Universiteit Leiden

Master Computer Science

Hierarchical Multistep and $Q(\lambda)$

Name:	Joery A. de Vries
Student ID:	s2420384
Date:	25/06/2021
Specialisation:	Data Science
1st supervisor:	prof. dr. Aske Plaat
2nd supervisor:	dr. Thomas M. Moerland

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Hierarchical Reinforcement Learning holds a strong promise to improve sample efficiency and scalability of online learning methods. However, the problem of learning arbitrary numbers of hierarchical policies, each with varying levels of temporal abstraction, has remained notoriously difficult. This work aims to shed further insight into the interplay of multilevel policy hierarchies and credit assignment in scenarios where other beneficial characteristics of hierarchy are excluded — e.g., the benefits posed for transfer, guided exploration, or generalization. We return to simple discrete and finite MDPs and investigate a recent hierarchical algorithm by Levy et al. [15] from a more conventional perspective for temporally extended credit assignment, being the multistep or λ-returns with eligibility traces. Viewed within our framework, we can intuitively reason on how multiple levels of increasingly temporally abstract control policies assign rewards to previously taken actions and how this differs from the conventional flat multistep return algorithms. Our main result shows that hierarchy allows agents to jump over multiple environment actions for reward assignment in a way that could be seen as a remedy to trace-cutting; the bane of off-policy multistep return algorithms to perform trace truncation when the behaviour and target policies oppose eachother during learning. Our experimental results also show, opposed to previous approaches for unifying temporally extended credit assignment algorithms, that our method is able to jointly and independently learn multiple levels of hierarchical policies and drastically outperform the baseline or non-hierarchical alternatives.

Keywords: Hierarchical and Model-Free RL, Hindsight Learning, Eligibility Traces

Contents

List of Figures

Algorithms

Nomenclature

The next list describes several symbols that will be used within this document

Numbers, Arrays, and Sets

a	a scalar (usually real or an integer)
$\mathbf{a}$	a column vector with elements $\mathbf{a} = [a_1, a_2, \ldots]^\top$
A	a matrix with columns $A = [\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \ldots]^\top$
I	The identity matrix with implicit dimensionality.
I_n	The identity matrix with explicit dimensionality $n \times n$.
1_n	The all-ones vector of length n
$\mathbb{R}$	The set of all real numbers
$\{0, 1\}$	The set with elements 1 and 0
$[a, b]$	The inclusive subset of real values between a and b
$(a, b]$	The inclusive subset of real values between a and b, excluding a
$\mathcal{A}$	Some generic set specified by the context
$\subset$	The subset operator acting on sets

Linear Algebra

$\top$	The transpose operator transforming column vectors to row vectors, and vice-versa
$\mathbf{a}^\top \mathbf{b}$	The inner (dot) product between vectors $\mathbf{a}$ and $\mathbf{b}$, $\mathbf{a}^\top \mathbf{b} = \sum_i a_i \cdot b_i$
$\bigoplus$	Column-wise vector concatenation $\mathbf{a} \equiv \bigoplus_i a_i$
$\bigotimes$	The tensor product, $\mathbf{a} \otimes \mathbf{b} \equiv \mathbf{a}\mathbf{b}^\top$
$\bigodot$	The Hadamard (or elementwise) product, $\mathbf{a} \odot \mathbf{b} \equiv [a_1 \cdot b_1, a_2 \cdot b_2, \ldots]^\top$
Diag	A function that maps vectors to diagonal matrices, $\text{Diag}(\mathbf{a}) = I \odot (\mathbf{a} \otimes 1^\top)$

Probability Theory

$p(a)$	Marginal distribution of random variable a
$x \sim \mathcal{D}$	Random variable x being sampled from distribution over a set $\mathcal{D}$
$\mathbb{E}_{x \sim \mathcal{D}}[f(x)]$	The expected value of $f(x)$ of some distribution over the set $\mathcal{D}$.
$\mathbb{1}_{\text{Predicate}}$	The indicator function that maps to 1 if the predicate is true, otherwise 0
$\mathbf{1}_i$	The indicator vector of only zeros with element i set to 1

Other Symbols

$\sum_i$	Summation operator over elements i
$\nabla_x f(x)$	The partial derivatives of some function f with respect to argument x
$\arg\max_x f(x)$	The operator that yields the argument x that maximizes f
$\doteq$	The define operator, which means that some statement holds true by definition
$\equiv$	Denotes equivalency between two distinct statements

1. Introduction

Hierarchical Reinforcement Learning (HRL) is often held as an open frontier in RL for developing more sample-efficient control algorithms [28, 27, 34, 14, 23, 15]. In principle, hierarchy provides a natural way to solve complex problems by decomposing them into smaller, preferably simpler and recurring, subtasks [28]. This also allows agents to reuse and recombine control policies on previously unseen parts of an environment's state space. Recently, the work by Wen et al. [34] outlined the benefits posed by hierarchy as three distinct points: (1) the ability to compose and reuse previously learned control policies for transfer; (2) to allow for more efficient exploration throughout the statespace by organizing control policies in temporally consistent and guided ways [15, 19, 9]; and (3) to perform more efficient credit assignment through counterfactual and multi-contingency learning.

Each of these benefits have been studied in detail over numerous decades, yet effective learning of arbitrary policy hierarchies has remained difficult [15, 27]. It is imperative for reinforcement learning algorithms to learn such policy structures for any relevant practical scenarios due to the vast scaling issues imposed by *reward sparsity* and arbitrary *temporal granularity* of the real world (without getting into further philosophical details). The recent hierarchical actor-critic algorithm by Levy et al. [15] is one of the first algorithms that achieved stable and effective learning of multilevel policy hierarchies jointly and independently of one another. Though their method was effective, it remained vague and poorly understood how exactly their training method allowed such stable and efficient learning and what the effects of multiple levels of hierarchies were.

This thesis investigates credit assignment in hierarchical reinforcement learning following the method outlined by Levy et al. [15]. We subsume their algorithm into a more general k-level Hierarchical Q-learning framework, or HierQ_k for short, to contrast this algorithm with conventional approaches for *temporal credit assignment* [27, 6, 13, 14]. These conventional approaches for temporally extended credit assignment rely on multistep returns or λ-returns, well known algorithms that compute these involve Tree-Backup or eligibility traces [27]. One difficulty of these methods is how deeply to propagate rewards for evaluating the impact of certain actions, which is strongly related to the bias-variance trade off.

Our method is conceptually, and practically, similar to previous work by Jain et al. [14], which extended the option framework to eligibility traces. The difference is that our incorporation of the work by Levy et al. [15] allows us to train policies independently of one another. Essentially, our method can also be subsumed as a special case of intra-option learning [28, 14].

Our results firstly shows how hierarchical credit assignment with multistep returns can drastically improve the baseline method, opposed to previous work by Jain et al. [14] which suffered from trace-cutting due to the hierarchical policies *not* being trained independently from the flat policies. Furthermore, we show how the hierarchical policies attribute rewards to state-action pairs that lie outside or within a hierarchical policy's *reach*. When a particular goal-state (or option) lies within the temporal reach of an hierarchical policy, it is able to directly 'jump' over all intermediary environment steps and send rewards far backwards through time (ignoring everything inbetween). Opposed to conventional approaches, which send rewards backwards through a memory-vector (the eligibility trace) that is dependent on every action that occurs inbetween temporally separated events; for example, policy instantiation and reward observation. This characteristic allows hierarchical policies to prevent premature trace-cutting and learn from training episodes invariant to the suboptimality of the current effective environment policy.

We also found that when the training traces for the hierarchical policies strongly exceed their temporal reach, then hierarchy can reduce learning speed. This phenomenon is an implication of the familiar bias-variance trade off in conventional approaches, however the hierarchy exacerbates this problem in two ways: the hierarchy allows rewards to be sent even further back due to omitting potential trace-cutting steps and diluted exponential discounting; and the hierarchy imposes a delay to action sampling and a strict adherence to potentially suboptimal goals. When significantly increasing the number of hierarchy levels, and effectively the hierarchical agent's temporal reach, we found that this problem is mitigated. Effectively, our hierarchical framework supersedes the performance of their flat counterparts on all tested environments when appropriately configured.

This document is structured as follows: We introduce all necessary background in Chapter 2 to conceptually build up to the hierarchical Q-learning algorithm. The algorithm is explained by putting the previous method of Levy et al. [15] in a more conceptually friendly framework in Chapter 3, which we then subsume into our Hier$Q_k(\lambda)$ framework (analogously to how Watkin's $Q(\lambda)$ subsumes and generalizes one-step Q-learning [33, 32]). From our algorithmic description we first make a variety of observations on how credit assignment is performed, in Chapter 4 we illustrate how this effectively differs from conventional, flat, temporally extended credit assignment. In the same chapter we continue with an extensive analysis into a variety of parameter ablations that attempt to make the conventional reward assignment algorithms as similar as possible to their hierarchical counterparts. We discuss our observations and hypotheses in Chapter 5 which is followed by Related Work and suggestions for further analysis in Chapter 6. We end with some concluding remarks in the final Chapter 7.

2. Background

We consider sequential decision making problems which we formulate as Markov Decision Processes (MDPs) [27]. A MDP is represented as a tuple $M \doteq \langle \mathcal{S}, \mathcal{A}, p, R, \gamma \rangle$ and constitutes the logic/ behaviour of an *environment* and how an autonomous *agent* interacts with it. We consider the sets for the state-space $\mathcal{S}$ and action-space $\mathcal{A}$ to be finite and discrete. The function $p : \mathcal{S}, \mathcal{A} \to \mathcal{P}(\mathcal{S})$ defines the dynamics of the environment by mapping state-action pairs to the distribution of following states (possibly deterministic). Upon each state-transition $s \xrightarrow{a} s'$, the reward function $R(s, a, s')$ yields a scalar reward signal. Finally, denote $\gamma : \mathcal{S} \to [0, 1)$ as the termination function that weighs the importance of short-term over long-term rewards, for brevity we use as a shorthand $\gamma_t = \gamma(S_t)$.

The aim of Reinforcement Learning (RL) is to learn a policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ for our agent that maximizes its expected cumulative reward when employed within an environment M. We consider that applying π in M within the time-interval $t, \dots, T$ yields a trace of random variables $\tau_{t:T} = \{S_t, A_t, R_{t+1}, \dots, S_{T-1}, A_{T-1}, R_T, S_T\}$, where actions and new states are sampled from the policy $A_t \sim \pi(S_t)$ and transition function $S_{t+1} \sim p(S_t, A_t)$, respectively, and using $R_{t+1} = R(S_t, A_t, S_{t+1})$. From this trace we derive the agent's *return* from t until T as:

$$G_t \equiv G_{t:T} \doteq \sum_{i=1}^{T-t} \gamma_{t+i}^{i-1} R_{t+i}, \tag{2.1}$$

which is an exponentially discounted sum of rewards using γ_i. This full return G_t is called the Monte-Carlo return [27, 24, 13] and is an unbiased sample of the expected return when following policy π starting at state S_t. We can write this expectancy as a *value function* for any state s as:

$$v_\pi(s) \doteq \mathbb{E}[G_t \mid S_t = s, \pi]. \tag{2.2}$$

We desire a policy that maximizes v_π, we denote this as the optimal policy $\pi_* \doteq \max_\pi v_\pi(s), \forall s \in \mathcal{S}$, where we abbreviate its value as $v_* \doteq v_{\pi_*}$.

