OPTIMAL HITTING TIME IN MARKOV CHAINS WITH REWINDING: A THEORETICAL FRAMEWORK FOR LLM TEST-TIME COMPUTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We study the optimal hitting-time problem in Markov chains with rewinding Azarmehr et al. (2026), motivated by verifier-guided test-time computation. Originally introduced in the study of lower bounds for sublinear-time graph algorithms, Markov chains with rewinding provide a natural model for the interaction between an algorithmic agent and randomness: at any point, the algorithm may resume generation from any previously observed state. That is, unlike standard Markov chains where the states are drawn "passively", we allow the algorithm to "backtrack" to any previously observed state of the Markov chain at any time. This model further extends value-guided backtracking Rohatgi et al. (2025), which implements backtracking through a verifier-guided random walk on the tree of partial generations.

Large language models (LLMs) can often produce substantially better outputs when allowed to use additional test-time computation through methods such as sampling, chain of thought, backtracking, or revising partial solutions. Despite the growing empirical success of such techniques, there is limited theoretical understanding of how test-time computation should be structured, or what constitutes an optimal use of budget. Many widely used test-time methods, including Chain-of-Thought (CoT) Wei et al. (2023), Tree-of-Thoughts (ToT) Yao et al. (2023), or Best-of-k Brown et al. (2024) could be seen as specific algorithms in this model.

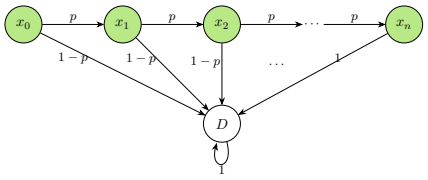
We fully characterize the optimal strategy in this model and show that it always generates a "caterpillar" tree. That is, if we remove the leaves of the state tree generated by the optimal algorithm, we obtain a path. We also construct Markov chains in which backtracking yields an exponential reduction in the number of generations, as well as instances in which adversarial verifier noise causes an exponential slowdown. Guided by the structure of the optimal strategy and these hard instances, we introduce a heuristic which we call Caterpillar of Thoughts (CaT), a new test-time computation algorithm, reducing the number of token/state generations. Our empirical evaluation shows that CaT, compared to ToT, achieves a better success rate while also reducing the number of token generations.

1 INTRODUCTION

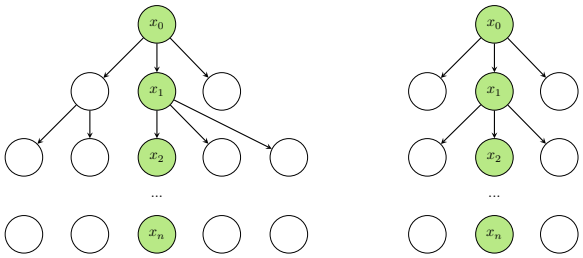
Recent advances in large language models have demonstrated that performance can often be improved not by additional training, but by allocating more computation at inference time. Techniques such as self consistency Wang et al. (2023), tree-of-thought exploration Yao et al. (2023), iterative refinement Madaan et al. (2023), recursive self-aggregation Venkatraman et al. (2025), verifier guided search Lightman et al. (2023); Botta et al. (2025); Snell et al. (2024); Wang et al. (2025), and backtracking von Rütte et al. (2025); Yang et al. (2025) allow an LLM to sample multiple continuations, revisit earlier partial solutions, and selectively pursue promising directions. These methods have proven effective across various reasoning, planning, and creative generation tasks, suggesting that inference time computation plays a central role in enabling more capable and adaptive language agents.

Despite this progress, the algorithmic foundations of test-time computation remain poorly understood. Most existing approaches are motivated heuristically, and it is unclear how close they are to optimal,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) A Markov chain where rewinding is crucial, consisting of $n+1$ states $x_0 \to x_1 \to \ldots \to x_n$ on a path, and one dummy state D. The arrows denote the transition probabilities. The first state on the path, x_0, is the starting point of the algorithm, and the last state, x_n, is the target state.



(b) This figure illustrates the tree explored by two different algorithms, with Figure 1a being the input Markov chain. White circles represent state D, and green states represent the optimal path to the target state. The right side tree represents the explored tree by CaT, and the left side tree represents a BFS-like algorithm that expands without pruning the dummy state.

or how performance should scale with additional inference-time resources. More fundamentally, *given a fixed but non-trivial budget, what is the best possible way to deploy test-time compute?*

1.1 Our Contributions

A Theoretical Model of Test-Time Computation: To answer the question above, we need a theoretical model for test-time computation. We model this problem as exploring a (massive) Markov chain, which is defined based on a pretrained LLM. Each state in the Markov chain represents a *partial solution*, i.e., a sequence of tokens. From any state, the LLM induces a distribution over successor states via sampling new tokens. That is, the Markov chain randomly transitions to a new state, corresponding to the solution obtained by appending the new tokens to the current partial solution. Crucially, modern inference time methods allow the algorithm to rewind: rather than continuing only from the most recently generated state, it may resume generation from any previously observed state.

This leads naturally to the model of a fully observable Markov chain with rewinding, in which an algorithm adaptively chooses both where to rewind and how long to explore. The goal is to reach a designated target state representing, for example, a correct answer or a sufficiently high-quality output, while minimizing the expected number of steps (i.e., total number of observed states). We remark that allowing the algorithm to rewind can exponentially decrease the number of steps (This is discussed further in Section 3.3, see Figure 1a for an example).

The set of observed states can be interpreted as a tree: each observed state corresponds to a node of the tree, and rewinding to a previous state opens a new branch at the corresponding node. Many existing test-time inference strategies fit into this framework. For example, Chain of Thoughts (CoT) Wei et al. (2023) explores a single path in the Markov chain, starting at a carefully chosen initial state. Best of k and self-consistency Wang et al. (2023) can be interpreted as exploring multiple paths that share the same root, and are otherwise independent. Other approaches such as Tree of Thoughts (ToT) Yao et al. (2023) and Monte Carlo Tree Search (MCTS) Dainese et al. (2024); Park et al. (2024); Liu (2025) employ more intricate methods for exploring the tree. Our rewinding Markov chain model can simulate these strategies by appropriate choices of the rewinding distribution over the observed set. Moreover, recent verifier-guided backtracking methods can also be expressed within our framework: in Section A we show that the proposed Markov chain of Rohatgi et al. (2025) is recovered as a special case of our model, and our framework generalizes it.

The Optimal Algorithm: Our main contribution is characterizing the optimal solution in this model, i.e., presenting an algorithm for exploring the Markov chain that reaches the target state with the minimum number of steps, in expectation. We show that no matter what the Markov chain looks like, the optimal generated tree is always a *caterpillar*. That is, all the nodes are within distance 1 of a central path, and the algorithm may rewind only to withdraw the most recently generated state. More precisely, for each state x, we define a value $\mathsf{OPT}(x)$, which is the optimal number of steps that an algorithm requires in expectation to reach the target state from x. This value guides our algorithm in exploring the Markov chain, and can be computed exactly given full knowledge of the Markov

chain, or otherwise approximated. That is, the algorithm maintains a *minimizer state* x^*, initialized to the starting state. Then, it repeatedly generates the next state g from x^*, until it obtains a state with $\mathsf{OPT}(g) < \mathsf{OPT}(x^*)$, which in turn replaces x^* and undergoes the same process. This is formalized below as Algorithm 1.

Theorem 1.1. *Given an initial state x_0, Algorithm 1 reaches the target state using the optimal number of steps in expectation, i.e., in $\mathsf{OPT}(x_0)$ steps.*

Observe that the algorithm always keeps track of only two states, the minimizer state x^*, and the most recently generated state g. It may rewind to x^* or otherwise continue from g, resulting in a tree that consists of leaves hanging from a central path, i.e., a caterpillar.

Massive Markov Chains: We note that, in practice, computing the exact value of $\mathsf{OPT}(x)$ might not be feasible given that the Markov chain corresponding to a pre-trained LLM could be huge. A nice property of Algorithm 1 is that only access to the values of $\mathsf{OPT}(\cdot)$ is sufficient for its implementation. In other words, given oracle access to $\mathsf{OPT}(\cdot)$, additional knowledge of the Markov chain's structure is not helpful. In this paper, we also study whether approximations of this oracle are sufficient. In particular, in Section 4, we present an algorithm that reaches the target state in $\text{poly}(\mathsf{OPT}(x))$ steps, given only noisy approximations of OPT.

Experiments: In Section 5, we evaluate a variant of the optimal Algorithm 1 (which is more robust when we only have approximations of $\mathsf{OPT}(x)$) empirically on two standard reasoning benchmarks of the literature: the Game of 24 and 5×5 crosswords. See Table 1 for a direct comparison between this algorithm and Tree-of-Thoughts (ToT). While our algorithm improves the success rate significantly from 74% to 81%, it also reduces the number of generated tokens (and thus the cost of evaluations) by more than 20%.

Algorithm 1 Caterpillar of Thoughts

Input: An initial state x_0, and a target state z
$x^* \leftarrow x_0$
while $x^* \neq z$ **do**
  $g \leftarrow$ next state drawn from x^*
  if $\text{opt}(g) < \text{opt}(x^*)$ **then**
    $x^* \leftarrow g$
  end if
end while

Table 1: In the Game of 24 benchmark Tree-of-Thoughts uses a beam size of 5. The proposed method reports best of 2 runs, for maximum budget of iterations set to 10 and 15. By tokens we mean the sum of generated and prompt tokens.

