Planning and Learning with Adaptive Lookahead

Aviv Rosenberg[1]*, Assaf Hallak[2], Shie Mannor[2,3], Gal Chechik[2,4], Gal Dalal[2]

[1] Amazon Science,
[2] Nvidia Research,
[3] Technion,
[4] Bar-Ilan University
avivros@amazon.com

Abstract

Some of the most powerful reinforcement learning frameworks use planning for action selection. Interestingly, their planning horizon is either fixed or determined arbitrarily by the state visitation history. Here, we expand beyond the naive fixed horizon and propose a theoretically justified strategy for adaptive selection of the planning horizon as a function of the state-dependent value estimate. We propose two variants for lookahead selection and analyze the trade-off between iteration count and computational complexity per iteration. We then devise a corresponding deep Q-network algorithm with an adaptive tree search horizon. We separate the value estimation per depth to compensate for the off-policy discrepancy between depths. Lastly, we demonstrate the efficacy of our adaptive lookahead method in a maze environment and Atari.

1 Introduction

The celebrated Policy Iteration (PI) and Value Iteration (VI) (Sutton and Barto 2018) algorithms are the basis for most state-of-the-art reinforcement learning (RL) algorithms. Since both PI and VI are based on a one-step greedy approach for policy improvement, so are the most commonly used policy-gradient (Schulman et al. 2017; Haarnoja et al. 2018) and Q-learning (Mnih et al. 2013; Hessel et al. 2018) based approaches. In each iteration, they perform an improvement of their current policy by looking one step forward and acting greedily. While this is the simplest and most common paradigm, stronger performance was recently achieved using multi-step lookahead. Notably, in AlphaGo (Silver et al. 2018) and MuZero (Schrittwieser et al. 2020), the multi-step lookahead is implemented via Monte Carlo Tree Search (MCTS) (Browne et al. 2012). In MCTS, the search depth is not chosen adaptively but gradually increases with the aggregation of state visitations.

Several recent works rigorously analyzed the properties of multi-step lookahead in common RL schemes (Efroni et al. 2018; Moerland et al. 2020; Hallak et al. 2021; Sikchi, Zhou, and Held 2022). These and other related literature studied a fixed planning horizon chosen *in advance*. However, both in simulated and real-world environments, a large variety of states benefit differently from various lookahead horizons. A grasping robot far from its target will learn very little from looking a few steps into the future, but if the target is within reach, much more precision and planning are required to grasp the object correctly. Similarly, at the beginning of a chess game, lookahead grants little information about which move is better, while agents in mid-game intricate situations benefit immensely from considering all future possibilities for the next few moves. Indeed, in this work, we devise a methodology for adaptive selection of the planning horizons in each state and show it achieves a significant speed-up of the learning process.

Algorithm	#Iterations	Iteration complexity
PI (Scherrer 2016)	1	$c(1)$
H-PI (Efroni et al. 2018)	$\frac{1}{H}$	$c(H)$
TLPI (this paper)	$\frac{1}{h^{(\kappa)}}$	$c(1) + \theta^{(\kappa)} c(h^{(\kappa)})$
QLPI (this paper)	$\frac{\log \gamma}{\log \kappa} \left(\leq \frac{1}{h^{(\kappa)}}\right)$	$O\left(\sum_{h=1}^{H} \theta_h c(h)\right)$

Table 1: Algorithm comparison summary. The iterations count (number of iterations) is divided by $S(A-1)\frac{\log(1-\gamma)}{\log \gamma}$. The iteration complexity (computational complexity per iteration) is divided by S. $c(k)$ is the computational complexity of k-step planning. $h^{(\kappa)}$ is the smallest integer h s.t. $\gamma^{h^{(\kappa)}} \leq \kappa$. $\theta^{(\kappa)}$ is the fraction of κ-contracting states. $\theta_1, \ldots, \theta_H$ are the contraction quantiles sizes.

We propose two complementing approaches to determine the suitable horizon per state in each PI iteration. To do so, we keep track of the room for improvement for the value function. Our first algorithm, Threshold-based Lookahead PI (TLPI), ensures the desired convergence rate and minimizes the computational complexity for each iteration. Alternatively, our second algorithm, Quantile-based Lookahead PI (QLPI), takes the per-iteration computational complexity as a given budget and aims for the best possible convergence rate. We prove that both TLPI and QLPI converge to the optimum and achieve a significantly lower computational cost than their fixed-horizon alternative (see Table 1).

Next, we devise QL-DQN: a DQN (Mnih et al. 2013) variant of QLPI, where the policy chooses an action by employ-

*Research conducted while the author was an intern at Nvidia Research.

ing an exhaustive tree search (Hallak et al. 2021) looking h steps into the future. The tree-depth h is chosen adaptively per state to achieve an overall improved convergence rate at a reduced computational cost. To sustain on-policy consistency while generalizing over the multiple depths, we use a different value network per depth, where the first layers are shared across networks. We test our method on Atari and show it improves upon a fixed-depth tree search.

To summarize, our contributions are the following. First, we propose to use adaptive state-dependent lookahead and devise corresponding algorithms. Our analysis shows they converge with improved computational complexity. Second, we extend our approach to online learning with a DQN variant that uses an exhaustive tree search of adaptive depth and per-depth value network. Third, we evaluate the proposed methods on maze and Atari environments and show better results than a fixed lookahead horizon.