2.1 Linear Temporal Difference Learning

In practice, the value function v_π is often unknown and must be estimated through sampling, we denote its approximation as $V \approx v_\pi$. Consider using a linear parametric model to estimate v_π with associated weights $\mathbf{w} \in \mathbb{R}^D$, we can write the estimated value for any state s as an inner product between our weights and the corresponding state features $\mathbf{x}(s)$,

$$V^{\mathbf{w}}(s) = \mathbf{w}^\top \mathbf{x}(s) = \sum_{i=1}^{D} w_i \cdot x_i(s), \tag{2.3}$$

where D is the dimensionality of the state-feature vector. In the simplest case, the state feature vector is an indicator $\mathbf{x}(s) \in \{0, 1\}^{|\mathcal{S}|}$, which can be interpreted as a *table-lookup*, $V^{\mathbf{w}}(s) = w_s$. It then follows that an unbiased estimation of v_π (given a fixed π) should minimize the sum of squared differences between v_π and $V^{\mathbf{w}}$, we can write this objective as a *loss function* over our weights,

$$E(\mathbf{w}) \doteq \frac{1}{2} \sum_{s \in \mathcal{S}} d_\pi(s) \left(v_\pi(s) - \mathbf{w}^\top \mathbf{x}(s) \right)^2, \tag{2.4}$$

where d_π is the state distribution induced by the dynamics p and policy π over all $s \in \mathcal{S}$. This loss function can be minimized with *Stochastic Gradient Descent* (SGD), this allows the agent to update its weights $\mathbf{w}$ at each step inside the trace according to the incremental update rule,

$$\begin{aligned} \mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2} \alpha \nabla_{\mathbf{w}} \left(v_\pi(S_t) - \mathbf{w}_t^\top \mathbf{x}(S_t) \right)^2 \\ &= \mathbf{w}_t + \alpha \left(v_\pi(S_t) - \mathbf{w}_t^\top \mathbf{x}(S_t) \right) \nabla_{\mathbf{w}} (\mathbf{w}_t^\top \mathbf{x}(S_t)) \\ &= \mathbf{w}_t + \alpha \left(v_\pi(S_t) - \mathbf{w}_t^\top \mathbf{x}(S_t) \right) \mathbf{x}(S_t), \end{aligned} \tag{2.5}$$

where α is a step-size parameter. When learning from samples, we swap v_π in the update above out for the unbiased samples G_t, this yields the final update rule,

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[G_t - \mathbf{w}_t^\top \mathbf{x}(S_t)\right] \mathbf{x}(S_t). \tag{2.6}$$

In the case of indicator features, this simplifies to an incremental average of every observed G_t at state S_t (in the case of $\alpha = 1$).

The Monte-Carlo returns G_t are unbiased estimates of v_π, however, they are prone to high variance and requires waiting until the termination of the trace $\tau_{t:T}$. Both problems can be sidestepped through *bootstrapping*, a dynamic programming trick that exploits the fact that the value function can be equivalently rewritten as:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[G_{t:T} \mid S_t = s, \pi] \\ &= \mathbb{E}[R_{t+1} + \gamma_{t+1} G_{t+1:T} \mid S_t = s, \pi] \\ &= \mathbb{E}[R_{t+1} + \gamma_{t+1} v_\pi(S_{t+1}) \mid S_t = s, \pi], \end{aligned} \tag{2.7}$$

where $\gamma_T \doteq 0$. With bootstrapping, the complete returns can be estimated at any time step in the trace. Given the partial returns $G_{t:t+n}$ for some environment trace, we can estimate the full return by letting,

$$\hat{G}_{t:t+n} \doteq G_{t:t+n} + V^{\mathbf{w}}(S_{t+n}), \quad 1 < t \leq T. \tag{2.8}$$

This allows us to update our current value estimation with only a $n < T$ step delay. This yields the following incremental update rule for $V^{\mathbf{w}}$ when using e.g., $n = 1$:

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha[\hat{G}_{t:t+1} - \mathbf{w}_t^\top \mathbf{x}(S_t)]\mathbf{x}(S_t) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha[R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}(S_{t+1}) - \mathbf{w}_t^\top \mathbf{x}(S_t)]\mathbf{x}(S_t), \end{aligned} \tag{2.9}$$

which is also known as (linear) *Temporal Difference* (TD) learning.

The full return G_t and its 1-step estimation $\hat{G}_{t:t+1}$ exist at the ends of two extremes; the full return can have high variance whereas the 1-step target can be strongly biased to the current (possibly faulty) value estimation. Both extremes can cause learning instability, especially when coupled with function approximation or with high learning rates α [27, 12]. Furthermore, both methods commit to an extreme timescale for performing updates (1-step or T-step), both can slow down learning. Usually, intermediate values for n when estimating $\hat{G}_{t:t+n}, 1 < n < T$ perform better in practice, this is called *atomic* multistep TD [6].

Finally, to learn a 'good' control policy, *Bellman's policy improvement theorem* [4, 3] shows that alternating between *policy evaluation* $V^{\mathbf{w}_t} \approx v_\pi$ and acting greedily $\pi^{(t+1)}(s) \leftarrow \max_\pi V^{\mathbf{w}_t}(s)$ will always yield a new policy that is no worse than the former $v_{\pi^{(t+1)}}(s) \geq v_{\pi^{(t)}}(s), \forall s \in \mathcal{S}$. Iterating this will eventually lead to a fixed point $\pi^{(t)}(s) = \pi^{(t-1)}(s)$ as $t \to \infty$, which under some technical conditions can be shown to yield the optimal policy π_* [27, 4]. This paradigm falls under *Generalized Policy Iteration* (GPI) and most modern RL algorithms employ this principle in some way [3, 27].

2.2 Temporal Difference Control

Instead of focusing on values over states, it is also possible to integrate policy evaluation directly with policy improvement (control). For this, denote the *state-action value function* as the expected return over G_t conditioned on a state and action pair,

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}[G_t \mid S_t = s, A_t = a, \pi] \\ &= \mathbb{E}[R_{t+1} + \gamma_{t+1} v_\pi(S_{t+1}) \mid S_t = s, A_t = a, \pi]. \end{aligned} \tag{2.10}$$

Though q_π and v_π are conceptually inseparable, it can be more practical to derive new control policies from $Q \approx q_\pi$ for GPI. As an example, for Q we can define $\pi(s) = \arg\max_a Q(s, a)$ instead of $\pi(s) = \arg\max_a \sum_{s'} p(s, a, s')[R(s, a, s') + \gamma(s')V(s')]$.

Similarly to $V^{\mathbf{w}}$ we can parameterize our state-action value estimation as $Q^{\mathbf{w}}$ and optimize in a similar fashion. When using a 1-step target we can update $Q^{\mathbf{w}}$ with,

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha[\hat{G}_{t:t+1} - \mathbf{w}_t^\top \mathbf{x}(S_t, A_t)]\nabla_{\mathbf{w}} Q^{\mathbf{w}}(S_t, A_t) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha[R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}(S_{t+1}, A_{t+1}) - \mathbf{w}_t^\top \mathbf{x}(S_t, A_t)]\mathbf{x}(S_t, A_t) \qquad (2.11)\end{aligned}$$

This update rule for $Q^{\mathbf{w}}$ is known as SARSA(0), implying $\lambda = 0$, which naturally refers to the $\langle S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}\rangle$ tuple derived from the environment trace. As the SARSA(0) update is derived directly from the environment trace, the state-action value estimates are those of the policy generating this trace — this is called *sampled on-policy* learning.

The SARSA(0) update is conveniently derived from environment samples, however, such samples can have high variance due to stochasticity induced by both the environment and the behaviour policy. One method that eliminates the variance in action sampling is Expected SARSA(0). This rule computes an expectation over $Q^{\mathbf{w}}$ weighted by π for the bootstrap target, yielding:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha[R_{t+1} + \gamma_{t+1}\sum_{a\in\mathcal{A}} \pi(a|S_{t+1})Q^{\mathbf{w}}(S_{t+1}, a) - Q^{\mathbf{w}}(S_t, A_t)]\nabla_{\mathbf{w}} Q^{\mathbf{w}}(S_t, A_t). \qquad (2.12)$$

While generally improving SARSA(0) [25, 6, 27], Expected SARSA(0) also provides a natural way to perform *off-policy* learning without having to compute importance sampling ratios [27]. In off-policy learning an environment trace is generated according to some behaviour policy b and we estimate Q (or V) for another policy π. The most well known off-policy control method is Q-learning [32, 33, 27] which is a special case of Expected SARSA(0) that attempts to learn the optimal policy $\hat{\pi}_* \approx \pi_*$ by maximizing over $Q^{\mathbf{w}}$,

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha[R_{t+1} + \gamma_{t+1}\max_a Q^{\mathbf{w}}(S_{t+1}, a) - Q^{\mathbf{w}}(S_t, A_t)]\nabla_{\mathbf{w}} Q^{\mathbf{w}}(S_t, A_t). \qquad (2.13)$$

Multistep target computation becomes more subtle when not strictly sampling from the generated environment trace [6, 20, 11, 27]. For example, a direct aggregation towards $\hat{G}_{t:t+n}$ within some trace would amount to a *sampled on-policy* target — i.e., multistep SARSA. Expected updates can reduce variance of action-sampling, but it has to compute $\hat{G}_{t:t+n}$ by backing up partial returns $\hat{G}_{t+i:t+n}$ after averaging these out at each backup step with $Q^{\mathbf{w}}$ and the target policy,

$$\hat{G}_{t+i-1:t+n} = R_{t+i} + \gamma_{t+1}\left(\pi(A_{t+i}|S_{t+i})\hat{G}_{t+i:t+n} + \sum_{a\in\mathcal{A}} \pi(a|S_{t+i})Q^{\mathbf{w}}(S_{t+i}, a)\right), \qquad (2.14)$$

for $i = n-1, \ldots, 1$; this method is also known as Tree-Backup [22, 27]. Tree-Backup can make learning slower, compared to multistep SARSA, as the repeated averaging can dilute the update targets[1]. Conversely, it can also speed up learning by allowing higher learning rates α due to the reduced variance [25]. In the case of off-policy learing, this method can also result in having to cut the multistep target prematurely. This implies truncating n and computing the bootstrap target at an intermediary timestep $1 < i < n$ when the target policy places zero probability on the sampled action, $\pi(A_{t+i} \mid S_{t+i}) \ = \ 0$.

2.3 Eligibility Traces and Watkin's $Q(\lambda)$

Another way to perform multistep updates is to define a mixture over all n-step returns from $n = 1, \ldots, T$, we distinguish this as a *compound* multistep TD target. The most commonly used compound return is the λ-return,

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1}\hat{G}_{t:t+n} + \lambda^{T-t-1}G_{t:T}, \qquad (2.15)$$

[1]There do exist algorithms that unify sampled and expected backups for 1-step or multistep returns, e.g., $Q(\sigma)$ or $Q(\sigma, \lambda)$ achieve this through a weighted average of the two [6, 8].

which is essentially a weighted average over all $\hat{G}_{t:t+n}$ with weights $(1-\lambda)\lambda^{n-1}$ fading over time. The $\lambda \in [0,1]$ parameter allows for straightforward tuning of the bias-variance trade-off where 1-step TD exists on one end of the spectrum and Monte Carlo estimation at the other [26, 27, 24]. The downside of the λ-return is that G_t^λ is only known at the end of the trace. Despite that, it is possible to perform value updates that build towards G_t^λ during generation of the trace. Instead of looking *forward* on the trace for determining update targets — which requires waiting until G_t^λ is known — we can instead signal intermediate rewards *backward* along the trace to perform incremental updates.

This can be done with the use of an *eligibility trace*, this is a vector $\mathbf{z}_t \in \mathbb{R}^D$ which accumulates all previously seen (discounted) state features $\mathbf{x}(s,a)$ (or $\mathbf{x}(s)$ for V). This is a form of visitation memory, and is updated at each timestep according to,

$$\mathbf{z}_t = \gamma_t \lambda \mathbf{z}_{t-1} + \nabla_\mathbf{w} Q^\mathbf{w}(S_t, A_t), \tag{2.16}$$

where $\mathbf{z}_{-1} \doteq 0$. With indicator features this simplifies to a table with discounted (λ-decayed) visitation counts. Using this memory vector, our weights for $Q^\mathbf{w}$ are updated by computing the 1-step TD-error $\delta_t = \hat{G}_{t:t+1} - Q^\mathbf{w}(S_t, A_t)$ and casting this error over $\mathbf{z}_t$, which gives us:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t. \tag{2.17}$$

Through this mechanism, the weights $\mathbf{w}$ are adjusted proportional to the recency of each state's feature that was observed in the past. The update mechanism in Equation 2.16 is known as an *accumulating trace*, it aggregates all observed (decayed) state features over time. An alternative to this (when using tabular features) would be to decay $\mathbf{z}_{t-1}$ with $\gamma_t \cdot \lambda$ and setting the active element from $\mathbf{x}(S_t, A_t)$ in $\mathbf{z}_t$ to 1, this is known as a *replacing trace*. Both these trace variants yield final weights $\mathbf{w}$ that are approximately equivalent to the weights that would've resulted from updating on the full return G_t^λ.