Method	Success Rate(%)	Avg. Tokens
Tree-of-Thoughts Yao et al. (2023)	74	19.2k
CaT (15 steps)	**81**	**15.3k**
CaT (10 steps)	**78**	**14.2k**

2 PRELIMINARIES

The Formal Model: We consider a finite-state fully observable Markov chain $M = (\Omega, P)$ with transition probabilities $P : \Omega \times \Omega \to [0, 1]$. In the rewinding model, unlike a standard Markov chain, in each step the process is allowed to *rewind* to any previously visited state.

Given a fully observable Markov chain $M = (\Omega, P)$, an initial state $x_0 \in \Omega$, and a target state $z \in \Omega$, the goal is to devise a rewinding algorithm that minimizes the expected number of steps to obtain the target state z.

Definition 2.1 (Rewinding Algorithms)**.** Given a fully observable Markov chain $M = (\Omega, P)$, and an initial state x_0, in every step $t \geq 0$, a rewinding algorithm chooses a rewinding time $t' \leq t$, and obtains a new state x_{t+1} according to the transition probabilities of $x_{t'}$. We refer to $x_{t'}$ as the parent state of x_{t+1}, and denote it by $p(x_{t+1})$.

Our main contribution is characterizing the optimal algorithm, which reaches the target state in the minimum number of steps.

Definition 2.2 (The Optimal Algorithm)**.** Given a tree of states observed so far, $X = (x_0, \cdots, x_t)$, a rewinding algorithm for the remainder of the process is optimal if it minimizes the expected number of steps before obtaining the target state z. We use $\mathsf{OPT}(X)$ to denote the expected hitting time of such an optimal algorithm.

For the remainder of the paper, we use $\mathsf{O}(\cdot)$ instead of $\mathsf{OPT}(\cdot)$ for brevity.

3 Fully Observable Markov Chain

3.1 Optimal Algorithm

This section is devoted to the proof of our main theorem, Theorem 1.1. First, we establish that the optimal algorithm depends only on the set of distinct states observed so far, and the exact structure of the tree is inconsequential. The proof is deferred to Section C.

Lemma 3.1. *There exists an optimal rewinding algorithm such that given a sequence of states $X = (x_0, \cdots, x_t)$, the rewinding time $t' \leq t$ chosen for obtaining the next state is determined solely based on the set of distinct states S in X. We use $\mathsf{O}(S) := \mathsf{O}(X)$ to denote the optimal hitting time corresponding to S, and $\mathsf{O}(x)$ as a shorthand for $\mathsf{O}(\{x\})$.*

Next, we define non-branching rewinding algorithms, which are algorithms that evolve a caterpillar tree.

Definition 3.2 (Non-branching Algorithms)**.** A rewinding algorithm is **non-branching** if in any step $t \geq 0$, it either rewinds to the most recently obtained state x_t, or to $p(x_t)$, i.e. to the same rewinding time as the previous step. For a state $a \in \Omega$, we let $\mathsf{O}_{\mathcal{NB}}(a)$ denote the minimum expected hitting time of z for non-branching algorithms starting at a.

The key idea behind the proof of Theorem 1.1 is that the optimal algorithm is a non-branching one.

Theorem 3.3. *For any set of states $S \subseteq \Omega$, there exists a state $x \in S$ such that*

$$\mathsf{O}(S) = \mathsf{O}_{\mathcal{NB}}(x).$$

This directly implies that, with a set of states S observed so far, the optimal algorithm is a non-branching one, which starts at the state $x \in S$ that minimizes $\mathsf{O}(x)$.

Corollary 3.4. *Given an initial set of states S, we have $\mathsf{O}(S) = \min_{x \in S} \mathsf{O}(x) = \min_{x \in S} \mathsf{O}_{\mathcal{NB}}(x)$.*

We restate Theorem 1.1 here.

Theorem 1.1. *Given an initial state x_0, Algorithm 1 reaches the target state using the optimal number of steps in expectation, i.e., in $\mathsf{O}(x_0)$ steps.*

We use Theorem 1.1 to obtain a recursive expression of $\mathsf{O}(\cdot)$, which we then utilize to compute the values of $\mathsf{O}(\cdot)$, in Section 3.2.

Corollary 3.5. *For a state $x \neq z$, let $L \subseteq \Omega$ be the set of states y such that $\mathsf{O}(y) < \mathsf{O}(x)$. Then, it holds that $\mathsf{O}(x) = \frac{1 + \sum_{y \in L} P(x,y)\mathsf{O}(y)}{P(x,L)}$.*

3.2 Computing $\mathsf{O}(x)$

We present a Dijkstra-like algorithm for computing the value of $\mathsf{O}(x)$ for every state x. The algorithm maintains an upper bound d_x for $\mathsf{O}(x)$ along with a set of states T that grows in every step. At any point, for every $x \in T$, it holds that $d_x = \mathsf{O}(x)$. The algorithm is formalized as Algorithm 2.

Theorem 3.6. *Algorithm 2 computes $\mathsf{O}(x)$ for every state $x \in \Omega$ and runs in time $O(n^2)$, where $n = |\Omega|$ denotes the number of states in the Markov chain. If m entries of the transition probabilities $P(\cdot, \cdot)$ are non-zero, then the algorithm can be implemented in time $O((m + n) \log n)$.*

3.3 The Necessity of Rewinding

In this section, we discuss the role of rewinding in the efficiency of algorithms. We show that allowing the algorithm to rewind to previous states can exponentially improve the expected run time, compared to algorithms that do not allow rewinding or allow limited forms of rewinding. Essentially, without rewinding, the algorithm may get "lost" and never recover.

Consider, as an example, the Markov chain depicted in Figure 1a, with $p = \frac{1}{2}$. The optimal algorithm operates as follows. At any point, it maintains a state x_i, which is the furthest state observed in the

Algorithm 2 Computes the optimal hitting time for each initial state.

Input: A Markov chain $M = (\Omega, P)$, and a target state $z \in \Omega$.
$T \leftarrow \varnothing$
$d_x \leftarrow \begin{cases} 0 & \text{if } x = z, \text{ and} \\ \infty & \text{otherwise} \end{cases}$
while $T \neq \Omega$ and $\min_{x \notin T} d_x < \infty$ **do**
  $x^* \leftarrow \arg\min_{x \notin T} d_x$ (break ties arbitrarily)
  $T \leftarrow T \cup \{x^*\}$
  for $x \notin T$ **do**
    if $P(x, T) > 0$ **then**
      $d_x \leftarrow \frac{1 + \sum_{y \in T} P(x,y) d_y}{P(x,T)}$
    end if
  end for
end while
return $\{d_x\}_{x \in \Omega}$

Algorithm 3 Stable Algorithm (for noisy approximations)

Input: A Markov chain $M = (\Omega, P)$, a target state $z \in \Omega$, an upper bound N for $\mathsf{O}(x_0)$, and a Laplacian approximator of $\widehat{\mathsf{O}}(\cdot)$ of the hitting times
$x^* \leftarrow x_0$
while $x^* \neq z$ **do**
  $g \leftarrow$ next state drawn from x^*
  $X_{x^*}, X_g \leftarrow$ mean-median approximations obtained from $\widehat{\mathsf{O}}$ (Lemma 4.4), with parameters $\varepsilon = \frac{1}{10}$ and $\delta = \frac{1}{10N}$
  if $X_g < X_{x^*} - \frac{1}{2}$ **then**
    $x^* \leftarrow g$
  end if
end while

path (initially, x_0). It repeatedly rewinds to x_i to generate the next state, until x_{i+1} is obtained. Since each increase takes $\frac{1}{p}$ generations in expectation, the algorithm reaches x_n in $\frac{n}{p} = 2n$ steps.

To compare, we consider an algorithm that allows no rewinding. That is, the algorithm starts at x_0 and explores in a single trajectory in hopes of reaching x_n. Observe that in each step, the trajectory either moves one state further down the path, or moves to the dummy state D. In case it moves to D, it is absorbed forever, and never reaches the target state x_n. Therefore, the algorithm reaches the target state with a negligible probability of $\frac{1}{2^n}$.

Similarly, we can consider an algorithm with limited backtracking that explores a number of independent trajectories in parallel. Based on the analysis above, each trajectory reaches the target state with a probability of $\frac{1}{2^n}$. Hence, an algorithm that explores k trajectories in parallel reaches the target state with a probability of $1 - \left(1 - \frac{1}{2^n}\right)^k \leq \frac{k}{2^n}$. Therefore, any algorithm that reaches the target state with a constant probability must explore at least $\Omega(2^n)$ states, which is exponentially larger than the expected number of steps for the optimal rewinding algorithm, $2n$.

4 Massive Markov Chains

Theorem 1.1 shows that an optimal strategy can be devised given oracle access to the exact values of $\mathsf{O}(x)$. However, such oracle access may be infeasible for large Markov chains, since examining the structure of the entire chain is computationally inefficient. Therefore, in this section, we consider access to noisy approximations of $\mathsf{O}(x)$ under two different noise models.