2 Preliminaries

We consider a discounted MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is a finite state space of size S, $\mathcal{A}$ is a finite action space of size A, $r : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the reward function, $P : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$ is the transition function, and $\gamma \in (0, 1)$ is the discount factor. Let $\pi : \mathcal{S} \to \mathcal{A}$ be a stationary policy, and $V^\pi \in \mathbb{R}^S$ be the value function of π defined by $V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t) \mid s_0 = s\right]$.

The goal of a planning algorithm is to find the optimal policy $\pi^\star$ such that, for every $s \in \mathcal{S}$,

$$V^\star(s) = V^{\pi^\star}(s) = \max_{\pi:\mathcal{S}\to\mathcal{A}} V^\pi(s).$$

Given a policy π, let $T^\pi : \mathbb{R}^S \to \mathbb{R}^S$ be the Bellman operator: $T^\pi[V] = r^\pi + \gamma P^\pi V$, where $r^\pi(s) = r(s, \pi(s))$ and $P^\pi(s'|s) = P(s'|s, \pi(s))$. It is well known that the value of policy π is the unique solution to the linear equations: $T^\pi[V^\pi] = V^\pi$. Let $T : \mathbb{R}^S \to \mathbb{R}^S$ be the optimal Bellman operator defined as:

$$T[V](s) = \max_{a\in\mathcal{A}} r(s, a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s, a)V(s').$$

Then, the optimal value is the unique solution to the nonlinear equations $T[V^\star] = V^\star$ and T is a γ-contraction in the max-norm over the state space:

$$\|V^\star - T[V^\pi]\|_\infty \le \gamma\|V^\star - V^\pi\|_\infty.$$

2.1 PI and h-PI

PI starts from an arbitrary policy π_0 and performs iterations that consist of: (1) an evaluation step that evaluates the value of the current policy, and (2) an improvement step that performs a 1-step improvement based on the computed value. That is, for $n = 0, 1, 2, \dots$,

$$\pi_{n+1}(s) = \arg\max_{a\in\mathcal{A}} r(s, a) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s, a)V^{\pi_n}(s').$$

By the contraction property of the Bellman operator, one can prove that PI finds the optimal policy after at most $\lceil(\log \frac{1}{\gamma})^{-1} S(A - 1) \log {}^{1}\!/_{1-\gamma}\rceil$ iterations (Scherrer 2016).

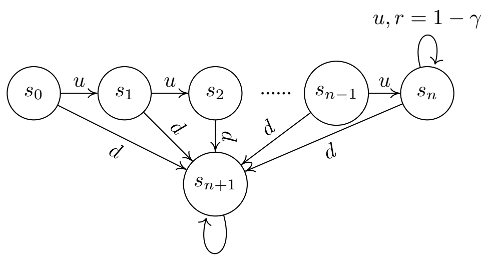

Figure 1: Chain MDP example with deterministic transitions and rewards 0 everywhere except for $r(s_n, u) = 1-\gamma$. Using a fixed horizon h per state for each PI iteration leads to the same convergence rate as using horizon ℓ in only a single state for $\ell = 2, \dots, h$, but at a much higher computational cost.

The PI algorithm can be extended to h-PI by performing h-step improvements (instead of 1-step). Formally, define the Q-function of policy π with a h-step lookahead as

$$Q_h^\pi(s, a) = \max_{\pi_t} \mathbb{E}^{s,a} \left[\sum_{t=0}^{h-1} \gamma^t r(s_t, \pi_t(s_t)) + \gamma^h V^\pi(s_h)\right],$$

where $\mathbb{E}^{s,a}[\cdot] = \mathbb{E}[\cdot|s_0 = s, \pi_0(s) = a]$. Then, the update rule of h-PI is $\pi_{n+1}(s) = \arg\max_{a\in\mathcal{A}} Q_h^{\pi_n}(s, a)$. The operator induced by h-step lookahead is a γ^h-contraction which allows to reduce a factor of h from the bound on the number of iterations until convergence (Efroni et al. 2018), i.e., it is bounded by $\left\lceil(h \log \frac{1}{\gamma})^{-1} S(A - 1) \log \frac{1}{1-\gamma}\right\rceil$.

Multi-step lookahead guarantees that the number of iterations to convergence is smaller than the 1-step lookahead, but it comes with a computational cost. Computing the h-step improvement may take exponential time in h. In tabular MDPs, this can be mitigated with the use of dynamic programming (Efroni, Ghavamzadeh, and Mannor 2020), while in MDPs with large (or infinite) state space, MCTS (Browne et al. 2012) or the alternative exhaustive tree-search (Hallak et al. 2021) are used in forward-looking fashion. To compare our algorithms, in the rest of the paper we measure the computational complexity as follows:

Definition 2.1. Let $c(h)$ be the computational cost of performing a h-step improvement in a single state. For example, in a deterministic full A-ary tree we have $c(h) = O(A^h)$.

3 Motivating Example

To show the potential of our approach, consider the chain MDP example in Figure 1:

Example 3.1 (Chain MDP). Let $\mathcal{M}$ be an MDP with $n + 1$ states $s_0, s_1, \dots, s_n$ and a single sink state s_{n+1}. Each of the $n+1$ states transitions to the consecutive state with action u, and to the sink state with action d. All rewards are 0 except for state s_n in which u yields reward $1 - \gamma$.

Now consider the standard PI algorithm initialized with $\pi_0(s_i) = d$ for all $i \in \{0, \dots, n\}$. Since the reward at the end of the chain needs to propagate backward, in each iteration the value of only a single state is updated. Thus, PI takes exactly n iterations to converge to the optimal policy $\pi^\star(s_i) = u$ for all i. Instead, with a fixed horizon $h = 2$, the reward propagates through two states in each iteration (instead of one) and therefore convergence takes $\lceil n/2 \rceil$ iterations. Generally, PI with a fixed horizon h, i.e., h-PI, converges in $\lceil n/h \rceil$ iterations.

