponential lower bound for the reachability criterion in the worst case, without perturbations. The proof is based again on the full construction of Section 3.2. However, MDPs with the reachability criterion have no rewards and costs. To adapt our full construction for the reachability criterion, we introduce gadgets that simulate rewards and costs using random actions. The simulations are approximate, not exact, thus to ensure the correctness of the reduction, the starting MDP with rewards must behave robustly for parameter values within certain ranges. We use in this section the following ranges for the parameters of the full construction: $r(i) \in [2^{4i+2-5n}, 2^{4i+3-5n}]$; $c(i) \in [2^{4i-5n}, 2^{4i+1-5n}]$; $\epsilon \in [2^{-100n-1}, 2^{-100n}]$; $\delta_j \in [2^{-200n+2j}, 2^{-200n+2j+1}]$; $p_j = 2^{-400j^2}$; $\alpha = 2^{-400n-400f(n)^2}$. We again reprove Propositions 5-9 for these parameter ranges to show that Greedy PI takes exponentially many iterations.

We then introduce gadgets to eliminate the rewards and costs. These gadgets require known bounds on the minimum and maximum values of any nodes.

Claim 1. Given the parameters in this section, the maximum value of any node is at most $\frac{1}{4}$.

Now, observe that if we add a reward of $\frac{1}{4}$ upon reaching the sink, the behavior of policy iteration is not affected. This is because it increases the value of every vertex by exactly $\frac{1}{4}$, and also increases the appeal of every action by $\frac{1}{4}$. Thus, adding this reward does not affect any appeals or values relative to each other. Since policy iteration depends only on relative appeals, its behavior does not change.

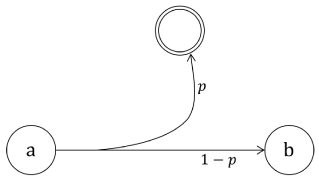

Figure 5: Gadget $g_4(p)$ for simulating positive rewards under the reachability criterion. The vertex with a double-line border is the sink that we are trying to reach. p is the probability of reaching the sink from the depicted action.

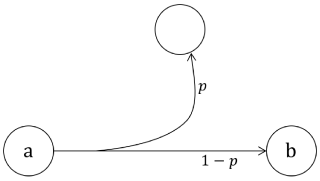

Figure 6: Gadget $g_5(p)$ for negative rewards under the reachability criterion. The top vertex is the 0-sink that we get no reward for reaching. p is the probability of reaching the sink from the depicted action.

After adding this reward of $\frac{1}{4}$ upon reaching the sink, the value of every node lies in $[\frac{1}{4}, \frac{1}{2}]$ throughout the duration of policy iteration. We use these bounds in our transformation to the reachability criterion, using the following gadgets.

Lemma 9. Assume that the value of every vertex lies in $\left[\frac{1}{4}, \frac{1}{2}\right]$. Then, for any $p \in [0, 1]$ and any deterministic action between vertices a and b, $g_4(p)$, shown in Figure 5, achieves $\mathrm{Val}(a) = \mathrm{Val}(b) + r$ where $r \in \left[\frac{p}{2}, \frac{3p}{4}\right]$.

We also create a gadget $g_5(p)$ that simulates a negative reward, shown in Figure 6.

Lemma 10. Assume that the value of every vertex lies in $\left[\frac{1}{4}, \frac{1}{2}\right]$. Then, for any $p \in [0, 1]$ and any deterministic action between vertices a and b, $g_5(p)$ achieves $\mathrm{Val}(a) = \mathrm{Val}(b) - r$ where $r \in \left[\frac{p}{4}, \frac{p}{2}\right]$.

We can convert the full construction to use the reachability criterion by using gadget g_4 in place of any action with a positive reward, and using gadget g_5 in place of any action with a negative reward. We slightly edit the actions a^i_j so that the rewards fall on deterministic actions, in order to apply the gadgets. This does not affect the behavior of policy iteration.

The full construction has $N = O(n)$ nodes in total, where n is the number of bit-nodes b_i. This conversion to the reachability criterion introduces at most $O(N)$ new nodes. Thus, we still have $O(N)$ vertices in total, giving us the following lower bound:

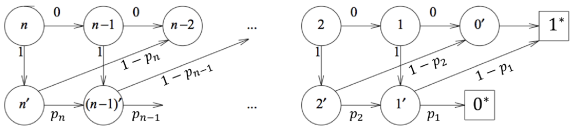

Figure 7: The basic graph on n vertices

Theorem 4. There exists an MDP on N nodes on which Greedy PI under the reachability criterion takes $2^{\Omega(N)}$ iterations to arrive at the optimal policy.

