ON CREDIT ASSIGNMENT IN HIERARCHICAL REINFORCEMENT LEARNING

Joery A. de Vries
Algorithmics Group
TU Delft
Delft, 2628 XE, The Netherlands
J.A.deVries@tudelft.nl

Thomas M. Moerland & Aske Plaat
Leiden Institute of Advanced Computer Science
Leiden University
Leiden, 2333 CA, The Netherlands
{t.m.moerland, a.plaat}@liacs.leidenuniv.nl

ABSTRACT

Hierarchical Reinforcement Learning (HRL) has held longstanding promise to advance reinforcement learning. Yet, it has remained a considerable challenge to develop practical algorithms that exhibit some of these promises. To improve our fundamental understanding of HRL, we investigate hierarchical credit assignment from the perspective of conventional multistep reinforcement learning. We show how e.g., a 1-step 'hierarchical backup' can be seen as a conventional multistep backup with n skip connections over time connecting each subsequent state to the first independent of actions inbetween. Furthermore, we find that generalizing hierarchy to multistep return estimation methods requires us to consider how to partition the environment trace, in order to construct backup paths. We leverage these insight to develop a new hierarchical algorithm Hier$Q_k(\lambda)$, for which we demonstrate that hierarchical credit assignment alone can already boost agent performance (i.e., when eliminating generalization or exploration). Altogether, our work yields fundamental insight into the nature of hierarchical backups and distinguishes this as an additional basis for reinforcement learning research.

1 INTRODUCTION

Hierarchical Reinforcement Learning (HRL) is often regarded as an open frontier in RL for developing more sample-efficient control algorithms (Sutton et al., 1999; Dietterich, 1998; Bakker & Schmidhuber, 2004; Sutton & Barto, 2018; Levy et al., 2018; Kulkarni et al., 2016). Hierarchy provides innate structure for solving complex problems by decomposing tasks into smaller, simpler and recurring, subtasks. In turn, this allows the decision making algorithm to reason or plan over temporally distant events instead of only the (arbitrarily granular) environment actions.

A vast body of literature has documented and demonstrated potential benefits of hierarchy. There have also been a few impressive empirical results (e.g., (Jaderberg et al., 2019)). Despite all this, it has remained difficult to design and work with hierarchical agents due to the additional challenges that these methods introduce. In essence, in HRL we often attempt to design algorithms that try to unite multiple, possibly non-stationary and unstable, policies into a more efficient whole. Research is occasionally hindered by issues relating to training instability, goal misspecification, or even a collapse of the hierarchy to a flat agent (Vezhnevets et al., 2017; Sutton et al., 1999; Nachum et al., 2018). While numerous methods effectively patch such encountered issues *ad-hoc*, a better fundamental understanding of the learning dynamics will be crucial in advancing HRL research.

The concept of learning in RL relates to the *policy evaluation* problem: the estimation of the expected future return for a particular policy. Numerous successful return estimation algorithms have been proposed for flat agents, such as TD(λ) or Retrace (Sutton & Barto, 2018; Munos et al., 2016). However, it is not directly obvious how these methods translate to the hierarchical setting. Hierarchical policies operate on varying levels of time granularity, and essentially can 'jump' over flat actions when backing up rewards.

We take as a practical example the hierarchical Q-learning algorithm by Levy et al. (2018) and show how flat reward estimation methods can be adapted to hierarchy. We do this for the Tree-Backup operator (Sutton & Barto, 2018) and an adaptation of Watkin's $Q(\lambda)$ (Watkins & Dayan, 1992), we

generalize this into a new algorithm called Hier$Q_k(\lambda)$ where k refers to the number of hierarchy levels. Finally, we analyze these methods in environment domains where we isolate the benefit of hierarchy to *just* the credit assignment, in order to compare the hierarchical agents to similarly formulated flat approaches.

In short, we make the following contributions: 1) we systematically study hierarchical credit assignment patterns, and compare them to flat back-ups, 2) we propose a new algorithm, Hierarchical $Q(\lambda)$, which integrates these insights with Tree-Backup, and 3) we empirically compare the performance of this algorithm on a range of tasks, which shows that hierarchy provides a fundamental performance benefit over flat agents through reward assignment alone (i.e., when removing other benefits of hierarchy provided by generalization, exploration, or transfer).

2 BACKGROUND

We consider solving Markov Decision Processes (MDP) which are 5-tuples $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, R, \gamma \rangle$ where $\mathcal{S}$ and $\mathcal{A}$ are a set of states and actions; $p : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is a stationary transition distribution; R maps transitions to rewards $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ which we abbreviate as $R_t = R(S_t, A_t, S_{t+1})$; and $\gamma \in [0, 1]$ is a discount factor (Sutton & Barto, 2018). Let $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ denote a control policy from which we can generate experiences[1] in $\mathcal{M}$: $\tau_{t:T:k} = \{S_{t+ki}, A_{t+ki}, R_{t+ki+1}\}_{i=0}^{T-t}, k = 1$. Then we seek the optimal policy π^* that maximizes the value function,

$$q_\pi(s, a) = \mathbb{E}_{\pi,p}[G_{t:T}|S_t = s, A_t = a] = \mathbb{E}_{\pi,p}[\sum_{i=0}^{T-t} \gamma^i R_{t+i+1}|S_t = s, A_t = a], \tag{1}$$

which, in practice, is estimated from samples $Q^\pi \approx q_\pi$.

A famous algorithm that learns q_π from data is Q-learning (Watkins & Dayan, 1992). At any transition $S_t, A_t, R_{t+1}, S_{t+1}$ it computes a biased estimation error using the recursive Bellman property of Q^π, $\delta_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$. This error term is then used to adjust the current estimate to q_π using the online update rule $Q^\pi = Q^\pi + \alpha\delta$, where $\alpha \in (0, 1]$ is a step-size parameter. Under some technical assumptions, Q-learning will converge to π_* in the limit of infinite data (Watkins & Dayan, 1992).

Multistep Backups: Though Q-learning may converge to π^* eventually, this method converges quite slowly as the estimation error δ is considered for 1-step transitions only. More clever algorithms leverage experience gathered over multiple timesteps, e.g., Tree-Backup (TB(n)) generalizes Q-learning by updating towards

$$\delta_{t:t+n} = \sum_{k=0}^{n-1} \delta_{t+k} \prod_{i=1}^{k} \gamma\pi(A_{t+i}|S_{t+i}), \tag{2}$$

which essentially computes n single-transition errors δ_{t+k} and sums them according to their policy probabilities (Sutton & Barto, 2018). If we were to utilize a greedy target policy π^*, then Tree-Backup simply sums the rewards n steps along the trace for as long as the actions are always greedy w.r.t. Q^π.

For online learning, it should be obvious that TB(n) also incurs a time-delay for updates as we need to wait for subsequent transitions. This consequence is inherent to this type of *forward view* method: at S_t we are looking n-steps ahead in time to construct $\delta_{t:t+n}$. It is also possible to express multistep updates using a *backward view* where we cast only the *current* estimation error δ_t towards our Q^π estimates for previous experiences. Consider an eligibility trace $z(s, a) \in [0, 1], \forall(s, a)$ (by default set to zero), where we keep track of past transitions by updating the recency values,

$$z(s, a) = \begin{cases} 1, & (s, a) = (S_t, A_t) \\ \gamma\lambda\pi(A_t|S_t)z(s, a), & (s, a) \neq (S_t, A_t) \end{cases} \tag{3}$$

[1]Notice that we make use of *time-slicing* for sequences according to the Python convention. For example $\tau_t \equiv \tau_{t:T:k}$ means: the sequence τ starting at t up until T with jumps of k.

where $\lambda \in [0, 1]$ is a decay parameter that governs a bias-variance trade-of. This particular update rule for $z(s, a)$ is known as the *replacing trace*. We can then update Q^π with,

$$Q^\pi(s, a) = Q^\pi(s, a) + \alpha\delta_t z(s, a), \quad \forall(s, a) \tag{4}$$

which yields the Tree-Backup(λ) algorithm for general policy learning or Watkin's $Q(\lambda)$ for learning π^* (Munos et al., 2016; Sutton & Barto, 2018).

3 HIERARCHICAL Q-LEARNING

Multistep reward algorithms considerably improve over 1-step methods, yet, policy learning can still become arbitrarily slow. For example, when no rewards are observed or when Tree-Backup prematurely truncates backups when the behaviour policy diverges from the target policy (Munos et al., 2016). Hierarchical Q-learning must handle *credit assignment* differently due to its recursive structure that decomposes the full MDP task into distinct subtasks — it must estimate returns for each task-specialized policy and handle multiple time resolutions.

Each specialized policy is updated to maximize a pseudo-reward for reaching its goal state $s \in \mathcal{S}$. Thus, instead of searching for one policy that maximizes q_π, we require a set of policies $\Pi = \{\pi_{\langle s_1 \rangle}, \pi_{\langle s_2 \rangle}, \ldots, \pi_{\langle s_{|\mathcal{S}|} \rangle}\}$ that each maximize their respective value function. We consider binary pseudo-rewards of the form $\mathbf{r}_t \equiv \mathbf{1}_{S_t}$, which is also known as the Successor Representation pseudo-reward (Dayan, 1993). Alternatively, we could write this as a vector of zeros with a one at the index corresponding to state S_t, i.e., the Kronecker delta $(\mathbf{r}_t)_i = 1_{S_i = S_t}$. As a result, we are guaranteed to observe a reward (intrinsically) at every state transition. So, when we do observe an environment (extrinsic) reward, we can leverage the knowledge contained within Π to pursue this reward.

Algorithm 1: Hierarchical training procedure.