While h-PI converges faster (in terms of iterations) as h increases, in most states, performing h-step lookahead does not contribute to the speed-up at all. In our example, we can achieve exactly the same convergence rate as 2-PI by using a 2-step lookahead in only a single state in each iteration (and 1-step in all other states). Specifically, we need to pick the state that is exactly 2 steps behind the last updated state in the chain. For general h, consider applying ℓ-step lookahead in only one state — the one that is ℓ steps behind the last updated state in the chain — for $\ell = 2, \dots, h$ and 1-step in the others. This guarantees the same number of iterations until convergence as h-PI, but with much less computation time. Namely, while the per-iteration computational cost of h-PI is $O\big(n \cdot c(h)\big)$, we can achieve the same convergence rate with just $O\big(n \cdot c(1) + \sum_{\ell=2}^{h} c(\ell)\big)$. In practice, when n is large and $c(h)$ scales exponentially with h, this gap can be immense: $O\big(n \cdot 2^h\big)$ versus $O\big(n + 2^h\big)$.

4 Contraction-Based Adaptive Lookahead

In this section, we introduce the concept of dynamically adapting the planning lookahead horizon during runtime, based on the online obtained contraction. In Example 3.1, h-PI convergence rate can be achieved when using a lookahead larger than 1 in just h states. The question is how to choose these states? In the example, the chosen states are evidently those with the maximal distance between their 1-step improvement and optimal value, i.e., $\arg\max_{s\in\mathcal{S}} |V^\star(s) - T[V^{\pi_t}](s)|$. In this section, we show that this approach also leads to theoretical guarantees on the convergence of the PI algorithm.

To understand how the convergence rate depends on the distance of the 1-step improvement from the approximate optimal value, we delve into the theoretical properties of PI. Since the standard 1-step improvement yields a contraction of γ while the h-step improvement gives γ^h, h-PI converges h times faster than standard PI (Efroni et al. 2018). Importantly, this contraction is with respect to the L_∞ norm; i.e., the states with the worst (largest) contraction coefficient determine the convergence rate of PI. This behavior is the source of weakness of using a fixed lookahead. Example 3.1 shows that one state may slow down convergence, but it also hints at an elegant solution: *use larger lookahead in states with larger contraction coefficient.*

We leverage this observation and present two new algorithms: TLPI which aims to achieve a fixed contraction in all states with a reduced computational cost, and QLPI which aims to achieve maximal contraction in every iteration within a fixed computational budget. While both algorithms seek to optimize a similar problem, their analyses differ and shed light on the problem from different perspectives: TLPI depends on the contraction factor per state, while QLPI considers only the ordering of the states with respect to their contraction factors.

Algorithm 1: TLPI

```
1: Input: $\mathcal{S}, \mathcal{A}, r, P, \gamma, \kappa, \beta, \widetilde{V}^\star$.
2: Initialization: Arbitrary $\pi_0$, $t \leftarrow 0$.
3: while $\pi_t$ changes do
4:     Evaluation: compute $V^{\pi_t}$, and set $U(s,a) \leftarrow \infty$.
5:     1-step improvement: $U(s,a) \leftarrow Q_1^{\pi_t}(s,a)\ \forall (s,a)$.
6:     $h^{(\kappa)}$-step improvement: $U(s,a) \leftarrow Q_{h^{(\kappa)}}^{\pi_t}(s,a)$ for
       every $(s,a)$ s.t.: (here $U(s) = \max_a U(s,a)$)
           $|\widetilde{V}^\star(s) - U(s)| > \kappa \|\widetilde{V}^\star - V^{\pi_t}\|_\infty - \beta.$   (1)
7:     Set $\pi_{t+1}(s) \leftarrow \arg\max_{a\in\mathcal{A}} U(s,a)$ for every $s \in \mathcal{S}$.
8: end while
```

Since we do not have access to the optimal value $V^\star(s)$, the algorithms rely on warm-starts or an approximation of the optimal value $V^\star$, denoted by $\widetilde{V}^\star$. While obtaining a good approximation of the value function is hard, we aim at a simpler task: find an approximation that is informative for allocating the depth resources. The approximated values may be far from optimal, as long as they yield similar allocation across depths. In many cases, we can obtain such an approximation through, e.g., state aggregation, training agents on similar tasks, or by running PI for a small number of iterations. In Sections 5 and 6, we show that these approximation methods are indeed effective in practice.

4.1 Threshold-Based Lookahead Policy Iteration

TLPI (Algorithm 1) takes as input the approximated value $\widetilde{V}^\star$, a desired contraction factor κ and a correction term β. We assume that $\|V^\star - \widetilde{V}^\star\|_\infty \le \epsilon$. This implies we can measure contraction up to some approximation error that scales with ϵ. The algorithm ensures that in each iteration, the value in every state contracts by at least κ. This is achieved by first performing 1-step improvement in all states and then performing $h^{(\kappa)}$-improvement in states whose measured contraction is less than κ, where $h^{(\kappa)}$ is the smallest integer h such that $\gamma^h \le \kappa$. Since we do not have an accurate estimate of the optimal value, we use the correction term β to make sure that no states falsely seem to achieve the desired contraction due to the approximation error ϵ (see Equation (1)).

The following result states that TLPI converges at least as fast as h-PI (Efroni et al. 2018) with h set to $h^{(\kappa)} - 1$, and with improved computational complexity. To measure the trade-off between the contraction factor (that determines the convergence rate) and the computational complexity needed to achieve it, Definition 4.1 presents $\theta^{(\kappa)}$ as the fraction of states in which we perform a large lookahead.