5 SMOOTHED LOWER BOUNDS FOR SIMPLE, DIFFERENCE, AND TOPOLOGICAL POLICY ITERATION

Melekopoglou and Condon [24] constructed MDPs on which policy iteration requires exponential time in the worst case when using the simple, topological, and difference switching rules, which switch only one state in each step. The MDPs in these constructions are reachability MDPs (thus there are no rewards). The construction involves two sink nodes, labeled 0^* and 1^*. The goal is to minimize the probability of reaching 1^*. There are $2n+1$ other vertices: $0', 1', \ldots, n'$, and $1, 2, \ldots, n$; see Fig. 7. Vertices $1, 2, \ldots, n$, which we call *min-vertices*, have two deterministic actions 0, 1. Vertices $1', \ldots, n'$, which we call *random vertices*, have one action with two probabilistic transitions as shown in Fig. 7. In [24] all transition probabilities were 1/2, but here we will parameterize them as $p_i, 1-p_i$ for vertex i'. Vertex $0'$ has one deterministic transition to $1*$

Let S_k denote the action chosen at vertex k. We write a policy as a string $S_n S_{n-1} \ldots S_1$. Thus, the policy where every vertex k takes action 0 is denoted $00\ldots0$. Observe that the optimal policy is $00\ldots01$, with vertex 1 set to action 1, and all other vertices set to action 0.

We use slightly different notation here than in the earlier sections, because the MDPs we consider are 2-action MDPs. We use $V(k)$ or $V(k')$ to denote the cost of a vertex k or k', i.e. the probability of reaching the target sink 1^*, starting from vertex k or k'; this is the same notation used in the original paper [24]. For each min-vertex k, we use $\mathrm{diff}(k)$ to denote the difference in value between the children of k. This is well-defined for this construction since each vertex k has at most two children. More precisely, we let $\mathrm{diff}(1) := V(1') - V(0')$, and for $k \geq 2$, we let $\mathrm{diff}(k) := V(k') - V(k-1)$.

We show that with small modifications, the Melekopoglou-Condon (MC) construction requires exponential time even when the probabilities assigned to random edges can be perturbed.

5.1 Simple Policy Iteration Algorithm

Consider Simple PI on the MDP of Fig. 7, where the fixed priority ordering of the min-vertices is $n, \ldots, 2, 1$; i.e., in each step, we choose to switch the highest-labeled switchable vertex.

Our proof follows the same format as the analogous one of MC. First, we derive an expression for $\mathrm{diff}(k)$. Then we use this expression to reprove lemmas from MC, which are sufficient to prove the lower bound.

Lemma 11. For every $k \geq 1$, $\mathrm{diff}(k) = \mathrm{diff}(1) \prod_{i=2}^{k} (p_i - S_{i-1})$.

Observe that k is switchable if and only if $S_k = 0$ and $\mathrm{diff}(k) < 0$ or $S_k = 1$ and $\mathrm{diff}(k) > 0$, since the goal is to minimize the cost. We can use Lemma 11 to compute $\mathrm{sgn}(\mathrm{diff}(k))$ and thus determine whether any vertex k is switchable given the actions of the other states. The original proof from [24] that policy iteration requires exponentially many iterations still holds because Lemmas 2.6 and 2.7 from [24] still hold. We state these below as Lemmas 12 and 13.

Lemma 12. If k is switched, and $S_n \ldots S_{k+2} S_{k+1} = 0 \ldots 01$, then all the vertices $k+1, k+2, \ldots, n$ are switchable.

Lemma 13. The following two statements hold for every $k \geq 1$:

(1) If $S_n \ldots S_{k+1} S_k = 0 \ldots 01$, and the vertices $k, k+1, \ldots, n$ are switchable, the next $2^{n-k+1} - 1$ switches of the simple policy improvement algorithm are made on these vertices, to reach the policy where $S_n \ldots S_{k+1} S_k = 0 \ldots 00$.
(2) If $S_n \ldots S_{k+1} S_k = 0 \ldots 00$, and the vertices $k, k+1, \ldots, n$ are switchable, the next $2^{n-k+1} - 1$ switches of the simple policy improvement algorithm are made on these vertices, to reach the policy where $S_n \ldots S_{k+1} S_k = 0 \ldots 01$.