In Section 4.1, we consider a model in which the algorithm can obtain independent approximations of $\mathsf{O}(x)$ with Laplacian noise, and present a non-branching rewinding strategy with a hitting time comparable to $\mathsf{O}(x)$. In Section 4.2, we show that adversarially chosen $(1 \pm \varepsilon)$-approximations of $\mathsf{O}(x)$ are not sufficient for an efficient algorithm, implying that some assumptions about the noise (such as in the first noise model) are necessary.

4.1 A Polynomial Algorithm under Laplacian Noise

First, we formalize the noise model.

Definition 4.1 (Approximations with Laplacian Noise)**.** For each state $s \in \Omega$, the algorithm can repeatedly obtain independent noisy evaluations of $\mathsf{O}(s)$:

$$\widehat{\mathsf{O}}(s) \sim \text{Laplace}(\mathsf{O}(s), \lambda),$$

which has a Laplacian distribution of mean $\mathsf{O}(s)$ and scale λ (i.e., variance $2\lambda^2$), each evaluation incurs the same runtime cost as generating a state.[1]

Of special interest may be the case where $\sqrt{2}\lambda = \varepsilon\mathsf{O}(s)$, as $\widehat{\mathsf{O}}(s) \in (1 \pm \varepsilon)\mathsf{O}(s)$ with constant probability.

Before stating the algorithm for this model, henceforth referred to as the stable algorithm, we describe an auxiliary algorithm which is used in the analysis. The auxiliary algorithm is non-branching and utilizes exact values of $\mathsf{O}(x)$, in a manner that enables the stable algorithm to simulate it using only noisy approximations of $\mathsf{O}(x)$. As opposed to the optimal algorithm, which replaces the minimizer state x^* with any state that has a smaller optimal hitting time, the auxiliary algorithm replaces x^* only when the new state improves the optimal hitting time non-negligibly.

Definition 4.2. Given an initial state $x_0 \in \Omega$ and a parameter $\varepsilon > 0$, the auxiliary algorithm operates as follows:

1. Maintain a special state x^* (serving as an "approximate minimizer"), initially equal to x_0.
2. Repeatedly rewind to x^* to obtain a state g, until the new state satisfies $g = z$ or $\mathsf{O}(g) < \mathsf{O}(x) - c$, where $c = \frac{\varepsilon}{1+\varepsilon}$.
3. If g is equal to the target state z, halt. Otherwise, let $x^* := g$ and go back to step 2.

We use $\mathsf{O}'(x_0)$ to denote the expected hitting time of z with this algorithm.

We establish that the hitting time of the auxiliary algorithm is at most $(1 + \varepsilon)$ times larger than the optimal hitting time. The proof is deferred to Section C.

Lemma 4.3. *For any state x, it holds that $\mathsf{O}'(x) \leq (1 + \varepsilon)\mathsf{O}(x)$.*

Now, we move on to the stable algorithm. First, we note that the mean-median technique can be used to strengthen the approximation guarantee of $\widehat{\mathsf{O}}(\cdot)$ (as defined in Definition 4.1). The proof is deferred to Section C.

Lemma 4.4. *For any state $s \in \Omega$ and parameters ε and δ, given access to estimations $\widehat{\mathsf{O}}(s) \sim \mathrm{Laplace}(\mathsf{O}(s), \lambda)$, the algorithm can use $O(\frac{\lambda^2}{\varepsilon^2} \log \frac{1}{\delta})$ estimates to obtain a value X_s such that $|X_s - \mathsf{O}(s)| \leq \varepsilon$, with probability δ.*

With the mean-median subroutine at hand, we are ready to present the stable algorithm (Algorithm 3).

Theorem 4.5. *Given an initial state x_0, Algorithm 3 reaches the target state with an expected computation cost (number of steps plus number of approximations) of at most $O(\lambda^2 \cdot \mathsf{O}(x_0) \log \mathsf{O}(x_0))$. In particular, when $\widehat{\mathsf{O}}(s) \sim \mathrm{Laplace}\,(\mathsf{O}(s), \varepsilon\mathsf{O}(s))$, the total cost is at most $O(\varepsilon^2 \cdot \mathsf{O}(x_0)^3 \log \mathsf{O}(x_0))$, in expectation.*

4.2 A Lower Bound for Adversarial Noise

In this section, we present a lower bound for adversarial noise in partially observable (i.e., massive) Markov chains, showing that access to arbitrary $(1 \pm \varepsilon)$-approximations of $\mathsf{O}(x)$ is not sufficient for reaching the target state efficiently, when the errors are set arbitrarily by an adversary. That is, we construct a partially observable Markov chain $M = (\Omega, P)$, with an initial state $x_0 \in \Omega$ and a target state z. For any generated state x, the algorithm can only observe $\widehat{\mathsf{O}}(x)$, which is an adversarially set value within $(1 \pm \varepsilon)\mathsf{O}(x)$. We show that to reach the target state with constant probability, the algorithm must generate at least $\exp(\mathsf{O}(x_0))$ states.

Theorem 4.6. *Given a parameter $\varepsilon < 1$, there exists a partially observable Markov chain $M = (\Omega, P)$, with observations $\widehat{\mathsf{O}}(x) \in (1 \pm \varepsilon)\mathsf{O}(x)$ for any state x, such that any algorithm that can only access the observations requires $\exp(\mathsf{O}(x_0))$ steps to reach the target, where $\mathsf{O}(x_0)$ can be taken arbitrarily large.*

[1] The Laplace distribution has a density function of $f(x) = \frac{1}{2\lambda} e^{|x-\mu|/\lambda}$. We remark that the algorithm in this section can be easily adapted to any independent noisy evaluation of variance λ.

Here, we describe the Markov chain and defer the proofs to Section C. To do so, we use a Δ-regular tree of depth $10d$. Here, Δ is a sufficiently large constant, and d is taken as large as desired to increase $\mathsf{O}(x_0)$. The root of the tree, which has degree Δ, corresponds to the initial state x_0. Each descendant of x_0 has $\Delta - 1$ children. This continues up to a depth of $10d$, where each descendant of depth $10d$ is a leaf and has no children. Each vertex in the tree corresponds to a state in the Markov chain. For the transition probabilities, we let any non-leaf state transition to its neighbors in the tree with uniform probability. A leaf vertex transitions to its parent with probability $\frac{1}{\Delta}$, and remains in the same state otherwise. Finally, we choose one of the leaves as the target state z. Before presenting the observations $\widehat{\mathsf{O}}(x)$, we calculate the optimal hitting times $\mathsf{O}(x)$ for each state.

Lemma 4.7. *For every state x, it holds that $\mathsf{O}(x) = \Delta \cdot \mathrm{dist}(x, z)$, where $\mathrm{dist}(x, z)$ denotes the length of the shortest path between x and z in the tree.*

The observations (with arbitrary noise) are set as follows. Let $\mathcal{P} = (x_0, x_1, \ldots, x_{10d})$ be the path that starts at the root x_0 and ends at the target state $x_{10d} = z$. We divide the states into three groups: (1) those within a distance of εd from the root; (2) those in the subtree of $x_{\varepsilon d}$; and (3) everything else, i.e., states with a depth strictly larger than εd, that are not in the subtree of $x_{\varepsilon d}$. For a state s in the first group, we let the observation $\widehat{\mathsf{O}}(x)$ be equal to $\Delta \cdot (10d - d_s)$, where d_s is the depth of s (its distance from the root). That is, the true hitting time as if s lied on the path from x_0 to z. For the second group, we let $\widehat{\mathsf{O}}(s) := \Delta \cdot \mathrm{dist}(s, z)$, i.e., the observation is set without error to the optimal hitting time. Finally, for the third group, we let $\widehat{\mathsf{O}}(s) := 10d - 2\varepsilon d + d_s$. That is, set to the hitting time as if s were in the subtree of $x_{\varepsilon d}$ but not in the subtree of $x_{\varepsilon d+1}$ (i.e., a node with depth d_s that branches off $\mathcal{P}$ at $x_{\varepsilon d}$). It can be easily confirmed that these observations are within a $(1 \pm \varepsilon)$-multiplicative error of the optimal hitting time (as characterized in Lemma 4.7).

To prove Theorem 4.6, in Section C, we show that any algorithm that reaches the target state with constant probability using only the observations, must explore a constant fraction of the states in the first group (the vertices within distance εd of x_0). That is, it must take at least about $\Delta^{\varepsilon d}$ steps which is exponentially larger than the optimal hitting time $10\Delta d$.

5 Experiments

In this section, we evaluate a practical variant of our optimal rewinding algorithm (Algorithm 1) that is better suited for practical settings, where the exact number of optimal steps from a state is not available. Our experiments and all the compared methods use the same underlying language model (GPT-4), the same prompting strategy for child generation, and comparable evaluator prompts. We used ToT as the base code, and implemented CaT. The implementations are available on github.

