There is no Accuracy-Interpretability Tradeoff in Reinforcement Learning for Mazes

Yishay Mansour∗ Michal Moshkovitz† Cynthia Rudin‡

Abstract

Interpretability is an essential building block for trustworthiness in reinforcement learning systems. However, interpretability might come at the cost of deteriorated performance, leading many researchers to build complex models. Our goal is to analyze the cost of interpretability. We show that in certain cases, one can achieve policy interpretability while maintaining its optimality. We focus on a classical problem from reinforcement learning: mazes with \( k \) obstacles in \( \mathbb{R}^d \). We prove the existence of a small decision tree with a linear function at each inner node and depth \( O(\log k + 2^d) \) that represents an optimal policy. Note that for the interesting case of a constant \( d \), we have \( O(\log k) \) depth. Thus, in this setting, there is no accuracy-interpretability tradeoff. To prove this result, we use a new “compressing” technique that might be useful in additional settings.

1 Introduction

Deploying interpretable policies for reinforcement learning (RL) tasks, where the policy is humanly-understandable, is an important aspiration. However, one might worry that restricting only to interpretable policies might come at a price. Thus, a fundamental question arises: what is the price of interpretability? There are complex domains for which uninterpretable policies are known while an interpretable policy has not been found, for instance, protein folding [AlphaFold, Jumper et al., 2021] or games like Go [Silver et al., 2016], chess [Silver et al., 2018] and Atari [Mnih et al., 2013]. However, the price of interpretability is unknown. Perhaps, an interpretable policy with good performance exists for these domains, only that no one has found one yet.

To understand whether the same policy can be both optimal and interpretable, we return to one of the first domains RL was studied in — mazes. In this domain, the agent plans a policy that takes it to a goal state. The agent moves up, down, left, or right at each state while not running into any obstacles.

Mazes are significant both from the theoretical perspective we consider here and from the practical perspective: numerous real-world robotic problems can be cast as mazes. Examples include robots traversing warehouse floors [Everett et al., 1995]; robots assembling products in factories [Fukuda, 1985] could also be modeled as a multi-dimensional maze, where the obstacles are constraints on how parts are assembled. In a sense, mazes can be used to model the decisions of AI-aided doctors providing drugs to critically ill patients over the timeframe they are in the intensive care unit of a hospital [Liu et al., 2017]; patients cannot receive more than a certain dose of a drug at a time, which can be viewed as an obstacle in a maze of doses that could be assigned to the patient.

In this paper, we analyze the price of interpretability for mazes. There are several types of interpretable models, but the gold standard is decision trees, which is our focus [see, e.g., Roth et al., 2019, Silva et al., 2020]. The input to the decision tree is the current state, and the output, at the leaf, is the action to perform. At each inner node, there is a linear function of the current state. The complexity of the policy is the number of steps until reaching a decision, i.e., the depth of the tree. We show the existence of an interpretable optimal
policy and show an efficient algorithm for finding such a policy. The algorithm’s output is a decision tree which implements an optimal policy, which guarantees that the agent reaches the goal state at minimum cost. The depth of the decision tree is provably bounded, but the depth is not necessarily the smallest. Namely, we prove the following.

**Theorem 1** (main theorem, informal). *For every maze with d dimensions and k obstacles, there is an optimal interpretable (i.e., sparse decision tree) policy with depth \(O(\log k + 2^d)\), which for a constant \(d\) is \(O(\log k)\). Additionally, if the starting state is sparse with at most \(n\) non-zero coordinates, then the policy depth is \(O(\log k + 2^n)\).*

To prove the theorem, we find a small subset of states, which depends only on \(k\) and \(d\), and prove that there is a structured optimal policy that traverses from the start state to one of those states, and continue traversing between states in this set until reaching the goal state. This observation reduces the original maze problem to only those states, and standard algorithms can solve mazes with this smaller number of states. This “compressing” technique for finding interpretable models might assist in additional settings. See the figure on the right for an example of the agent’s trajectory from Start to G, using the new technique. The small subset of states are the intersections of the gray lines.

An advantage of the policy we construct is that the agent does not need to traverse the entire decision tree when encountering a new state. More specifically, the agent traverses the tree once to find the action for the starting state, and all the following actions are found deterministically in time \(O(1)\). Additionally, we consider a “uniform noise” scenario, where the agent action might “fail.” More precisely, there is a fixed probability that the agent stays in the same place regardless of the action performed. We show that under this specific noise model the optimal policy does not change, and hence our results immediately carry over. (See Appendix C.)

**Summary of Contribution**

- **Main conceptual result**: There is an optimal interpretable policy for mazes, so no need to tradeoff interpretability with accuracy in mazes.

- **Quantitatively**: For mazes in \(d\) dimension and \(k\) obstacles, there is an optimal policy defined by a decision tree with depth \(O(\log k + 2^d)\). For the interesting case of 2-d and 3-d mazes this gives \(O(\log k)\) depth. If the starting state is sparse with at most \(n\) non-zero coordinates, then the depth of the decision tree is \(O(\log k + 2^n)\). The same results hold in the “uniform noise” model.

- **Running the agent**: The interpretable policy allows the agent to traverse efficiently in the decision tree — no need to traverse the entire tree upon reaching every new state. For the first step the agent will spend time \(O(\log k + 2^d)\), and in following steps it will spend only \(O(1)\) time.

- **Proof methodology**: We suggest a new “compressing” technique to build (and prove) an interpretable policy.

**Related Work**

**Interpretable policies.** Several works [Roth et al., 2019, Bastani et al., 2018, Liu et al., 2018, Verma et al., 2018, Coppens et al., 2019] build interpretable policies, either tree-based policies or “programmatically interpretable.” Unfortunately, these works are mostly heuristics, i.e., they do not provide guarantees for their
quality. Alternatively, Silva et al. [2020] focuses on a simple environment (simplified 1-d pole problem) with a straightforward optimal policy. In contrast, we provide provable guarantees for a general d-dimension maze problem.