Theorem 5. The simple policy improvement algorithm applied to the MDP of Fig. 7 requires 2^n iterations in the worst case, even when the probabilities associated with the random vertices can be perturbed arbitrarily within the open interval $(0, 1)$.

5.2 Topological Policy Iteration Algorithm

Modify the basic graph of Fig. 7, by assigning probability p_0 to the edge $(0', 1^*)$ and adding an edge $(0', n)$ with probability $1 - p_0$. Note that in the resulting MDP, all the min-vertices $1, \ldots, n$ are in the same strongly connected component (SCC). Consider the Topological PI algorithm on this MDP with the minimum reachability criterion. Since all min-vertices are in the same SCC, Topological PI switches in each step the highest-numbered switchable vertex.

It turns out that adding the edge from $0'$ to n does not affect the switches made by Simple PI (and therefore Topological PI), which we argue to obtain the following theorem.

Theorem 6. The topological policy iteration algorithm on the above MDP requires 2^n iterations in the worst case, even when the probabilities and cost are perturbed, as long as we have that $p_0, p_1, \ldots, p_n \in (0, 1)$, and the probabilities p_0 and p_1 satisfy $p_0 > 1 - p_1$.

5.3 Difference Policy Iteration Algorithm

The difference policy iteration algorithm. This vertex is chosen to maximize the difference between the costs of its two children; the chosen vertex v^* from the set of switchable min-vertices V satisfies $v^* = \arg\max_{v \in V} |\mathrm{diff}(v)|$ In this section, we use the basic graph of Fig. 7, and we insert gadgets used in [24], with different parameters to make the construction robust under perturbations.

For each min-vertex k, we add two copies of the gadget $g_1(f(k))$ shown in Figure 8 to the basic graph where f is a function that is defined later: one copy of the gadget is inserted between k and k', and one between k and $k-1$, replacing the corresponding edges of the basic graph. We assume that the probabilities q_i within the gadgets are between $\frac{1}{2}$ and $\frac{1}{2} + \frac{1}{n}$.

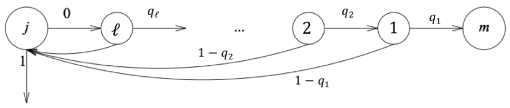

Figure 8: The gadget $g_1(\ell)$ from j to m, with ℓ intermediate nodes. The edges shown from j are actions 0 and 1. From each intermediate gadget node i, there is a single zero-reward probabilistic action going to $(i-1)$ with probability q_i and back to j with probability $1-q_i$.

Definition 4. Let k be a min-vertex. Let a be the child of k in the gadget between k and $k-1$; let b be the child of k in the gadget between k and k'. We define $\mathrm{diff}'(k) := V(b) - V(a)$

Let $q_1, \ldots, q_{f(k)}$ be the probabilities in the gadget from k to $k-1$, and let $r_1, \ldots, r_{f(k)}$ be the probabilities in the gadget from k to k'. Let a be the child of k in the gadget between k and $k-1$, and let b be the child of k in the gadget between k and k'. We first derive expressions for the values of diff yielded by the gadget in Lemma 14. We then use this expression to show that diff is increasing in n, so difference policy iteration follows the same sequence of policies as simple policy iteration, flipping higher-numbered bits first.

Lemma 14. If $S_k = 1$, $\mathrm{diff}'(k) = \left(\prod_{i=1}^{f(k)} q_i\right) \mathrm{diff}(k)$. If $S_k = 0$, $\mathrm{diff}'(k) = \left(\prod_{i=1}^{f(k)} r_i\right) \mathrm{diff}(k)$

Theorem 7. Let $f(n) = 0$, and let $f(k) = \lceil \log_{(\frac{1}{2}+\frac{1}{n})} ((\frac{1}{2})^{f(k+1)} \cdot (\frac{1}{3})) \rceil$ for $1 \le k < n$. On the MDP obtained from the basic graph by adding gadgets $g_1(f(v))$ between each min-vertex v and its two children, the difference policy improvement algorithm requires 2^n iterations, as long as all the probabilities q_j and r_j for each gadget lie in the open interval $(\frac{1}{2}, \frac{1}{2}+\frac{1}{n})$, and the probabilities p_i lie in $(0,1)$. Furthermore, for all k such that $1 \le k \le n$, $f(k) = O(\mathrm{poly}(n))$, and the MDP has size $O(\mathrm{poly}(n))$.