5.1 The Algorithm

For our experiments, we implement a variant of the optimal algorithm Algorithm 1 that instead of selecting the "best" explored state deterministically, selects one via the *softmax* distribution. This modification is more suitable for our empirical studies, as the optimal hitting time $\mathsf{O}(x)$ from a given state x is not readily available and can only be approximated. More formally, the distribution is based on $\widehat{\mathsf{O}}(x)$, an estimator of $\mathsf{O}(x)$, and prioritizes states with smaller estimated hitting times. See Algorithm 4.

Algorithm 4 Caterpillar of Thoughts (via softmax)

Input: Initial state x_0, target state z, and hitting time estimator $\widehat{\mathsf{O}}(\cdot)$
$S \leftarrow \{x_0\}$
while $z \notin S$ **do**
  Sample a parent state $x^* \in S$ according to the softmax distribution

$$\Pr[x^*] \propto \exp(-\widehat{\mathsf{O}}(x^*))$$

  Sample a child state $g \sim P(x^*, \cdot)$
  $S \leftarrow S \cup \{g\}$.
end while

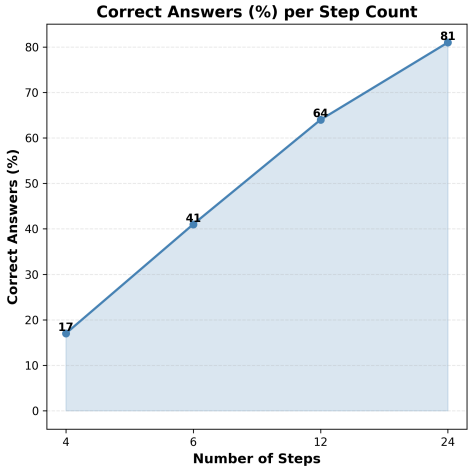


Figure 2: Illustration on the right shows log linear relation between number of valid expressions and number of steps.

In each iteration, Algorithm 4 rewinds to a previously observed state and expands it by sampling new child states according to the underlying transition probabilities. The use of a softmax distribution heavily biases exploration toward states that are estimated to be closer to target, while still allowing exploration over the entire observed set.

We note that the main advantage of Algorithm 4 compared to the non-branching algorithm of Theorem 1.1 is robustness to estimation error. Without exact access to O, a non-branching algorithm may commit early to an incorrectly evaluated state x and get stuck exploring a suboptimal subtree rooted at x, with no mechanism to recover from this mistake. In contrast, Algorithm 4 allows the set of observed states to occasionally branch, preventing the algorithm from being confined to a single trajectory. At the same time, the rewinding rule ensures that exploration remains exponentially biased toward states with smaller estimated rewinding hitting times.

5.2 Task: Game of 24

In the *Game of 24*, the model is given four integers and must construct a valid arithmetic expression using the operations $\{+, -, \times, /\}$ that evaluates exactly to 24. Each number must be used exactly once, and intermediate results must be valid real numbers. A solution is naturally constructed incrementally by applying operations step by step, which makes the task suited for modeling reasoning as a search process over partial solutions. For example, given input "4 4 6 8", a solution output could be "(6 - 4) * (4 + 8) = 24".

The Markov Chain: A state corresponds to a *partial solution*, consisting of the sequence of arithmetic operations applied so far, and the resulting multiset of remaining values. From a given state, transitions correspond to sampling a valid next arithmetic operation and two of the remaining values using the language model, producing a new partial solution. For example given input "4 4 6 8", a valid state is represented as "4 + 8 = 12 (left: 4 6 12)" or "4 + 8 = 12 (left: 4 6 12), 6 - 4 = 2 (left: 2 12)". Any state representing a complete arithmetic expression that evaluates to 24 and uses the input numbers each exactly once, is considered a target state.

Data: We use data from 4nums[2], the same data as Yao et al. (2023) which has 1,362 games that are sorted from easy to hard, and use a subset of relatively hard games indexed 901-1,000 for testing. We report the success rate across 100 games as the metric. **Algorithm Implementation** In our experimental implementation, we use a version of Algorithm 4 with bounded number of iterations. We fix a maximum number of iterations as a hyperparameter, we execute the algorithm until a target state is reached, or terminate the algorithm once this budget of maximum iterations is exhausted. The

[2] 4nums.com

Table 2: Performance on the 5×5 crossword benchmark: cost matched ToT vs. CaT.

Method	Word Acc. (%)	Letter Acc. (%)	Games Solved (%)	Avg. Tokens
ToT Yao et al. (2023)	39.5	64.8	5	73.4k
CaT (This paper)	**50.0**	**68.6**	**15**	**66.8k**

output is considered successful if a valid solution state is discovered within this budget. In contrast, the BFS like algorithm of ToT, cannot terminate when reaching the target, since ToT prioritizes breadth instead of depth, and to hit the target needs to reach a specific depth and exhaustively process each layer.

In Algorithm 4, at each iteration, after sampling a parent state according to the softmax distribution with temperature 1, we expand the selected state by sampling multiple child states. The number of children sampled from a parent is not fixed a priori and is instead determined by the language model, following the same protocol used in Tree of Thoughts. This allows the model to adaptively propose multiple candidate next steps from a given partial solution.

For each observed state s, we estimate $\mathsf{O}(s)$ using the language model itself. Concretely, the model is prompted to assess how likely the current partial solution is to lead to a correct final answer. The resulting scalar score is treated as a noisy estimate $\widehat{\mathsf{O}}(s)$ of the optimal remaining cost to go from s.

We run two independent executions (best of 2), each with a fixed iteration budget, and report success if either produces a valid solution. Independent runs mitigate stochastic concentration on an unproductive region while keeping average token use below Tree of Thoughts.

Results: Table 1 summarizes performance on the Game of 24 benchmark. Algorithm 4 gets a 81% accuracy in Game of 24 while reducing token usage to $15.3k$ tokens. In Figure 2, we illustrate how many of the inputs were solved per new iterations.

5.3 Task: Crossword

We evaluate our method on a 5×5 crossword completion benchmark. Each instance consists of 10 clues whose answers must be placed on a shared 5×5 grid, with 5 entries oriented horizontally and 5 entries oriented vertically. We report three metrics: (i) *letter accuracy*, measured over the 25 grid cells; (ii) *word accuracy*, measured over the 10 entries; and (iii) *game success*, the fraction of instances for which the entire crossword is solved.

The Markov Chain: We model the search process as a Markov chain that allows rewinding over partially completed boards. A state is a partially filled crossword grid (equivalently, a set of assigned clue answers that are mutually consistent). A transition selects one clue and proposes an answer using the language model; if the proposed entry is compatible with all currently placed letters at intersecting cells, it is inserted into the grid, producing a new (more complete) state. Incompatible proposals are rejected (the state remains unchanged).

Data: We use the 5×5 crossword dataset from GooBix,[3] which is the same source used in the Tree-of-Thoughts (ToT) experiments. The dataset contains 156 crossword instances. Following the ToT setup, we evaluate on 20 held-out puzzles with indices $1, 6, \ldots, 96$, and we use the same subset as ToT for in-context prompting.

Implementation: We use the model to score partial boards, sample boards from the resulting softmax distribution, and run 20 iterations of Algorithm 4 per input. Full implementation details appear in Section B.

Results: Table 2 compares our approach (CaT) against a truncated version [4] of ToT with 20 search steps instead of 100 steps reported by ToT. Across all metrics, CaT achieves higher accuracy and solves more puzzles while using fewer tokens on average.

[3] https://www.goobix.com/crosswords/0505/

[4] This is from reports of Chi et al. (2025) on truncated ToT

AI USE STATEMENT

In this work, generative AI tools were used to adapt the LaTeX source to the ICLR format and to assist with light editing and reorganization of existing text. They were not used to generate the paper's research ideas, theoretical claims, proofs, experimental results, or data. The authors reviewed all AI-assisted changes and take responsibility for the final content of the work, including all text, claims, and artifacts.

REFERENCES

Amir Azarmehr, Soheil Behnezhad, Alma Ghafari, and Madhu Sudan. Markov chains with rewinding. In Mohit Singh and Tom Gur (eds.), *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX 2026, Boston University, Boston, Massachusetts, USA, and RANDOM 2026, Boston University, Boston, Massachusetts, USA, August 19-21, 2026, August 19-21, 2026*, volume 392 of *LIPIcs*, pp. 43:1–43:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026. doi: 10.4230/LIPICS.APPROX/RANDOM.2026.43. URL https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2026.43.

Edoardo Botta, Yuchen Li, Aashay Mehta, Jordan T. Ash, Cyril Zhang, and Andrej Risteski. On the query complexity of verifier-assisted language generation, 2025. URL https://arxiv.org/abs/2502.12123.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Yizhou Chi, Kevin Yang, and Dan Klein. ThoughtSculpt: Reasoning with intermediate revision and search. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 7685–7711, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.428. URL https://aclanthology.org/2025.findings-naacl.428/.

Nicola Dainese, Matteo Merler, Minttu Alakuijala, and Pekka Marttinen. Generating code world models with large language models guided by monte carlo tree search. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Lihui Liu. Monte carlo tree search for graph reasoning in large language model agents. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, CIKM '25, pp. 4966–4970, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400720406. doi: 10.1145/3746252.3760854. URL https://doi.org/10.1145/3746252.3760854.

Aman Madaan et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Sungjin Park, Xiao Liu, Yeyun Gong, and Edward Choi. Ensembling large language models with process reward-guided tree search for better complex reasoning, 2024. URL https://arxiv.org/abs/2412.15797.