Definition 4.1 (Def. of $\theta^{(\kappa)}$). Let $\{\pi_t\}_{t=1}^T$ be the sequence of policies generated by TLPI with correction term β and approximated value $\widetilde{V}^\star$. Let $\kappa \in (0,1)$, and define

$$\mathcal{X}_t = \{s : |\widetilde{V}^\star(s) - T[V^{\pi_t}](s)| \le \kappa \|\widetilde{V}^\star - V^{\pi_t}\|_\infty - \beta\}$$

as the set of states, that after 1-step improvement in iteration t, are β-close to be contracted by κ with respect to $\widetilde{V}^\star$. Then,

Algorithm 2: QLPI

```
1: **Input:** $\mathcal{S}, \mathcal{A}, r, P, \gamma, (\theta_1 \dots \theta_H), m, \widetilde{V}^\star$.
2: **Initialization:** Arbitrary $\pi_0$, $t \leftarrow 0$.
3: **while** $\pi_t$ changes **do**
4:     Evaluation: compute $V^{\pi_t}$, and set $U(s,a) \leftarrow \infty$.
5:     **for** $h = 1, 2, \dots, H$ **do**
6:         Compute $q_h$ as the $(1 - \theta_h - m/S)$ quantile of $\{|\widetilde{V}^\star(s) - \max_a U(s,a)|\}_{s \in \mathcal{S}}$.
7:         $h$-step improvement: $U(s,a) \leftarrow Q_h^{\pi_t}(s,a)$ for every $(s,a)$ s.t.: $|V^\star(s) - \max_{a \in \mathcal{A}} U(s,a)| \geq q_h$.
8:     **end for**
9:     Set $\pi_{t+1}(s) \leftarrow \arg\max_{a \in \mathcal{A}} U(s,a)$ for every $s \in \mathcal{S}$.
10: **end while**
```

denote by $\theta^{(\kappa)} = \max_{1 \leq t \leq T} |\mathcal{S} \setminus \mathcal{X}_t|/S$ the largest fraction of states with contraction less than κ, observed along all policy updates.

Theorem 4.2. *The TLPI algorithm with approximated value $\widetilde{V}^\star$ and correction term $\beta = \epsilon(\kappa + 1)$ converges in at most $\left\lceil \left((h^{(\kappa)} - 1)\log\frac{1}{\gamma}\right)^{-1} S(A-1)\log\frac{1}{1-\gamma} \right\rceil$ iterations. Moreover, its per-iteration computational complexity is bounded by $S \cdot \left(c(1) + \theta^{(\kappa)} c(h^{(\kappa)})\right)$.*

Proof sketch. The proof to bound the number of iterations follows Scherrer (2016) while utilizing two key observations. First, the convergence analysis of PI only uses the contraction property of the Bellman operator w.r.t. $V^\star$, and not w.r.t. an arbitrary pivot vector. The distance to $V^\star$ can be approximated using $\widetilde{V}^\star$, and the approximation error is handled by the correction term β. Second, by the construction of the algorithm, a contraction of at least κ in every state is guaranteed. The computational complexity follows because we perform 1-step lookahead in all states and $h^{(\kappa)}$-step lookahead in $\theta^{(\kappa)}$ of the states by Definition 4.1. For the complete proof, see Appendix A.1. □

To illustrate the merits of TLPI and Thoerem 4.2, consider the Chain MDP in Example 3.1 where we set $\kappa = \gamma^h$ for some $h \in \mathbb{N}$ (and assume $\widetilde{V}^\star = V^\star$ for simplicity). In every iteration, the states not contracted by κ after 1-step improvement are the h states closest to the end of the chain that have not been updated yet (recall that each state reaches the correct optimal value after just one non-idle update). Thus, $\theta^{(\kappa)} = h/S$ and the per-iteration computational complexity is $S \cdot c(1) + h \cdot c(h)$.

4.2 Quantile-Based Lookahead Policy Iteration

QLPI (Algorithm 2) resembles TLPI, but instead of a contraction coefficient κ, it takes as input a vector of quantiles (budgets) $(\theta_1, \dots, \theta_H) \in [0,1]^H$ for some predetermined maximal considered lookahead H. Instead of the actual distance to the optimal value, QLPI relies only on the ordering of the states in terms of distance from the optimum. This allows for weaker requirements on the approximated value $\widetilde{V}^\star$ as it should only preserve the order.

Definition 4.3. Let p_s and $\tilde{p}_s$ be the positions of state s in the orderings of $\{|V^\star(s) - V^{\pi_t}(s)|\}_{s \in \mathcal{S}}$ and $\{|\widetilde{V}^\star(s) - V^{\pi_t}(s)|\}_{s \in \mathcal{S}}$, respectively. We define the approximation $\widetilde{V}^\star$ to be *m-order-preserving* if, for every $s \in \mathcal{S}$, $|p_s - \tilde{p}_s| \leq m$.

State-aggregation is an example of an approximation that preserves orders and that is available in many domains where states are based on locality (like the maze environment considered in Section 5). Assume that we have access to a state-aggregation scheme that splits the state space into S/m groups of size m such that for every two states s_1, s_2 in the same group $|V^\star(s_1) - V^\star(s_2)| \leq \epsilon$ and for any state s_3 from a different group $|V^\star(s_1) - V^\star(s_3)| > 2\epsilon$. Then the optimal value of the aggregated MDP $V^\star_{agg}$ is m-order-preserving as long as $|V^\star_{agg}(s) - V^\star(s)| \leq \epsilon$ for every $s \in \mathcal{S}$, since the position of a state can be shifted due to the aggregation by at most the size of its group m.