The eligibility trace update in Equation 2.16 defines the SARSA(λ) algorithm, seeing as it performs pure on-policy sampling. We can rewrite this for Expected SARSA, which gives us Tree-Backup(λ). This requires that δ_t is calculated with Equation 2.12 and the eligibility is updated according to,

$$\mathbf{z}_t = \gamma_t \lambda \pi(A_t|S_t) \mathbf{z}_{t-1} + \nabla_\mathbf{w} Q^\mathbf{w}(S_t, A_t). \tag{2.18}$$

When learning the optimal policy $\hat{\pi}_*$, Tree-Backup(λ) yields Watkin's $Q(\lambda)$, this version cuts traces, $\mathbf{z}_t = 0$, when $A_t \neq \max Q^\mathbf{w}(S_t, a)$. The general outline of Watkin's $Q(\lambda)$ algorithm is given in Algorithm 1 for a static discount parameter γ. This is an extremely general procedure and can easily be modified to either Tree-Backup(λ), SARSA(λ), or other variants [27, 32, 18].

Algorithm 1 Outline of Watkin's $Q(\lambda)$.

```
    Requires: A behaviour policy $b$, discount factor $\gamma$, decay rate $\lambda$, and learning rate $\alpha$.
 1: Initialize $Q^\mathbf{w}$ arbitrarily, and $\mathbf{z} \leftarrow 0$
 2: Initialize $S$
 3: while not terminate do
 4:     Sample an action $A \sim b(a|s)$
 5:     Take action $A$, observe transition $S'$ and reward $R$
 6:     $\mathbf{z} \leftarrow \mathbb{1}_{Q^\mathbf{w}(S,A)=\max_a Q^\mathbf{w}(S,a)} \gamma\lambda\mathbf{z} + \nabla_\mathbf{w} Q^\mathbf{w}(S,A)$
 7:     $\delta \leftarrow R + \gamma \max_a Q^\mathbf{w}(S',a) - Q^\mathbf{w}(S,A)$
 8:     $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
 9:     Update behaviour policy $b$ based on $Q^\mathbf{w}$                    ▷ e.g., $\epsilon$-greedy
10:     Update state $S \leftarrow S'$
```

2.4 Goal Conditioning and Hindsight Learning

The previously introduced concepts can effectively tackle credit (or rewards) assignment when performing GPI in some environment M. By doing so, these methods can swiftly identify and follow the gradient of interest imposed by R. However, consider a *sparse* reward function R, of which the output is almost always zero, none of the previously introduced concepts offer a solution for effectively organizing control until an actual reward is observed. As a result, an agent may have to resort to random control until a reward is observed.

This phenomenon is often tackled through *reward shaping*, these methods augment the environment's reward function through e.g.: score-based metrics, exploration bonuses, or novelty based rewards. We focus on a scenario where we define sparse reward functions for all reachable states $s \in \mathcal{S}$. This method naturally lends itself to hierarchical RL (as will be explained in the following chapter) by allowing such agents to instantiate specialized policies for achieving specific, desired, environment states (goal conditioning) [23, 3, 5] and update multiple control policies based on singular transitions (hindsight learning) [1, 15].

Concretely, we desire an agent that can learn to reach any desired state $s \in \mathcal{S}$ within M. To prevent confusion, we change notation for such desired states as desired *goals* $g \in \mathcal{S}$. We define reward functions for each distinct goal as $R^g(s, a, s') \equiv R^g(s') = \mathbb{1}_{s'=g}$ and formulate the problem as learning the set of optimal state action value functions $\{q_*(s, a, g)\}_{g \in \mathcal{S}}$, where each

$$q_*(s, a, g) \doteq \mathbb{E}[\gamma_{t+n}^n \mid S_t = s, A_t = a, g, \pi_*], \tag{2.19}$$

models the expected number of steps n needed to reach goal g following that goal's optimal policy on a γ_t-logarithmic scale. The idea is also illustrated in Figure 2.1, which shows that given some environment trace we are able to learn about each contingency within the environment in case we need to return to such states in the future.

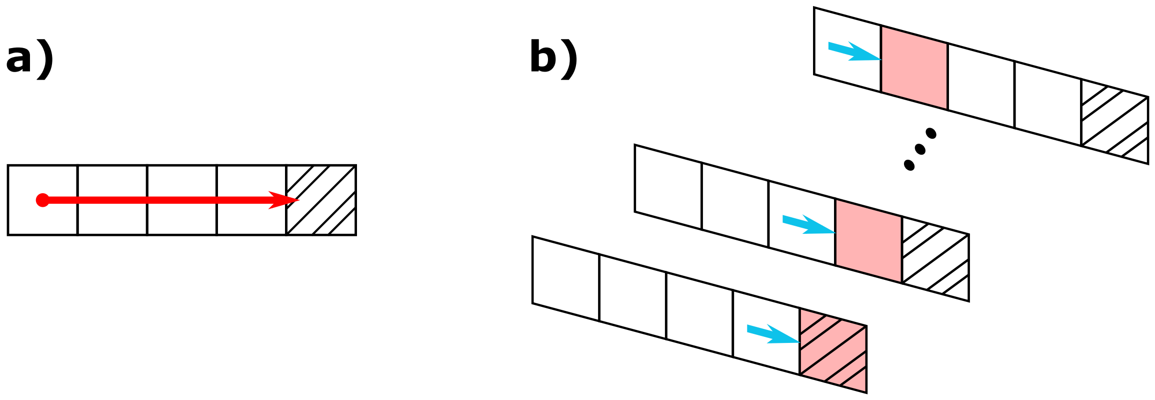

Figure 2.1: **a)** Given some arbitrary environment trace **b**) we perform Q-updates conditioned on every observed state along the trace. Over time, the agent will learn multiple policies, each of which can return the agent to a desired goal-state following the optimal path.

Consider using a linear Q-learning agent with associated weights $W \in \mathbb{R}^{d \times |\mathcal{S}|}$. Our aim is to learn the weight matrix $W = [\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \dots, \mathbf{w}^{(|\mathcal{S}|)}]^\top$ such that each column g directly estimates $Q^{\mathbf{w}^{(g)}} \equiv Q_g^W \approx q_*(s, a, g)$. By writing all our Q-function estimations as a singular matrix, we can conveniently rewrite the reward function as the next-state indicator vector (for discrete and finite $\mathcal{S}$),

$$\mathbf{r}_{t+1} \equiv \mathbf{r}(S_t, A_t, S_{t+1}) \doteq \mathbf{1}_{S_{t+1}}, \tag{2.20}$$

along with the termination function $\boldsymbol{\gamma}_{t+1} \doteq \gamma(1 - \mathbf{1}_{S_{t+1}})$, where γ is a base value. From this, we can write out the goal-conditioned TD-errors (in the case of Q-learning) as a single vector,

$$\boldsymbol{\delta}_t \doteq \mathbf{r}_{t+1} + \boldsymbol{\gamma}_{t+1} \bigoplus_{g \in \mathcal{S}} \max_a Q_g^W(S_{t+1}, a) - Q^W(S_t, A_t), \tag{2.21}$$

where $\bigoplus$ denotes a column-wise vector concatenation. This mechanism essentially enables us to update all columns in W (all goal-conditioned Q-function estimations) by casting the estimation errors $\boldsymbol{\delta}_t$ over W according to the update rule,

$$W_{t+1} \doteq W_t + \alpha(\boldsymbol{\delta}_t \otimes \nabla_W Q^W(S_t, A_t))^\top, \tag{2.22}$$

where $\otimes$ denotes the tensor product (or outer product) operator which generates a matrix of dimensions $|\mathcal{S}| \times D$. By updating all columns in this way, we can simultaneously learn about all possible contingencies within M based on a single transition, this concept falls under hindsight learning [1]. It should be clear that we can derive new control policies for reaching some goal $g \in \mathcal{S}$ through,

$$\begin{aligned} \hat{\pi}_*(s, a, g) &= \arg\max_a Q_g^W(s, a) \\ &= \arg\max_a (W^\top \mathbf{x}(s, a))_g, \end{aligned} \tag{2.23}$$

seeing as the above inner product yields a column vector with each element containing the estimation to $q_*(s, a, g)$.

This definition naturally extends itself to multistep returns and eligibility traces. In such a scenario, the eligibility trace would be a matrix $Z \in \mathbb{R}^{D \times |\mathcal{S}|}$ that would be updated in a similar way to the non goal-conditioned methods from the previous section. For example, when using an eligibility trace inspired by Watkin's $Q(\lambda)$, we can write,

$$Z_t \doteq \lambda Z_{t-1} \text{Diag}\left(\boldsymbol{\gamma}_t \bigoplus_{g \in \mathcal{S}} \hat{\pi}_*(A_t | S_t, g)\right) + \nabla_W Q^W(S_t, A_t). \tag{2.24}$$

where Diag : $\mathbb{R}^n \to M_{nn}$ and $\nabla_W Q^W(S_t, A_t)$ is added to Z_t column-wise[2]. With this, the weight matrix can be updated according to,

$$W_{t+1} \doteq W_t + \alpha Z_t \text{Diag}(\boldsymbol{\delta}_t), \tag{2.25}$$

where $\boldsymbol{\delta}_t$ contains the 1-step TD-errors for each goal-state (according to Equation 2.21).

Note, that in case of hindsight learning, that most goal-conditioned updates become inherently *off-policy* (disregarding the method used for bootstrapping over Q_s^W). Hypothetically, it is possible to align certain policies through their trajectories to do on-policy hindsight updates (by filtering out the samples with equal action probabilities), however it is more practical to simply resort to off-policy learning. In some way, the goal conditioning on states can be seen as a remedy for trace-cutting in Watkin's $Q(\lambda)$, or multistep Q-learning with Tree-Backup. An action with zero probability under one target policy specialized for goal g may just as well have maximum probability under another target policy specialized for $g' \neq g$, this allows the entire agent trace to be split up into the different goal conditioned Q estimations.

[2]I.e., vector to matrix addition: $M_{nm} + \mathbf{v}_n$ is implicit for $M_{nm} + (1_m \otimes \mathbf{v}_n)^\top$, where 1_m is the all-ones vector of length m; not to be confused with the indicator vector $\mathbf{1}_m$ with all-zeros except element m being 1.

3. Hierarchical Q-Learning

This chapter presents the general idea of hierarchical Q-learning, following the Hierarchical Actor-Critic framework of Levy et al. [15], and extends it to a broader class of control algorithms. We first describe the basic method: how the hierarchical framework recursively decomposes a problem into smaller subMDPs, and how it jointly assigns rewards/ credits to the control policies independent of temporal scale. After this, we introduce the concept of multistep returns for this class of algorithms and subsume the base method into the general Hier$Q_k(\lambda)$ algorithm. This algorithm is equivalent to Tree-Backup(λ), or Watkin's $Q(\lambda)$, for $k = 1$, and enables efficient learning of multiple policies operating at varying temporal scales.

3.1 Hierarchical Q-Learning

We consider the scenario where the reward function of a MDP is extremely sparse, e.g. due to: the problem description; timescaling; or unavailability of a shaped reward function. As explained earlier, agent behaviour can become relatively unproductive with methods like Q-learning if relying on ϵ-greedy behaviour policies. As alluded to in Section 2.4, conditioning multiple policies on desired goal-states can allow RL agents to organize behaviour by learning to achieve desired observations. Through this, another policy that operates on a less granular temporal scale can instruct the lower level policy (policies) to reach a certain state in the form of actions.

Algorithm 2 Environment loop for learning hierarchical weights $\{W^{(0)}, W^{(1)}, \dots, W^{(k-1)}\}$.