6 CONNECTIONS TO THE SIMPLEX ALGORITHM

Finding an optimal policy in an MDP can be formulated as a linear program (LP). First, we present a matrix encoding of the MDP given by Hansen in [19, Definition 2.1.2]. Let the MDP have N states and m actions. Let $\mathbf{e} \in \mathbb{R}^N$ be an all-one vector. Let $\mathbf{J}, \mathbf{P} \in \mathbb{R}^{m \times N}$. $\mathbf{J}$ represents the adjacency matrix, where for each action a, $\mathbf{J}_{a,i} = 1$ if $a \in A_i$, and $\mathbf{J}_{a,i} = 0$ otherwise. $\mathbf{P}$ represents the probabilities associated with the various actions. $\mathbf{P}_{a,i}$ is the probability of ending up in state i from action a. Let $\mathbf{c} \in \mathbb{R}^m$ represent the rewards of the actions. That is, $\mathbf{c}_a$ is the reward of taking action a.

We can solve the following linear program to obtain the optimal value y_i of each state i:

$$\begin{aligned} &\text{minimize } \mathbf{e}^T \mathbf{y} \\ &\text{subject to } (\mathbf{J} - \mathbf{P})\mathbf{y} \ge \mathbf{c} \end{aligned}$$

This is equivalent to the dual LP in [17].

Several single-state switching rules for policy iteration have been shown to be equivalent to pivot rules for the simplex algorithm. We show that two of our lower bounds, for Simple Policy and Improvement and Difference Policy Improvement, imply equivalent smoothed lower bounds for the simplex algorithm using Bland's and Dantzig's pivot rules respectively.

We note that the perturbations in our MDP setting translate to non-standard perturbations in the simplex setting. In the celebrated result by Spielman and Teng that the simplex method has polynomial smoothed complexity [30], all entries in the constraint matrix and right-hand-side are perturbed. In our MDP formulation, this would mean perturbing $(\mathbf{J} - \mathbf{P})$ and $\mathbf{c}$. With such perturbations, deterministic actions could become probabilistic, and perturbations of zero entries in $\mathbf{J}$ could create new edges between states.

Our smoothed MDP lower bounds translate to semi-smoothed simplex lower bounds, where weights can be perturbed but the general structure must be preserved (e.g., zero entries stay zero). More precisely, we do not perturb the adjacency matrix $\mathbf{J}$ at all. We perturb only the nonzero and non-one entries of $\mathbf{P}$. For each row a representing a random action, let i be the first state with nonzero $\mathbf{P}_{a,i}$. We define $\mathbf{P}_{a,i} = 1 - \sum_{j \ne i} \mathbf{P}_{a,j}$. We perturb all non-zero $\mathbf{P}_{a,j}$ for $j \ne i$. This ensures that the probabilities associated with each random edge sum to 1. We perturb all nonzero entries of $\mathbf{c}$. We say that an LP parameterized by $\mathbf{e}, \mathbf{J}, \mathbf{P}, \mathbf{c}$ that is perturbed in this way is *MDP-smoothed*.

Hansen shows that Bland's pivoting rule is equivalent to making the first improving switch according to some fixed permutation of the edges [19, Section 5.8]. This is exactly the simple policy improvement algorithm, where the edges are ordered according to the vertices' numbers. Thus, our result in Section 5.1 implies a 2^n lower bound on the number of iterations for the simplex algorithm using Bland's rule for LPs generated from MDPs, even when the probabilities and rewards can be perturbed.

Theorem 8. The worst-case MDP-smoothed complexity of the simplex algorithm with Bland's pivoting rule for LPs with dimension N, number of constraints $O(N)$, and allowed perturbations of up to $\frac{1}{2}$, is $2^{\Omega(N)}$.

There is in fact an exponential smoothed lower bound for Bland's pivoting rule under all zero-preserving perturbations of LPs, rather than our more structured MDP smoothing. Spielman notes in a lecture that the Klee-Minty cube is robust under zero-preserving perturbations, yielding an exponential lower bound [28].