Interpretability in machine learning. There are many works on machine learning interpretability, see Rudin et al. [2022], and most of them are heuristics. Our focus is on provably interpretable machine learning and mainly the price of interpretability. Dasgupta et al. [2020], Makarychev and Shani [2022], studied this price for unsupervised learning, specifically, for k-means clustering. The tradeoff is between interpretability and the quality of the model, measured by the k-means cost. These works explored the price, by proving lower and upper bounds on the k-means cost as a function of the tree’s complexity or when it is the smallest possible. Moshkovitz et al. [2021] proved an upper bound on the price of interpretability for supervised learning, under distributional assumptions. In this paper, we explore the price of interpretability for reinforcement learning.

Interpretable policies with human-in-the-loop. Different works find interpretable policies by utilizing additional external information. For example, Shu et al. [2017] builds hierarchical policies composed of different subtasks, where a human provides the subtasks. As the human feedback is limited, the optimality of the policies can be guaranteed. In this paper, we find an interpretable policy based solely on the Markov decision process.

Post-hoc policies explanations. The aim of interpretability is to make policies transparent. Another approach to gain (partial) transparency is by adding explanations to preexisting policies. Amir and Amir [2018], Sequeira and Gervasio [2020], Juozapaitis et al. [2019] explored different types of explanations, differing in what and how to explain (e.g., explaining rewards, states, or trajectories). The disadvantages of post-hoc explanations for critical systems is elaborated by Rudin [2019].

2 Setup and Problem Formulation

2.1 Maze Definition

An obstacle is an axis-aligned hyperrectangle \( R_{a,b} \), defined by two vectors \( a, b \in \mathbb{R}^d \) as

\[
R_{a,b} = \{ x \in \mathbb{R}^d : \forall i, a_i < x_i < b_i \},
\]

where \( x_i \) is the \( i \)-th coordinate of \( x \). The \((k, d)\)-Maze is a Markov Decision Process (MDP) which is a tuple \( \mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C} \rangle \) defined by \( k \) obstacles, \( O^1, \ldots, O^k \). When \( k \) and \( d \) are clear from the context, we sometimes refer to the \((k, d)\)-Maze simply as the maze. For ease of notation, we denote by \( O \) the set of all points comprising all of the obstacles, \( O = \bigcup_{i=1}^{k} O^i \). The (infinite) state space is \( \mathcal{S} = \mathbb{Z}^d \), where one of the states is the (absorbing) goal state, \( g \in \mathcal{S} \). There are \( 2d \) actions: \( \mathcal{A} = [d] \times \{\pm 1\} \). For \( d = 2 \), the actions correspond to the four actions: up, down, right, and left.

The transition function \( \mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S} \) is deterministic defined by the action chosen. If the agent tries to go through an obstacle, or move from the goal state, it will stay in place. More formally, for a state \( s \) and an action \( (i,b) \in [d] \times \{\pm 1\} \) we first define \( s' \in \mathbb{Z}^d \) as the move from \( s \), in the \( i \)-th feature by \( b \), i.e., \( s'_i = s_i + b \) and \( s'_j = s_j \) for all \( j \neq i \). Then, the transition function is defined as

\[
\text{Figure 1: 2D-maze. Goal state at } (0,0), \text{ denoted by } G. \text{ Grey: two obstacles. Policy navigates the agent from current square to the goal state.}
\]

\(^1\)for convenience, and w.l.o.g., the edges of the hyperrectangles are not part of the obstacle.
The cost $C: S \times A \rightarrow \mathbb{R}$ is 1 for each action, except actions at the goal state with cost 0 for each action, i.e., $C(s, a) = 1 \cdot \mathbb{1}_{s \neq g}$, where $\mathbb{1}$ is the indicator function.

A policy is a function $\pi: S \rightarrow A$. Any start state $s_0 \in S$ defines an episode $(s_0, s_1, \ldots, s_T)$ where $s_{t+1} = T(s_t, \pi(s_t))$. The episode terminates once $s_T = g$. We call $s_0$ the starting state. The total cost of a policy for this episode is the sum of the costs throughout, which is the number of actions in the episode, i.e., $T$. The agent aims to minimize the total cost of all possible episodes. The value function, $V^\pi: S \rightarrow \mathbb{R}$, of policy $\pi$ is the cost to reach the goal state from each state, when following policy $\pi$. The value of an optimal policy is denoted by $V^\star$.

### 2.2 Interpretable Policy

An interpretable policy is defined by a decision tree. Each internal node contains a linear threshold function of the current state. Thus, it iteratively partitions the data, leading to leaves. The leaves are labeled by the actions that the policy executes in the current state. The policy’s depth defines its complexity. In the paper, we aim to find an optimal interpretable policy with a small complexity.

#### 2.2.1 Efficient Implementation

In its most straightforward implementation, the interpretable policy traverses the tree upon reaching every new state, which is costly. Our interpretable policy is more efficient. First, as long as the leaf remains the same when moving to another state, there is no need to traverse the tree again, and the same action applies. For that, we use the notation

$$(i, b, \text{while condition}),$$

which means that we use the action $(i, b)$ (i.e., change feature $i$ by $b$) repeatedly as long as condition is satisfied. Once the leaf of the starting state is located, the next leaf can be found in $O(1)$. In fact, after traversing the tree once to locate the starting state, the policy does not need to traverse the full tree from the top ever again.

### 2.3 Problem Formulation and Main Theorem

Given $k$ obstacles and a goal state $g$, we aim to find an optimal policy $\pi^\star: S \rightarrow A$ that is defined by a decision tree with small depth. One cannot apply standard methods like policy iteration, since it does not find interpretable policies. Nonetheless, we design an algorithm that finds an interpretable optimal policy. We are now ready to state the main theorem.

**Theorem 2** (main theorem, formal). For every $(k, d)$-maze, there is an optimal interpretable policy $\pi^\star: S \rightarrow A$ with complexity $O(\log k + 2^d)$, which implies that for a constant $d$ it is $O(\log k)$. If the starting state has at most $n$ non-zero features, then the complexity of the policy is $O(\log k + 2^n)$.