Dhruv Rohatgi, Abhishek Shetty, Donya Saless, Yuchen Li, Ankur Moitra, Andrej Risteski, and Dylan J. Foster. Taming imperfect process verifiers: A sampling perspective on backtracking, 2025. URL https://arxiv.org/abs/2510.03149.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Siddarth Venkatraman, Vineet Jain, Sarthak Mittal, Vedant Shah, Johan S. Obando-Ceron, Yoshua Bengio, Brian R. Bartoldson, Bhavya Kailkhura, Guillaume Lajoie, Glen Berseth, Nikolay Malkin, and Moksh Jain. Recursive self-aggregation unlocks deep thinking in large language models. *CoRR*, abs/2509.26626, 2025. doi: 10.48550/ARXIV.2509.26626. URL https://doi.org/10.48550/arXiv.2509.26626.

Dimitri von Rütte, Janis Fluri, Yuhui Ding, Antonio Orvieto, Bernhard Schölkopf, and Thomas Hofmann. Generalized interpolating discrete diffusion, 2025. URL https://arxiv.org/abs/2503.04482.

Kaiwen Wang, Jin Peng Zhou, Jonathan Chang, Zhaolin Gao, Nathan Kallus, Kianté Brantley, and Wen Sun. Value-guided search for efficient chain-of-thought reasoning, 2025. URL https://arxiv.org/abs/2505.17373.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Xiao-Wen Yang, Xuan-Yi Zhu, Wen-Da Wei, Ding-Chu Zhang, Jie-Jing Shao, Zhi Zhou, Lan-Zhe Guo, and Yu-Feng Li. Step back to leap forward: Self-backtracking for boosting reasoning of language models, 2025. URL https://arxiv.org/abs/2502.04404.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

A Comparison to Related Work

In this section, we argue that Verifier Guided Backtracking (VGB) Rohatgi et al. (2025) is a special case of the Markov chain with rewinding. We restate the VGB in a self-contained way. Fix: (1) a prompt x; (2) a discrete token alphabet $\mathcal{A}$; (3) a reference policy (base model) $\pi_{\mathrm{ref}}(\cdot \mid x, y_{1:h})$, which for every prefix $y_{1:h} \in \mathcal{A}^h$ is a probability distribution over next tokens $a \in \mathcal{A}$; and (4) a nonnegative base value function $V_b(x, y_{1:h}) \geq 0$ defined for every prefix $y_{1:h} \in \mathcal{A}^h$ (and similarly $V_b(x, y_{1:h}, a) \geq 0$ defined for extended prefixes).

Define the *prefix tree* $\mathcal{T} := \bigcup_{h=0}^{H} \mathcal{A}^h$, whose root is the empty prefix $\varnothing \in \mathcal{A}^0$. For $y_{1:h} \in \mathcal{A}^h$ with $h \geq 1$, its parent is $\mathrm{par}(y_{1:h}) := y_{1:h-1}$. For $h \leq H-1$ where H is the maximum depth of the tree, its children are $\mathrm{ch}(y_{1:h}) := \{(y_{1:h}, a) : a \in \mathcal{A}\}$.

Define the *neighborhood* of a node $y \in \mathcal{T}$ by $N(y) := \{\mathrm{par}(y)\} \cup \mathrm{ch}(y)\}$. Given the current node $y = y_{1:h}$, define non-normalized weights on $N(y)$ by

$$w(u \mid x, y) := \begin{cases} V_b(x, y) & \text{if } u = \mathrm{par}(y) \text{ and } y \neq \varnothing, \\ \pi_{\mathrm{ref}}(a \mid x, y) \cdot V_b(x, y, a) & \text{if } u = (y, a) \in \mathrm{ch}(y), \\ 0 & \text{otherwise.} \end{cases}$$

Let $p(u \mid x, y) := \frac{w(u|x,y)}{\sum_{u \in N(y)} w(u|x,y)}$ for $u \in N(y)$.

The VGB algorithm operates by evolving a Markov chain $(Y_t)_{t \geq 0}$. The chain is initialized at $Y_0 = \varnothing$ and evolves as follows: given $Y_t = y$, sample $Y_{t+1} \sim p(\cdot \mid x, y)$. The process continues until a target state is reached.

Lemma A.1. *The VGB process is an instance of the Markov chain with rewinding.*

Proof. Instantiate our framework with the nodes of the prefix tree $\mathcal{T}$ as the state space. The initial state is the root of the prefix tree corresponding to the prompt x, and an empty string $y = \varnothing$ of tokens

generated so far. For a node in the tree, the transition probabilities are defined on the children *only*, proportional to the non-normalized weights.

With this setup, the VGB algorithm can be interpreted in our model as follows. Given the most recently produced state $u_t = (x, y)$ (corresponding to the prompt x, and the string of generated tokens y), the algorithm either rewinds to the parent $\mathrm{par}(u_t)$ with probability $p(\mathrm{par}(y) \mid x, y)$, or evolves a child otherwise. As a result, the next state u_{t+1} has the desired distribution as in VGB. □

B Crossword Implementation Details

For evaluation, the model is prompted to assess how likely an answer to a clue is to be compatible with the partially completed board. We score each partial crossword by adding the scores of its filled words. The resulting scalar is treated as a noisy estimate $\widehat{\mathsf{O}}(s)$ of the optimal remaining cost from s. We use these scores to form the softmax distribution from which partially completed boards are sampled. Given a sampled board, we generate new boards by sorting the scores for each remaining clue and selecting the top three. We use the same clue-scoring prompts as ToT.

C Omitted Proofs

This section includes the proofs deferred from the main body of the paper. The claims are restated here for ease of reference.

Lemma 3.1. *There exists an optimal rewinding algorithm such that given a sequence of states $X = (x_0, \cdots, x_t)$, the rewinding time $t' \le t$ chosen for obtaining the next state is determined solely based on the set of distinct states S in X. We use $\mathsf{O}(S) := \mathsf{O}(X)$ to denote the optimal hitting time corresponding to S, and $\mathsf{O}(x)$ as a shorthand for $\mathsf{O}(\{x\})$.*

To prove the lemma, we first show that the optimal algorithm depends only on the set of distinct elements of a sequence X and not on the order or multiplicity of states in the history.

Lemma C.1. *Let $X^{(1)}$ and $X^{(2)}$ be two different sequences of obtained states with the same set of distinct states S. We have*

$$\mathsf{O}(X^{(1)}) = \mathsf{O}(X^{(2)}).$$

Proof. Fix an arbitrary algorithm σ used after history $X^{(1)}$. We construct a corresponding algorithm σ' to be used after $X^{(2)}$ such that the trajectories under σ and σ' can be perfectly coupled while maintaining the following invariant. Let $S_t^{(1)}$ and $S_t^{(2)}$ denote the sets of states observed up to time t under σ and σ', respectively. We maintain the invariant

$$S_t^{(1)} = S_t^{(2)} = S_t \qquad \text{for all } t \ge 0.$$

This holds initially because $X^{(1)}$ and $X^{(2)}$ have the same set of distinct states S.

Now assume the invariant holds at step t. algorithm σ rewinds to some time $t' \le t$, equivalently, it chooses a state $a \in S_t$. We define σ' so that it also rewinds to the same state a. Since $a \in S_t^{(1)} = S_t^{(2)}$, this is always possible. Because the transition probabilities depend only on the chosen state and not on the history, both runs now sample $y \sim P(a, \cdot)$. We couple them so that both runs obtain the same next state y. Therefore,

$$S_{t+1}^{(1)} = S_t^{(1)} \cup \{y\} = S_t^{(2)} \cup \{y\} = S_{t+1}^{(2)},$$

so the invariant holds at time $t + 1$. Thus, under this coupling, the sets of distinct states for $X^{(1)}$ and $X^{(2)}$ evolve identically at any time t and therefore, both algorithm σ and σ' hit z at the same time.

Let $\sigma(X^{(1)})$ be the expected hitting time of algorithm σ for $X^{(1)}$. We define $\sigma'(X^{(2)})$ respectively. Pick σ be the algorithm with optimal expected hitting time on $X^{(1)}$. For the coupled algorithm σ' on $X^{(2)}$, we have

$$\mathsf{O}(X^{(2)}) \le \sigma'(X^{(2)}) = \sigma(X^{(1)}) = \mathsf{O}(X^{(1)}).$$

Symmetrically, we also obtain $\mathsf{O}(X^{(2)}) \ge \mathsf{O}(X^{(1)})$. Hence $\mathsf{O}(X^{(1)}) = \mathsf{O}(X^{(2)})$. □

Proof of Lemma 3.1. Let X be an arbitrary sequence of obtained states, and let S be its set of distinct states. Since S is precisely the set of distinct states in X, we can construct a sequence $\widetilde{X}$ that visits exactly the states in S (each at most once) and has the same set of distinct states as X. We denote this sequence by S as well, viewed now as some fixed sequence whose distinct-state set is S. Applying Lemma C.1 with $X^{(1)} = X$ and $X^{(2)} = S$, we obtain $\mathsf{O}(X) = \mathsf{O}(S)$. This shows that the optimal hitting time starting from any history X depends only on its set of distinct states S. □

Theorem 3.3. *For any set of states $S \subseteq \Omega$, there exists a state $x \in S$ such that*

$$\mathsf{O}(S) = \mathsf{O}_{\mathcal{NB}}(x).$$

Proof. We prove the statement by a downward induction on $|S|$. It holds trivially for any set of states S that contains z since we have that $\mathsf{O}(S) = \mathsf{O}_{\mathcal{NB}}(z) = 0$. This includes the base case, $S = \Omega$. For the induction step, given a set S with $|S| < |\Omega|$, fix any optimal algorithm, and let $a \in S$ be the rewinding state chosen for obtaining the next state. Let g be the state that is obtained as a result. Observe that if $g \in S$, the optimal algorithm remains the same (by Lemma 3.1). Therefore, without loss of generality, we can assume that the optimal algorithm rewinds to a, until it obtains a state $g \notin S$.