Input: Hierarchy size k and horizons $H_i, \forall i < k$

Initialize π_i for each level $i = 0, \ldots, k-1$
Initialize S and S_{goal}
while $S \neq S_{goal}$ **do**
  $S \leftarrow$ `Recurse` $(k-1, S, S_{\text{goal}})$

Function `Recurse` *(i, S, $S^{(i)}_{goal}$)*:
  Set counter $n \leftarrow 0$
  while $n < H_i$ **and** $S \neq S^{(j)}_{goal}, \forall j, j \geq i$ **do**
    $A^{(i)} \sim \pi_i(a \mid S, S^{(i)}_{\text{goal}})$
    if $i > 0$ **then**
      $S' \leftarrow$ `Recurse` $(i-1, S, A^{(i)})$
    else
      Apply $A^{(0)}$, observe transition S', R
      Update $\Pi_i, i = 0, \ldots, k-1$
    $n \leftarrow n+1, S \leftarrow S'$
  return S

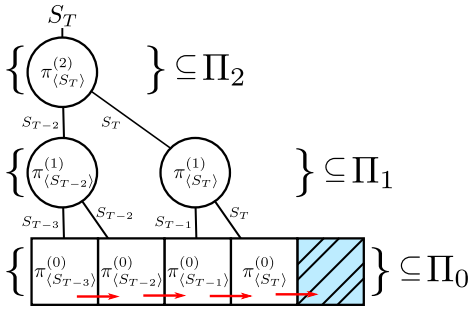

Figure 1: Example structure of the hierarchical task decomposition with $k = 3$ levels of hierarchy. Given the final goal S_T, the top level $i = 2$ sequentially samples two sub-policies at $i = 1$ that stipulate a goal-trajectory towards S_T. These sub-policies in turn perform a similar procedure, but conditioned on a different goal $\{S_{T-2}, S_T\}$ and on a more granular time-scale. Finally, the flat $i = 0$ policies take environment actions to achieve their instructed goal (red arrows).

From the structure of Π we can define policies over policies (akin to Options (Sutton et al., 1999)) in order to sample goals to reach, the agent can then stipulate a trajectory of goals that maximize the environment reward (see Figure 1). Hierarchical Q-learning does this in recursive fashion, as depicted in Figure 1 and Algorithm 1. The method creates a tree of k goal-sampling policies $\Pi_i, i = 0, \ldots, k-1$ that each instruct their lower level[2] (direct child node) to achieve some state in finite time (e.g., in H_i steps). In other words, for any $\pi_i \in \Pi_i$, its action space is $\mathcal{A}_i \subseteq \Pi_{i-1}, i > 0$, with $\mathcal{A}_0 \equiv \mathcal{A}$, and its episode horizon is H_i. The hierarchy prunes branches when nodes exceed their budget H_i or when a goal state is achieved (at any level). As a result, any level $i > 0$ is *semi-Markov* with a maximum atomic horizon (the time-span of executed actions) of $H^a_i = \prod_{j=0}^{i-1} H_j$

[2]We denote a hierarchy level i with subscripts. For overloaded indices (e.g., time), we use superscripts (i).

(Sutton et al., 1999). For $k = 1$ we collapse to a flat agent, making this formulation a recursive generalization to flat RL.

3.1 HIERARCHICAL POLICY EVALUATION

A desirable property of Algorithm 1 is that it allows us to update *all* policies (that is for each hierarchy level) after every environment step (line 13). In contrast to conventional update mechanisms (such as TB(λ)), hierarchical actions do not have to be completed when performing updates. Levy et al. (2018), showed that we can relabel hierarchical actions (in hindsight) as the states that were reached rather than those that were instructed. This philosophy leads to a dense and counterfactual learning mechanism in the sense that: if the current state had been an instructed goal, then this *would have been* a valid *hierarchical* action at previously encountered states (and optimal if the current state was a goal state).

The general idea is illustrated in Figure 2 in a *forward view* for (a, b, c) and a *backward view* in (d). Given some environment trace, the conventional way of looking at a multistep (n-step; Equation 2) backup is to consider n environment actions being applied sequentially (Figure 2a). In contrast, the hierarchical action relabelling implies that, counterfactually, a state S_t *could* have been a goal-action at each of the preceding H_i^a states— i.e., the states that lie within the hierarchical policy's atomic horizon. As an example, for $H_i^a = 2$, a 1-step update implies that the set $\{S_{t-1}, S_{t-2}\}$ contains all valid preceding states for the action S_t.

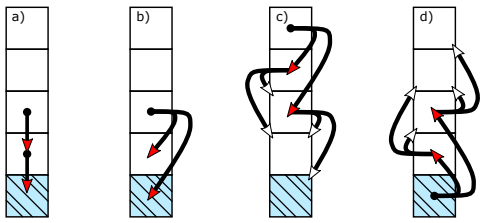

Figure 2: Comparison of a flat $n = 2$ backup at $i = 0$ (a), a hierarchical $n = 1$ and $H_1^a = 2$ backup at level $i = 1$ (b) and the set of all *possible* backups for a hierarchical $n = H_1^a = 2$ backup (c) along with its backward view (d) both at level $i = 1$. Here the white arrows indicate that the reward backup paths are inferred through partitioning of the trace (multistep) whereas the red arrows indicate states that are reachable for the hierarchy (1-step) starting from the reference state (black dots). Heuristically, we could sparsify the *white* arrows by only considering those with maximum time-span: $H_i^a = 2$.

When we consider hierarchical n-step backups through this lens, we can see that the number of backup paths grows quite swiftly (as shown for $n = 2$ in Figure 2c; now, there are 4 backup paths of length n). We can actually show that the number of all possible backup paths grows super-exponentially with respect to k, n, H_i and t (for more details, see Appendix B). Hence, for online learning, considering all possible backup paths will quickly become infeasible, however, most of these paths will show overlap. As illustrated by the backward view in Figure 2d, looking back from the goal-tile, it doesn't make much sense to perform a $n = 2$ backup from S_{T-1} to S_{T-2} when S_{T-2} lies within the H_i^a horizon, we can jump directly from S_T to S_{T-2}.

For this reason, we suggest to sparsify the multistep backups by only considering those with maximum valid length. To view the effect of this sparsification in Figure 2c,d, imagine pruning all white arrows that do not make time-jumps of length H_i^a (maximum valid time-span). This is somewhat heuristic of course, but it makes the return computation tractable and incentivizes the hierarchical policies to take actions that make maximal use of their action-budget. Naturally, if we'd consider backup paths of $H_i^a = 1$, we would end up with a flat backup and effective policy.

3.2 ALGORITHM FORMULATION

Denote k estimates for q_π at each level in the hierarchy as $Q_i \subset \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}_i|\times|\mathcal{S}|}, i = 0, \dots, k-1$. Then, observe that for a hierarchical policy $\pi_i \in \Pi_i, i > 0$ at state S_t we can look forward on the trace to find the set of reachable actions $\mathbf{A}_t^{(i)} = \{S_{t+j}\}_{j=1}^{H_i^a}$. If we extend the Q-learning update to consider the hierarchical time skips and the specialized goal policies contained in Π_i, we get:

$$\boldsymbol{\delta}_t^j = \underbrace{\mathbf{r}_{t+j} + \boldsymbol{\gamma}_{t+j}\mathbb{E}_\pi Q_i(S_{t+j}, \cdot, \mathbf{g})}_{=\mathbf{G}_t^j} - Q_i(S_t, A_{t+j-1}^{(i)}, \mathbf{g}), \tag{5}$$

where $A^{(i)}_{t+j-1} \equiv S_{t+j}, i > 0$ and $\mathbb{E}_\pi Q_i$ is short-hand for the expectation of Q_i under the target policy π. Note that $\boldsymbol{\delta}^j_t \in \mathbb{R}^{|\mathcal{S}|}$ is now a vector of errors for each policy $\pi_i \in \Pi_i$, this is made explicit by the goal-vector $\mathbf{g}$. Accordingly, the vector $\mathbf{r}_{t+j} = (1 - \boldsymbol{\gamma}_{t+j})/\gamma = \mathbf{1}_{S_{t+j}}$ is the state indicator as discussed before and bold $\boldsymbol{\gamma}$ is a termination function that ends (restarts) episodes when a S_t-specialized policy achieves S_t. If we then map the set of state-action pairs $\{S_t\} \times \mathbf{A}^{(i)}_t$ to the set of corresponding update targets $\Delta^{(i)}_t = \{\boldsymbol{\delta}^1_t, \dots, \boldsymbol{\delta}^{H^a_i}_t\}$ using a greedy target policy π^*, we get the update target for HierQ_k (Levy et al., 2018).

HierTB$_k(n)$: We can extend the single step update targets in $\Delta^{(i)}_t$ to multistep ones through the Tree-Backup operator. To reduce clutter, denote $h = H^a_i$, then we can write,

$$\boldsymbol{\delta}^j_{t:t+nh:h} = \sum_{k=0}^{n-1} \boldsymbol{\delta}^j_{t+kh} \prod_{l=1}^{k} \boldsymbol{\gamma}_{t+j+lh} \pi(A^{(i)}_{t+j+lh} | S_{t+j+lh}, \mathbf{g}) \tag{6}$$

with $A^{(i)}_{t+j+lh} \equiv S_{t+j+(l+1)h}, i > 0$, as the general HierTB$_k(n)$ error for each $\Pi_i, i = 0, \dots, k-1$ (with corrected h) and for any well-defined target policy π. For brevity, we use subscripts for $\pi_{t+j+lh} = \pi(A^{(i)}_{t+j+lh} | S_{t+j+lh}, \mathbf{g})$, where i is defined implicitly from the context.

Our sparsification can be observed from the time-jumps towards the farthest allowed state on the trace from each reference state, i.e., from S_t towards $S_{t+H^a_i}$ as indicated by the subscripts (see also Figure 2c,d for a visual reference; all white arrows with span $< H^a_i$ get pruned). Our implementation of Tree-Backup performs hierarchical updates with a similar time complexity as conventional Tree-Backup. This is a drastic improvement over any naive implementation, which would be of exponential time-complexity (see Appendix B.1).

Hier$Q_k(\lambda)$: Let us first define the eligibility trace as the matrix $Z^{(i)} \subseteq [0,1]^{|\mathcal{S}| \times |\mathcal{A}_i| \times |\mathcal{S}|}$ for each level $i = 0, \dots, k-1$. So, each column in $Z^{(i)}$ tracks an individual eligibility trace (c.f., Equation 3) for every separate policy in Π_i. Denote the set of valid preceding states $\mathbf{S}^{(i)}_t = \{S_{t-j}\}^{H^a_i}_{j=0}$ with valid action $A^{(i)}_t$, we can then write the update for $Z^{(i)}$, for each element $(s,a) \in \Omega \equiv \mathbf{S}^{(i)}_t \times \{A^{(i)}_t\}$, according to,

$$Z^{(i)}(s,a,\mathbf{g}) = \begin{cases} 1, & (s,a) \in \Omega \\ \lambda \boldsymbol{\gamma}_k \boldsymbol{\pi}_k Z^{(i)}(s,a,\mathbf{g}), & \text{Otherwise} \end{cases} \tag{7}$$

where $\boldsymbol{\pi}_k$ contains the transition (state-action) probabilities for each goal, $\boldsymbol{\gamma}_k$ terminates achieved state-goals, and $k \in \{t, t-1, \dots, t - H^a_i + 1\}$. Like before (the sparsification from hierarchical Tree-Backup) we can assume that the policy always takes actions with maximal temporal span.