### 3 Gentle Start: The Case of $d = 2$

To better understand our main result, we start with the special case that the dimension is $d = 2$. In this case the points are in the plane and obstacles are axis-aligned rectangles. The main idea would be to identify a subset of points, which we call corners. The set of corners would be of size at most $O(k^2)$, and the optimal policy would go from the start state to a corner, and continue going between corners until it reaches the goal. The main theoretical challenge would be to prove that indeed there is such an optimal policy, and then to compute it. Finally, we would implement the policy using a decision tree of depth $O(\log k)$.
We start by defining the set of points that we will call corners and related notions required for our presentation. We begin by introducing a few key terms. A horizontal (vertical) line is defined by an integer, e.g., the line $y = 1$ is encoded by the integer 1. An obstacle is defined by two horizontal and two vertical lines. For example, in Fig. 2a there are two obstacles. The top one is defined by two horizontal lines: $-1, 1$, and two vertical lines $3, 6$. The second obstacle is defined by the horizontal lines $-3, -6$ and vertical lines $1, 3$. We denote an obstacle as two pairs $(h_1, v_1), (h_2, v_2) \in \mathbb{R}^2$ where $h_1 < h_2$ are the horizontal lines and $v_1 < v_2$ are the vertical lines. (Note that the points $(h_1, v_1)$ and $(h_2, v_2)$ are the two opposite corners of the obstacle rectangle.)

We now define $H$ to be the set of all $h_i$ of the obstacles, and $V$ the set of all the $v_i$ of the obstacles. We also add the goal state $(g_1, g_2)$, namely add $g_1 \in H$ and $g_2 \in V$. The set of corners is $C = V \times H$ and the number of corners is $O(k^2)$. For example, the horizontal line $-6$ and the vertical line 3 define the corner $(3, -6)$.

Using sets $H$ and $V$ we can define the surface of a point, which will be an axis-aligned rectangle which is obstacles free. Fix a point $(x, y) \in \mathbb{R}^2$, its surface is defined by the following two segments, $X$ and $Y$. If $x \in V$, then $X = \{x\}$. Otherwise, $v^j < x < v^{j+1}$ and $X$ is the segment between $v^j$ and $v^{j+1}$, i.e., $X = [v^j, v^{j+1}]$. Similarly one can define $Y$. The surface of $(x, y)$ is

$$S((x, y)) = X \times Y.$$ 

See example in Fig. 2b. From the definition of the surface we have that the surface of a corner $c$ is the corner itself, $S(c) = \{c\}$.

The next claim proves that the agent can freely go from a state $s \in S$ to any point in its surface, $S(s)$, without encountering obstacles. Namely, for any $s' \in S(s)$ the agent can modify first the $x$ coordinate from $s_x$ to $s'_x$ and then from $s_y$ to $s'_y$, without hitting any obstacle in $\mathcal{O}$. This will give us the freedom to leave the surface in many different states. We utilize it to implement the interpretable policy. The general proof for any dimension appears in Section 4.

**Claim 1.** For any $s \in \mathbb{R}^2$, if $s \notin \mathcal{O}$ then $S(s) \cap \mathcal{O} = \emptyset$.

Another key term is adjacent corners. Two corners are adjacent if equal on one feature and the other feature has two consecutive lines in $H$ or $V$, while not having an obstacle between them, see Fig. 2b.

---

For ease of presentation we assume that $-\infty, \infty \in H, V$, where $\infty$ (respectively $-\infty$) is larger (smaller) than any number; and the segment $[a, \infty]$ ($[-\infty, a]$) is the half-open interval $[a, \infty)$ (respectively $(-\infty, a]$).
3.1 The Existence of a Structured Optimal Policy

The set of optimal policies is not necessarily a singleton, as exemplified in Fig. 3. When there are multiple optimal policies, there is flexibility in choosing among them. The next section uses this flexibility to construct one policy which has an interpretable (i.e., sparse and easy-to-understand) structure.

We focus on a policy $\tilde{\pi}$, see Algorithm 1, where the agent moves from one corner to an adjacent one, until reaching the goal state. The agent iteratively chooses its next move (i.e., which corner to go to next) using a new MDP $\mathcal{M}'$ defined on the $O(k^2)$ corners. There are four actions, one for each adjacent corner. The optimal policy, $\pi^*_{\mathcal{M}'}$, for the finite maze problem, $\mathcal{M}'$, can be found efficiently, e.g., using Dijkstra’s algorithm [Cormen et al., 2022] that finds shortest paths to the goal state.

The formal proof that policy $\tilde{\pi}$ is an optimal one appears in Section 4.

3.2 Building the Decision Tree

Building the decision tree is composed of two steps. Recall that the input to the decision tree is $(x, y)$. First, the decision tree detects the surface the starting state $(x, y)$ is in. Second, given the detected surface, the decision tree chooses which action to take. These two steps are illustrated in Fig. 4 for the starting state $(2, -2)$. The left decision tree finds the surface. The right decision tree continues and computes the optimal distance to the goal and the corresponding corner the agent should visit next (we elaborate more next).

**Algorithm 1 Policy $\tilde{\pi}$**

1. if $s$ is a corner then
2. return $\pi^*_{\mathcal{M}'}(s)$
3. else
4. $F \leftarrow$ corners in $S(s)$
5. $c = \arg \min_{c \in F} V^*_{\mathcal{M}'}(c) + \sum_{i=1}^2 |s_i - c_i|$  
6. return go to corner $c$
7. end if

**Step 1: finding the surface.** In the first step, we want to detect the location of the current state $(x, y)$ in the list of vertical lines, $V$. Namely, we want to find index $i$ such that $v^i \leq x < v^{i+1}$. The binary search technique enables us to find this $j$ using a decision tree of depth $O(\log k)$. In Fig. 4 (left), these are the first two levels of the decision tree. Similarly, we use the binary search technique over the horizontal lines and find an index $j$ such that $h^j \leq y < h^{j+1}$. At the end of this step each leaf corresponds to indexes $i, j$ and all points in the same leaf have exactly the same surface.

**Step 2: taking an action.** Our agent should go to one of the (at most) four corners in the starting state’s surface. For each of the corners we compute the distance from the starting state to that corner plus the distance from the corner to the goal. (In Fig. 4 we use $d(s, c_i)$ to be the distance to corner $c_i$ and $V(c_i)$ to be the distance from $c_i$ to the goal.) We then select as the next corner as the corner that minimizes total distance. Since Algorithm 1 is an optimal policy, we are guaranteed that there is an optimal policy that goes to one of the corners. Showing that indeed Algorithm 1 is optimal is deferred to the general case and appears in Lemma 2.
Putting it all together. The decision tree we build implements the optimal policy $\tilde{\pi}$, see Algorithm 1, and thus it is an optimal policy, see Section 3.1. The policy complexity is controlled by the first step, which is $O(\log k)$. For general dimension, see next section.