Requires: Number of hierarchy levels k, policy horizon $H^{(i)}$ and goal-conditioned behaviour policies $b^{(i)}$ for each hierarchy level $i = 0, \dots, k-1$, and an environment state to achieve $g^{(k-1)} \in \mathcal{S}$.

```
1: Derive atomic horizons $H^{(i)a} = \prod_{j=0}^{i} H^{(j)}$ for all levels $i = 0, \dots, k-1$
2: Initialize $Q^{W^{(i)}} \leftarrow 0$ for all levels $i = 0, \dots, k-1$
3: Initialize $S$
4: while $S \neq g^{(k-1)}$ do
5:     $S \leftarrow$ Recurse($k-1$, $S$, $g^{(k-1)}$)
```

Recurse(i :: level, S :: state, $g^{(i)}$ :: goal-state)

```
1: Initialize counter $n \leftarrow 0$
2: while $n < H^{(i)}$ and $S \neq g^{(j)}, i \leq j \leq k-1$ do        ▷ i.e., not terminate options $j \geq i$
3:     Sample an action $A^{(i)} \sim b^{(i)}(S, g^{(i)})$
4:     if $i > 0$ then
5:         $S' \leftarrow$ Recurse($i-1$, $S$, $A^{(i)}$)
6:     else
7:         Apply $A^{(i)}$ as an action $A$, observe transition $S'$ and environment reward $R$
8:         for $i = 0, \dots, k-1$ do
9:             Update($i$, $W^{(i)}$, $H^{(i)a}$)        ▷ e.g., see Algorithm 3
10:            Derive new goal conditioned behaviour policies $b^{(i)}$        ▷ e.g., $\epsilon$-greedy
11:    Update state $S \leftarrow S'$ and counter $n \leftarrow n + 1$.
12: return $S$
```

This form of Hierarchical RL (HRL) was effectively demonstrated by Levy et al. [15], their algorithm attempts to learn a recursively optimal hierarchical policy based on goal-conditioning and hindsight learning. The proposed recursive relation naturally extends the usual RL agent-environment loop, as illustrated in Algorithm 2. For learning a set of, goal-conditioned, Q-function weights, they decompose an MDP into multiple smaller subMDPs that are sampled by the agent policies. Each weight matrix within this set defines a policy that samples desired states $g \in \mathcal{S}$ as terminal states for the subMDPs — except for the 0th weight matrix $W^{(0)}$ which is defined over the environment actions. Every subMDP is given a fixed action-budget, called the policy-horizon H. Thus, when the policy at hierarchy-level i samples a state to achieve, then the subset of lower level policies $\{W^{(j)} | j < i, j = 0, \dots, k-1\}$ must reach this state within this budget (as shown in line 2 of the **Recurse** function in Algorithm 2). These subMDPs are created in a top-down fashion, $i = k-1, \dots, 0$, until the atomic level is reached (line 3 and 5 of the **Recurse** function in Algorithm 2) at which an environment action is applied. The number of environment actions that are implied by the i^{th} subMDP's action-budget $H^{(i)}$ is called the environment or atomic horizon $H^{(i)a} = \prod_{j=0}^{i} H^{(j)}$. For a 1-level hierarchy, this algorithm simplifies to learning a flat goal-conditioned weight matrix for estimating Q^W.

In order to learn the hierarchical weights, as shown in line 9 of the **Recurse** function in Algorithm 2, all weight matrices are updated after an environment transition. Opposed to the atomic weights $W^{(0)}$, the hierarchical weights are updated based on the *observed* state transitions and not the goal-actions that were proposed. In other words, an hierarchical policy i that proposes state $g^{(i-1)}$ for the lower level policies to achieve, is never updated towards this action explicitly (unless this state was reached). Instead, the hierarchical actions are inferred from the environment trace. Levy et al. [15] refer to this as hindsight action transitions: because the entire hierarchy is learned jointly, policies are very likely unable to reach proposed goal-states because these are either impossible, or not yet learned. Inferring the hierarchical actions is a simple way to jointly learn which states can be reached at each level of temporal abstraction.

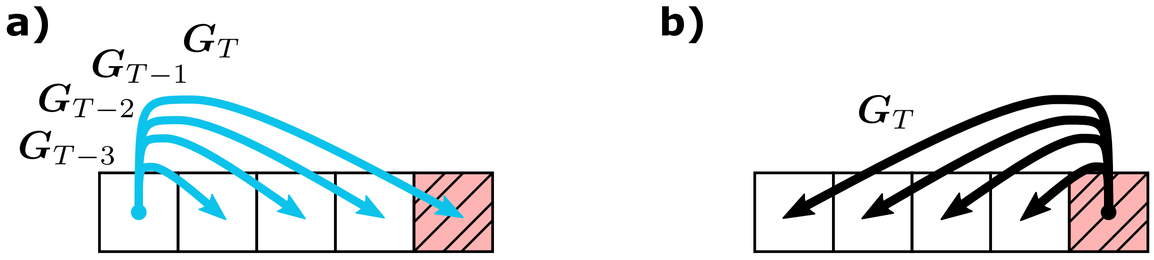

Figure 3.1: Reusing the environment trace illustrated in Figure 2.1, **a)** shows a forward view for all valid state-action updates for the hierarchical agent with an environment horizon of $H^a = 4$; each state within the temporal horizon is inferred as an action. **b)** A more efficient backward view makes use of the fact that the forward returns are reused at multiple states — seeing as, G_{t-2} at state S_0 is equivalent to G_{t-1} at S_1, and so forth — and updates each trailing state-action pair towards the same, immediate, return.

By inferring hierarchical actions from the environment trace, all observed states within a policy's atomic horizon H^a can be considered as valid goals. Due to the use of hindsight learning, each of these actions are optimal w.r.t. to that particular goal-state. In a forward view (looking forwards in the trace), this would imply that a hierarchical weight matrix $W^{(i)}$ will receive $H^{(i)a}$ state-action updates for most of the observed states on the environment trace, as is illustrated in the left of Figure 3.1 for a policy with an atomic horizon of $H^a = 4$.

It should be apparent that, as the number of hierarchy levels increases the atomic horizon will increase exponentially for $H^{(i)} > 1, \forall i$. This would imply long delays between updates akin to the offline λ-return calculation. Instead, it can be observed that a policy's atomic horizon implies a sliding window of states, which in a backward view can be considered as valid preceding states. Coincidentally, each of these trailing states can be updated towards the same return, as is illustrated in the right of Figure 3.1. Algorithm 3 gives a more formal outline of this depicted update procedure. This function can be called in line 9 of the **Recurse** function in Algorithm 2, and overall provides a general layout for updating the full set of weights $\{W^{(0)}, W^{(1)}, \dots, W^{(k-1)}\}$.

3.1.1 Action-Space Bounding

Until now we considered that each of the hierarchical weights govern a policy that samples goal-states $g \in \mathcal{S}$, a view that was directly adopted from Levy et al. [15]. However, it doesn't make much

Algorithm 3 Hier$Q(0)$: Procedure for updating hierarchical weights with 1-step returns.

```
    Requires: A base discount factor $\gamma \in [0, 1]$ and a learning rate $\alpha \in (0, 1]$.
    Requires: An environment trace $\tau = \{S_0, A_0, R_1, S_1, A_1, \dots\}$ and current timestep $t$.

Update($i$ :: level, $W$ :: weights, $H^a$ :: environment horizon)
 1: Infer action: $A \leftarrow S_{t+1}$ if $i > 0$ else $A \leftarrow A_t$
 2: Compute reward $\mathbf{r} \leftarrow \mathbf{1}_{S_{t+1}}$ and discount factor $\boldsymbol{\gamma} \leftarrow \gamma(1 - \mathbf{1}_{S_{t+1}})$      ▷ Equation 2.20
 3: Estimate returns $\widehat{\mathbf{G}} \leftarrow \mathbf{r} + \boldsymbol{\gamma} \bigoplus_{g \in \mathcal{S}} \max_a \left(W^\top \mathbf{x}(S_{t+1}, a)\right)_g$      ▷ Equation 2.21
 4: for $j = 0, 1, \dots, \min(H^a - 1, t)$ do      ▷ (omit duplicate $S$)
 5:     $\boldsymbol{\delta} \leftarrow \widehat{\mathbf{G}} - Q^W(S_{t-j}, A)$
 6:     $W \leftarrow W + \alpha \left(\boldsymbol{\delta} \otimes \nabla_W Q^W(S_{t-j}, A)\right)^\top$      ▷ Equation 2.22
```

sense to define each hierarchical policy $W^{(i)}$ over all potential goal-states in the environment's state space. A hierarchical policy with an atomic horizon of $H^{(i)a} = 4$ will never be able to reach states that lie outside of this horizon in one action-step. As the state-space becomes larger, the probability of sampling reachable goals will decrease proportionally to $|\mathcal{S}|$. While it is of course possible to let the agent learn which goal-states seem fruitful, it can be more efficient to constrain learning.

Instead, we consider that each hierarchical set of weights $W^{(i)}$ with an atomic horizon $H^{(i)a}$ defines an action space that is bounded by some neighborhood (e.g., l_1 or l_∞) with radius $H^{(i)a}$ in $\mathcal{S}$ centered around the current observed state s. As a result, each generated subMDP in the recursion of Algorithm 2 is guaranteed to become simpler as $i \leftarrow 0$. Simultaneously, this also allows for a large reduction in the dimensionality of the hierarchical weights. We provide an empirical comparison to hierarchical agents with bounded and unbounded action spaces in Appendix A.1.

3.1.2 Miscellaneous Extensions

Aside from action space bounding, there exist numerous simple extensions that can complement Algorithm 2. We introduce two additional extensions that are of relevance: stationary transition filtering and intermediate goal termination. The former modifies line 9 in the **Recurse** function of Algorithm 2 to only update the hierarchical weights $W^{(i)}, i > 0$ iff $S' \neq S$. The idea behind this is that if the agent runs into a wall or does not move, on a hierarchical level we can infer that no action was taken. The latter is a modification to the goal termination in line 2 of the **Recurse** function in Algorithm 2 which considers the value of the previously sampled goals and potential new goals. When sampling new state-actions we modify the algorithm to store the Q-value of this particular action and over time track whether this action can be terminated prematurely if another state-action is found with a higher Q-value. Hence, we interchangeably refer to this as *greedy* subgoal termination. The effect of the intermediate goal termination has a subtle effect on the effective environment policy of the agent, we return to this characteristic in Chapter 4.

3.2 Hierarchical Multistep and $Q(\lambda)$

In order to introduce multistep returns to the Hierarchical Q-learning framework, as outlined in Algorithm 3, observe the following: for every inferred hierarchical action inside the atomic horizon H^a within an environment trace τ, it follows that another H^a subsequent actions can be inferred from every state-action onwards. Regardless of the fact that many of these subsequent state-actions already lie within the current horizon H^a, this creates an extremely dense tree of potential action paths, as is illustrated by the black-white arrows in the top-left of Figure 3.2.

While it may be relevant to consider every such path, and update the current value estimates such that every possible contingency is learned, computationally this can swiftly become quite complicated. Though it is possible to create some form of transposition table to hash shared return values to, note that many of these considered paths would be suboptimal anyway. For example, the top-left example in Figure 3.2 shows that one can compute a multistep return starting from

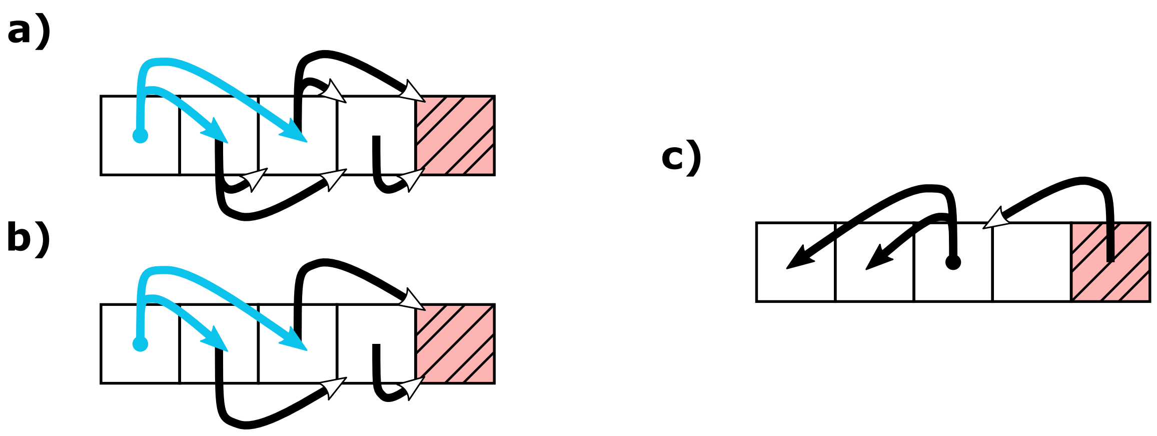

Figure 3.2: **a)** In a forward view, multistep targets can be derived from a dense tree of potential action paths. **b)** To reduce this density, and increase the probability of the inferred actions under the target policies, only the paths with maximal temporal span can be considered (in this scenario $H^a = 2$) **c)** In a backward view, this implies that the immediate targets can be sent H^a steps back at once using a Tree-Backup.

the left-most tile until the red goal-tile with singular steps of 1. If the base discount parameter γ is appropriately set, then this path would be suboptimal under a broad class of reward functions seeing as it would require the largest number of hierarchical actions. Hence, a derived policy would usually opt for a path that is temporally extended, as is illustrated in the bottom-left of Figure 3.2. For this reason, it makes sense to compute multistep returns assuming that the agent will always attempt to select hierarchical actions with a maximal temporal span w.r.t. its atomic horizon H^a.