For a state $g \notin S$, let S_g denote $S \cup \{g\}$. Since $|S_g| > |S|$, by the induction hypothesis, there exists a state $x_g \in S_g$ such that

$$\mathsf{O}(S_g) = \mathsf{O}_{\mathcal{NB}}(x_g). \tag{1}$$

That is, there exists a non-branching algorithm starting at x_g that is optimal for S_g. We consider two scenarios. First, we consider the case that for all $g \notin S$ it holds that $x_g = g$ (this includes the case where $g = z$, and the process stops), i.e., the target is obtained or there is an optimal algorithm for S_g that starts at x_g and does not branch. In this case, we have a non-branching algorithm which is optimal for S that starts at a, meaning $\mathsf{O}(S) = \mathsf{O}_{\mathcal{NB}}(a)$. This algorithm, rewinds to a until a state $g \notin S$ is obtained, and then proceeds according to the non-branching algorithm at $x_g = g$.

Now, consider the case where there exists a state $g \notin S$ such that $x_g \neq g$. Then, it holds that $x_g \in S$. We show that $\mathsf{O}(S) = \mathsf{O}_{\mathcal{NB}}(x_g)$ (the algorithm could have started at x_g instead of a to begin with). First, observe that since $S \subseteq S_g$, any algorithm for S is a valid algorithm for S_g. Therefore we have

$$\mathsf{O}(S_g) \leq \mathsf{O}(S). \tag{2}$$

Second, since $x_g \in S$, the optimal non-branching algorithm starting at x_g is a valid algorithm for S. Hence,

$$\mathsf{O}(S) \leq \mathsf{O}_{\mathcal{NB}}(x_g). \tag{3}$$

Finally, combining Equations (1) to (3), we have $\mathsf{O}(S) \leq \mathsf{O}_{\mathcal{NB}}(x_g) = \mathsf{O}(S_g) \leq \mathsf{O}(S)$, which implies that the above inequalities hold with equality, and hence, $\mathsf{O}(S) = \mathsf{O}_{\mathcal{NB}}(x_g)$. □

Corollary 3.4. *Given an initial set of states S, we have $\mathsf{O}(S) = \min_{x \in S} \mathsf{O}(x) = \min_{x \in S} \mathsf{O}_{\mathcal{NB}}(x)$.*

Proof. First, note that for any state $x \in \Omega$, Theorem 3.3 implies $\mathsf{O}(x) = \mathsf{O}_{\mathcal{NB}}(x)$ (by letting $S = \{x\}$). Therefore, $\mathsf{O}(x)$ and $\mathsf{O}_{\mathcal{NB}}(x)$ can be used interchangeably and the second equality in the claim follows.

Now, we show $\mathsf{O}(S) = \min_{x \in S} \mathsf{O}_{\mathcal{NB}}(x)$. Observe that a non-branching algorithm starting at a state $x \in S$ is a valid algorithm for S. Hence, $\mathsf{O}(S) \leq \min_{x \in S} \mathsf{O}_{\mathcal{NB}}(x)$. Moreover, by Theorem 3.3, there exists a state $x^* \in S$ such that $\mathsf{O}(S) = \mathsf{O}_{\mathcal{NB}}(x^*)$. Therefore, $\mathsf{O}(S) = \min_{x \in S} \mathsf{O}_{\mathcal{NB}}(x)$. □

Theorem 1.1. *Given an initial state x_0, Algorithm 1 reaches the target state using the optimal number of steps in expectation, i.e., in $\mathsf{O}(x_0)$ steps.*

Proof. Note that the Algorithm 1 is non-branching since in every step, it either rewinds to x^*, the same rewinding state as the last step, or to g, the most recently obtained state. To show it is the optimal non-branching algorithm, by Corollary 3.4, it suffices to prove that whenever the algorithm rewinds to x^*, it satisfies $x^* = \arg\min_{x \in S} \mathsf{O}_{\mathcal{NB}}(x)$, where S denotes the set of states obtained so far. This can be proven inductively as follows. For the base case, it holds trivially since x^* is equal to x_0, the only state in S. For the induction step, observe that if $\mathsf{O}(g) \geq \mathsf{O}(x^*)$, then x^* is still the minimizer of $\mathsf{O}_{\mathcal{NB}}(x)$. Otherwise, if $\mathsf{O}(g) < \mathsf{O}(x^*)$, then g is the new minimizer of $\mathsf{O}_{\mathcal{NB}}(x)$ in S, and x^* is changed to S. □

Corollary 3.5. *For a state $x \neq z$, let $L \subseteq \Omega$ be the set of states y such that $\mathsf{O}(y) < \mathsf{O}(x)$. Then, it holds that $\mathsf{O}(x) = \frac{1+\sum_{y \in L} P(x,y)\mathsf{O}(y)}{P(x,L)}$.*

Proof. Consider the hitting time of the optimal algorithm described in Theorem 1.1, and let y be the first state drawn by the algorithm (obtained by rewinding to x_0). There are two possibilities for y. Either $\mathsf{O}(y) \geq \mathsf{O}(x)$, in which case it is ignored and the expected number of remaining steps is $\mathsf{O}(x)$, or $\mathsf{O}(y) < \mathsf{O}(x)$, in which case x^* is set equal to y and the expected remaining number of steps is $\mathsf{O}(y)$. Therefore, it holds that

$$\begin{aligned}\mathsf{O}(x) &= 1 + \sum_{y \notin L} P(x,y)\mathsf{O}(x) + \sum_{y \in L} P(x,y)\mathsf{O}(y) \\ &= 1 + P(x, \Omega \setminus L)\mathsf{O}(x) + \sum_{y \in L} P(x,y)\mathsf{O}(y).\end{aligned}$$

The claim follows from moving $P(x, \Omega \setminus L)\mathsf{O}(x)$ to the left-hand side and dividing both sides by $P(x, L)$. □

Theorem 3.6. *Algorithm 2 computes $\mathsf{O}(x)$ for every state $x \in \Omega$ and runs in time $O(n^2)$, where $n = |\Omega|$ denotes the number of states in the Markov chain. If m entries of the transition probabilities $P(\cdot, \cdot)$ are non-zero, then the algorithm can be implemented in time $O((m+n)\log n)$.*

Proof. We prove the following invariant inductively: At any point, for all states $x \in \Omega$, it holds that $d_x \geq \mathsf{O}(x)$, and this holds with equality for all states $x \in T$. For the base case, we consider the algorithm at the end of the first iteration. At this time, it holds that $T = \{z\}$ and $d_z = \mathsf{O}(z) = 0$. For all $x \neq z$, we have

$$d_x = \begin{cases} \frac{1}{P(x,z)} & \text{if } P(x,z) \neq 0, \text{ and} \\ \infty & \text{otherwise.} \end{cases}$$

This value is an upper bound for $\mathsf{O}(x)$ as it corresponds to the simple algorithm which rewinds to x until it obtains the target state z.

For the induction step, given the set T, we define a corresponding algorithm for each initial state $x \notin T$. The algorithm starts by repeatedly rewinding to x, until it obtains a state $g \in T$. Then, it proceeds by following the optimal algorithm for g. The expected hitting time of this algorithm can be analyzed as follows. In expectation, it takes $\frac{1}{P(x,T)}$ to obtain a state in T from x. Then, conditioned on that the newly obtained state g is in T, we have $g = y$ with probability $\frac{P(x,y)}{P(x,T)}$, in which case the remainder of the algorithm takes $\mathsf{O}(y)$ steps to terminate in expectation. Therefore, the expected hitting time of this algorithm is equal to:

$$\frac{1}{P(x,T)} + \sum_{y \in T} \frac{P(x,y)}{P(x,T)} \mathsf{O}(y).$$

Observe that if $\mathsf{O}(y) = d_y$ for all $y \in T$, then the value above is equal to d_x.

Hence, it remains to show that when x^* is added to T, it holds that $d_{x^*} = \mathsf{O}(x^*)$. Let $A \subseteq \Omega \setminus T$ be the set of states x outside T that minimize $\mathsf{O}(x)$, and let m denote the minimum. For any $x \in A$, the algorithm corresponding to d_x (as defined in the previous paragraph) is the same as the optimal algorithm outlined in Theorem 1.1 since T contains every state x' with $\mathsf{O}(x') < \mathsf{O}(x)$. Therefore, we have that $d_x = \mathsf{O}(x) = m$ for all $x \in A$. Furthermore, for any $x \notin A$, it holds that $d_x \geq \mathsf{O}(x) > m$. As a result, $x^* := \arg\min_{x \notin T} d_x$ is a member of A, and it holds that $d_{x^*} = \mathsf{O}(x^*) = m$.