QLPI attempts to maximize the contraction in every iteration while using ℓ-step lookahead in at most $\theta_\ell \cdot S + m$ states. This is achieved by performing ℓ-step improvement on the $(\theta_\ell + m/S)$ portion of states that are furthest away from $\widetilde{V}^\star$.

The following result is complementary to Theorem 4.2: now, instead of choosing the desired iteration complexity (via κ in TLPI), we choose the desired computational complexity per iteration via budgets $(\theta_1, \dots, \theta_H)$. For the resulting iteration complexity we define the induced contraction factor:

Definition 4.4 (Def. of $\kappa^{(\theta)}$)**.** Let $\{\pi_t\}_{t=1}^T$ be the sequence of policies generated by QLPI. Let $h_t^\theta(s)$ be the largest lookahead applied in state s in iteration t when running QLPI with quantiles $(\theta_1, \dots, \theta_H)$. For a given κ, define

$$\mathcal{Y}_t(\kappa) = \{s : |V^\star(s) - T^{h_t^\theta(s)} V^{\pi_t}(s)| \leq \kappa \|V^\star - V^{\pi_t}\|_\infty\}$$

as the set of states contracted by κ in iteration t. The induced contraction factor $\kappa^{(\theta)}$ is defined as the minimal κ such that $\mathcal{Y}_t(\kappa) = \mathcal{S}$ for every t.

Though its formal definition may seem complex, $\kappa^{(\theta)}$ is simply the effective contraction obtained by QLPI.

Theorem 4.5. *The QLPI algorithm converges in at most $\left\lceil (\log\frac{1}{\kappa^{(\theta)}})^{-1} S(A-1)\log\frac{1}{1-\gamma} \right\rceil$ iterations. Moreover, its per-iteration computational complexity is bounded by $S \cdot \sum_{h=1}^H (\theta_h + m/S) c(h)$.*

We provide the proof in Appendix A.2; it is based on similar ideas as the proof of Theorem 4.2.

To illustrate the merits of QLPI and Theorem 4.5, consider the Chain MDP in Example 3.1 where we set $\theta_1 = 1$ and $\theta_2 = \dots = \theta_h = 1/S$ for some $h \in \mathbb{N}$ (again $\widetilde{V}^\star = V^\star$ for simplicity). In every iteration QLPI first performs 1-step lookahead in all states, and then, for each $\ell = 2, \dots, h$, it performs ℓ-step lookahead in exactly one state – the state that is ℓ steps behind the last updated state in the chain. As explained in Section 3, the induced contraction is $\kappa^{(\theta)} = \gamma^h$ and QLPI converges in n/h iterations with *optimal* per-iteration complexity of $S \cdot c(1) + \sum_{\ell=2}^h c(\ell)$.

Finally, we highlight the complementary nature of the two algorithms: while in TLPI the complexity parameter

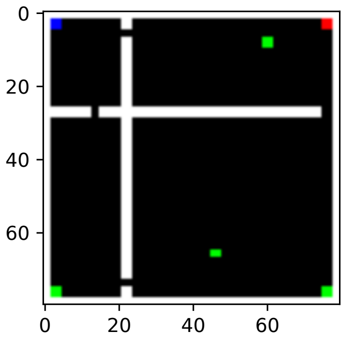

Figure 2: Snapshot of our maze environment, a 30×30 grid world. The agent is spawned in the top left corner (blue) and needs to reach one of four randomly chosen goals (green), while avoiding the trap (red). White pixels denote walls. Upon reaching a goal state, the agent re-appears in a new random location.

$\theta^{(\kappa)}$ is governed by the desired contraction coefficient, in QLPI the induced contraction $\kappa^{(\theta)}$ is the outcome of the predetermined computational budget.

5 Maze Experiments

In the first set of experiments, we evaluate our adaptive lookahead algorithms, TLPI and QLPI, on a grid world with walls (Tennenholtz et al. 2022). Specifically, we used a 30×30 grid world that is divided to four rooms with doors between them; see Figure 2. The agent is spawned in the top left corner (blue) and needs to reach one of four randomly chosen goals (green) where the reward is 1, while avoiding the trap (red) that incurs a reward of -1. There are four deterministic actions (up, down, right, left). Upon reaching a goal, the agent is moved to a random state. We set $\gamma = 0.98$.

Fixed lookahead. We begin with testing the fixed-horizon h-PI with values $h = 1, 2, \ldots, 7$. To corroborate that larger lookahead values reduce the number of PI iterations required for convergence, in Figure 3, we show the distance from the solution as the function of iteration for the different depths. The plot demonstrates the effect of the lookahead in a less pathological example than Example 3.1.

In Figure 5, we compare the overall computational complexity, and not only the number of iterations, of the different fixed lookahead values. To measure performance, we count the number of *queries to the simulator* (environment) until convergence to the optimal value. More efficient lookahead horizons will require fewer overall calls to the simulator.