It turns out, to get an exact generalization of our version of Hierarchical TB(n) to hierarchical TB(λ), we need to keep track of H^a_i disjoint eligibility matrices $Z^{(i)}_h$ for $h = 1, \dots, H^a_i$ in order to correctly track the sparsified back-up paths for each hierarchy level. We can then circulate through each eligibility matrix by only utilizing the eligibility $Z^{(i)}_h, h = t \mod H^a_i$ at timestep t. In the case of a greedy target policy π yields a hierarchical generalization of Watkin's $Q(\lambda)$, which we dub Hier$Q_k(\lambda)$. For further reference see also the provided pseudocode in Appendix E.1.

4 Empirical Evaluation

From our formulation of the hierarchical backup, along with their algorithmic implementations, we can see how hierarchical credit assignment differs from conventional (flat) credit assignment in a number of ways. The hierarchy induces skip-connections over the environment trace when computing returns. As a result, we don't need to account for correction terms at each environment transition which allows us to propagate rewards much further back in time.

It is a well known phenomenon that conventional Tree-Backup (or Watkin's $Q(\lambda)$) truncates traces too often due to possible divergence between behaviour policies and the target policy (Munos et al., 2016; Peng & Williams, 1996; Kozuno et al., 2021). Naturally, this trace truncation is for good reason from a conservative perspective: we would otherwise be estimating a different policy. A similar reasoning applies to the λ parameter for the eligibility trace and the parameter n for the

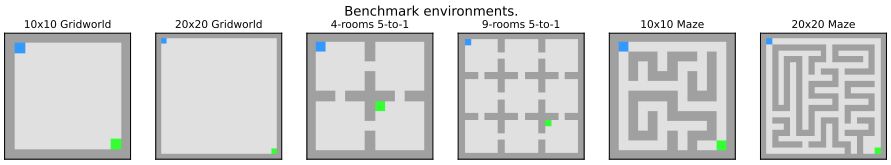

Figure 3: Experiment domain: the blue tile and green tiles are the start and end state, respectively.

multistep return, which both interpolate between having a more biased estimate of the return or one with potentially high variance. When rewards are propagated far backwards in time, the resulting update can become arbitrarily noisy and unstable.

All in all, this may raise the question whether a hierarchical backup only yields a benefit (or detriment) to the agent through deeper reward propagation (see also Appendix C for a direct comparison of a flat and hierarchical agent trained on the same trace). Hypothetically, we could achieve a similar effect of deeper reward propagation by just increasing n or γ, λ for the flat agent (not exactly of course, due to the aforementioned truncation issue). Thus, we ran experiments to analyze the difference in behaviour and performance of these agents over various backup parameters.

Experimental Setup: We evaluated the Hierarchical Tree-Backup and $Q(\lambda)$ implementations from Algorithm 2 and Algorithm 3 over two separate parameter grids on each of the discrete gridworld environments portrayed in Figure 3 (See also Appendix D for additional experiment details). Each environment was chosen to provide either a distinct structure to the state-action spaces or to their scale. We opted for a tabular setting to eliminate most other benefits posed by hierarchy (Wen et al., 2020), e.g., to eliminate possible generalization between goals or goal-directed exploration. The agents were initialized (with $Q_i = \mathbf{0}, \forall i$) at the blue-tile and could traverse the environment deterministically with the actions $\mathcal{A}^{(0)} = \{\uparrow, \downarrow, \leftarrow, \uparrow\}$. Upon reaching the green-tile, the agents received a sparse binary reward.

The first, and largest, parameter grid evaluated our algorithms over various hierarchy levels $k \in \{1, 2, 3, 4\}$, backup steps $n \in \{1, 3, 5, 8\}$, decay values $\lambda \in \{0, 0.5, 0.8, 1\}$, and behaviour policies $\{\Pi_{k-1}, \pi_0\}$ (only during training). This study aimed to quantify the marginal performance benefit of evaluating and training with additional hierarchy for various reward backup depths. The second parameter grid was evaluated for $k \in \{1, 2, 3\}$, behaviour policies $\{\Pi_{k-1}, \pi_0\}$ (only during training), and decay $\lambda = 1$, with derived parameters $\gamma = (\gamma_0)^{1/H^a_{k-1}}$ and $n = n_0/H^a_{k-1}$. These formulas were intended to adjust the discount and backup parameters such that every hierarchy level k in this ablation study sent credits back equally far. For a complete overview of all parameters and their descriptions, see Table 7 in the supplementary material. We utilized ϵ-greedy exploration with $\epsilon = 0.25$ for all flat policies $k = 1$ ($i = 0$) during training and $\epsilon = 0.05$ during evaluation. All hierarchical polices utilized a fully greedy policy $\epsilon = 0$ (uniform tie-breaking) such that exploration was mostly handled by the flat level.

This choice for hierarchical exploration is well motivated in our case seeing as uniform random exploration compounds on each level, $Pr(\text{Random-Policy}) = 1 - (1 - \epsilon)^k$. Considering the fact that hierarchical policies sample actions that carry over multiple time-steps, uniformly random exploration would result in erratic behaviour of the effective environment policy — especially as the dimensionality of the action-space increases. Of course we could have opted for a different exploration policy (e.g., Boltzmann exploration), but this would have more strongly confounded our results due to, possibly, more efficient hierarchical exploration.

All configurations were evaluated over 200 random seeds on each environment with a train-test loop of 50 iterations. Episodes were terminated when the agent exceeded a budget of 10^5 steps, this was sufficiently large to let every agent configuration observe a reward in their first episode. The performance, measured in the number of $\log(\text{steps})$ needed to reach the goal, was aggregated over all repetitions per time-step to produce loss-curves. We log-transform these loss-curves to stabilize the variance of the means, which we then averaged to quantify the marginal performance — this could be interpreted as an unnormalized area under the *mean* performance curve.

Experiment Results: The mean results (with standard errors) for the first parameter grid, conditioned on the hierarchical training policy, are visualized fully in Figure 4. Generally speaking, the

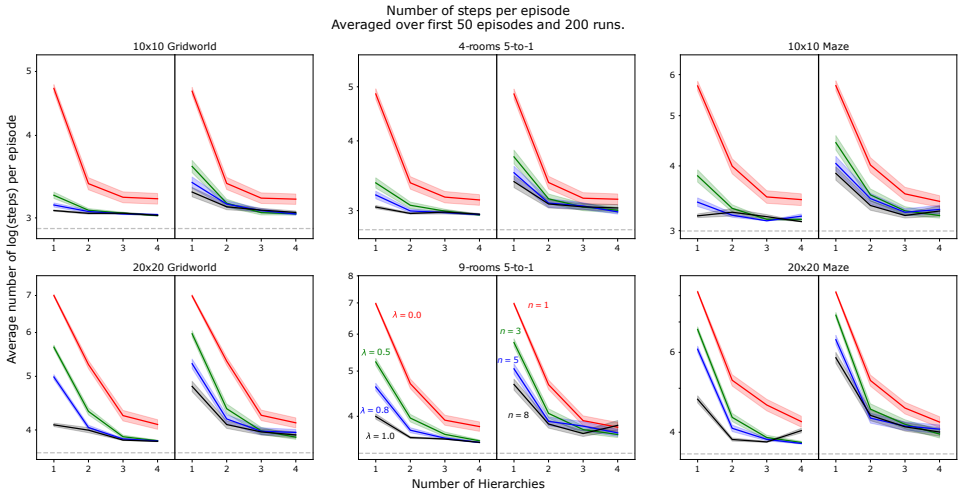

Figure 4: Marginal log-performance of each experiment configuration for each environment from Figure 3 (lower is better). All agents utilized a base-discount of $\gamma = 0.95$. The configurations are split left for the eligibility trace agents (Algorithm 3) and right for Tree-Backup agents (Algorithm 2) as indicated by the colored annotations. Shaded regions indicate ½-standard errors of the mean and grey-dashed lines indicate optimal performance per environment.

pattern indicates that hierarchy near-monotonically improves upon its flat counterpart $k = 1$ for any parameter setting (in expected log-score). Though, hierarchy generally did indeed improve upon the flat counterparts, it also exhibited much higher variance; in the first number of episodes we found that hierarchy could actually degrade performance. Regularly, the hierarchical agents would take a marginally longer time to finish their first few training episodes compared to the flat agents (see also Table 2 in the Supplementary Material). In general, we found that additional hierarchy levels prolong the first episode. However, this pattern occasionally jumps back when the hierarchy level is taken to an extreme, which was observed on the 10×10 Maze environment.

It is also noteworthy that the flat $\lambda = 1$ agents consistently performed in a competitive manner to the hierarchical policies, especially on the small environments. The environments were deterministic after all, and these agents were capable of sending rewards back the farthest of the flat policies. As also noted in Table 2, these agents generally spent the least amount of time wandering about in the first episode.

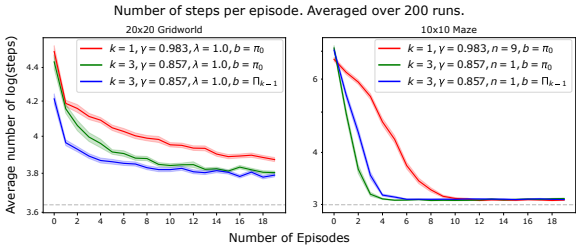

Figure 5: Average log(score) of the evaluated agents at each training episode (lower is better) when credit assignment *depth* is appropriately balanced for each hierarchy level k and when training data is generated with either a flat policy $b = \pi_0$ or with the full hierarchy $b = \Pi_{k-1}$. Data for the hierarchical agent was generated with two different policies to account for the observation that hierarchy could prolong episodes (see also Table 2). Shaded regions indicate 1-standard error of the mean.