Analogously, we can adapt this to a backward view Tree-Backup algorithm, as illustrated in the right of Figure 3.2. Following the outline of Algorithm 4, starting from the current transition, $S_t \xrightarrow{A_t} S_{t+1}$, the computed partial returns are propagated backwards with time-jumps of H^a steps. Then, given the final multistep return, each state-action pair within the trailing window $n \cdot H^a$ steps back is updated towards the same return, similarly to Algorithm 3. The Tree-Backup algorithm is in fact essential for computing $\widehat{\mathbf{G}}_{t:t+nH^a}$ as it is necessary to terminate the states that are reached inside the trace.

Algorithm 4 Tree-Backup for Goal-Conditioned Multistep Hier$Q$.

```
**Requires:** A base discount factor $\gamma \in [0,1]$ and a learning rate $\alpha \in (0,1]$.
**Requires:** The number of backup steps $n$, some target policy $\pi$, an environment trace $\tau = \{S_0, A_0, R_1, S_1, A_1, \dots\}$, and the current timestep $t$.

**Update**($i$ :: level, $W$ :: weights, $H^a$ :: environment horizon)
1:  $t_n \leftarrow t - H^a(n-1)$                                   ▷ Sweep $n$ to 1 if $t = T$
2:  **if** $t_n \geq 0$ **then**
3:      $\widehat{\mathbf{G}} \leftarrow \mathbf{r}_{t+1} + \boldsymbol{\gamma}_{t+1} \bigoplus_{g \in \mathcal{S}} \left(\sum_{a \in \mathcal{A}} \pi(a|S_{t+1}, g) Q^W_g(S_{t+1}, a)\right)$
4:      **for** $k = t + 1 - H^a \geq t_n$ with jumps of $-H^a$ **do**
5:          Infer action $A \leftarrow S_{k+H^a}$ **if** $i > 0$ **else** $A \leftarrow A_k$
6:          $\widehat{\mathbf{G}} \leftarrow \mathbf{r}_k + \boldsymbol{\gamma}_k \bigoplus_{g \in \mathcal{S}} \left(\pi(A|S_k, g)\widehat{G}_g + \sum_{a \in \mathcal{A}} \pi(a|S_k, g) Q^W_g(S_k, a)\right)$
7:      Infer action: $A \leftarrow S_{t_n+1}$ **if** $i > 0$ **else** $A \leftarrow A_{t_n}$
8:      **for** $j = 0, 1, \dots, \min(H^a - 1, t_n)$ **do**                ▷ (omit duplicate $S$)
9:          $\boldsymbol{\delta} \leftarrow \widehat{\mathbf{G}} - Q^W(S_{t_n - j}, A)$
10:         $W \leftarrow W + \alpha \left(\boldsymbol{\delta} \otimes \nabla_W Q^W(S_{t_n-j}, A)\right)^\top$      ▷ Equation 2.22
```

3.2.1 An Hierarchical Eligibility Trace

Deriving an efficient online algorithm that updates all hierarchical policies towards the λ-return through eligibility traces is, unfortunately, not as straightforward as the hierarchical generalization to Tree-Backup. For the atomic weights, we can directly apply the method as introduced in Section 2.4. However, for the hierarchical weights this would cause two problems when utilized in this way. 1) If we were to make use of one trace matrix $Z \in \mathbb{R}^{D \times |\mathcal{S}|}$ and decay the trace at every timestep according to Equation 2.24, then all action paths will be decayed as if singular steps were taken. However, if we were to decay the trace every H^a steps, then many of the state-action pairs within the policy's horizon would be decayed prematurely. 2) Another issue is that, each state-action pair within the policy's horizon define a different TD-error w.r.t. the current return. When looking beyond the 1-step update this no longer becomes a problem (as illustrated in the right of Figure 3.2) as all these trailing states congregate to the same path.

To tackle these two problems we firstly propose to utilize an individual trace matrix $Z^{(h)}$ for each atomic horizon step H^a, and secondly to split up the trace updates from the 1-step updates. To ensure that each state is correctly decayed and follows an action-path with maximal temporal span, we circulate through a set of eligibility traces $\{Z^{(0)}, Z^{(1)}, \dots Z^{(H^a)}\}$ by utilizing trace $h = t\%H^a$ at timestep t. Unfortunately this implies that we have to keep track of H^a matrices for each level in the hierarchy, we leave this shortcoming as an avenue for future work. As outlined in Algorithm 5, at timestep t we update the eligibility according to,

$$Z_t^{(i,h)} \doteq \underbrace{\lambda Z_{t-H^a}^{(i,h)} \mathrm{Diag}\left(\boldsymbol{\gamma}_{t-t_{\min}} \bigoplus_{g \in \mathcal{S}} \hat{\pi}_*(A|S_{t-t_{\min}}, g)\right)}_{\text{Interim trace matrix } Z'} + \underbrace{\sum_{j=0}^{t_{\min}} \nabla_W Q^W(S_{t-j}, A)}_{\text{Trailing State-Actions}}, \tag{3.1}$$

where $t_{\min} = \min(H^a - 1, t)$ and A is either A_t for the atomic level ($i = 0$) or S_{t+1} for the hierarchical levels ($i > 0$). The weight matrix is then updated by performing 1-step updates for the trailing state-actions according to Algorithm 3, and performing an eligibility trace update according to Equation 2.21 using the interim trace Z' (left term in Equation 3.1) and the estimation errors $\boldsymbol{\delta}_t$ with maximal temporal span: $\langle S_{t-t_{\min}}, A\rangle$. We call this final algorithm Hier$Q_k(\lambda)$ where k refers to the number of hierarchy levels used, and λ naturally refers to the decay parameter. For $k = 1$ we essentially fall back to a goal-conditioned instance of Watkin's $Q(\lambda)$ or Tree-Backup(λ).

Algorithm 5 Hier$Q_k(\lambda)$ update inspired by Watkin's $Q(\lambda)$ (Algorithm 1)

```
    Requires: A base discount factor $\gamma \in [0,1]$ and a learning rate $\alpha \in (0,1]$.
    Requires: The number of backup steps $n$, some target policy $\pi$, an environment trace $\tau = \{S_0, A_0, R_1, S_1, A_1, \dots\}$, and the current timestep $t$.

Update($i$ :: level, $W$ :: weights, $H^a$ :: environment horizon)
  1: Initialize helper variables $h \leftarrow t \,\%\, H^a$ and $t_{\min} \leftarrow \min(H^a - 1, t)$
  2: Infer action: $A \leftarrow S_{t+1}$ if $i > 0$ else $A \leftarrow A_t$
  3: Estimate returns $\widehat{\mathbf{G}} \leftarrow \mathbf{r}_{t+1} + \boldsymbol{\gamma}_{t+1} \bigoplus_{g \in \mathcal{S}} \max_a \left(W^\top \mathbf{x}(S_{t+1}, a)\right)_g$   ▷ Equation 2.21
  4: $\boldsymbol{\delta} \leftarrow \widehat{\mathbf{G}} - Q^W(S_{t-t_{\min}}, A)$
  5: $Z^{(i,h)} \leftarrow \lambda Z^{(i,h)} \mathrm{Diag}(\boldsymbol{\gamma}_{t-t_{\min}} \bigoplus_{g \in \mathcal{S}} \hat{\pi}_*(A|S_{t-t_{\min}}, g))$   ▷ Interim trace of Equation 3.1
  6: $W \leftarrow W + \alpha Z^{(i,h)} \mathrm{Diag}(\boldsymbol{\delta})$   ▷ Equation 2.25
  7: $Z^{(i,h)} \leftarrow Z^{(i,h)} + \sum_{j=0}^{t_{\min}} \nabla_W Q^W(S_{t-j}, A)$   ▷ Trailing State-Actions of Equation 3.1
  8: Perform 1-step updates according to Algorithm 3
```

4. Algorithm Analysis

As discussed in Section 2.4, the policy specialization for every distinct state allows an agent to split up its environment traces into numerous specialized traces. An interesting implication of this is that it alleviates some of the issues caused by (online) off-policy learning; instead of cutting-traces completely, traces are split up counterfactually. Hier$Q_k(\lambda)$ exploits the resulting *Feudal* structure by learning temporally extended actions over these specialized policies at each level [2, 31]. In this chapter we aim to gather evidence for two hypotheses that attempt to distinguish hierarchical credit assignment from that of conventional, flat, agents.

To articulate our hypotheses, we first make some observations for the flat and hierarchical agents. The trace portrayed in Figure 4.1 exemplifies the differences of credit assignment for a 2-level Hier$Q(\lambda = 1.0)$ agent with a temporal horizon (action budget) of $H = 3$ and a flat Watkin's $Q(\lambda)$ agent in a small artificial gridworld. In this environment, the agent starts out at the center of the left room and has to move to the green tile by moving up, left, right, or down. In this scenario, we observe that the flat agent's greedy policy would strictly follow its previously traversed path (as indicated by the arrows).

The hierarchical policy shows a similar pattern, however its actions are able to jump over multiple timesteps (as indicated by the colored boxes-to-circles on each encountered tile). As a result, the hierarchical policy becomes invariant, to some extent, to transpositions within the environment trace. It can also be seen that the greedy hierarchical policy imposes a rigid structure on the actions, the backed up return from the green tile has a clear temporal separation of $H = 3$ steps. Because of this, the hierarchical agent seems to propose a goal-state that is globally suboptimal at S_0 (as indicated by the black box-to-circle). This is an artifact of the action horizon and can of course be mitigated by terminating intermediate goals when better goals can be realized (as indicated by the red boxes). Another effect on credit assignment caused by the hierarchy is reward discounting, because actions are extended in the environment by a constant H^a, cumulative environment rewards can become significantly undiluted.

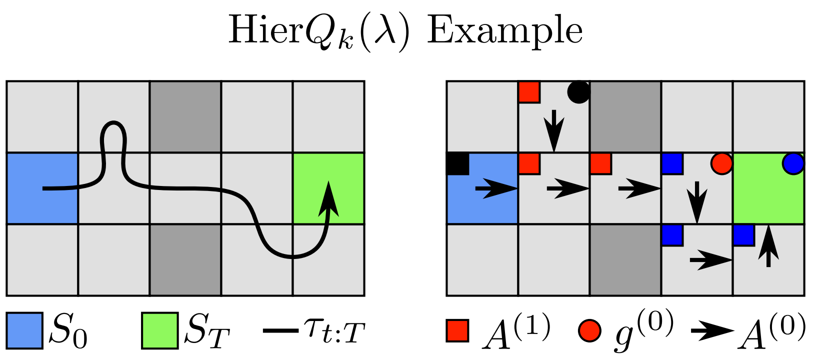

Figure 4.1: Greedy policy comparison of a flat (arrows) and hierarchical agent (boxes-to-circles) on an example environment trace with $\lambda = 1.0$. The hierarchical agent is instructing the flat agent to take two more environment steps to reach the green tile (starting from the center in the left 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).

All-in-all, this begs the question for when and how hierarchy provides significant benefits to credit assignment opposed to a conventional flat agent with *deep* reward propagation; credit assignment over a long temporal horizon without returns becoming zero as a result of repeated exponential discounting or trace-cutting. Hence, in the remaining part of this chapter we perform a variety of experiments and juxtapose results from hierarchical Q-learning agents with similar flat agents. Ultimately, we are interested in the following two (related) hypotheses:

Hypothesis 1): *Hierarchical and flat credit assignment create equivalent policies when discounting, multistep targets, and trace-cutting are properly balanced and sub-optimal goals are terminated.*

This conjecture follows from our observations from Figure 4.1. Though this illustration can't fully portray the difference in policy dimensionality, it is clear that the greedy hierarchical and flat policies resort to the same environment policy based on this trace. When not utilizing any form of *transfer* or *generalization* (assuming the MDP is fixed and we use tabular state features) this begs the question whether this hierarchical framework is any different to a flat agent with deep credit propagation.

Hypothesis 2): *Hierarchical agents benefit or suffer from their temporally extended credit assignment in a similar manner to shifting the λ-return in a flat agent.*

Consider an environment where there are a lot of actions that loop back towards the current states, which we dub *stationary transitions*, for a flat $Q(\lambda)$ agent this will cause the entire eligibility trace to be decayed. The hierarchical agent will instead decay this trace more slowly with a constant factor H^a, the environment horizon. As the Hier$Q_k(\lambda)$ agent discounts and decays its traces exponentially less compared to a flat agent, allowing for more straightforward tuning of the discounting parameter γ, credits are less diluted and less prone to cutting. However, propagating rewards far back like the MC-returns, can cause a larger amount of variance.