Beginning with the fixed lookahead results in the leftmost plot, we see the trade-off when picking the lookahead. For a lookahead too short (1 in this case), the convergence requires too many iterations such that even the low computational complexity of each iteration is not sufficient to compensate for the total compute time. Note that h-PI with $h = 1$ is the standard PI algorithm, which evidently performs worse than the best-fixed lookahead although it is overwhelmingly the most widely used version of PI. On the other extreme of a very large lookahead, each iteration is too computationally

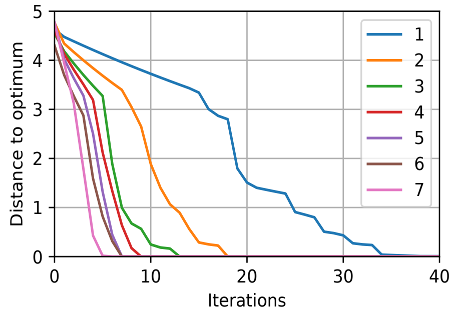

Figure 3: The distance to the optimum, captured by $\|V^\star - V^{\pi_t}\|_\infty$, as a function of the iteration number t for fixed lookahead values of $h = 1, 2, \ldots, 7$ in maze environment.

expensive, despite the smaller number of iterations.

TLPI. To verify our observation that a long lookahead is wasteful in large parts of the state space, we first plot a histogram of the contraction factor along several PI iterations in Figure 4. Here we see that indeed the effective contraction factor κ is much smaller than γ (i.e., more contractive than 1-step lookahead) in roughly 90% of the states.

Next, we run TLPI with $\kappa = \gamma^2, \gamma^3, \ldots, \gamma^7$ and an accurate approximated value $\widetilde{V}^\star = V^\star$. The results are given in Figure 5, second plot. By Theorem 4.2, when setting $\kappa = \gamma^h$, we expect the same number of iterations until convergence as h-PI but with better computational complexity. In fact, the results reveal even stronger behavior: TLPI(γ^h) for all $h = 1, 2, \ldots, 7$ achieves similar computational complexity compared to the *best* fixed lookahead witnessed in h-PI.

QLPI. In all our experiments we run QLPI with $\theta_1 = 1$ and $\theta_3 = \theta_5 = \theta_6 = \theta_7 = 0$ (again $\widetilde{V}^\star = V^\star$). For $(\theta_2, \theta_4, \theta_8)$ we set the following values: $(0.3, 0.2, 0.1)$, $(0.2, 0.15, 0.05)$, $(0.2, 0.05, 0.02)$ and $(0.1, 0.05, 0.02)$, which respectively depict decreasing weights to depths $2, 4, 8$. The results are presented in Figure 5, third plot. Again we can see that for all the parameters, QLPI performs as well as the best-fixed lookahead. Moreover, notice that for some of the choices of θ vectors, the performance significantly improves upon the best-fixed horizon.

Approximate $V^\star$ via state aggregation. We again run QLPI with budget values $(0.1, 0.05, 0.02)$, but replace $V^\star$ with an approximation we obtain with state aggregation. Namely, we merge squares of $k \times k$ into a single state, solve the smaller aggregated MDP, and use its optimal value as an approximation for $V^\star$.

We perform this experiment with $k = 2, 3, 4, 5$ and include the aggregated MDP solution process in the total simulator query count. This way, our final algorithm does not have any prior knowledge of $V^\star$. The results are presented in Figure 5, last plot. As expected, the performance is slightly worse than the original QLPI that uses the accurate $V^\star$, but for all different aggregation choices the algorithm still performs as well as the best-fixed lookahead in h-PI.

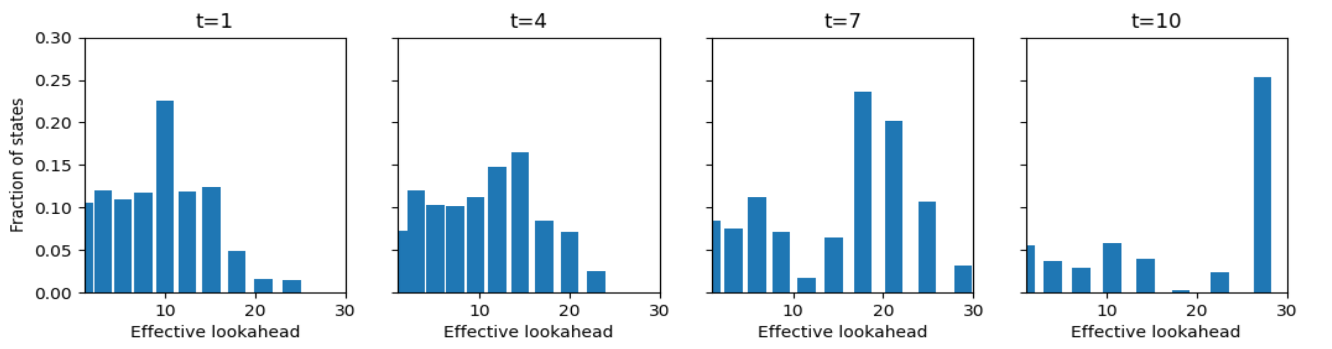

Figure 4: Histograms for the fraction of states per effective lookahead along several iterations of PI. The effective lookahead of contraction factor κ is defined as $h = \log_\gamma(\kappa)$, i.e., $\gamma^h = \kappa$.