4 Algorithm for Building an Optimal Interpretable Policy

In the rest of the section we describe the general algorithm that builds the interpretable policy in $d$ dimensions. Following Section 3, the high-level approach is to construct a smaller MDP, $\mathcal{M}'$, and use an optimal policy on $\mathcal{M}'$ to construct the interpretable policy. In Section 4.3 we construct $\mathcal{M}'$ using the terminology introduced in Section 4.1 and Section 4.2. Then, in Section 4.4 we describe the desired algorithm. Importantly, this will prove that there exists an interpretable policy. Thus, there is no accuracy/interpretability tradeoff in low dimensional mazes.

4.1 Direct Access

We say that there is direct access between a state $s$ and a state $s'$ if DirectGoTo algorithm can be applied to go from $s$ to $s'$. Note that if the DirectGoTo can be applied, this is a shortest path with smallest cost between them, $d(s, s')$, where $d$ is the Manhattan distance function defined as $d(x, y) = \sum_{i=1}^{d} |x_i - y_i|$.

4.2 Corners, Surfaces and Adjacent Corners

In this section we introduce key terms like a corner and a surface that will be essential in the main proof. Recall that a maze is defined using a goal state $g \in \mathcal{S}$ and $k$ obstacles $R_{a_1, b_1}, \ldots, R_{a_k, b_k}$. We first define for each feature $i \in [d]$ a list of scalars $L_i \subseteq \mathbb{Z}$, which come from the $i$th coordinate of each obstacle and the goal.
Algorithm 2 DirectGoTo

1: **Input:** $s, s' \in S$
2: **Output:** series of actions
3: for $i \in [d]$ do
4:    if $s_i < s'_i$ then
5:        Take action $(i, +1)$ while $s_i < s'_i$
6:    else if $s_i > s'_i$ then
7:        Take action $(i, -1)$ while $s_i > s'_i$
8:    end if
9: end for

\[ L_i = \bigcup_{j=1}^{k_i} \{a_i^j, b_i^j\} \cup \{g_i\} \]

For example, for the maze in Fig. 1, $L_1 = \{0, 1, 3, 6\}$ and $L_2 = \{-1, 0, 1, 3, 6\}$. Denote $n_i = |L_i|$. Assume without loss of generality that the elements in $L_i$ are sorted $L_i[1] \leq L_i[2] \leq \ldots \leq L_i[n_i]$. A corner is a vector $c \in \mathbb{Z}^d$ such that all of its coordinates are in the list, i.e., $\forall i \in [d], c_i \in L_i$. We denote the set of all corners by

\[ C = L_1 \times \ldots \times L_d. \]

The total number of corners is $|C| = \prod_{i=1}^{d} n_i$. Two corners $c_1, c_2$ are adjacent if they are the same in all features, except exactly one, $i$, where $c'_1$ and $c'_2$ are consecutive in $L_i$, and there is no obstacle between them. It implies that there is direct access between them. Clearly, each corner has at most $2d$ adjacent corners.

Given a point $x \in \mathbb{R}^d$, we define its surface, $S(x)$, as the set of points that can be reached from $x$ until (and including) encountering a value in one of the lists $L_i$. More formally, define for each feature $i$ the segment $S^i$ in the following way. If $L_i[j] < x_i < L_i[j+1]$ then $S^i = [L_i[j], L_i[j+1])$ and if $x_i = L_i[j]$ then $S^i = [L_i[j]]$. The surface of $x$ is defined as $S(x) = S^1 \times \ldots \times S^d$.

For example, if $x$ is a corner then $S(x) = \{x\}$. Any surface contains at most $2^d$ corners. A key advantage of this definition for surfaces is that if the agent reaches a point $x$, it can directly reach all of its surface, $S(x)$ without hitting an obstacle. In other words, if a point is not in $O$, then none of its surface is in $O$.

**Lemma 1.** For any $x \in \mathbb{R}^d$, if $x \notin O$ then $S(x) \cap O = \emptyset$.

**Proof.** Fix $x \notin O$ and $y \in S(x).$ We will show that $y \notin O$. Assume by contradiction that $y \in O$. Then, there is an obstacle $R_{a,b}$ such that for all $i \in [d]$, it holds that $a_i < y_i < b_i$. Fix $i \in [d]$. If $x_i = L_i[j]$ then $y_i = x_i$ and $a_i < x_i < b_i$. If $L_i[j] < x_i < L_i[j+1]$, then, $L_i[j] \leq y_i \leq L_i[j+1]$. From the definition of the grid, $b_i \geq L_i[j+1]$ and $a_i \leq L_i[j]$ which implies that $a_i < x_i < b_i$. Thus, for all $i \in [d], a_i < x_i < b_i$. I.e., $x \in R_{a,b}$, which is a contradiction.

A corollary of Lemma 1 is that DirectGoTo can be applied to any state $s$ and $s'$, in its surface, $s' \in S(s)$, by advancing one feature at a time, while not running into any obstacle.

**Claim 2.** Algorithm DirectGoTo implements the shortest path from $s$ to $s'$ in the following cases

- If $s \in S$ and $s' \in S(s)$
- If $s \in C$ and $s' \in C$ is an adjacent corner to $s$

**Proof.** The algorithm is the shortest path, assuming it does not run into obstacles. If $s \in C$ and $s' \in C$ is an adjacent corner to $s$, DirectGoTo will not run into obstacles, by the definition of adjacent corners. If $s \in S$ and $s' \in S(s)$, then all states DirectGoTo traverses are in $S(s)$ and the claim follows from Lemma 1. \hfill $\square$

---

3For ease of presentation we assume that $-\infty, \infty \in L_i$ for each list; $\infty (-\infty)$ is larger (smaller) than any number; and the segment $[a, \infty] ([-\infty, a])$ is the half-open interval $[a, \infty]$ $((-\infty, a])$ and $[-\infty, \infty]$ is the open interval $(-\infty, \infty)$. 8
4.3 The Existence of a Structured Optimal Policy

This section shows the existence of one optimal policy that has a special structure. Intuitively, we show that there is an optimal policy that traverses between areas by moving from one corner to another. In designing the interpretable model, this will allow us to focus only on corners and ignore all other (infinite number of) states.