Finally, we note that when the algorithm terminates, for any state $x \notin T$, there is no path of positive probability from x to z. Otherwise, the algorithm would have expanded T through one of the edges in the path that go from $\Omega \setminus T$ to T. As a result, for any $x \notin T$, it holds that $\mathsf{O}(x) = d_x = \infty$. Therefore, when Algorithm 2 terminates, d_x is equal to $\mathsf{O}(x)$ for all $x \in \Omega$. It can be easily seen that it can be implemented in $O(n^2)$ or $O((n+m)\log n)$, which concludes the proof. □

Lemma C.2. *For a state $x \neq z$, let $L \subseteq \Omega$ be the set of states y such that $\mathsf{O}(y) < \mathsf{O}(x) - c$. Then, it holds that*

$$\mathsf{O}'(x) = \frac{1 + \sum_{y \in L} P(x,y)\mathsf{O}'(y)}{P(x,L)}.$$

Proof. The proof is similar to that of Corollary 3.5. After the first state y is drawn, either $y \notin L$, in which case the expected remaining number of steps is $\mathsf{O}'(x)$, or $y \in L$, in which case the expected remaining number of steps is $\mathsf{O}'(y)$. □

Lemma 4.3. *For any state x, it holds that $\mathsf{O}'(x) \leq (1+\varepsilon)\mathsf{O}(x)$.*

Proof. Given that $c = \frac{\varepsilon}{1+\varepsilon}$ holds, $\mathsf{O}'(x) \leq (1+\varepsilon)\mathsf{O}(x)$ and $(1-c)\mathsf{O}'(x) \leq \mathsf{O}(x)$ are equivalent. We prove the latter inductively, ordering the states by $\mathsf{O}(x)$. More specifically, for the base case, we consider $x = z$ where $\mathsf{O}(x) = \mathsf{O}'(x) = 0$. Then, for the induction step, we assume the claim for all y with $\mathsf{O}(y) < \mathsf{O}(x)$, and prove it for x.

Take any state $x \neq z$. We divide the states into three groups based on their optimal hitting time:

$$\begin{aligned} H &:= \{y \in \Omega \mid \mathsf{O}(x) \leq \mathsf{O}(y)\}, \\ M &:= \{y \in \Omega \mid \mathsf{O}(x) - c \leq \mathsf{O}(y) < \mathsf{O}(x)\}, \text{ and} \\ L &:= \{y \in \Omega \mid \mathsf{O}(y) < \mathsf{O}(x) - c\}. \end{aligned}$$

Recall the recursive formulation of $\mathsf{O}(x)$ (Corollary 3.5, note the change in definition of L):

$$\mathsf{O}(x) = \frac{1 + \sum_{y \in M \cup L} P(x,y)\mathsf{O}(y)}{P(x, M \cup L)}.$$

For $y \in M$, it holds $\mathsf{O}(y) \geq \mathsf{O}(x) - c$ by definition. For $y \in L$, the induction hypothesis implies $\mathsf{O}(y) \geq (1-c)\mathsf{O}'(y)$. Therefore, $\mathsf{O}(x)$ can be lower-bounded as follows:

$$\begin{aligned} \mathsf{O}(x) &\geq \frac{1 + \sum_{y \in L} P(x,y)(1-c)\mathsf{O}'(y)}{P(x, M \cup L)} \\ &\quad + \frac{\sum_{y \in M} P(x,y)(\mathsf{O}(x) - c)}{P(x, M \cup L)} \\ &= \frac{1 + \sum_{y \in L} P(x,y)(1-c)\mathsf{O}'(y)}{P(x, M \cup L)} \\ &\quad + \frac{P(x,M)}{P(x, M \cup L)}(\mathsf{O}(x) - c). \end{aligned}$$

Moving $\frac{P(x,M)}{P(x,M \cup L)}\mathsf{O}(x)$ to the left-hand side and dividing both sides by $1 - \frac{P(x,M)}{P(x,M \cup L)}$, we get

$$\mathsf{O}(x) \geq \frac{1 + \sum_{y \in L} P(x,y)(1-c)\mathsf{O}'(y)}{P(x,L)} - \frac{P(x,M)}{P(x,L)}c.$$

Further algebraic manipulation of the inequality and plugging in the recursive formulation of $\mathsf{O}'(x)$ (Lemma C.2) yields

$$\begin{aligned} \mathsf{O}(x) &\geq \frac{1 + \sum_{y \in L} P(x,y)(1-c)\mathsf{O}'(y)}{P(x,L)} - \frac{P(x,M)}{P(x,L)}c \\ &= (1-c)\frac{1 + \sum_{y \in L} P(x,y)\mathsf{O}'(y)}{P(x,L)} \\ &\quad + \frac{1}{p(x,L)}c - \frac{P(x,M)}{P(x,L)}c \\ &\geq (1-c)\frac{1 + \sum_{y \in L} P(x,y)\mathsf{O}'(y)}{P(x,L)} \\ &= (1-c)\mathsf{O}'(x). \end{aligned}$$

This concludes the proof. □

Lemma 4.4. *For any state $s \in \Omega$ and parameters ε and δ, given access to estimations $\widehat{\mathsf{O}}(s) \sim \mathrm{Laplace}(\mathsf{O}(s), \lambda)$, the algorithm can use $O(\frac{\lambda^2}{\varepsilon^2}\log\frac{1}{\delta})$ estimates to obtain a value X_s such that $|X_s - \mathsf{O}(s)| \leq \varepsilon$, with probability δ.*

Proof. Let $N = \log \frac{1}{\delta}$. First, for each $1 \le i \le N$, let $X^{(i)}$ be the mean of $k = 32\frac{\lambda^2}{\varepsilon^2}$ independent approximations obtained from $\widehat{\mathsf{O}}(s)$. As a result, each $X^{(i)}$ has a mean of $\mathsf{O}(s)$ and a variance of $\frac{2\lambda^2}{k} = \frac{\varepsilon^2}{16}$. Therefore, by Chebyshev's inequality, it holds:

$$\Pr[|X^{(i)} - \mathsf{O}(s)| \ge \varepsilon] \le \frac{\varepsilon^2/16}{\varepsilon^2} \le \frac{1}{16}.$$

We let the final estimate X_s be equal to the median $\{X^{(i)}\}_{i \le N}$, and prove that it satisfies

$$\Pr[|X_s - \mathsf{O}(s)| > \varepsilon] \le \delta.$$

To show this, we note that for the median to be outside the interval $\mathsf{O}(s) \pm \varepsilon$, at least half of the means $\{X^{(i)}\}_{i \le N}$ must be outside the interval. Taking the union bound over all the possibilities for these $\lfloor N/2 \rfloor$ indices, the probability of this event is at most

$$\begin{aligned}\binom{N}{\lfloor N/2 \rfloor} \cdot \left(\frac{1}{16}\right)^{N/2} &\le 2^N \cdot \left(\frac{1}{16}\right)^{N/2} \\ &\le 2^{-N} \\ &\le \delta.\end{aligned}$$

Therefore, X_s has the desired property, which concludes the proof. □

Theorem 4.5. *Given an initial state x_0, Algorithm 3 reaches the target state with an expected computation cost (number of steps plus number of approximations) of at most $O(\lambda^2 \cdot \mathsf{O}(x_0) \log \mathsf{O}(x_0))$. In particular, when $\widehat{\mathsf{O}}(s) \sim \mathrm{Laplace}\,(\mathsf{O}(s), \varepsilon\mathsf{O}(s))$, the total cost is at most $O(\varepsilon^2 \cdot \mathsf{O}(x_0)^3 \log \mathsf{O}(x_0))$, in expectation.*

Proof. Without loss of generality, we assume that the guarantee of the mean-median approximation (Lemma 4.4) holds in every step of Algorithm 3 deterministically, and show that the expected number of steps is at most $2\mathsf{O}(x_0)$ when it does. That is,

$$|X_{x^*} - \mathsf{O}(x^*)| \le \frac{1}{10} \qquad \text{and} \qquad |X_g - \mathsf{O}(g)| \le \frac{1}{10}. \tag{4}$$

We show this is without loss of generality through a slight modification to the algorithm. We add a reset mechanism to the algorithm: If the target is not reached within $4N$ steps, then the algorithm discards the run and starts from the beginning. Each run involves $8N$ mean-median approximations (two per each step), and by Lemma 4.4 each of them satisfies the guarantee with probability $\delta = \frac{1}{10N}$. Therefore, the probability that every mean-median approximation in a run satisfies the guarantee is at least:

$$\left(1 - \frac{1}{10N}\right)^{8N} \ge \frac{e^{-8/10}}{2} \ge \frac{1}{5}.$$

When this event happens, we show that the expected number of steps before reaching the target state is at most $2N$. As a result, by the Markov inequality, such a run has a probability of at least $\frac{1}{2}$ to reach the target before the reset at $4N$ steps. That is, in expectation, it takes $\frac{1}{1/5} \cdot \frac{1}{1/2} = 10$ runs to reach the target state, incurring only a constant factor on the number of steps.