To analyze the effect of reward backup depth on the quality of the resulting policy, it must be noted that the results in Figure 4 are confounded by the fact that the training data was generated by entirely different policies for the flat and hierarchical agents (as should be evident from Table 2). For a more fair comparison to the effect of reward backup depth, Figure 5 (see also Figures 9,10 in the Supplementary Material) visualizes the mean performance of the agents over time/ training episodes for two of the tested environments (chosen for visual clarity). Still, this result shows a similar pattern as observed in Figure 4: even with appropriately balanced backup depths and training either with or without the hierarchical structure, the positive effect on performance due to hierarchy seems to persist. This result also shows that the test scores for the

hierarchical agent trained with the *flat* behaviour policy improved marginally slower in the large Gridworld and faster in the small Maze environments. We also see that the flat agent $k = 1$ (red line) performed worse during evaluation than both hierarchical policies.

5 Discussion and Conclusion

Though our results may hint that hierarchy supersedes conventional, flat, agents, they often exhibited larger variance in the first few episodes. The first reward was not always observed as swiftly by the hierarchical agents compared to the flat agents. These results are actually an artifact of our choice of policy parameterization, ϵ-greedy for level $i = 0$ and fully greedy for $i > 0$. For the conventional agents, as long as no reward was observed the agent would do an uniform random walk. For the hierarchical agent, this was also the case for $i = k - 1$ (the top-level). However, all levels below $k - 1$ would swiftly learn to follow their instructed goals as a result of their state-specialization; they became experts in reaching the proposed goals by level $i = k - 1$ even though this agent was still random. As a result, the agent became biased towards the states for which it could quickly accumulate knowledge. The top-level could then randomly sample a state, far from the goal-state, and the levels below would naively, skillfully, follow.

This issue could of course have been handled through alternative hierarchical exploration methods, however, for a fair comparison to the conventional agent we chose to simply generate training data with a flat policy (as shown in Figure 7 in the Appendix). This again showed that hierarchy generally improves upon the flat agents, though the effect seemed to become less pronounced with additional hierarchy levels. In a way, our results should emphasize that ϵ-based exploration becomes more and more unreliable as we decrease the temporal resolution. To effectively leverage the benefits posed by hierarchy, agents should follow a more principled way for exploration (c.f., (Kulkarni et al., 2016; Bellemare et al., 2016; Ecoffet et al., 2021)).

When we again juxtapose the definitions of flat and hierarchical multistep backups, the main distinctions that we can make are: how the rewards are sent back and how the trajectory is partitioned in order to perform updates. As mentioned before, Tree-Backup and its variants compute policy correction terms over state-transitions to estimate the returns for good reason. The return would otherwise be estimating a different policy, leading to e.g., a SARSA update (Sutton & Barto, 2018). Anecdotally, initial experiments showed that SARSA (or variants thereof) always performed worse than Q-learning in our experimental setting. Thus, it is interesting that hierarchy does not degrade performance in a similar way, but rather improves upon it. Possibly, we could view hierarchical structure to make Q-learning (or any other conservative return operator) amenable to non-conservative updates — i.e., akin to SARSA or Peng's $Q(\lambda)$ (Sutton & Barto, 2018; Peng & Williams, 1996; Pertsch et al., 2020).

Of course, we should not attribute the performance benefit of hierarchy merely to the update mechanism, but also to its innate structure. The recursive decomposition induces a form of 'stitching effect' between policies that contain some knowledge about the task. However, it is not clear if this effect can be measurably distinguished from the hierarchical backups.

Finally, it must be noted that our description for Hier$Q_k(\lambda)$ does not yield a practical algorithm. Memory does not scale well in the tabular setting, especially when considering that we track $|\mathcal{S}|$ policies at each level. The same reasoning goes for the eligibility traces, which in the worst-case scales proportional to $O(H^a_{k-1}|\mathcal{S}|^2|\mathcal{A}|)$, seeing as $Z \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}^{(i)}|\times|\mathcal{S}|}$ for which we keep H^a_{k-1} copies. Our implementation adopts some heuristic optimizations in order to make our experiments tractable (see Appendix E), further optimizations such as a LIFO buffer or approximations to the eligibilities are left as future work. Our version of Hierarchical Tree-Backup should be readily applicable to hierarchical Q-learning methods even if e.g., (non-linear) function approximation is used (Levy et al., 2018).

Conclusion. This paper takes a fundamental perspective on how credit assignment is performed in Hierarchical Reinforcement Learning through a direct comparison to multistep return estimation methods (Sutton & Barto, 2018). We take as a running example the Hierarchical Q-learning method by Levy et al. (2018) to illustrate how such an algorithm may construct estimations of its returns. Our perspective showed that the number of possible ways to construct update targets for hierarchical policies grows rapidly over time but can be sparsified by focusing only on a couple of heuristically

chosen paths. We used this insight to extend the original 1-step method to more general estimation operators like Tree-Backup and $Q(\lambda)$. Our experiments show how hierarchy can yield substantial performance gains over flat RL due to improved credit assignment and policy structure.

We conclude that hierarchy opens up an additional basis for research on return estimators. Canonically, almost all developed return estimation algorithms consider the sequential Markov Chain generated by some policy. Our hierarchical formulation of Tree-Backup and $Q(\lambda)$ shows that there exists a family of return estimation methods that also take into account the number of ways to partition traces of experiences rather than purely stepping through them.

REFERENCES

Bram Bakker and Jürgen Schmidhuber. Hierarchical Reinforcement Learning Based on Subgoal Discovery and Subpolicy Specialization. In *Proceedings of the 8-th Conference on Intelligent Autonomous Systems, IAS-8*, pp. 438–445, 2004.

Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying Count-Based Exploration and Intrinsic Motivation. *arXiv:1606.01868 [cs, stat]*, November 2016. URL http://arxiv.org/abs/1606.01868. arXiv: 1606.01868.

Peter Dayan. Improving Generalization for Temporal Difference Learning: The Successor Representation. *Neural Computation*, 5(4):613–624, July 1993. ISSN 0899-7667. doi: 10.1162/neco.1993.5.4.613. Conference Name: Neural Computation.

Thomas G. Dietterich. The MAXQ Method for Hierarchical Reinforcement Learning. In *In Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 118–126. Morgan Kaufmann, 1998.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, February 2021. ISSN 1476-4687. doi: 10.1038/s41586-020-03157-9. URL https://www.nature.com/articles/s41586-020-03157-9. Number: 7847 Publisher: Nature Publishing Group.

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, Reading, 1989.

Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, May 2019. doi: 10.1126/science.aau6249. URL https://www.science.org/doi/abs/10.1126/science.aau6249. Publisher: American Association for the Advancement of Science.

Ayush Jain and Doina Precup. Eligibility Traces for Options. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pp. 1008–1016, Richland, SC, July 2018. International Foundation for Autonomous Agents and Multiagent Systems.

Tadashi Kozuno, Yunhao Tang, Mark Rowland, Rémi Munos, Steven Kapturowski, Will Dabney, Michal Valko, and David Abel. Revisiting peng's q(λ) for modern reinforcement learning. 2021. URL https://arxiv.org/abs/2103.00107v1.

Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. *arXiv:1604.06057 [cs, stat]*, May 2016. URL http://arxiv.org/abs/1604.06057. arXiv: 1604.06057.

Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning Multi-Level Hierarchies with Hindsight. September 2018. URL https://openreview.net/forum?id=ryzECoAcY7.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and Efficient Off-Policy Reinforcement Learning. *arXiv:1606.02647 [cs, stat]*, November 2016. URL http://arxiv.org/abs/1606.02647. arXiv: 1606.02647.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-Efficient Hierarchical Reinforcement Learning. *arXiv:1805.08296 [cs, stat]*, October 2018. URL http://arxiv.org/abs/1805.08296. arXiv: 1805.08296.

Jing Peng and Ronald J. Williams. Incremental multi-step q-learning. 22(1):283–290, 1996. ISSN 1573-0565. doi: 10.1007/BF00114731. URL https://doi.org/10.1007/BF00114731.

Karl Pertsch, Oleh Rybkin, Frederik Ebert, Chelsea Finn, Dinesh Jayaraman, and Sergey Levine. Long-Horizon Visual Planning with Goal-Conditioned Hierarchical Predictors. *arXiv:2006.13205 [cs, stat]*, November 2020. URL http://arxiv.org/abs/2006.13205. arXiv: 2006.13205.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, Massachusetts, 2 edition, November 2018. ISBN 978-0-262-03924-6. URL http://www.incompleteideas.net/book/the-book-2nd.html.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, August 1999. ISSN 0004-3702. doi: 10.1016/S0004-3702(99)00052-1. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal Networks for Hierarchical Reinforcement Learning. *arXiv:1703.01161 [cs]*, March 2017. URL http://arxiv.org/abs/1703.01161. arXiv: 1703.01161.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698.

Zheng Wen, Doina Precup, Morteza Ibrahimi, Andre Barreto, Benjamin Van Roy, and Satinder Singh. On Efficiency in Hierarchical Reinforcement Learning. *Advances in Neural Information Processing Systems*, 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/4a5cfa9281924139db466a8a19291aff-Abstract.html.

Xingdong Zuo. mazelab: A customizable framework to create maze and gridworld environments. https://github.com/zuoxingdong/mazelab, 2018.

A Related Work

The pursuit for Hierarchical structure in Reinforcement Learning agents has long been motivated by both philosophical considerations and promising results (Wen et al., 2020; Dietterich, 1998; Kulkarni et al., 2016; Bakker & Schmidhuber, 2004; Pertsch et al., 2020; Nachum et al., 2018) and is perhaps most well known under the Options framework of Sutton et al. (1999). Though numerous RL algorithms have been designed that leverage benefits of hierarchy, such studies often put more emphasis in getting their method to work rather than to gain a better understanding of the underlying dynamics.

Our work takes as a running example a recent hierarchical Q-learning algorithm by Levy et al. (2018), which illustrated strong performance in both discrete and continuous environments coupled with neural networks for function approximation. This method is similar to earlier work by Bakker & Schmidhuber (2004), the difference is the usage of neural networks and assuming that each observable state can be a goal state. Though their method can work well, it remained unclear what kind of benefits multiple levels of hierarchy could bring forth (as also mentioned by some of the reviewers in the OpenReview ICLR-2019 submission of Levy et al. (2018)). This unclarity can be directly answered through our interpretation of hierarchical credit assignment.

A formulation for hierarchical agents that is similar to the multistep methods we develop in this paper is by Jain & Precup (2018), who extended the option framework to general return estimation operators (such as, Retrace (Munos et al., 2016), or Tree-Backup). They do this using an intra-option framework for updating their policies. However, their method was limited to the fact that individual levels could not train independently of one another, and it remained unclear whether their hierarchical structure actually provided any substantial benefits to conventional agents.