Our experiments were executed on six distinct environments varying in scale and difficulty[1], all are illustrated in Figure 4.2. In each experiment, agent-episodes were reset when the terminal state was reached or after 10^4 primitive actions regardless of whether a reward was observed. We focused on stationary and deterministic environments where the rewards were strictly intrinsic (goal-based), this simplified our analyses by allowing fixed learning rates of $\alpha = 1.0$. In principle, our methods extend naturally to non-deterministic environments and extrinsic rewards, but this is not necessary for the current scope (we discuss this in more detail in Section 6.1). In all our experiments, if mentioned, greedy subgoal termination was only utilized during agent *evaluation*, not during training. All statistical repetitions were performed using randomized seeds.

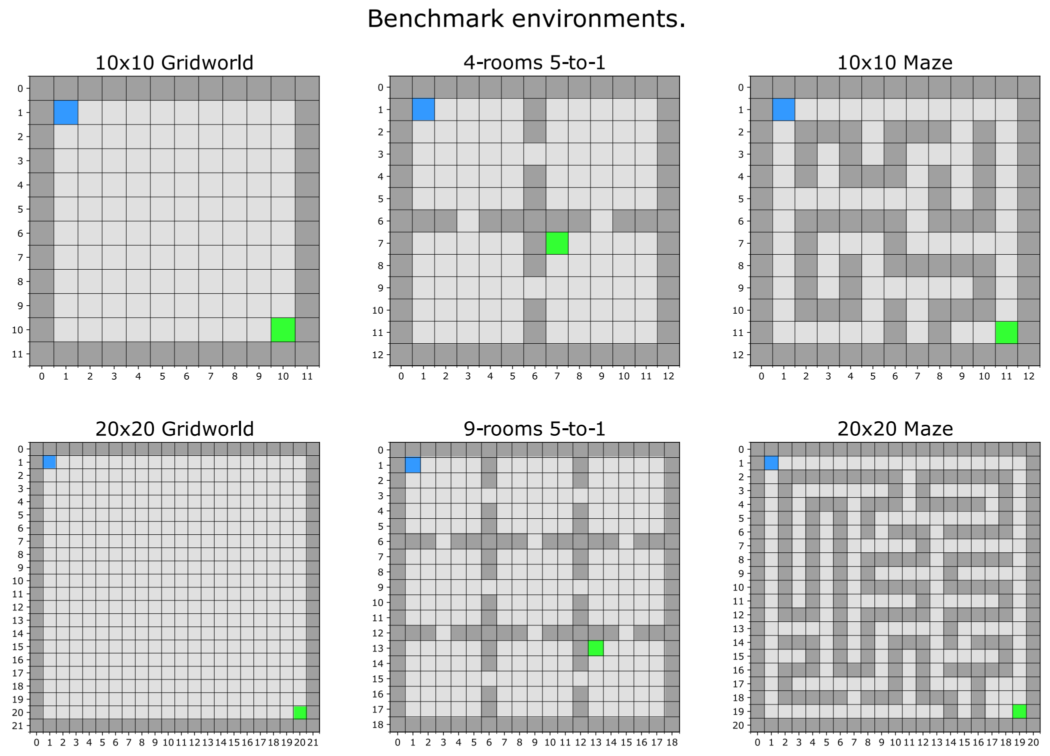

Figure 4.2: The six MDPs we considered in our experiments (start = blue tile; green = goal tile).

[1]The code for our experiments and Hier$Q_k(\lambda)$ implementation can be found at `https://github.com/joeryjoery`.

4.1 Backup Patterns

In our first experiment we utilized multistep and Hier$Q(\lambda)$ on the 10×10 Gridworld of Figure 4.2 for policy evaluation. A random agent that was able to move up, down, right, or left (including potentially moving into walls), generated a set of traces that we used to learn the Q-tables. In this scenario we focus on $k = 2$ levels for brevity and portray state-values as $\bar{V} = \max_a Q^W(s, a, g)$ with g being the environment goal (green tile). For the eligibility trace agent we utilized $\lambda = 0.9$ for both the atomic and hierarchical weights. The multistep agents were balanced in their multistep returns such that the flat n-step return equalled $H \cdot n_{\text{hier}}$ where $n_{\text{hier}} = 3$ and $H = 3$, implying a propagation depth of $n_{\text{flat}} = 9$. All agents utilized a base discount parameter of $\gamma = 0.95$.

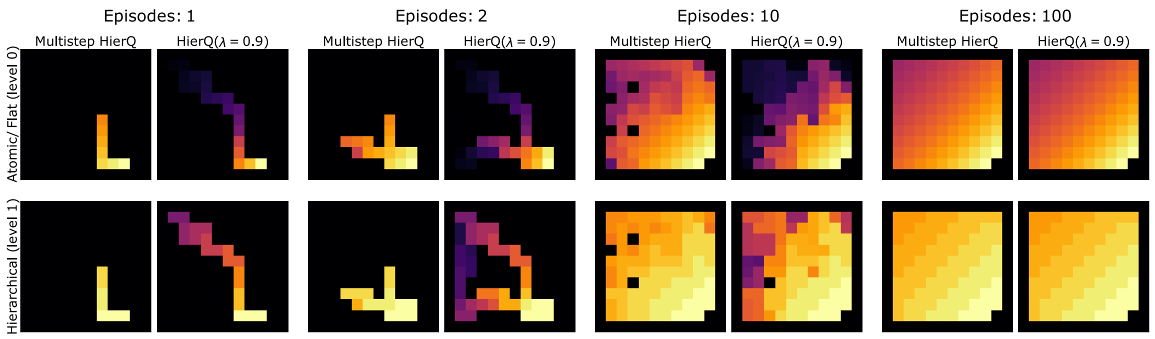

Figure 4.3: Small Gridworld: Backup Patterns for multistep Hier$Q_{k=2}$ and Hier$Q_{k=2}(\lambda = 0.9)$ by using $\bar{V} = \max_a Q^W(s, a, g)$ with $g = S_T$ for every state $s \in \mathcal{S}$; also using $\gamma = 0.95$ and $\alpha = \epsilon = 1.0$. The multistep returns were balanced such that the propagation depth was uniform, $n_{\text{flat}} = 9$ and $n_{\text{hier}} = 3$ using $H = 3$ which resulted in a propagation depth of $H \cdot n_{\text{hier}} = 9$ for the hierarchical policy. The final (near-converged) hierarchical table is a diluted version of the flat table.

Figure 4.3 shows the state value estimates after numerous training episodes for both agents. The top and bottom row separate the atomic and hierarchical weights, a clear distinction between the two is that the final (nearly-converged) value estimates are diluted by a temporal factor $H = 3$. This should of course be evident from the hierarchical TD-update description, but it also becomes very intuitive from the first backup (episode 1).

Over time, it seems that the hierarchical and atomic value estimates learned at a similar pace. A notable observation is the gaps in the value estimations at the tenth episode, comparing only the flat and hierarchical *multistep* agents. The hierarchical agent seems to have filled one gap in the value estimates here, a possible explanation is that this is a result of temporally extended bootstrapping of the hierarchical agent or trace-cutting in the flat agent. Like flat atomic and compounded returns, the λ-return based agent seemed to learn more incrementally on the atomic level, this is not as clear on the hierarchical level. Finally, the near-converged value estimates (episode 100) are only slightly off from their true value in the top-right and bottom-left corners of the hierarchical weights, the atomic value estimates illustrate the random policy's true value. Interestingly, these slight aberrations were a consistent result over numerous runs, this could allude to an exploration difficulty when increasing policy hierarchies and horizons H, we return to this observation in the Discussion.

4.2 Reward Propagation Balancing

Our second experiment sought to quantitatively illustrate the effect of hierarchy and reward propagation depth. Here we applied multistep HierQ_k for $k = 2$ and flat multistep Q-learning on the smaller benchmark environments (top row of Figure 4.2) over a variety of multistep returns such that the propagation depth and discounting was approximately equivalent. Following the setup for the backup pattern visualization, propagation depth balancing was done by using $n_{\text{flat}} = H \cdot n_{\text{hier}}$ where n_{hier} is the n-step return used at the hierarchical level; we did not filter stationary transitions (see Figure A.2 for the same experiment *with* stationary transition filtering). On top of this, we set the base discount parameter for the hierarchical agent to $\gamma_{\text{hier}} = \gamma^H$ to balance the exponential discounting. Finally, we utilized ϵ-greedy exploration *only* on the flat level, hierarchical levels explored through tie-breaking.

Figure 4.4 shows the results for applying the multistep Hier$Q_{k=2}$ for $n_{\text{hier}} = \{1, 2, 3, 4, \infty\}$ with implied values for $n_{\text{flat}} = \{3, 6, 9, 12, \infty\}$ on the smaller benchmark environments. We applied the agents for 50 training episodes in the environment, following the GPI protocol, and measured the trajectory length of their greedy policies after each episode. The lines and approximation errors in Figure 4.4 show the aggregated lengths over all episodes over 100 repetitions, which is an indicator for how fast the algorithm approaches a good control policy. The top row shows the results for standard multistep Hier$Q_{k=2}$, additionally the bottom row shows a naive version for multistep hierarchical SARSA — this method backs up multistep returns by ignoring off-policy corrections.

The current pattern seems to indicate that the hierarchical agents usually outperformed or were on par with the flat agents, a notable exception is $n_{\text{flat}} = 12$ for the Hier$Q$ agent. However, the standard errors for this exception overlaps between the flat and hierarchical agents and may be due to variance. Interestingly, for the naive hierarchical SARSA agent we see on the 10×10 Gridworld that the balanced propagation depth results also strongly overlap. Furthermore, for the SARSA agent, the effect of the hierarchy seems to be more pronounced for the extreme depths $n_{\text{flat}} = \infty$.

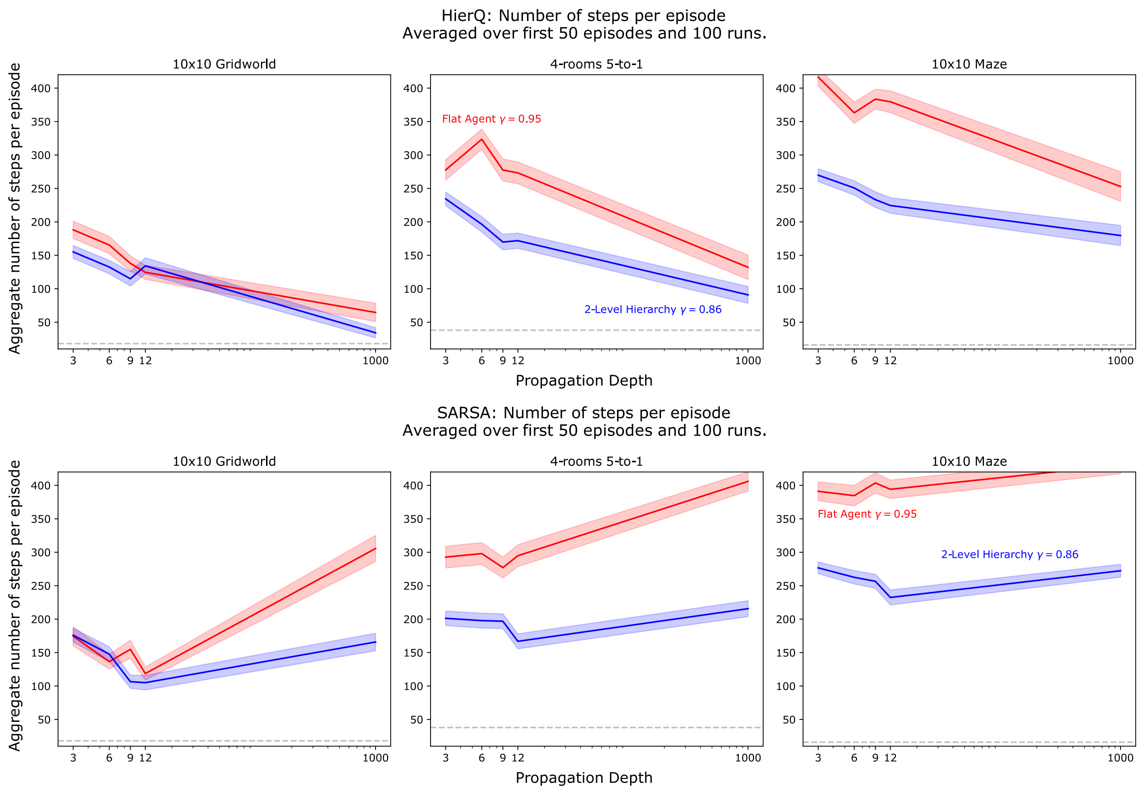

Figure 4.4: Results for the flat Vs. hierarchical reward propagation depth balancing experiment. The top row shows the results for multistep Hier$Q$ and the bottom row for multistep naive Hierarchical SARSA (without off-policy corrections). The x-axis for the Hier$Q$ multistep returns is only an indicator of propagation depth due to trace-cutting inherent in off-policy Tree-Backup, for the SARSA agent the x-axis is exact. The y-axis shows the average greedy policy trace-length aggregated over all repetitions and episodes.