Fearnley and Savani [16] show that Dantzig's pivot rule corresponds to policy iteration where the action with the greatest appeal is switched. Their definition of appeal is exactly our diff. Thus, the simplex algorithm with Dantzig's pivot rule is equivalent to the difference policy improvement algorithm. Our result in Section 5.3 implies an equivalent semi-smoothed result for the simplex algorithm with Dantzig's pivot rule.

Theorem 9. The worst case MDP-smoothed complexity of the simplex algorithm with Dantzig's pivot rule for LPs with dimension N, number of constraints $O(N)$, and allowed perturbations of up to $\frac{1}{\Omega(N)}$, is $2^{\Omega(\sqrt{N})}$.

7 CONCLUSION

We showed that under a natural smoothed model, several common variants of policy iteration have subexponential or exponential

lower bounds. Our main and most involved result concerns the standard Howard's PI (Greedy PI) algorithm and holds even when the perturbations are chosen arbitrarily (rather than randomly) within a certain inverse polynomial range. We further obtain an exponential lower bound on the number of iterations required by Howard's PI under the reachability criterion in the worst case, without perturbations. We also extend results from [24] to show that several single-switch PI variants (Simple PI, Topological PI, Difference PI) take at least exponential or subexponential time under large perturbations.

One natural direction for future work is to investigate where such lower bounds are not possible. Which perturbations yield polynomial expected runtime – in our model, do constant perturbations suffice? While we focused on the total reward, average reward, and reachability criteria, the discounted reward criterion is also popular. We suspect that similar results hold for discount rates γ that are exponentially close to 1, as the behavior of such discounted MDPs is similar to the total reward. On the other hand, if $1 - \gamma$ is at least inverse polynomial then we know that Greedy PI converges in polynomial time by the results of [18, 34].

For the reachability criterion, we showed a lower bound for Howard's PI only in the worst case. Can our result for the reachability criterion be extended to the smoothed and/or robust model? In the case of several PI variants that switch a single state in each iteration, Simple PI, Topological PI, and Difference PI, the bounds hold for reachability MDPs in the robust (and smoothed) model. Are there similar smoothed/robust lower bounds for other single-switch policy iteration variants, such as the Random-Facet and Random-Edge switching rules?

ACKNOWLEDGMENTS

This research was supported in part by NSF Grants CCF-2107187, CCF-1763970, and CCF-2212233, by JPMorgan Chase & Co, by LexisNexis Risk Solutions, and by the Algorand Centres of Excellence programme managed by Algorand Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are solely those of the authors.