B The Number of Backup Paths

For multistep hierarchical methods we need to consider how to look forward (or backward) on the trace in order to compute an estimate of the value function. Whereas conventional methods do this step-by-step through time, a hierarchical backup can make time 'jumps'. The backup path is thus defined by the time-deltas, suppose we have collected a trace τ_t up until time t and write the time-deltas as Δ_τ — for flat RL this is simply $\{1\}_{j=0}^{t}$. The hierarchical backup needs to consider how to construct Δ_τ such that $\sum_j (\Delta_\tau)_j = t$, this is a special case of an integer partitioning problem.

In a forward sense, the problem of computing the number of all hierarchical backup paths can also be framed as a dice rolling problem. The hierarchical environment horizon H_i^a defines the maximum value that each die can take, i.e., it defines the range of the time-deltas at any level i. Then, the backup parameter n defines the number of dice that we have. If we now assume n to be sufficiently large, and write $h = H_i^a$ for shorthand, then the number of possible hierarchical backup paths of depth t is given by,

$$\alpha_t(h, n) = \sum_{j=0}^{n} (-1)^j \binom{n}{j} \binom{t - hj - 1}{n - 1} \tag{8}$$

where $\binom{n}{k}$ is the binomial coefficient.

Proof. The quantity $\alpha_t(h, n)$ is the coefficient of the t-th monomial in the power expansion of rolling n die with ranges 1 to h, this is a result that follows from the binomial theorem (see e.g., Chapter 7 of the book by Graham et al. (1989) for a similar treatment on these types of combinatorial problems). This sequence of coefficients is captured by the ordinary generating function (o.g.f.),

$$\begin{aligned} F(x) &= (x^1 + x^2 + \cdots + x^h)^n, \\ &= x^n \left(\textstyle\sum_{j=0}^{h-1} x^j \right)^n \end{aligned} \tag{9}$$

This can be rewritten using the closed-form solution to the finite geometric series,

$$\begin{aligned} F(x) &= x^n \left(\frac{1 - x^h}{1 - x} \right)^n \\ &= x^n (1 - x^h)^n (1 - x)^{-n}. \end{aligned}$$

Written as a binomial series, this yields,

$$F(x) = x^n \sum_{j=0}^{\infty} \binom{n}{j} (-x^h)^j \sum_{k=0}^{\infty} \binom{-n}{k} (-x)^k$$
$$= x^n \sum_{j=0}^{\infty} (-1)^j \binom{n}{j} x^{hj} \sum_{k=0}^{\infty} (-1)^k \binom{-n}{k} x^k,$$

Then see that, $\binom{-n}{k} = (-1)^k \binom{k+n-1}{k}$, which we can use to cancel out the other power of -1 (due to the even power $2k$) to give us,

$$F(x) = x^n \sum_{j=0}^{\infty} (-1)^j \binom{n}{j} x^{hj} \sum_{k=0}^{\infty} \binom{n+k-1}{k} x^k.$$

To extract the coefficient of the monomial with power t from this o.g.f., note that that the powers in $F(x)$ being x^n, x^{hj}, x^k should together sum to t. In other words, $t = n + hj + k$. If we then also note that the first summation in $F(x)$ sums infinitely many zeros after $j > n$ because $\binom{n}{k} = 0$ for $k > n$. Then, to get the coefficient α_t for the t-th power in $F(x)$ we can omit the second summation in $F(x)$ and simply substitute for $k = t - n - hj$, which yields,

$$\alpha_t x^t = x^n \sum_{j=0}^{n} (-1)^j \binom{n}{j} x^{hj} \binom{n+k-1}{k} x^{t-n-hj},$$
$$= x^t \sum_{j=0}^{n} (-1)^j \binom{n}{j} \binom{n + (t-n-hj) - 1}{t-n-hj}.$$

Then applying the identity $\binom{n}{k} = \binom{n}{n-k}$ and dividing the above expression by x^t we get the solution in Equation 8. □

The quantity from Equation 8 is of course for a fixed backup depth, because h can be varied between $1, \ldots, h$ there are multiple possible backup depths. When we use a smaller n when computing the multistep returns, then the backup paths get truncated after n steps. Then, it follows trivially that the shortest backup path is bounded by n whereas the longest backup path is bounded by hn. We can write the total number of possible paths to compute returns on as,

$$S = \sum_{d=n}^{nh} \alpha_d(h, n). \tag{10}$$

This of course yields $S = h^n$. The number of hierarchical backup paths grows at least exponentially.

With that said, we should also keep in mind that the atomic horizon $h \equiv H_i^a$ too grows exponentially with increasing levels of hierarchy according to the hierarchical action budget, $H_i^a = \prod_{j=0}^{i-1} H_j$. For example, a hierarchy with $k = 3$ levels and a policy budget of $H_i = 4$ for all levels i gives $H_{k-1}^a = 16$. If we then want to do a $n = 4$ multistep update, then in total there are approximately $\sum_{d=n}^{nh} \alpha_d(16, 4) = 16^4 \approx 65 \cdot 10^3$ possible updates. Hence, the number of backup paths actually grows super-exponentially w.r.t., n, H_i and k. This is not practical for any algorithm.

B.1 Number of Tree-Backup Paths

For our formulation of hierarchical Tree-Backup (Section 3.2), we provide the backup parameter n and horizon h when computing multistep returns. Thus, to compute all possible Tree-Backup paths, we can simply fall back to Equation 10. However, this does not take into account the severe overlap between backup paths, the truncation of the traces due to the target policy probabilities, and our proposed sparsification.

In fact, our sparsification of the multistep returns are able to reduce the quantity S for any n and h to an upper bound of just $S_{TB} = nh$ backup paths. Along with some implementation optimizations, this can be implemented in $O(n)$ time which is equivalent in time-complexity to conventional (flat) Tree-Backup (as illustrated in Algorithm 2).

C EXAMPLE HIERARCHICAL POLICY

To provide a better illustration to the joint effect of hierarchy on the effective environment policy, in Figure 6 we draw a $k = 2$ hierarchical and $k = 1$ flat policy (both conditioned on the end-goal $\langle S_T \rangle$) after one training update with a shared trace. The decomposition that the hierarchy induces was portrayed earlier in Figure 1, however, that structure centered around the trace whereas Figure 6 illustrates this in the actual environment. In the left side of Figure 6, we see that the agent moves from the blue to the green tile in a slightly sub-optimal way, it loops before the corridor, and takes a detour before reaching S_T. After encountering S_T, the flat and hierarchical policy are both updated in parallel with backup parameters $\lambda = 1.0, \gamma = 0.95$ and $H_i^a = 3$.

The $k = 1$ flat policy exhibits a familiar pattern as indicated in the right side of Figure 6, its newly updated policy greedily follows along the previously traversed path (greedily by omitting the loop before the corridor). In contrast, the hierarchy seems to sample a hierarchical action that would send the lower level $i = 0$ towards the loop-tile before the corridor (the one that the flat agent now omitted). Of course, this possible issue can be avoided by simply greedily re-sampling a new goal after each transition. The reason that the hierarchy sends the flat agent to this loop is actually an artifact of partitioning the hierarchical backup according to the maximal horizon $H_i^a = 3$, this can be seen in the number of colored squares (left-upper aligned in each tile) that point towards the colored circles (right-upper aligned in each tile). Each hierarchical action $A^{(1)}$ is repeated on three distinct tiles. The hierarchy provides a benefit to the flat agent in that it can jump from the left-tile next to S_T straight to S_T, without the detour that the trace made. Surely, the flat agent would still follow the detour when instructed with S_T, but this only shows that the hierarchical agent can look further forwards on the trace during learning.

Thus, when the hierarchy greedily resamples goals when better ones are encountered, the hierarchical agent fundamentally supersedes the flat agent in this example. The greedy hierarchy omits the loop before the corridor, just like the flat agent. However, the hierarchy can jump over the tile left to S_T, straight to S_T whereas the flat policy is forced to follow its previous tracks. This may indicate that it may be smart for the lower level policy to 'forget' slightly in this environment, such that when S_T is instructed that the agent does not greedily follow its previous trajectory. Naturally, this is closely tied to the tuning of the backup parameters γ, λ, n, and H_i^a and the bias-variance trade off.

D FURTHER EXPERIMENT DETAILS

This section enumerates all further experimental details that did not fit in the main narrative. For an overview of all hyperparameters and the domain we considered for our ablations, see Table 1.

For all our main experiments we utilized a greedy hierarchy during training $\epsilon = 0$, but we also greedily terminated and resampled goals for the lower levels to follow *during evaluation time*. So sub-optimal goals could be sampled and followed during training, but were resampled greedily according to Q_i during test-time. Furthermore, for all eligibility trace based agents, we truncated/ cut the traces when the eligibility value would shrink to $z < 10^{-8}$ to reduce computation time (see Appendix E for further reference on our implementation). We also did not allow hierarchical policies to sample a state as goal that they were already positioned at $\pi_i(S_t|S_t, \mathbf{g}) = 0, \forall i > 0$. If the instructed goal state of a hierarchical policy i was within reach of the agent, than that policy would directly sample the goal (if the agent sees the goal, then it can greedily sample the goal). We utilized uniformly random tie breaking in case of shared Q values for various state-action pairs, at every level $i = 0, \ldots, k-1$.

All experiments were run on an AMD Ryzen 9 5900x CPU which took approximately a 1-2 days of compute when running multiple ablations in parallel. Our framework was built on Python 3.9.7 with Numpy 1.20.3, OpenAI Gym 0.19.0, and we utilized environment examples from the public MazeLab library by Zuo (2018). Code for implementation and reproduction of our experiments is available at: `https://github.com/joeryjoery/HierQ/tree/main`, see also the supplementary material for the experiment data used for visualization.

Table 1: Table of parameter values (and ablation set) along with a short description. This table does not display every involved variable; e.g., the parameterization of the policies or the tie-breaking method in case of shared Q-values.