Formally, we define a new MDP, $M'$. The states are all the corners, $C$ in the grid. The actions are “go to adjacent corner.” There are at most 2$d$ actions from each state. The cost of action that moves from corner $c^1$ to an adjacent corner $c^2$ that differs on the $i$-th feature is $d(c^1, c^2) = |c^1_i - c^2_i|$, which is exactly the cost to move from state $c^1$ to state $c^2$.

Denote by $V^*_M: S \rightarrow \mathbb{R}$ the optimal value and by $\pi^*_M$ the optimal policy for MDP $M$. We will focus on the policy $\tilde{\pi}$ defined as Algorithm 3. The main theorem of this section is that $\tilde{\pi}$ is an optimal policy for $M$.

To prove it, we first show that the optimal value with respect to the full maze MDP, $M$, can be calculated using the optimal value for the smaller MDP, $M'$. Algorithm 3

Algorithm 3 Policy $\tilde{\pi}$

1: **Input:** $s \in S$
2: **Output:** actions
3: if $s \in C$ then
4: return $\pi^*_M(s)$
5: else
6: $c = \arg\min_{c \in S(s) \cap C} d(s, c) + V^*_M(c)$
7: return $\text{DirectGoTo}(s, c)$
8: end if

**Lemma 2.** For any state $s \in S$, it holds that

$$V^*_M(s) = \min_{c \in S(s) \cap C} d(s, c) + V^*_M(c).$$

This proves that indeed $\tilde{\pi}$ is an optimal policy. The proof of this lemma is in Appendix B.

4.4 The Algorithm

In this section, we show how to implement the optimal policy $\tilde{\pi}$ with an interpretable policy, i.e., a low depth decision tree. This interpretable policy is built in our main algorithm, Algorithm 4. It is composed conceptually of the following steps. (See Appendix A)

We first compute the set of corners $C$ from the set of obstacles. Then we build the MDP, $M'$, defined by the corners $C$, and compute an optimal policy and value function for $M'$. Finding the optimal policy $\pi^*_M$ for the finite, smaller MDP can be done by any of the standard planning algorithms [Sutton and Barto, 2018] in time $O(d|C|)$. Given the optimal policy $\pi^*_M$ for $M'$, the algorithm then calls the $\text{build_grid_tree}$ subroutine, which builds the decision tree.

The $\text{build_grid_tree}$ subroutine partitions the state space such that at each part, all states have the same surface. The subroutine, for each feature $i$, builds a search tree over $L^i$ to find an index $j$ such that $L^i[j] \leq x_i < L^i[j+1]$. This decision tree has depth $O(\log |C|)$. Special care is taken to ensure that in each leaf we have a single surface (details are in Appendix A). We then compute all the corners in each leaf (which is a single surface). If we are already in a corner, we continue to the next corner that $\pi^*_M$ defines, which we encode in the leaf. Otherwise, we continue to the $\text{build_direction_tree}$ subroutine.

The subroutine $\text{build_direction_tree}$ receives as input the corners of the current leaf. It chooses the best corner among them, where best minimizes the sum of the minimal distance to the corner plus the

---

Note that standard algorithms cannot be implemented directly to find the optimal policy of the original MDP, $M$ as $M$ has infinite number of states. This adds another layer of complexity for finding the optimal interpretable policy for $M$. 

9
minimal distance from the corner to the goal. (It is critical for our decision tree construction that the next state is a corner, which is guaranteed by Lemma 2). Finally, we encode in the current leaf the identity of the best corner.

**Analysis.** One of the main challenges in the proof is proving Lemma 2. The lemma shows that an optimal policy can move first to a corner and then continue as \( \pi^*_M \) indicates, i.e., moving only between corners. Put differently, the agent leaves each surface from a corner. Next, we outline the high-level idea for proving this lemma.

We show how to take an optimal policy \( \pi^*_M \) to \( M \) and modify it to ensure the agent leaves any surface from a corner. The modification is done in a backward manner from the goal state to the starting state. Denote the current state by \( s \). If the goal state is in the surface of \( s \), i.e., \( g \in S(s) \), the claim is immediate, as there is direct access between \( s \) and \( g \). Otherwise, focus on the backward path the agent takes from \( s \) to \( g \). Before reaching \( g \), we pause this path at the last time the agent reaches a state \( s' \) such that in some feature \( i \), it holds that \( s'_i \in L' \). This \( s' \) must exist because \( g \) is not in the surface of \( s \). We need to show that we can replace the path \( s \rightarrow s' \rightarrow g \) by a path \( s \rightarrow s' \rightarrow c \rightarrow g \), where \( c \) is a corner, and the cost of the two paths is identical.

We consider a corner \( c \), which is equal to \( g \) at every feature except \( i \), i.e., \( c_j = g_j \) for all \( j \in [d] - \{ i \} \). In the \( i \)-th feature \( c \) equals to \( s' \), i.e., \( c_i = s'_i \). Notably, the corners \( c \) and \( g \) are both in the surface of \( s' \). This means that the agent can freely go from \( s' \) to \( c \) and then to \( g \). Note that this path has the same cost as the path induced by \( \pi^*_M \) from \( s' \) to \( g \). The main benefit is that now the modified policy leaves from corner \( c \). The proof can continue by induction where the new goal state is the corner \( c \). (The full analysis and proof are in Appendix B.)

5 Conclusion and Open Problems

In this paper, we found an optimal policy with complexity \( O(2^d + \log k) \) for \((k,d)\)-mazes, implying that for mazes, there is no price for interpretability. This result holds even in the face of uniform noise: at each state, the agent stays at the same place with some probability.

Several problems are left open. Under what conditions can one reduce the complexity of the policy (or show a matching lower bound). In the paper we explored one type of noise. Nevertheless, building interpretable policies for other types of noise is of interest. Lastly, can this result be generalized to different settings beyond mazes, allowing for optimal and interpretable policies.