4.3 Performance Ablations

In our third and final experiment we ran multiple instances of multistep HierQ_k and Hier$Q_k(\lambda)$ with varying parameters and hierarchy levels to evaluate their overall performance. We applied multistep HierQ_k for $n \in \{1, 3, 5, 8\}$ and Hier$Q_k(\lambda)$ for $\lambda \in \{0, 0.5, 0.8, 1.0\}$, for both algorithms we then took the Cartesian product with the parameter sets and the maximum number of hierarchy levels $k \in \{1, 2, 3\}$. We utilized a policy horizon of $H^{(i)} = 3$ for all $i = 0, \dots, k-1$. All agent instances were tested over 100 repetitions on all benchmark environments shown in Figure 4.2, the data was collected and aggregated in the same way as in the previous section. We did not terminate intermediary goals if better ones were encountered during testing (see Figure A.3 in the Appendix for the same experiment *with* intermediary goal termination), however, we did utilize stationary transition filtering. Exploration was handled similarly to the previous section.

The hierarchy level was capped at $k = 3$ seeing as the implied atomic horizons would've otherwise exceeded the environment's optimal path; on the 10×10 Gridworld, the optimal has length $|\tau_*| = 18$ whereas the atomic horizon of a policy hierarchy with $k = 4$ and $H = 3$ would imply $H^{(k-1)a} = 27 > |\tau_*|$. Exceeding the horizon of the environment would not make much sense as the environment goal would most often be equal to the top-policy's selected subgoal — i.e., there is no more decomposition of the MDP to simpler subMDPs. To properly evaluate higher levels for k we would require larger environments, but with tabular features this doesn't scale well storage-wise.

The results for this experiment are shown in Figure 4.5, which now shows the number of hierarchies (split by the two distinct algorithms) on the x-axis opposed to the propagation depth of Figure 4.4. The y-axis again shows the aggregate number of steps for the greedy policy to reach the environment goal with approximation errors. On most of the tested environments, we see that the atomic multistep returns are monotonically improved by adding more hierarchy levels; with the exception of the 20×20 Gridworld and for $n = 5$ on the 9-rooms 5-to-1 environment. We also see that on both Gridworld environments that the flat Monte Carlo ($\lambda = 1$) agent seems to be hindered by the additional hierarchy levels. It is notable that we only see significant performance degradation with multiple hierarchies on the larger Gridworld environment. There is also a clear increase in measurement variance on the larger MDPs, this is notable as some of the $\lambda = 0$ and $n = 1$ runs seemed dissimilar in their average performance despite being algorithmically equivalent.

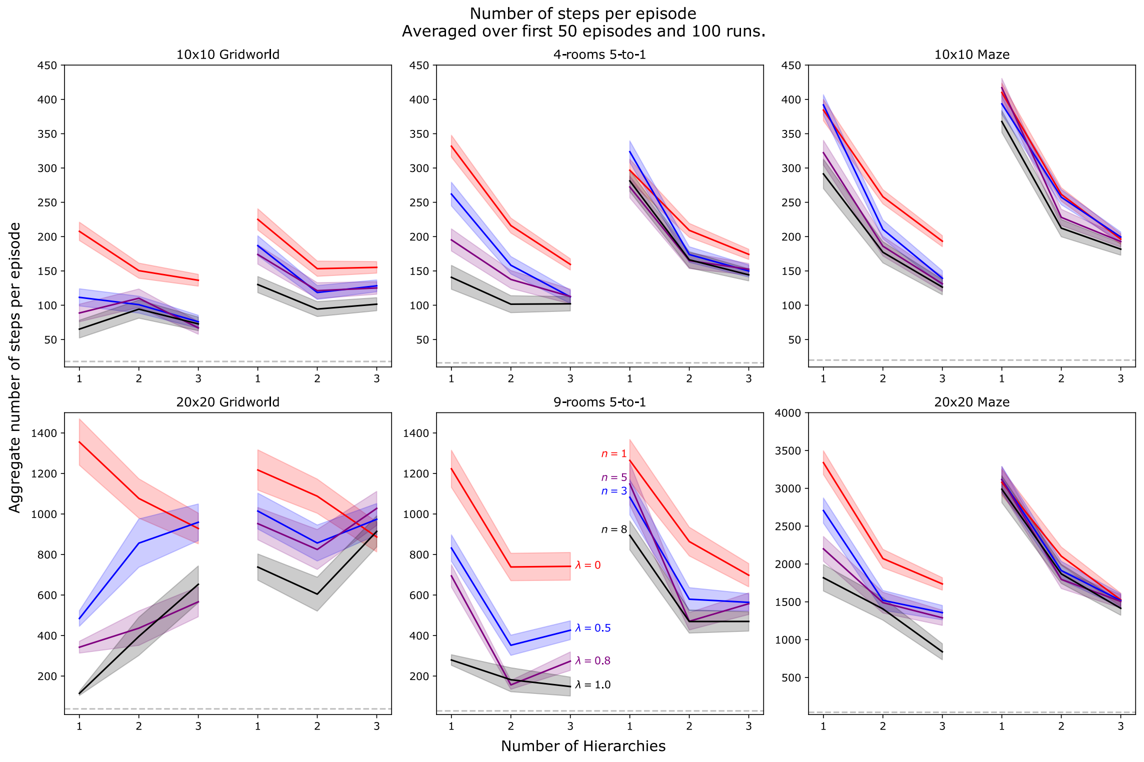

Figure 4.5: Results for the hierarchy ablations. The x-axis denotes per column the level of used hierarchies split between the online λ-returns (left) and the slightly delayed multistep returns (right). The y-axis shows the average number of steps needed to reach the environment goal aggregated over all runs and episodes.

5. Discussion

From our experiments we gathered a few notable observations: hierarchy and flat credit assignment can generate equivalent policies; hierarchy can propagate credits further back compared to flat policies due to omitting trace-cutting steps and diluted discounting; and hierarchy does not necessarily supersede flat agents when training in similar environments. This chapter discusses our experimental observations and relates them back to our conjectured hypotheses. Furthermore, we propose multiple avenues for further investigation and how our analysis should be extended.

Hierarchical and flat credit assignment do generate distinct policies (Hypothesis 1):

Although this result may sound obvious, the fact that all hierarchy levels for the agent are learned by deriving actions from the same trace raised the question on whether a hierarchical agent would generate different effective policies from training than a flat agent, as noted in Figure 4.1. The results for the extreme values of n_{flat} provide strong evidence that the effective environment policy for an hierarchical agent is part of a multivariate equation rather than just the reward propagation depth. Though, a clear difference in reward assignment between flat and hierarchical agents is the possible omission of trace-cutting, the results from the naive Hierarchical SARSA agent show that reward backup depth isn't necessarily the explaining factor for the agents' performance.

A simple explanation for the observed differences in performance is that the hierarchical and flat training procedures are different, the hierarchical agents did not terminate intermediate goals during training (only during evaluation). During training, the hierarchical agents could sample environment policies that did not see the environment goal as their specialized goal. One subtle effect of this is that the goal specialized policies can become very effective in reaching their goal-states very rapidly, especially when these states have a high probability of being observed (e.g., states that lie close to the MDP's starting state) and are updated with hindsight learning. When the hierarchical policies have not yet observed a reward or are highly suboptimal, it is possible that the agent gets stuck following the wrong goals. As these specialized goal-policies become so effective in reaching these suboptimal goals, the agent may not encounter novel states sufficiently often. The policy budgets exacerbate this issue as the hierarchical policies will receive H^a fewer opportunities to sample actions (explore) opposed to the flat agents. In summary, exploration in the environment is organized in an inherently different manner in the hierarchical agents, ultimately resulting in significantly distinct training data compared to a flat agent.

It is of course possible to terminate subgoals during training, at the cost of some additional computation, but this still doesn't guarantee that the hierarchical and flat agents will produce equivalent environment policies due to the trace-cutting differences. Our hypothesis, thus, holds by definition despite in some scenarios producing similar observations. It would be interesting to extend our analysis by quantifying how intermediary goal termination would affect control performance within our framework.

Hierarchy and deeply propagated rewards (Hypothesis 2):

This leads us directly to our second hypothesis, on how deeply propagated rewards influence the quality of the effective environment policies for hierarchical and flat agents. On the more constrained environments (small and large rooms and maze environments) we observed that additional levels of hierarchy generally resulted in improved or non-deteriorated performance, whereas we saw an opposite pattern on the open gridworld environments. Due to the stationary transition filtering and diluted exponential discounting this would imply that rewards would be propagated even further back on the constrained environments compared to the open gridworld, which contradicts our initial hypothesis. Yet, this does not explain the poor performance of the hierarchical agents on the gridworld environments.

Relating back to the first hypothesis, we surmised that the overall performance was confounded by the differences in exploration for hierarchical and flat agents and that the actual reward backups differ in how they cut traces. Because suboptimal goals are not terminated during training, it is likely that this also confounded the results in this experiment. However, we believe that there is another explanation at the core of this phenomenon that relates to the effective atomic horizon of the hierarchical agents. Hypothetically, when the maximum atomic horizon encapsulates the agent's goal it will most likely result in superior performance to flat agents. This is motivated by the fact that the performance decrease was less pronounced on the small gridworld opposed to the large gridworld experiment. In essence, the rooms environments consist of multiple (smaller) gridworld environments in sequence to one another. When the atomic horizon does not sufficiently encapsulate the generated traces within the environment, the agent may suffer during training for the same reasons as for our first hypothesis: the agent may generate and train on suboptimal paths and strictly adhere to these suboptimal goals during training without the option of intermediary termination.

If the atomic horizon is sufficiently large, and encapsulates the environment goal, the hierarchical agent will not incur deterioration's in its performance as the subgoals will amount to the environment goal. This disregards the fact that the point of smaller goal horizons is to guide the agent with incremental goals as to reduce the drastic sparsity of this environment goal. Relating this back to our hypothesis about reward propagation depth and hierarchy, the hierarchical agents most likely behave exactly the same with regards to the λ-parameter as a flat agent, but on a different temporal scale. However, when rewards are propagated within the atomic horizon, this effect is either positive or unaffected. The decreases in performance that we observed in our experiments are strongly linked to the exploration differences and the lack of goal termination capabilities, rather than *purely* the depth of the reward propagation.

5.1 Concerning Exploration

As alluded to earlier, the fact that we utilize hierarchy policies that sample goal-states from (subsets of) $\mathcal{S}$ implies that there are many more actions to consider for exploration. Munos et al. [18] showed that Watkin's $Q(\lambda)$ algorithm, which generalizes standard Q-learning, converges to q_* with probability 1 as a corollary as long as all state-action pairs are tried sufficiently often in the limit — which is achieved for example by a context independent ϵ-greedy policy.

Through bounding of the action spaces, as explained in Section 3.1.1, we essentially eliminate all impossible actions which gives us better guarantees for convergence as exploration is focussed on only the possible states. Nevertheless, when moving up through the hierarchy we get an exponentially increasing atomic horizon, which in turn implies a considerable growth in the number of possible actions. With hierarchical policies the action-space grows proportionally to this temporal budget H^a, meaning that more actions have to be explored to guarantee convergence to q_*. Though each trace allows updating multiple state-action pairs, this is only linear in H^a whereas the action space can grow quadratically to H; for example a l_1 bounded action space within a 2-dimensional lattice, $\mathbb{Z}^2$, defines the Moore neighborhood and grows quadratically according to $(2H^a + 1)^2$.