REFERENCES

[1] Omer Angel, Sébastien Bubeck, Yuval Peres, and Fan Wei. 2017. Local max-cut in smoothed polynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 429–437.
[2] D. Avis and O. Friedmann. 2017. An exponential lower bound for Cunningham's rule. *Math. Program.* 161, 1-2 (2017), 271–305.
[3] Christel Baier, Luca de Alfaro, Vojtech Forejt, and Marta Kwiatkowska. 2018. Model Checking Probabilistic Systems. In *Handbook of Model Checking*, Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). Springer, 963–999.
[4] R. Bellman. 1957. *Dynamic Programming*. Princeton University Press.
[5] Ali Bibak, Charles Carlson, and Karthekeyan Chandrasekaran. 2021. Improving the Smoothed Complexity of FLIP for Max Cut Problems. *ACM Trans. Algorithms* 17, 3, Article 19 (July 2021), 38 pages. https://doi.org/10.1145/3454125
[6] Xi Chen, Chenghao Guo, Emmanouil-Vasileios Vlatakis-Gkaragkounis, Mihalis Yannakakis, and Xinzhi Zhang. 2020. Smoothed complexity of local Max-Cut and binary Max-CSP. In *Proceedings of the 52th Annual ACM SIGACT Symposium on Theory of Computing*.
[7] Costas Courcoubetis and Mihalis Yannakakis. 1995. The Complexity of Probabilistic Verification. *J. ACM* 42, 4 (1995), 857–907.
[8] Costas Courcoubetis and Mihalis Yannakakis. 1998. Markov decision processes and regular events. *IEEE Trans. Autom. Control.* 43, 10 (1998), 1399–1418.
[9] Daniel Dadush and Sophie Huiberts. 2020. A Friendly Smoothed Analysis of the Simplex Method. *SIAM J. Comput.* 49, 5 (2020).
[10] C. Derman. 1972. *Finite State Markov Decision Processes*. Academic Press.
[11] Amit Deshpande and Daniel A. Spielman. 2005. Improved Smoothed Analysis of the Shadow Vertex Simplex Method. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS*. IEEE Computer Society, 349–356.
[12] Yann Disser, Oliver Friedmann, and Alexander V Hopp. 2022. An exponential lower bound for Zadeh's pivot rule. *Mathematical Programming* (2022), 1–72.
[13] Matthias Englert, Heiko Roglin, and Berthold Vocking. 2016. Smoothed Analysis of the 2-Opt Algorithm for the General TSP. *ACM Transactions on Algorithms* 13, 1 (2016).
[14] Michael Etscheid and Heiko Röglin. 2017. Smoothed Analysis of Local Search for the Maximum-Cut Problem. *ACM Trans. Algorithms* 13, 2 (2017), 25:1–25:12.
[15] John Fearnley. 2010. Exponential lower bounds for policy iteration. In *International Colloquium on Automata, Languages, and Programming*. Springer, 551–562.
[16] John Fearnley and Rahul Savani. 2015. The complexity of the simplex method. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 201–208.
[17] Oliver Friedmann, Thomas Dueholm Hansen, and Uri Zwick. 2011. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*. 283–292.
[18] T. Hansen, P. Miltersen, and U. Zwick. 2013. Strategy Iteration Is Strongly Polynomial for 2-Player Turn-Based Stochastic Games with a Constant Discount Factor. *J. ACM* 60, 1 (2013), 1:1–1:16.
[19] Thomas Dueholm Hansen. 2012. *Worst-case analysis of strategy iteration and the simplex method*. Ph. D. Dissertation. Department Office Computer Science, Aarhus University.
[20] Romain Hollanders, Jean-Charles Delvenne, and Raphaël M Jungers. 2012. The complexity of policy iteration is exponential for discounted Markov decision processes. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 5997–6002.
[21] Romain Hollanders, Balázs Gerencsér, Jean-Charles Delvenne, and Raphaël M. Jungers. 2016. Improved bound on the worst case complexity of Policy Iteration. *Oper. Res. Lett.* 44, 2 (2016), 267–272.
[22] R. Howard. 1960. *Dynamic Programming and Markov Processes*. MIT Press.
[23] G. S. Lueker. 1975. Unpublished manuscript. (1975). Princeton University.
[24] Mary Melekopoglou and Anne Condon. 1994. On the complexity of the policy improvement algorithm for Markov decision processes. *ORSA Journal on Computing* 6, 2 (1994), 188–192.
[25] M. Puterman. 1994. *Markov Decision Processes*. Wiley.
[26] Alejandro A Schäffer and Mihalis Yannakakis. 1991. Simple local search problems that are hard to solve. *SIAM J. Comput.* 20, 1 (1991), 56–87.
[27] Bruno Scherrer. 2013. Improved and Generalized Upper Bounds on the Complexity of Policy Iteration. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems*. 386–394.
[28] Daniel A Spielman. 2002. The Behavior of Algorithms in Practice: Lecture 14. Scribe: Brian Sutton. http://www.cs.yale.edu/homes/spielman/BAP/lect14.pdf.
[29] Daniel A. Spielman and Shang-Hua Teng. 2009. Smoothed analysis: an attempt to explain the behavior of algorithms in practice. *Commun. ACM* 52, 10 (2009), 76–84.
[30] Daniel A Spielman and Shang-Hua Teng. 2004. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)* 51, 3 (2004), 385–463.
[31] Meet Taraviya and Shivaram Kalyanakrishnan. 2019. A Tighter Analysis of Randomised Policy Iteration. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI (Proceedings of Machine Learning Research, Vol. 115)*. AUAI Press, 519–529.
[32] Moshe Y. Vardi. 1985. Automatic Verification of Probabilistic Concurrent Finite-State Programs. In *26th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 327–338.
[33] Yue Wu and Jesús A. De Loera. 2022. Geometric Policy Iteration for Markov Decision Processes. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM.
[34] Y. Ye. 2011. The Simplex and Policy-Iteration Methods Are Strongly Polynomial for the Markov Decision Problem with a Fixed Discount Rate. *Mathematics of Operations Research* 36, 4 (2011), 593–603.

Received 2022-11-07; accepted 2023-02-06