Parameter	Set of Values	Description
α	1	Step-size/ learning-rate for updating value estimates (Q-tables) during learning (see Section 2).
H_i	3	The number of *hierarchical* steps before termination of the policy/ pruning this policy-branch. E.g., if policy $H_0 = 3$, then policy π_0 is terminated if it hasn't reached its goal in 3 steps. This value was kept constant at every level $i = 0, \ldots, k-1$ in the hierarchy.
$b : \mathcal{S} \times \mathcal{A} \to [0, 1]$	$\{\Pi_{k-1}, \pi_0\}$	Behaviour policy for generating training data. We used either the full hierarchy, or just the flat policy by fixing $A^{(i)} = S_{\text{goal}}, \forall i > 0$.
ϵ_0	$0.05, 0.25$	Exploration probability of the level $i = 0$ policy during evaluation (0.05) and during training (0.25).
$\epsilon_{i>0}$	0.0	Exploration probability of the level $i > 0$ policies (greedy).
$t_{\max}$	10^5	Number of allowed steps within one training episode, after which the agent is reset to the initial state. This value is kept large enough to let the agent almost always observe a reward.
Main Ablation Study		
λ	$\{0, 0.5, 0.8, 1.0\}$	Decay parameter for the eligibility trace (c.f., Equation 3).
n	$\{1, 3, 5, 8\}$	Number of steps to perform Tree-Backup on for computing multistep returns (c.f., Equation 2).
$\gamma \equiv \gamma_0$	0.95	Base discount factor for the cumulative returns (c.f. Equation 1).
k	$\{1, 2, 3, 4\}$	Number of hierarchy levels to define policies over policies (to sample states as goals/ actions).
Backup Depth Study		
λ	1.0	See Above.
n	$\{9, 3, 1\}$	Number of adjusted Tree-Backup steps for the multistep returns such that each hierarchy level k in the ablation study sent rewards back equally far (see definition of the atomic horizon H_i^a in Section 3).
γ	$\{0.983, 0.95, 0.857\}$	Base discount factors adjusted proportional to the propagation depth of credits/ rewards, these were calculated as $(\gamma_0)^{1/H^a_{k-1}}$ (see Above).
k	$\{1, 2, 3\}$	See Above.

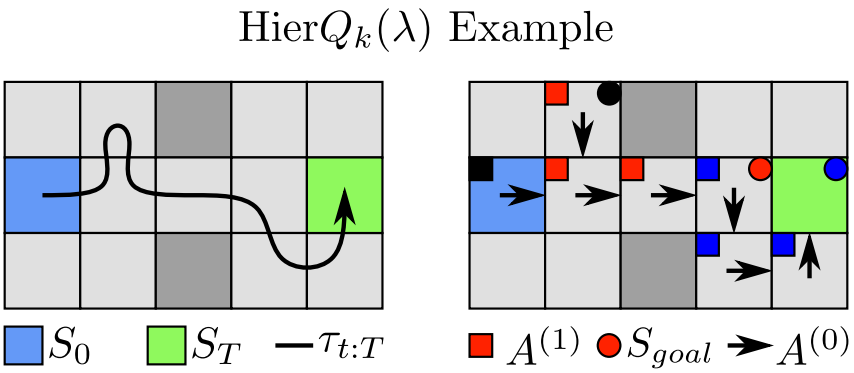

Figure 6: Greedy policy comparison of a flat (arrows) and hierarchical agent (boxes-to-circles) on an example environment trace with $\lambda = 1.0$. The hierarchical agent is instructing the flat agent to take two more environment steps to reach the green tile (starting from the center in the left room) if the agent were to strictly adhere to every subgoal — i.e., without intermediate termination. Also, the flat agent aims to follow its previously traversed path, only eliminating the small loop, whereas the hierarchical policy wants to 'jump' over these redundant actions (as shown by the blue boxes-to-circle).

E IMPLEMENTATION NOTES

This section discusses implementation specific details and considerations. The next subsection describes important points about our hierarchical Q-learning algorithm and HierTB$_k(n)$. The subsections afterward discuss our policy representation and choice for reward function. We provide an additional experimental result/ ablation there to illustrate and back our choice for these particular implementations.

E.1 ALGORITHM DETAILS

In Algorithm 3 one may have noticed that we can send the rewards back in a form of 'semi-backward' view in order to update 1-step state-action pairs (line 9). It turns out that for all our hierarchical return operators, we don't need to wait for the entire set $\Delta_t^{(i)}$ to be computed before being able to update Q-estimates. In fact, we can perform the hierarchical update by only computing $\mathbf{G}_t^1$ (Equation 5) and updating the Q-estimates towards this return value for all state-action pairs in $\{S_{t-j+1}\}_{j=1}^{H_i^a} \times \{A_t^{(i)}\}$. This mechanism is 'semi-backward' in the sense that $\mathbf{S}_t^{(i)} = \{S_{t-j+1}\}_{j=1}^{H_i^a}$ contains all *past* states for which $A_t^{(i)}$ is a reachable action. The motivation behind this particular mechanism can also be seen by noting that $\mathbf{G}_t^1 \equiv \mathbf{G}_{t-j}^j, \forall j > 0$; there is exact overlap between returns within a rolling window of states.

We also adopted this efficient semi-backward view for our Hierarchical Tree-Backup implementation, which is shown in Algorithm 2. The algorithm is of almost the exact same form as conventional Tree-Backup (Sutton & Barto, 2018), but differs in the fact that: at each backup-step we perform 'jumps' of length H_i^a in time, and given the return $\mathbf{G}$ we update multiple state-action pairs (inferred from the trailing states $\mathbf{S}_{t_n}^{(i)}$.

Like conventional Tree-Backup, our algorithm incurs a time-delay proportional to n before being able to compute update targets. This is exacerbated by the hierarchy horizon H_i^a, we need to wait nH_i^a time-steps before being able to compute the multistep return. Despite this, we can still compute returns/ update targets at each timestep due to the observed pseudo-rewards. This allows us to continually update all goal-specialized levels $\Pi_i, i = 0, \dots, k-1$. As mentioned before and as illustrated by Algorithm 2, due to our sparsification assumption and the overlap between hierarchical returns at different timesteps, $\mathbf{G}_t^1 \equiv \mathbf{G}_{t-j}^j$ (Equation 5), we only have to backup along a single path to update the Q-values of the elements in $\mathbf{S}_t^{(i)} \times \{A_t^{(i)}\}$.

The hierarchical $Q(\lambda)$ procedure is illustrated in Algorithm 3. In the pseudocode, we keep the 1-step updates separate from the eligibility trace update initially (line 9; Algorithm 3), seeing as each element $(s, a) \in \Omega$ induces a different error $\boldsymbol{\delta}$ even though they share the same returns $\mathbf{G}$. Of course, this is only relevant for the 1-step errors, after which these trailing pairs are simply added to the eligibility trace (line 8; Algorithm 3).

Algorithm 3: Hierarchical $Q(\lambda)$/ Hier$Q_k(\lambda)$

1 **Input :** Environment trace $\tau_{t+1} = \{S_j, A_j\}_{j=0}^{t+1}$, the hierarchy level i, eligibilities $Z^{(i)}$, a target policy π, and Q-table $Q_i(s, a, g)$
2 **Parameters :** Discount factor $\gamma \in [0, 1)$, step-size $\alpha \in (0, 1]$, decay $\lambda \in [0, 1]$, and policy reach H^a
3 Let $h \leftarrow t \mod H^a$, $t_{\min} \leftarrow \min(H^a - 1, t)$
4 $A \leftarrow A^{(0)}$ or S_{t+1} if $i > 0$
5 $\mathbf{G} \leftarrow \mathbf{r}_{t+1} + \boldsymbol{\gamma}_{t+1} \mathbb{E}_\pi Q_i(S_{t+1}, \cdot, \mathbf{g})$
6 $\boldsymbol{\delta} \leftarrow \mathbf{G} - Q_i(S_{t-t_{\min}}, A, \mathbf{g})$
7 $Z_h^{(i)}(s, a, \mathbf{g}) \leftarrow \lambda \boldsymbol{\gamma}_{t-t_{\min}} \boldsymbol{\pi}_{t-t_{\min}} Z_h^{(i)}(s, a, \mathbf{g})$
8 $Q_i(s, a, \mathbf{g}) \leftarrow Q_i(s, a, \mathbf{g}) + \alpha \boldsymbol{\delta} Z_h^{(i)}(s, a, \mathbf{g})$,
9 **for** $j = 0, \dots, \min(H^a - 1, t_{\min})$ **do**
10 $\quad Z_h^{(i)}(S_{t_{\min}-j}, A, \mathbf{g}) \leftarrow 1$
11 $\quad Q_i(S_{t_{\min}-j}, A, \mathbf{g}) \leftarrow (1 - \alpha) Q_i(S_{t_{\min}-j}, A, \mathbf{g}) + \alpha \mathbf{G}$

Algorithm 2: Tree-Backup for Hierarchical Q-learning.

Input: Environment trace $\tau_{t+1} = \{S_j, A_j\}_{j=0}^{t+1}$, the hierarchy level i, a target policy π, and Q-table $Q_i(s, a, g)$

Parameter: Discount factor $\gamma \in [0, 1)$, step-size $\alpha \in (0, 1]$, backup depth $n \in \mathbb{N}$, and policy reach H^a

$t_n \leftarrow t - H^a(n-1)$ {Sweep n to 1 if $t = T$ }

if $t_n \geq 0$ **then**

  $\mathbf{G} \leftarrow \mathbf{r}_{t+1} + \boldsymbol{\gamma}_{t+1}\mathbb{E}_\pi Q_i(S_{t+1}, \cdot, \mathbf{g})$

  for $k = t - H^a + 1 \geq t_n$ *with steps of* $-H^a$ **do**

    $A \leftarrow S_{k+H^a}$ **if** $i > 0$ **else** $A \leftarrow A_k$

    $\mathbf{G} \leftarrow \mathbf{r}_k + \boldsymbol{\gamma}_k(\boldsymbol{\pi}_k\mathbf{G} + (1 - \boldsymbol{\pi}_k)\mathbb{E}_\pi Q_i(S_k, \cdot, \mathbf{g}))$

  $A \leftarrow S_{1+t_n}$ **if** $i > 0$ **else** $A \leftarrow A_{t_n}$

  for $j = 0, 1, \ldots, \min(H^a - 1, t_n)$ **do**

    $Q_i(S_{t_n-j}, A, \mathbf{g}) \leftarrow (1 - \alpha)Q_i(S_{t_n-j}, A, \mathbf{g}) + \alpha\mathbf{G}$

ELIGIBILITY TRACES

At a first glance, it might seem that an eligibility trace would make the extension for 1-step hierarchical updates to multistep methods even simpler opposed to Tree-Backup. With eligibility traces we can just update recency values within the trace and cast a 1-step error backwards (see Equation 7). However, the hierarchical structure actually makes the extension to Tree-Backup(λ) quite intricate.