Hence, it remains to show that the expected number of steps to reach the target state is at most $2\mathsf{O}(x_0)$, assuming (4) holds in every step. The analysis is through comparing the stable algorithm (Algorithm 3) to the auxiliary algorithm (Definition 4.2). Invoking Lemma 4.3, with $\varepsilon = 1$ and $c = \frac{1}{2}$, yields

$$\mathsf{O}'(x_0) \le 2\mathsf{O}(x_0).$$

We show that the stable algorithm has at most as many steps as the auxiliary rewinding algorithm. More precisely, let $\mathsf{O}''(x)$ denote the expected number of steps of the stable algorithm, assuming (4) holds. We prove that $\mathsf{O}''(x) \le \mathsf{O}(x)$ for all x.

We prove this claim using induction. Ordering the states increasingly based on $\mathsf{O}(x)$, we assume $\mathsf{O}''(y) \le \mathsf{O}'(y)$ for all y before x (i.e., states y with $\mathsf{O}(y) < \mathsf{O}(x)$), and show the claim holds for x.

It holds trivially for the base case: $\mathsf{O}''(z) = \mathsf{O}'(z) = 0$. For the induction step, recall the recursive formulation of $\mathsf{O}'(x)$:

$$\mathsf{O}'(x) = \frac{1 + \sum_{y \in L} P(x,y)\mathsf{O}'(y)}{P(x,L)} = 1 + \sum_{y \notin L} P(x,y)\mathsf{O}'(x) + \sum_{y \in L} P(x,y)\mathsf{O}'(y), \tag{5}$$

where $L \subseteq \Omega$ is the set of states y such that $\mathsf{O}(y) < \mathsf{O}(x) - c$.

Similarly, we can obtain a recursive formulation for $\mathsf{O}''(x)$. Let $M \subseteq \Omega$ be the set of states y with $\mathsf{O}(x) - c < \mathsf{O}(y) < \mathsf{O}(x)$. We note that since (4) holds, the stable algorithm replaces x^* by g when $g \in L$, and it does not replace x^* by g when $g \notin M \cup L$. The only difference between the stable algorithm and the auxiliary algorithm is that the stable algorithm might replace x^* for some of the states $g \in M$, with a probability of $Q(x^*, g)$, in which case it gains a benefit over the auxiliary algorithm. Therefore, we obtain

$$\mathsf{O}''(x) = 1 + \sum_{y \notin M \cup L} P(x,y)\mathsf{O}''(x) + \sum_{y \in M} P(x,y)(Q(x,y)\mathsf{O}''(y) + (1 - Q(x,y))\mathsf{O}''(x)) + \sum_{y \in L} P(x,y)\mathsf{O}''(y).$$

Using the induction hypothesis we replace $\mathsf{O}''(y)$ with $\mathsf{O}'(y)$ to obtain:

$$\mathsf{O}''(x) \leq 1 + \sum_{y \notin M \cup L} P(x,y)\mathsf{O}''(x) + \sum_{y \in M} P(x,y)(Q(x,y)\mathsf{O}'(y) + (1 - Q(x,y))\mathsf{O}''(x)) + \sum_{y \in L} P(x,y)\mathsf{O}'(y).$$

Rewriting (5), we have:

$$\mathsf{O}'(x) = 1 + \sum_{y \notin M \cup L} P(x,y)\mathsf{O}'(x) + \sum_{y \in M} P(x,y)(Q(x,y)\mathsf{O}'(x) + (1 - Q(x,y))\mathsf{O}'(x)) + \sum_{y \in L} P(x,y)\mathsf{O}'(y).$$

Combining the two implies $\mathsf{O}''(x) \leq \mathsf{O}'(x)$, which concludes the proof of the induction. This yields the desired bound of $\mathsf{O}''(x) \leq 2\mathsf{O}(x)$ on the number of steps. Furthermore, each step uses $O(\frac{\lambda^2}{\varepsilon^2} \log \frac{1}{\delta}) = O(\lambda^2 \log \mathsf{O}(x_0))$ Laplacian approximations. Therefore, the total cost is $O(\lambda^2 \cdot \mathsf{O}(x_0) \log \mathsf{O}(x_0))$. In particular, when $\lambda = \varepsilon \mathsf{O}(x_0)$, the total cost is $O(\varepsilon^2 \cdot \mathsf{O}(x_0)^3 \log \mathsf{O}(x_0))$. This completes the proof.

We also remark that when an upper bound N is not available for $\mathsf{O}(x_0)$, a scaled-up approximation of $2\widehat{\mathsf{O}}(x_0)$ can be used for each run. □

Lemma 4.7. *For every state x, it holds that $\mathsf{O}(x) = \Delta \cdot \mathrm{dist}(x, z)$, where $\mathrm{dist}(x, z)$ denotes the length of the shortest path between x and z in the tree.*

Proof. The optimal hitting time is characterized by Theorem 1.1. Take any non-leaf state x, and let $N(x)$ be its neighbors in the tree. Since the underlying graph is a tree, there exists a neighbor $y^* \in N(x)$ such that $\mathrm{dist}(x, z) = \mathrm{dist}(y, z) + 1$, and for any other neighbor $y \neq y^*$ it holds that $\mathrm{dist}(x, z) = \mathrm{dist}(y, z) - 1$. Note that any path starting x or any other neighbor in $N(x) \setminus \{y^*\}$ and ending in z, must go through y^*. Therefore, it holds $\mathsf{O}(x) \geq \mathsf{O}(y^*)$, and $\mathsf{O}(y) \geq \mathsf{O}(y^*)$ for any neighbor $y \neq y^*$.

Based on this, we can invoke Theorem 1.1. There exists an optimal non-branching algorithm that operates as follows:

1. Maintain a "minimizer" state x^*, initially equal to x_0.
2. Repeatedly rewind to x^* to obtain a state y, until the new state satisfies $\mathrm{dist}(y, z) < \mathrm{dist}(x^*, z)$.
3. If y is equal to the target state z, halt. Otherwise, let $x^* := y$ and go back to step 2.

Equivalently, the algorithm moves on the shortest path from x_0 to z. At each vertex x, it repeatedly generates neighbors until the next vertex on the shortest path is drawn. Since the next vertex on the shortest path is drawn with probability $1/\Delta$, this takes Δ steps in expectation, and the entire process lasts $\Delta \cdot \mathrm{dist}(x_0, z)$ in expectation. □

Theorem 4.6. *Given a parameter $\varepsilon < 1$, there exists a partially observable Markov chain $M = (\Omega, P)$, with observations $\widehat{\mathsf{O}}(x) \in (1 \pm \varepsilon)\mathsf{O}(x)$ for any state x, such that any algorithm that can only access the observations requires $\exp(\mathsf{O}(x_0))$ steps to reach the target, where $\mathsf{O}(x_0)$ can be taken arbitrarily large.*

Proof. We let $\Delta = 2^{1/\varepsilon}$, so that $\mathsf{O}(x_0) = 2^{1/\varepsilon}10d$, and d can be set arbitrarily large. By Yao's minimax lemma, it suffices to present an input distribution, such that any *deterministic* algorithm that reaches the target state with at least constant probability uses at least $\Omega(2^{\mathsf{O}(x_0)})$ queries. In fact, we show that this many queries is necessary for reaching $x_{\varepsilon d}$ with constant probability, where recall that $x_0, \ldots, x_{10d}$ is the path starting at the initial state x_0 and ending at the target state $x_{10d} = z$.

We construct the input distribution based on the Markov chain outlined in Section 4.2. The structure of the tree and the node labels are fixed, then one of the leaves is chosen uniformly at random to be the target state z, and the observations are set accordingly. The observations in the shallow part of the tree (i.e., among the vertices within distance εd of the root) the observations $\widehat{\mathsf{O}}(\cdot)$ are symmetric, in the sense that any two states of the same depth have the same observation. Furthermore, considering any two nodes at depth εd, the observations in the two subtrees are identical, except for the node $x_{\varepsilon d}$. We refer to the nodes at depth εd as the middle nodes, and call $x_{\varepsilon d}$ the special middle node. As such, the observations made within depth εd reveal nothing about the location of the $x_{\varepsilon d}$, and any observations made in the subtree of a middle node s only reveals whether s is special or not, and reveals no further information about the location of $x_{\varepsilon d}$ relative to s.

At any point in the run time of the algorithm, we say that a middle node has been revealed if any observation is made in its subtree. By the previous paragraph, conditioned on the observations made so far, if none of the revealed vertices is special (i.e., if $x_{\varepsilon d}$ has not been reached), then the special node could be any of the remaining middle nodes uniformly at random. As such, any algorithm that reveals a c-fraction of the middle nodes, reaches $x_{\varepsilon d}$ with a probability of at most c.

Therefore, any algorithm that reaches the target state z (and hence the special middle node $x_{\varepsilon d}$) with constant probability, must reveal at least $\Omega(\Delta^{\varepsilon d}) = \Omega(\exp(\mathsf{O}(x_0))$ middle nodes, and as a result, take at least as many steps. This concludes the proof. □