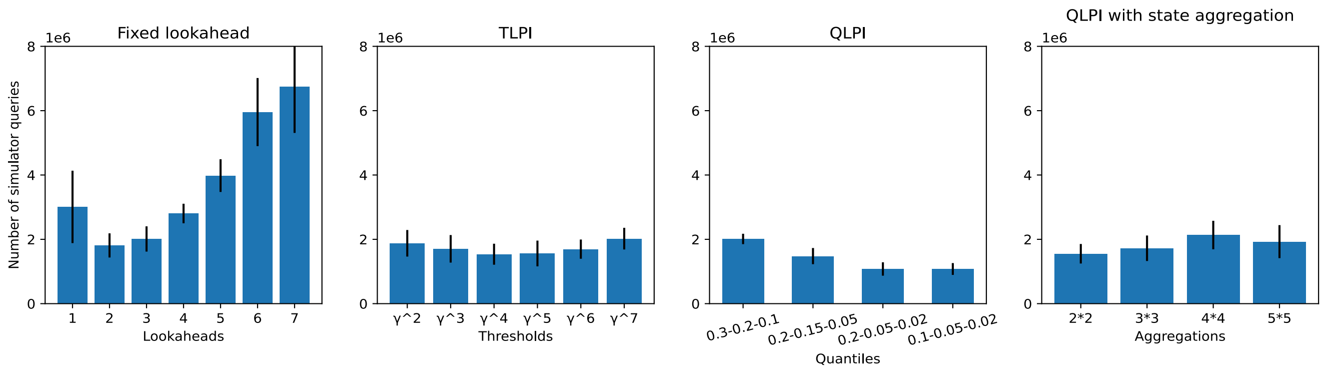

Figure 5: Number of simulator queries until convergence. Lower is better. Results are averaged across 10 runs and error bars represent standard deviation. QLPI is run with lookaheads 1, 2, 4, and 8, where the quantiles in the x-axis represent $\theta_2, \theta_4, \theta_8$.

To summarize, the maze experiments show that adaptive planning lookaheads manage to reach the solution with better sample complexity (i.e. number of simulator queries) compared to fixed-horizon h-PI. More importantly, our methods are robust to hyperparameter choices: the improved results are obtained uniformly with *all* various tested parameters of TLPI and QLPI. This alleviates the heavy burden of finding the best-fixed horizon for a given environment.

6 QL-DQN and Atari Experiments

In this section, we extend our adaptive lookahead algorithm QLPI to neural network function approximation. We present Quantile-based lookahead DQN (QL-DQN): the first DQN algorithm that uses state-dependent lookahead that is dynamically chosen throughout the learning process. QL-DQN is visualized in Figure 7 and operates as follows:

1. We introduce a *per-depth Q-function*. Technically, maintain H parallel Q-networks (where H is the maximal tree depth) and use the h-th network to predict the value of a leaf in depth h. To improve generalization and data re-use, the networks share initial layers (feature extractors).
2. Maintain a sorted replay buffer according to the distance between the current value estimate and approximated optimal value (for efficiency we utilize a priority queue). For $\tilde{V}^\star$, we train a standard DQN (depth 0) agent for only 1M steps – a relatively quick process.
3. For a replay buffer of size N, maintain H quantiles based on the above ordering. The quantile sizes are $\theta_1 \cdot N, \ldots, \theta_H \cdot N$. The values θ_i are hyper-parameters.
4. Choosing tree depth: Per state during simulation, start by spanning the 1st level of the tree. If the best leaf value distance from the approximate optimal value is in the 1st quantile, end the tree-search. Otherwise, go one level deeper, compare to the 2nd quantile, and re-iterate with the same logic. Continue possibly until the max depth H is reached. Notice that the tree-search is feasible in reasonable run-time thanks to highly efficient parallel Atari simulation on GPU (Hallak et al. 2021).
5. Choosing action given depth: After spanning the tree as described above, return the first action (from the root) that corresponds to the highest leaf value.
6. Store (state, action, cumulative reward to leaf, leaf state, tree-depth) to the replay buffer.
7. For training, set the bootstrap target to be the cumulative reward from the tree-search plus the Q-value corresponding tree-depth calculated on the leaf-state.

We note that the per-depth Q-function is crucial in order to keep online consistency and achieve convergence. We found

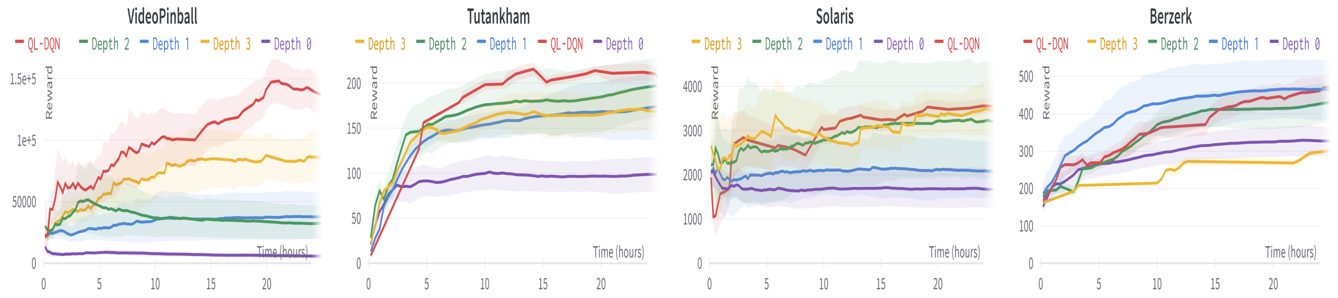

Figure 6: Average and std of training reward of QL-DQN (in red) and DQN with fixed tree-depths $0, 1, 2, 3$ in various Atari environments. x-axis is time in hours (a plot with step-based x-axis is given in Appendix B).