As alluded to in the main paper (Section 3.2), there are numerous subtleties in extending our version of Tree-Backup to a full backward method. Since every level in the hierarchy estimates the returns at different temporal resolutions, this means that we need to track k separate eligibilities — in fact, any level i can be interpreted to estimate the return up to a $(\gamma)^{1/H_i^a}$ diluted discount (and/ or decay rate). Moreover, in order to correctly update all policies $\pi \in \Pi_i$, at every level i, we need to account for the policy correction terms at each transition for each goal-specialized policy. For example, an action A_t can be optimal at S_t for reaching goal S' but be suboptimal for reaching another goal S''. Hence, why we argued for utilizing $|\mathcal{S}|$ separate eligibility traces at each separate level i.

Another interesting implication was that we had to use another H_i^a separate eligibility traces for each temporal horizon. This is an artifact of our heuristic choice of dealing with the exorbitant number of possible backup paths by making fixed time-jumps of H_i^a steps. Suppose we applied the update in Equation 7 at every environment step, then it is easy to see that our sparse structure gets violated: we do not only add new paths to the previously added ones, but we also decay all previous path exponentially fast due to the repeated multiplication with $(\gamma\lambda)$. For example, if we added a state-action pair of time-length H_i^a in the previous time-step, and in the next time-step we added the state-action pairs for all trailing states, we have a backup-path that connects through a H_i^a-time jump to an intermediate 1-step path. From Algorithm 2 it should be obvious that intermediate time-jumps should always be H_i^a.

E.2 POLICY REPRESENTATION AND MEMORY

As explained in Section 3 each level within the hierarchy was parameterized as $\mathcal{A}_i \subseteq \Pi_{i-1}, i > 0$. The action space of the hierarchical levels was the set of goal-specialized polices at the level below. This is quite trivial to implement simply by using the *full* set of polices $\mathcal{A}_i \equiv \Pi_{i-1}, i > 0$ such that the agent can figure out itself which goals are reachable (which actions are viable) through the backed up rewards. Seeing as we utilized fully greedy hierarchical policies, after observing the first reward, the agent would always sample valid state-actions.

We restricted this set of policies by defining $\mathcal{A}$ to be a function over the power set of the state-space (in terms of goal-policies), $\mathcal{A}_i : \mathcal{S} \to \mathcal{P}(\Pi_{i-1})$ such that each $\pi \in \mathcal{A}_i(S_t)$ could actually reach its goal within H_i^a steps. In other words, we restricted the action set of each hierarchical policy at every state such that any sampled policy's goal state lied within a H_i^a neighborhood of the current state in terms of the state-space (we utilized l_1 neighborhoods). This makes sense because any policy that the

hierarchical level could sample that lied outside this neighborhood would be terminated prematurely due to the hierarchical action budgets H_i. Of course, this is an *ad-hoc* detail, and is not applicable in every domain; this either requires knowledge of the transition function or an assumption on the environment's geometry. Due to our tabular domain, this assumption was necessary to cut down on the memory usage.

Another memory optimization is noting that we do not utilize *all* policies in the top-level Π_{k-1}, only the policy conditioned on the environment reward. This cut down the memory complexity from $O(|\mathcal{S}|^2 \times |\mathcal{A}_{k-1}|)$ to $O(|\mathcal{S}| \times |\mathcal{A}_{k-1}|)$ for the top-level. Combined with the restricted action-spaces at the lower levels $i < k-1$, this provided a tractable implementation for the larger domains (e.g., 20×20 Gridworld).

E.3 Reward Function and Discounting

A common theme in our main paper was whether hierarchy simply adds depth to the reward estimation similarly to flat multistep methods. However, we argued that hierarchy creates skip-connections over time whereas conventional methods simply step-through the trace. An important factor that balanced the depth of the return in either case, was the discount factor γ. Due to our choice of binary pseudo-rewards, i.e., the Successor Representation (Dayan, 1993), after sufficient time the rewards will get diluted due to the exponential discounting. For our experiments/ ablations it may have made sense to utilize another reward function, such that γ could be set to $\gamma = 1$. A simple alternative is $\mathbf{r}'_t = 1 - \mathbf{r}_t$, i.e., a penalizing reward of -1 at each time-step and 0 when the goal is observed.

Though this choice for reward function eliminates one confounder γ, it introduces another one. The penalizing reward has a similar effect to count-based exploration methods. During training episodes, states will get penalized continuously until goals are observed, this has the effect that the agent will steer away from frequently visited states if it doesn't observe rewards for said goals. Our choice of the binary rewards ensured that the agents would stay uniformly random until actual rewards were observed.

E.4 Policy and Reward Ablation

Our choice for the relative policy parameterization and binary pseudo-rewards is motivated by our additional ablation results in Figure 8. These results were generated with the *exact* same experimental setup as discussed before in Section 4. However, here we performed ablations over the choice of reward function (binary Vs. penalizing) and the policy parameterization (restricted Vs. unrestricted) only for the 1-step Hierarchical Q-learning algorithm. In a sense, this can be seen as a direct comparison to the original algorithm by Levy et al. (2018) and our adaptation. All-in-all, these results simply strengthen our implementation considerations that we used to generate our main results.

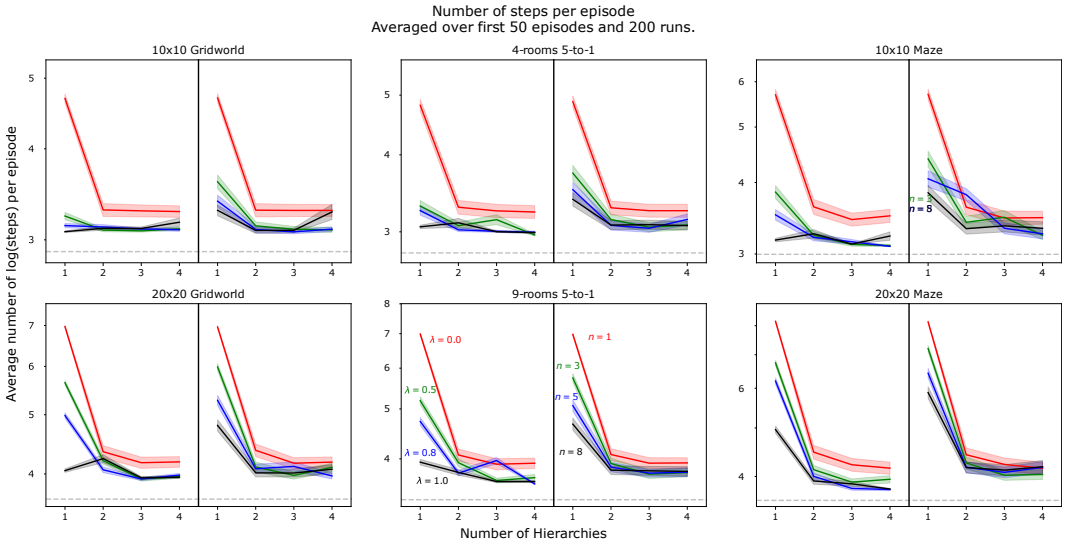

Figure 7: Marginal log-performance of each experiment configuration for each environment from Figure 3 (lower is better). This figure is identical to the main paper (including the y-scale; c.f., Figure 4), however this data portrays the agent's performance when training proceeded purely with the flat policy (i.e., by keeping the hierarchy fixed during training: $A^{(i)} = S_{\text{goal}}, \forall i > 0$) and evaluation with the hierarchical policy.

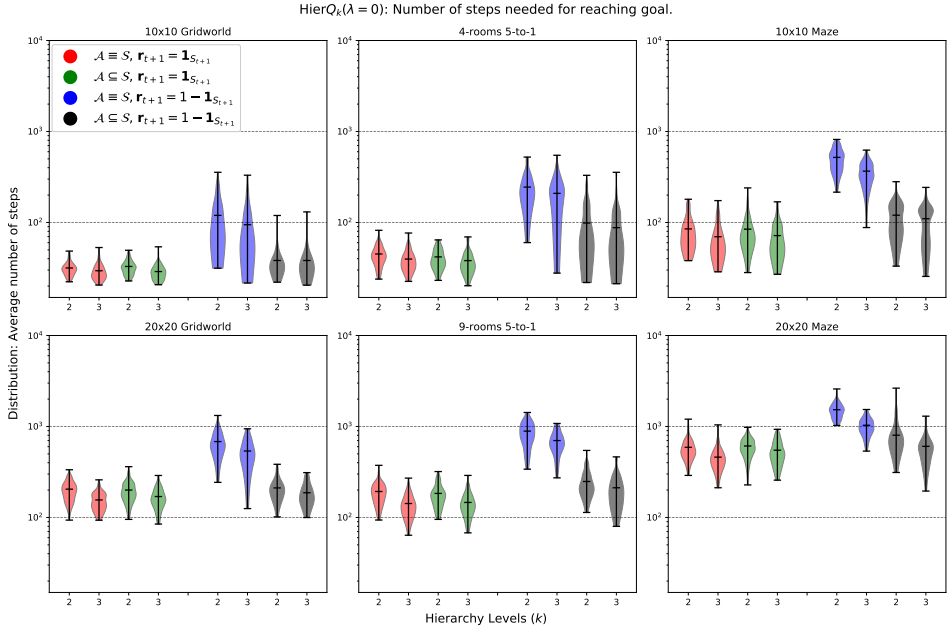

Figure 8: Comparison of our implementation for Hier$Q_k(\lambda = 0)$ to that of Levy et al. (2018). The x-axis shows the number of hierarchy levels k, split by the reward function and as indicated in the legend. The y-axis shows the aggregate number of steps to reach the environment goal of the greedy policy over the first 50 training episodes, the violinplots illustrate their distribution over 200 random seeds.

Table 2: Table of all first episode reward statistics from the main ablation study, i.e., the recorded number of environment steps in the first training episode. The cell values contain the mean and standard errors $\mu \pm s^{-1}$ rounded to integers. These means can also be interpreted as the scale parameter β for an exponential distribution $\text{Exp}(\beta^{-1})$, though this data is best captured by a scaled Poisson (or Gamma). The lowest means are emphasized with bold fonts whereas the highest means are emphasized with italics, see also the supplementary material for further visualizations.