Acknowledgements

Yishay Mansour and Michal Moshkovitz has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (grant agreement No. 882396), by the Israel Science Foundation (grant number 993/17), Tel Aviv University Center for AI and Data Science (TAD), and the Yandex Initiative for Machine Learning at Tel Aviv University. Cynthia Rudin acknowledges funding under NSF IIS: 2147061, NSF CCF: 1934964, and NIDA R01 DA054994.
References

Dan Amir and Ofra Amir. Highlights: Summarizing agent behavior to people. In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, pages 1168–1176, 2018.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In Proceedings of Conference on Neural Information Processing Systems (NeurIPS), pages 2499–2509, 2018.

Youri Coppens, Kyriakos Efthymiadis, Tom Lenaerts, Ann Nowé, Tim Miller, Rosina Weber, and Daniele Magazzeni. Distilling deep reinforcement learning policies in soft decision trees. In Proceedings of the IJCAI Workshop on Explainable Artificial Intelligence, pages 1–6, 2019.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to Algorithms. MIT press, 2022.

Sanjoy Dasgupta, Nave Frost, Michal Moshkovitz, and Cyrus Rashtchian. Explainable k-means and k-medians clustering. In International conference on machine learning, pages 7055–7065. PMLR, 2020.

Hobart R Everett, Douglas W Gage, Gary A Gilbreath, Robin T Laird, and Richard P Smurlo. Real-world issues in warehouse navigation. In Mobile Robots IX, volume 2352, pages 249–259. SPIE, 1995.

Toshio Fukuda. Flexibility control of elastic robotic arms. Journal of Robotic Systems, 2:73–88, 1985.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. Nature, 596(7873):583–589, 2021.

Zoe Juozapaitis, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez. Explainable reinforcement learning via reward decomposition. In Proceedings of the IJCAI/ECAI Workshop on Explainable Artificial Intelligence, 2019.

Guiliang Liu, Oliver Schulte, Wang Zhu, and Qingcan Li. Toward interpretable deep reinforcement learning with linear model u-trees. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 414–429. Springer, 2018.

Ying Liu, Brent Logan, Ning Liu, Zhiyuan Xu, Jian Tang, and Yangzhi Wang. Deep reinforcement learning for dynamic treatment regimes on medical registry data. In 2017 IEEE International Conference on Healthcare Informatics (ICHI), pages 380–385. IEEE, 2017.

Konstantin Makarychev and Liren Shan. Explainable k-means. don’t be greedy, plant bigger trees! In Proceedings of the 54rd Annual ACM SIGACT Symposium on Theory of Computing, 2022.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

Michal Moshkovitz, Yao-Yuan Yang, and Kamalika Chaudhuri. Connecting interpretability and robustness in decision trees through separation. In International Conference on Machine Learning, pages 7839–7849. PMLR, 2021.

Aaron M Roth, Nicholay Topin, Pooyan Jamshidi, and Manuela Veloso. Conservative Q-improvement: Reinforcement learning for an interpretable decision-tree policy. arXiv e-print arXiv:1907.01180, 2019.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nature Machine Intelligence, 1(5):206–215, 2019.

Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. Statistics Surveys, 16:1–85, 2022.
Pedro Sequeira and Melinda Gervasio. Interestingness elements for explainable reinforcement learning: Understanding agents’ capabilities and limitations. Artificial Intelligence, 288:103367, 2020.

Tianmin Shu, Caiming Xiong, and Richard Socher. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. arXiv preprint arXiv:1712.07294, 2017.

Andrew Silva, Matthew Gombolay, Taylor Killian, Ivan Jimenez, and Sung-Hyun Son. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In International Conference on Artificial Intelligence and Statistics (AISTATS), pages 1855–1865, 2020.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587):484–489, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science, 362(6419):1140–1144, 2018.

Richard S Sutton and Andrew G Barto. Reinforcement Learning: An Introduction. MIT press, 2018.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In Proceedings of International Conference on Machine Learning (ICML), pages 5045–5054, 2018.

A Algorithms

In this section we provide the pseudocode of the algorithms described throughout the paper.

**Algorithm 4** Building Interpretable Optimal Policy

```plaintext
1: **Input:** lists \( L^1, \ldots, L^d \)
2: **Output:** interpretable policy
3: \( \# \text{ Construct MDP } \mathcal{M}' \)
4: \( \mathcal{C} \leftarrow L^1 \times \ldots \times L^d \) \( \# \text{ The states of } \mathcal{M}' \)
5: \( \mathcal{A}' \leftarrow \{ \text{“go to } j\text{-th adjacent corner”} \} \) \( \# \text{ The actions of } \mathcal{M}' \)
6: for \( c \in \mathcal{C} \) do
7: \( \# \text{ Construct transition and cost functions of } \mathcal{M}' \)
8: \( j \leftarrow 0 \)
9: for \( c' \) adjacent corner to \( c \) do
10: \( T'(c,j) \leftarrow c' \) \( \# j\text{th actions of } c, \text{ moves agent to } c' \)
11: \( C'(c,j) \leftarrow d(c,c') \) \( \# \text{ the cost of this action} \)
12: \( j \leftarrow j + 1 \)
13: end for
14: end for
15: \( \mathcal{M}' \leftarrow (\mathcal{C}, \mathcal{A}', T', C') \)
16: \( V_{\mathcal{M}'}, \pi_{\mathcal{M}'} \leftarrow \text{optimal value and policy for } \mathcal{M}' \)
17: **return** build_grid_tree(1, \( L^1, \ldots, L^d \), \( V_{\mathcal{M}'}, \pi_{\mathcal{M}'} \))
```
We start with a simplification of Lemma 2, and show it holds for any corner.