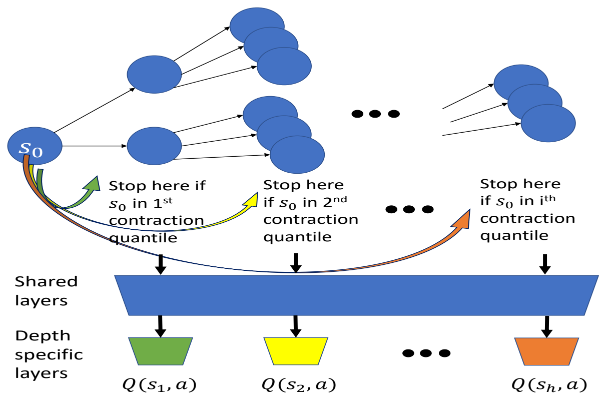
s_0
Stop here if s_0 in 1st contraction quantile
Stop here if s_0 in 2nd contraction quantile
Stop here if s_0 in ith contraction quantile
Shared layers
Depth specific layers
$Q(s_1, a)$
$Q(s_2, a)$
$Q(s_h, a)$

Figure 7: The QL-DQN algorithm. When choosing actions, the policy uses a tree depth based on the ranking of s_0's contraction coefficient in the replay buffer. The per-depth Q network has a shared basis and depth-specific heads.

that in practice, the Q-function learned by DQN is not the true cumulative reward of rollouts. Instead, it is some function that minimizes the Bellman error. This phenomenon is orthogonal to our work and was also recently studied in (Fujimoto et al. 2022). In our context of multiple-depth Q-network, if we bootstrap using the target from one depth for another, the above phenomenon causes inconsistencies that lead to divergence. To handle this inconsistency, we introduced the multi-head Q-network for multiple depths and found that it solves the issue. All other parts of the algorithm and hyper-parameter choices are taken as-is from the original DQN paper (Mnih et al. 2013).

We train QL-DQN on several Atari environments (Bellemare et al. 2013). Since our work aims to improve sample complexity over fixed-horizon baselines, our metric of interest here is the reward as a function of training time. Hence, in Figure 6 we present the convergence of QL-DQN versus DQN with fixed depths 0 through 3, as a function of time. The plots consist of the average score across 5 seeds together with std values. Note that depth 0 corresponds to standard DQN (the baseline). As seen, QL-DQN achieves better performance on VideoPinball and Tutankham, while on Solaris and Berzerk, it is on par with the best-fixed lookahead.

The conclusion here is again that we obtain a better or similarly-performing agent to a pre-determined fixed planning horizon. This comes with the benefit of robustness to the expensive hyper-parameter choice of the best-fixed horizon per a given environment.

7 Discussion

In this paper we propose the first planning and learning algorithms that dynamically adapt the multi-step lookahead horizon as a function of the state and the current value function estimate. We demonstrate the significant potential of adaptive lookahead both theoretically — proving convergence with improved computational complexity, and empirically — demonstrating their favorable performance in a maze and Atari. Our algorithms often perform as well as the best-fixed horizon in hindsight in almost all the experiments, while in some cases they surpass it. Future work warrants an investigation whether the best-fixed horizon can always be outperformed by an adaptive horizon.

Theoretically, our guarantees rely on prior knowledge of an approximate optimal value, raising the question whether one can choose lookahead horizons adaptively without any prior knowledge, e.g., using transfer learning based on similarity between domains. Moreover, when the forward model performing the lookahead is inaccurate or learned from data, the adaptive state-dependent lookahead itself may serve as a quantifier for the level of trust in the value function estimate (short lookahead) versus the model (long lookahead). This can offer a way for state-wise regularization of the learning or planning problem. Our work is also related to the growing Sim2Real literature. In particular, when having several simulators with different computational costs and fidelity levels. The lookahead problem then translates to choosing in which states to use which simulator with which lookahead.

Our focus in this paper was reducing iteration and overall complexity; we thus ignored more intricate details of the forward search itself. Additional practical aspects such as CPU-GPU planning efficiency trade-offs (Hallak et al. 2021) can also affect the lookahead selection problem. One promising direction is to expand at each step only the few most promising nodes, and keep the search width fixed after a certain value. This gives linear complexity in the search depth instead of exponential, at the risk of missing relevant paths.

References

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1): 1–43.

Efroni, Y.; Dalal, G.; Scherrer, B.; and Mannor, S. 2018. Beyond the one-step greedy approach in reinforcement learning. In *International Conference on Machine Learning*, 1387–1396. PMLR.

Efroni, Y.; Ghavamzadeh, M.; and Mannor, S. 2020. Online Planning with Lookahead Policies. *Advances in Neural Information Processing Systems*, 33.

Fujimoto, S.; Meger, D.; Precup, D.; Nachum, O.; and Gu, S. S. 2022. Why Should I Trust You, Bellman? The Bellman Error is a Poor Replacement for Value Error. *arXiv preprint arXiv:2201.12417*.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 1861–1870. PMLR.

Hallak, A.; Dalal, G.; Dalton, S.; Mannor, S.; and Chechik, G. 2021. Improve Agents without Retraining: Parallel Tree Search with Off-Policy Correction. *Advances in Neural Information Processing Systems*, 34.

Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Moerland, T. M.; Deichler, A.; Baldi, S.; Broekens, J.; and Jonker, C. M. 2020. Think too fast nor too slow: The computational trade-off between planning and reinforcement learning. *arXiv preprint arXiv:2005.07404*.

Scherrer, B. 2016. Improved and generalized upper bounds on the complexity of policy iteration. *Mathematics of Operations Research*, 41(3): 758–774.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sikchi, H.; Zhou, W.; and Held, D. 2022. Learning off-policy with online planning. In *Conference on Robot Learning*, 1622–1633. PMLR.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Tennenholtz, G.; Hallak, A.; Dalal, G.; Mannor, S.; Chechik, G.; and Shalit, U. 2022. On Covariate Shift of Latent Confounders in Imitation and Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*.