Hier$Q_k(\lambda)$		10x10 Gridworld	20x20 Gridworld	4-rooms 5-to-1	9-rooms 5-to-1	10x10 Maze	20x20 Maze
$\lambda = 0$	$k = 1$	$\mathbf{561 \pm 37}$	$\mathbf{2746 \pm 178}$	$\mathbf{931 \pm 56}$	$\mathbf{2556 \pm 142}$	2149 ± 173	9215 ± 599
	$k = 2$	834 ± 63	5188 ± 380	1135 ± 66	3949 ± 230	*5418 ± 734*	*14312 ± 982*
	$k = 3$	857 ± 64	*6996 ± 529*	*1293 ± 69*	*4178 ± 277*	2497 ± 209	12388 ± 791
	$k = 4$	*950 ± 71*	6704 ± 548	1200 ± 70	4035 ± 238	$\mathbf{1471 \pm 127}$	$\mathbf{10665 \pm 688}$
$\lambda = 0.5$	$k = 1$	$\mathbf{626 \pm 35}$	$\mathbf{3168 \pm 179}$	$\mathbf{1024 \pm 59}$	$\mathbf{2541 \pm 125}$	2797 ± 215	$\mathbf{9080 \pm 629}$
	$k = 2$	1043 ± 83	6014 ± 478	1222 ± 74	*4136 ± 239*	*4805 ± 564*	*14028 ± 908*
	$k = 3$	990 ± 70	8244 ± 590	1273 ± 91	4109 ± 283	2860 ± 262	11647 ± 784
	$k = 4$	*1245 ± 90*	*8770 ± 727*	*1339 ± 77*	3928 ± 270	$\mathbf{1308 \pm 95}$	10580 ± 643
$\lambda = 0.8$	$k = 1$	$\mathbf{629 \pm 36}$	$\mathbf{3300 \pm 189}$	$\mathbf{941 \pm 54}$	$\mathbf{2792 \pm 151}$	2525 ± 187	10384 ± 684
	$k = 2$	1130 ± 78	6528 ± 436	1308 ± 73	3717 ± 213	*6318 ± 665*	13167 ± 867
	$k = 3$	1134 ± 76	8254 ± 663	*1400 ± 99*	*4655 ± 303*	2914 ± 267	*14865 ± 944*
	$k = 4$	*1315 ± 105*	*8406 ± 788*	1331 ± 71	4492 ± 287	$\mathbf{1663 \pm 143}$	$\mathbf{9854 \pm 633}$
$\lambda = 1$	$k = 1$	$\mathbf{562 \pm 31}$	$\mathbf{3023 \pm 216}$	$\mathbf{838 \pm 46}$	$\mathbf{2937 \pm 162}$	2236 ± 189	$\mathbf{8922 \pm 574}$
	$k = 2$	1069 ± 76	6586 ± 504	1255 ± 72	3532 ± 198	*5617 ± 600*	*13826 ± 829*
	$k = 3$	1010 ± 70	*10385 ± 819*	1339 ± 76	*4102 ± 240*	2965 ± 260	11590 ± 711
	$k = 4$	*1380 ± 104*	8643 ± 766	*1370 ± 81*	3933 ± 235	$\mathbf{1785 \pm 151}$	10923 ± 707
HierTB$_k(n)$							
$n = 1$	$k = 1$	$\mathbf{637 \pm 43}$	$\mathbf{3251 \pm 221}$	$\mathbf{895 \pm 52}$	$\mathbf{2960 \pm 190}$	2255 ± 155	$\mathbf{9718 \pm 651}$
	$k = 2$	859 ± 70	5518 ± 368	1196 ± 80	3568 ± 219	*4464 ± 614*	*14190 ± 1040*
	$k = 3$	*959 ± 71*	6417 ± 526	*1297 ± 69*	*4658 ± 279*	2469 ± 199	12969 ± 832
	$k = 4$	895 ± 72	*6468 ± 605*	1251 ± 74	4244 ± 267	$\mathbf{1490 \pm 105}$	9919 ± 585
$n = 3$	$k = 1$	$\mathbf{698 \pm 44}$	$\mathbf{3064 \pm 210}$	$\mathbf{972 \pm 58}$	$\mathbf{3183 \pm 193}$	2493 ± 171	$\mathbf{9028 \pm 555}$
	$k = 2$	1069 ± 87	5176 ± 394	1174 ± 75	3627 ± 206	*5000 ± 555*	*12388 ± 936*
	$k = 3$	994 ± 76	*8010 ± 600*	*1310 ± 77*	*4776 ± 332*	2573 ± 247	11168 ± 748
	$k = 4$	*1163 ± 78*	6563 ± 628	1262 ± 71	3880 ± 279	$\mathbf{1681 \pm 143}$	9549 ± 584
$n = 5$	$k = 1$	$\mathbf{519 \pm 33}$	$\mathbf{3023 \pm 186}$	$\mathbf{946 \pm 55}$	$\mathbf{2873 \pm 154}$	2514 ± 185	$\mathbf{9440 \pm 644}$
	$k = 2$	1050 ± 86	6077 ± 480	1090 ± 63	3723 ± 219	*4097 ± 420*	*13469 ± 884*
	$k = 3$	869 ± 69	8179 ± 649	1234 ± 78	*4589 ± 385*	2824 ± 251	10385 ± 637
	$k = 4$	*1141 ± 91*	*8313 ± 731*	*1292 ± 78*	4236 ± 236	$\mathbf{1599 \pm 146}$	9846 ± 634
$n = 8$	$k = 1$	$\mathbf{632 \pm 36}$	$\mathbf{3139 \pm 191}$	$\mathbf{1064 \pm 61}$	$\mathbf{2789 \pm 161}$	2479 ± 193	$\mathbf{8984 \pm 597}$
	$k = 2$	968 ± 74	6094 ± 404	1131 ± 68	3591 ± 218	*3707 ± 439*	*14032 ± 933*
	$k = 3$	1030 ± 74	7691 ± 557	1271 ± 74	3891 ± 233	2900 ± 256	11853 ± 753
	$k = 4$	*1140 ± 80*	*8691 ± 830*	*1341 ± 78*	*4310 ± 243*	$\mathbf{1551 \pm 146}$	9077 ± 559

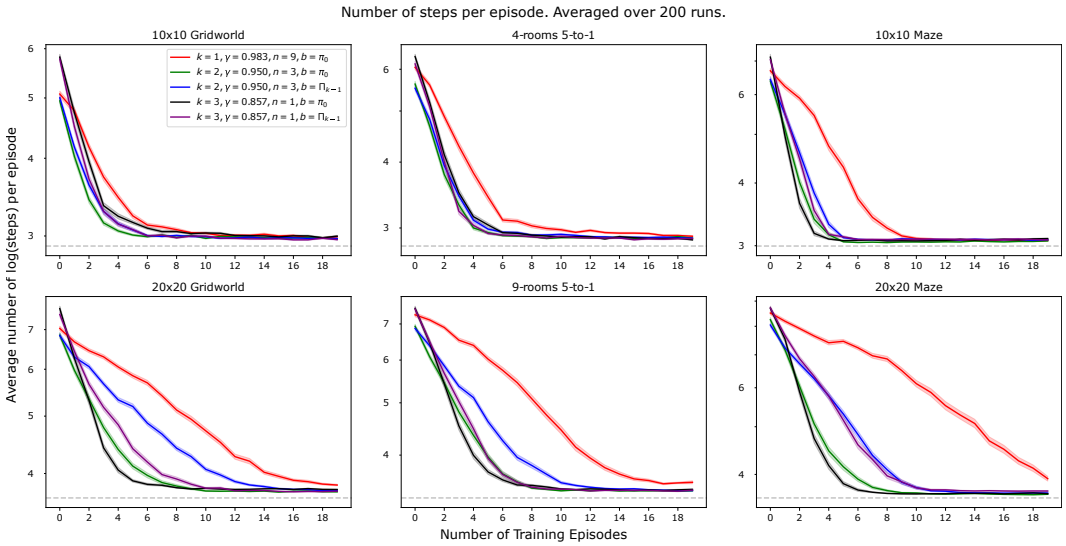

Figure 9: Average $\log$(score) of the evaluated hierarchical Tree-Backup agents at each training episode (lower is better) when credit assignment *depth* is appropriately balanced for each hierarchy level k. Shaded regions indicate 1-standard error of the mean.

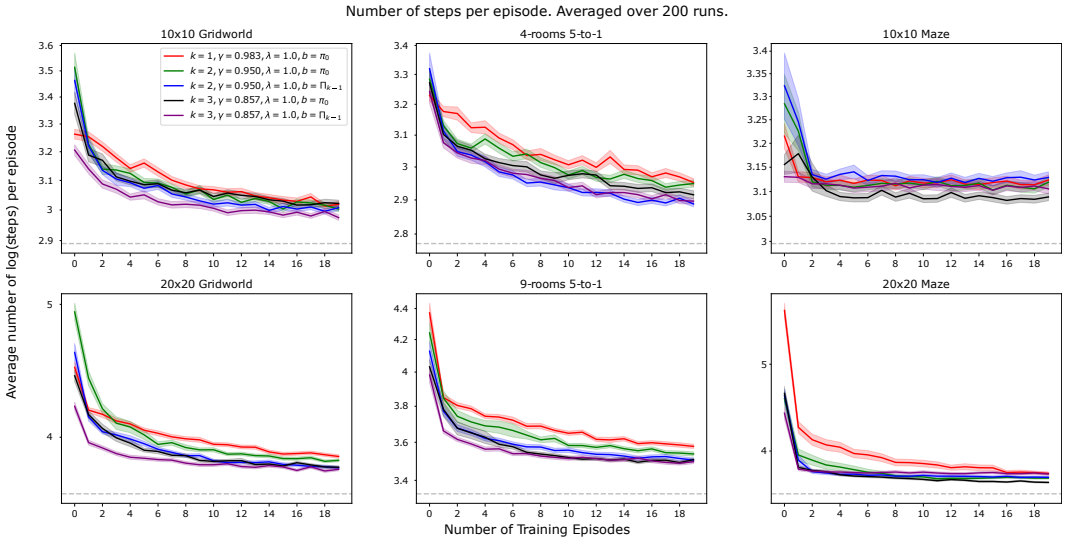

Figure 10: Average $\log$(score) of the evaluated hierarchical $Q(\lambda)$ agents at each training episode (lower is better) when credit assignment *depth* is appropriately balanced for each hierarchy level k. Shaded regions indicate 1-standard error of the mean.