Algorithm 5 Build Grid Tree

1: Input: $i, L^1, \ldots, L^d, V^*_M, \pi_M^*$
2: Output: interpretable policy
3: if $|L^i| > 2$ then
4:   # If current list is bigger > 2: add inner node and continue building the tree recursively
5:   mid ← $\lfloor \frac{|L^i|}{2} \rfloor$
6:   node.condition ← $x_i \in L^i[\text{mid}]$
7:   node.less ← build_grid_tree$(i, L^1, \ldots, L^{i-1}, L^i[\text{mid}], \ldots, L^d)$
8:   node.eq ← build_grid_tree$(i, L^1, \ldots, L^{i-1}, L^i[\text{mid}], L^{i+1}, \ldots, L^d)$
9:   node.large ← build_grid_tree$(i, L^1, \ldots, L^{i-1}, L^i[\text{mid}], \ldots, L^d)$
10: return node
11: else if $i < d$ then
12:   # Alg. is done with the current list. If not the last list, continue to the next one
13:   return build_grid_tree$(i + 1, L^1, \ldots, L^d)$
14: else
15:   # Now alg. can detect the surface
16:   if $\forall i, |L^i| = 1$ then
17:     # if a corner move to next corner, following policy $\pi_M^*$
18:     leaf.action = $\pi_M^*(x)$
19:     return leaf
20: else
21:   # if not a corner, choose a corner to move to
22:   return build_direction_tree$(L^1 \times \ldots \times L^d, V^*_M)$
23: end if
24: end if

B Analysis

We start with a simplification of Lemma 2 and show it holds for any corner.

Claim 3. For any corner $c \in C$, it holds that $V^*_M(c) = V^*_M(c)$.

Proof. Fix an optimal policy $\pi_M^*$ for the original MDP $M$. For any corner $c \in C$, the inequality $V^*_M(c) \geq V^*_M(c)$ follows immediately as any trajectory in $M'$ corresponds to a trajectory in the original MDP with a similar total cost. To prove the interesting part, $V^*_M(c) \leq V^*_M(c)$, we will take the feature-wise point of view. Recall that for each feature $i \in [d]$, there is a list $L^i \subseteq \mathbb{R}$ that is induced by the obstacles and goal state of the MDP $M$. A point is a corner, $c \in C$, if for each coordinate $c_i \in L^i$. For a trajectory to $g$, $x^0, x^1, \ldots, x^T = g$, which is induced by $\pi_M^*$, we say that a feature-event happens at time $t < T$ if there is $i \in [d]$ such that $x^i_t \notin L^i$ and $x^{i+1}_t \in L^i$. We will prove the claim that for every corner $c \in C$ it holds that $V^*_M(c) = V^*_M(c)$, by induction on the number of feature-events in the trajectory from $c$ to $g$. Once the agent reaches the goal state the number of feature-events is zero, which proves the basis of the induction.

We turn to proving the induction step. Fix a corner $c \in C$ with $n > 0$ feature-events. Focus on the first feature-event, at state $s$. So the path that $\pi_M^*$ induces is $c \rightarrow s \rightarrow g$. Denote by $D \subseteq [d]$ the features that differ between $s$ and $c$ and by $i \in D$ the feature that caused the feature-event (i.e., $s_i \in L^i$ and for the state $x$ that is immediately before $s$, $x_i \notin L^i$). Denote by $c'$ the corner which is equal to $c$ at all features except $i$, i.e., for $\forall j \neq i, c'_j = c_j$ and in the $i$-th coordinate it is equal to $s_i$, i.e., $c'_i = s_i$.

From the definition of feature-event $c, c' \in S(s)$. From Claim 2 we know that $s$ can reach $c$ and $c'$ directly. This implies that the trajectory $c \rightarrow c' \rightarrow s \rightarrow g$ has similar cost as the original trajectory from $c$ to $g$. Number of feature-events for $c'$ is smaller than $n$. Thus, from the induction hypothesis, $V^*_M(c') = V^*_M(c')$.
We proved the claim as the following inequalities hold
\[ V^*_M(c) = d(c, c') + V^*_M(c') \]
\[ = d(c, c') + V^*_M(c') \]
\[ \geq V^*_M(c') \]

A similar proof applies to Lemma 2. For completeness, we repeat it here.

**Lemma 2.** For any state \( s \in \mathcal{S} \), it holds that
\[ V^*_M(s) = \min_{c \in \mathcal{S}(s) \cap \mathcal{C}} d(s, c) + V^*_M(c). \]

**Proof.** We generalize the definitions of \( V^*_M(s), V^*_M(s), \pi^*_M(s) \) to allow an arbitrary corner \( c \in \mathcal{C} \) to be a goal state and denote it as \( V^*_M(s,c), V^*_M(s,c), \pi^*_M(s,c) \). Note that the two definitions coincide when \( c = g \), i.e., \( V^*_M(s,g) = V^*_M(s), V^*_M(s,g) = V^*_M(s), \pi^*_M(s,g) = \pi^*_M(s) \). We will show a stronger claim: For any state \( s \in \mathcal{S} \) and a corner \( c \in \mathcal{C} \), it holds that
\[ V^*_M(s,c) = \min_{c' \in \mathcal{S}(s) \cap \mathcal{C}} d(s, c') + V^*_M(c', c). \]

To prove that for any state \( s \in \mathcal{S} \) and corners \( c, c' \in \mathcal{S}(s) \cap \mathcal{C} \) the inequality \( V^*_M(s,c) \leq d(s, c') + V^*_M(c', c) \) holds, we will show a trajectory from \( s \) to \( c \) with cost \( d(s, c') + V^*_M(c', c) \). The trajectory is from \( s \) to \( c' \) and then from \( c' \) to \( c \). The cost of the first term is \( d(s, c') \), from Lemma 1 and the cost of the second term is \( V^*_M(c', c) \), by Claim 3.

To prove that the inequality \( V^*_M(s,c) \geq \min_{c' \in \mathcal{S}(s) \cap \mathcal{C}} d(s, c') + V^*_M(c', c) \) holds for any state \( s \in \mathcal{S} \) and a corner \( c \in \mathcal{C} \) we take the feature point of view and define the feature-event, similarly to the previous claim. For a trajectory \( x^0, x^1, \ldots, x^T \), a feature-event happens at time \( t < T \) if there is \( i \in [d] \) such that \( x^i \notin L^i \) and \( x^{i+1} \in L^i \). We will prove the claim by induction on the number of feature events for any \( s \in \mathcal{S}, c \in \mathcal{C} \) and a trajectory between them defined by \( \pi^*_M \). If the number of feature-events is zero, then the \( s = c \) and \( S(s) = c \), and the claim immediately follows as \( V^*_M(c,c) = 0 = d(c,c) + V^*_M(c,c) \).

To prove the claim by induction on the number of feature-events, we traverse the trajectory from \( s \) to \( c \) induced by \( \pi^*_M(s,c) \), backwards from \( c \) to \( s \). Focus on the first (from last) feature-event, at state \( s' \). So the
trajectory that $\pi^*_M(s,c)$ induces, $p$, is $s \rightarrow s' \rightarrow c$. Denote by $D \subseteq [d]$ the features that differ between $c$ and $s'$ and by $i \in D$ the feature that caused the feature-event.

Denote by $c'$ the corner that is the same as $c$ except feature $i$ where it is equal to $s'$. I.e., for all $j \neq i$, $c''_j = c_j$ and $c''_i = s_i$. We will show that the trajectory $p' = s \rightarrow s' \rightarrow c'' \rightarrow c$ has the same cost as the original trajectory between $s$ and $c$. Since no feature-event happened between $s'$ and $c$, there is a direct access between them and the cost from $s'$ to $c$ is equal to $d(s',c) = \sum_{j \in D} |s'_j - c_j|$. Denote the cost from $s$ to $s'$, induced by $\pi^*_M$, by $V$. Then, the cost of $p$ is $V_M(s,c) = V + d(s',c) = V + d(s',c') + d(c',c)$. The cost of $p'$ is $V + d(s',c') + d(c',c)$.

$$V^*_M(s,c) = V + d(s',c')$$
$$= V + d(s',c'') + d(c'',c')$$
$$\geq V^*_M(s,c') + d(c',c)$$

The trajectory from $s$ to $c''$ has less feature-events. Thus, from the induction hypothesis, $V^*_M(s,c'') = \min_{c' \in S(s) \cap C} d(s,c') + V^*_M(c',c'')$.

$$V^*_M(s,c) \geq \min_{c' \in S(s) \cap C} d(s,c') + V^*_M(c',c'')$$
$$\geq \min_{c' \in S(s) \cap C} d(s,c') + V^*_M(c',c),$$

which is what we wanted to prove. 

**Proof.** (of main theorem) We need to show that $\hat{\pi}$ is implemented by Algorithm 1. Importantly, each leaf in the grid has the same surface. Thus, finding the best corner, as done in Algorithm 1, implements policy $\hat{\pi}$.

The depth of the tree is bounded by the depth of finding the area, which is $O(d \log k)$, and finding the best corner. The depth of finding the best corner is the same as the number of corners in a surface, which is $L' \times L'$ for the sets $L'$ corresponding to the leaf with $|L'| = 1$ or $|L'| = 2$. In the worst case, $|L'| = 2$ for all $i$, so the number of corners is bounded by $2^d$. However, if the starting state is sparse, the bound can be highly improved. Specifically, assume that the starting state has at most $n$ coordinates which are nonzero. Then, add 0 to each each set $L'$. And the number of corners of the starting state is only $2^n$. For sparse starting states, $2^n \ll 2^d$.

We remark that a generalization of the theorem to nonsparse vectors is possible in the following sense. Suppose there are $O(1)$ vectors, $V$, such that every starting state $s$, its Manhattan distance to some vector $v \in V$ is at most $n$. Then, results still hold, by adding $V$ to the $L'$ lists.

## C Uniform Noise

This section proves that adding noise to each state does not alter the optimal policy. Thus, it is enough to focus on a deterministic MDP, as this paper does. More formally, fix an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, C)$. The uniformly noisy MDP is defined as $\mathcal{M}^N = (\mathcal{S}, \mathcal{A}, T, C, \alpha)$, where

$$P(s'|s,a) = (1 - \alpha) \cdot I_{T(s)=s'} + \alpha \cdot I_{s'=s},$$

for some fixed $\alpha \in [0,1]$ and $I$ is the indicator function. In words, for the uniform noisy MDP, there is a probability of $\alpha$ to stay at the same state and a probability of $1 - \alpha$ to follow the deterministic MDP. Same $\alpha$ for every state. We begin with a claim that will prove helpful later on; if all costs are multiplied by a scalar, the optimal policy does not change.

**Claim 4.** Fix MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, C, g)$. Suppose that all costs are multiplied by $c > 0$. Then, optimal policy for $\mathcal{M}^c = (\mathcal{S}, \mathcal{A}, T, c \cdot C, g)$ does not change.

**Proof.** Fix a policy $\pi$. Denote by $V^\pi$ the value function of $\pi$ for $\mathcal{M}$ and by $V^{c \cdot \pi}$ the value function of $\pi$ for $\mathcal{M}^c$. Then, $V^{c \cdot \pi}(s) = c \cdot V^\pi(s)$. An optimal policy is one that minimize the value function at each state and

$$\arg \min_{\pi} V^{c \cdot \pi}(s) = \arg \min_{\pi} V^\pi(s),$$

thus proving the claim.

---

15
Now for the main claim for this section.

**Lemma 3.** If $\pi$ is an optimal policy for $\mathcal{M}$, then $\pi$ is an optimal policy for $\mathcal{M}^N$.

**Proof.** For every $s \in \mathcal{S}$, denote by $s' = T(s)$. The Bellman equations for the noisy MDP are

$$
V^N(s) = 1 + \alpha V^N(s) + (1 - \alpha) V^N(s')
$$

$$(1 - \alpha) V^N(s) = 1 + (1 - \alpha) V^N(s').
$$

$$
V^N(s) = \frac{1}{1 - \alpha} + V^N(s').
$$

These are the same equations for the original MDP besides the cost multiplied by $1/(1 - \alpha)$, use previous claim to prove the lemma.

As a corollary we have the main theorem for the uniformly noisy setting. We are now ready to state the main theorem.

**Theorem 3.** For every uniformly noisy $(k, d)$-maze there is an optimal interpretable policy $\pi^* : \mathcal{S} \to \mathcal{A}$ with complexity $O(\log k + 2^d)$, which implies that for a constant $d$ it is $O(\log k)$.

Similarly to the non-noisy setting, if the starting state is sparse with at most $n$ non-zero coordinates, then the depth of the policy is $O(\log k + 2^n)$. 

16