When excluding feature generalization from the agent, it is evident that sufficient exploration will become more costly under the current framework. Yet, this resulting behaviour is somewhat strange, as the agent could still still visit every *state* a sufficient number of times, have converged flat Q-estimates, but still have poor hierarchical Q-estimates. This phenomenon was commonly observed, and is to a lesser extent illustrated in Figure 4.3 (in the 100th training episode in the bottom-left and top-right corners of the hierarchical V-estimates). Future work could investigate how to counterfactually distill knowledge from multiple levels of hierarchy in order to refine the current estimations of higher level policies, promising approaches include but are not limited to experience replay [16] or model-based methods like Dyna [27, 21].

6. Related Work

The pursuit for temporal abstraction in reinforcement learning agents has persisted for decades [28, 27] and in principle allows autonomous agents to scale much better in practical applications. Various benefits and caveats of hierarchy have already been described, recent work by Wen et al. [34] gives a good overview of this body of work and they also discuss theoretical benefits to HRL for model-based agents. Other recent algorithms within HRL that are similar to the approach by Levy et al. [15], and in turn to ours, are the HIRO algorithm [19] which also makes use of goal-state based subpolicy specialization, and various option oriented methods [14, 27, 30]. As will be discussed in the following section, our method closely resembles intra-option based learning but differs in the fact that all hierarchy levels are learned independently from one another. Of course, there are numerous other analogs to our work, such as the MAXQ method by Dietterich [7], but we do not provide an exhaustive enumeration of every such method here. To the best of our knowledge, this work is the first to investigate the interplay of reward propagation between hierarchical and flat agents.

The following sections discuss related work that is of interest to our method and analysis but are disparate to our discussion regarding the experimental results. We focus on interesting relations with other algorithms along with potential avenues for future work.

6.1 The Options Framework

Perhaps the most well known approach for Hierarchical Reinforcement Learning revolves around the Options framework of Sutton et al. [28]. Options can be described as a tuple $\mathcal{O} = \langle \pi_o, \mathcal{I}_o, \mathcal{B}_o \rangle$, where π_o is the atomic/ environment policy, $\mathcal{I}_o \subseteq \mathcal{S}$ is this option's initiation set, and $\mathcal{B}_o : \mathcal{S} \to [0, 1]$ is a termination function. In essence, the Hierarchical Q-learning framework that we considered in this work is a case of the options framework with some minor technical alterations.

First of all, in our framework the termination function $\mathcal{B}_o$ depends on both past and present events such as, an exceeded time-limit $H^{(i)}$, a reached goal-state, or the presence of a better option, this makes our policy over options *semi-markov* [28]. Furthermore, when not using action space bounding, we can consider $\mathcal{I}_o \equiv S$ for all options. Of course, we only considered bounded action spaces $\mathcal{I}_o \subseteq S$ in our analyses, which can be extended to the concept of an initiation set by also bounding the policies over $\mathcal{I}_o$. In our scenario we instantiate all options *a priori* to specialize in reaching one particular state in the environment, and this can be stacked to any arbitrary level of temporal abstraction.

A key difference is that the value functions for all option policies are assumed to be instantiated simultaneously, though only one policy is actually followed in the environment (hindsight learning), and following their own distinct reward function (goal-conditioning). On another note, view the Q-function over options, as described by [28], through,

$$q_\pi(s, o) \doteq \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots \mid \mathcal{E}(o\pi, s, t)], \tag{6.1}$$

where $\mathcal{E}(o\pi, s, t)$ denotes the event of option o being instantiated at state s and time t and following some policy π. We can redefine our multistep return algorithms, assuming a static discount parameter γ for now, by rewriting Equation 6.1 as:

$$q_\pi(s, o) \doteq \mathbb{E}[G_{t:t+h} + \gamma^h G_{t+h:t+2h} + \cdots + \gamma^{(n-1)h} G_{t+(n-1)h:t+nh} + \gamma^{nh} v_\pi(s) \mid \mathcal{E}(o\pi, s, t)], \tag{6.2}$$

where n is the multistep return parameter and $h \equiv H^a$ is the atomic horizon of this policy (assuming that the episode does not end intermediately). In this scenario our q_π function is not goal-conditioned, and we replaced the goals g with options o (goals are in fact options with a defined terminal state).

Jain et al. [14] also unify option algorithms with eligibility traces in a way that is similar to ours. Their method also makes use of intra-option learning, however, a strong difference is that during updates they consider the options that are actually being followed. Our presented framework is simpler in this sense, as the followed options are inferred from the environment trace (importance sampling ratios become equal to one) and are independent of $\mathcal{B}_o$ when performing hindsight updates — this is done by assuming $\mathcal{B}_o = 1, \forall o$, which implies that no options are currently being executed. Their method is simpler as they do not consider the fixed temporal horizons inside their eligibility traces for backing up rewards along the path of maximal temporal span. This allows them to only use one eligibility trace per hierarchy level. Simultaneously, their intra-option model does not allow each hierarchy level to train independently from eachother despite these benefits, they also did not investigate multilevel hierarchies. Hence, we consider it worthwhile to further investigate the intersection between our method with that of Jain et al. [14]. Potentially, this can ground Hier$Q_k(\lambda)$ more strongly in the ubiquitous options framework and considerably improve its practical aspects.

6.2 Goal Discovery, Dimensionality, and Successor Features

While our work extends the framework by Levy et al. [15] and as discussed in the previous section is closely related to the work by Jain et al. [14], the basic hierarchical framework draws from work by Bakker et al. [2] — which they dubbed as the HASSLE algorithm. The HASSLE algorithm is essentially extended by Levy et al. [15] by including hindsight experience replay [1] and explicit goal-conditioning following the principles of Universal Value Functions [23]. One particular feature that was implicitly left out by Levy et al. [15] is that subgoals are autonomously discovered by the algorithm rather than being specified *a priori*, for example by letting the goal space $\equiv \mathcal{S}$. Of course, it is possible to 'learn' the agent's goal space as novel states are discovered, it naturally leads to a considerably large set of goals and resulting goal-specialized policies.

While we did not consider feature approximation in this work, the results by [14] and our definition of the eligibility traces in Section 3.2.1 show no indication for why it shouldn't work with gradient methods. Note though, that Watkin's $Q(\lambda)$ and gradient TB(λ) are unstable under gradient methods [29]. Furthermore, it is also likely that the accumulate trace of Equation 3.1 will result in divergence for large learning rates; future work could look at alternative traces like *dutch-traces* for this [24]. An inherent limitation of letting the goal/ hierarchical-action spaces be a direct subset of the statespace implies that the dimensionality of $W^{(i)}$ is at least a multiple of $|\mathcal{S}|$ in some way. Following the options framework we can limit the weights only to the relevant subset for some option according to $\mathcal{I}_o$, however, this does not scale when we move up through the hierarchy.

Potentially, for discrete and finite $\mathcal{S}$, future work could look at aggregations over $\mathcal{S}$ to define goals. For example, a goal could be to reach a state within a subset of $\mathcal{S}$ rather than reaching the actual state, the precision for reaching particular states can become more granular when one moves down in the hierarchy. When paired with subgoal discovery like in the HASSLE algorithm [2], this could potentially allow for a scalable way to jointly learn arbitrary levels of hierarchy.

This directly leads us to the next point, when incorporating subgoal discovery over a limited subset of the state space, we also have to initialize new subgoal policies. The Successor Features (SF) framework could pose as an interesting avenue for knowledge transfer from the set of current policies to a new subgoal policy [3, 5, 17]. In short, SFs assume that the reward function R_{t+1} can be decomposed as,

$$R(S_t, A_t, S_{t+1}) \doteq \phi(S_t, A_t, S_{t+1})^\top \mathbf{b}, \tag{6.3}$$

where $\mathbf{b}$ is some learned (or known) weight vector, and $\phi : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^D$ are environment features for the transition $s \xrightarrow{a} s'$. The idea then is to model the Q-values as an inner product between the future discounted features and the weight vector $\mathbf{b}$,

$$\begin{aligned} q_\pi(s,a) = \mathbb{E}[G_t \mid S_t = s, A_t = a, \pi] &= \mathbb{E}\left[\textstyle\sum_{i=1}^{T-t} \gamma_{t+i}^{i-1} R_{t+i} \mid S_t = s, A_t = a, \pi \right] \\ &= \mathbb{E}\left[\textstyle\sum_{i=1}^{T-t} \gamma_{t+i}^{i-1} (\phi_{t+i})^\top \mathbf{b} \mid S_t = s, A_t = a, \pi \right] \\ &= \mathbb{E}\left[\textstyle\sum_{i=1}^{T-t} \gamma_{t+i}^{i-1} \phi_{t+i} \mid S_t = s, A_t = a, \pi \right]^\top \mathbf{b} = \boldsymbol{\psi}_\pi(s,a)^\top \mathbf{b}. \end{aligned} \tag{6.4}$$

The estimates in $\boldsymbol{\psi}_\pi$ are known as the successor features for any (s,a) under π, in the tabular case these are simply the discounted visit counts under some policy π. Goal-conditioning and the corresponding intrinsic rewards as considered in this work can be seen as a special case of SFs where $\phi(S_t, A_t, S_{t+1}) = \mathbf{x}(S_{t+1})$ and the weights $\mathbf{b}$ are rows of the indicator matrix $I_{|\mathcal{S}|}$. A nice theoretical result by SFs is the *Generalized Policy Improvement* theorem [3], which is an extension of Generalized Policy Iteration [27], and shows that new policies can be derived by maximizing over all currently available subgoal policies based on a new reward vector for $\mathbf{b}$. Other approaches towards this direction can also rely on meta-learning to get a strong novel subgoal policy initialization, for gradient methods one can consider the work by Finn et al. [10]. These methods can considerably speed up learning of the lower levels in the hierarchy, which will in turn enable the agent to actually exploit the hierarchical policy structure during training.

7. Conclusion

In this work we developed a method for implementing multistep returns for hierarchical control algorithms and investigated the interplay of temporally extended credit assignment with the depth of the reward propagation along the environment trace. We have contrasted this with conventional multistep return based methods, either with atomic or compound based returns, and documented how learning fundamentally differs when incorporating hierarchy.

When agents are unable to exploit other commonly documented benefits of hierarchy, such as transfer and generalization over temporally abstracted actions [34], we found that hierarchy can still provide strong benefits compared to flat agents purely through more effective reward assignment. Hierarchy allows the agents to send rewards back (or evaluate by looking forwards) over multiple timesteps in the environment trace without the detriments of conventional multistep algorithms which have to compute unstable importance sampling ratios or dilute/ cut traces with off-policy data. In our framework, the agents learn from the temporally extended actions by looking over potentially suboptimal actions and viewing each timejump within some horizon as a singular action. The hierarchical agents then utilize these extended actions with conventional multistep return algorithms by backing up the rewards only through the path of maximal temporal span. Compared to previous work [14], this allowed our hierarchical Q-learning based agents to actually supersede their flat counterparts in most of our tested environments.

Interestingly, we found that hierarchy started to underperform on the larger open grid environments which we conjectured was a result of strict adherence to suboptimal goals. In other words, faulty actions for the hierarchical agents are expected to lead to a stronger negative impact on control performance, when measured in terms of the environment policy, opposed to actions on a more granular temporal scale. We found that intermediate subgoal termination can alleviate this issue during evaluation, even occasionally leading to equivalent environment policies compared to flat agents. However, this still leads to a distinct dataset being generated during training. Along with the other proposed avenues for future work, our analysis could be extended to focus on how to explore the hierarchical action spaces during training.

Bibliography

[1] Marcin Andrychowicz et al. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[2] 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*. 2004, pp. 438–445.

[3] Andre Barreto et al. "Successor Features for Transfer in Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[4] Richard Bellman and Rand Corporation. *Dynamic programming*. OCLC: 526246. Princeton: Princeton University Press, 1957.

[5] Diana Borsa et al. "Universal Successor Features Approximators". In: International Conference on Learning Representations. Sept. 27, 2018.

[6] Kristopher De Asis et al. "Multi-step Reinforcement Learning: A Unifying Algorithm". In: *arXiv:1703.01327 [cs]* (June 11, 2018). arXiv: `1703.01327`.

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

[8] Markus Dumke. "Double Q(σ) and Q(σ, λ): Unifying Reinforcement Learning Control Algorithms". In: *arXiv:1711.01569 [cs, stat]* (Nov. 5, 2017). arXiv: `1711.01569`.

[9] Adrien Ecoffet et al. "First return, then explore". In: *Nature* 590.7847 (Feb. 2021). Number: 7847 Publisher: Nature Publishing Group, pp. 580–586. ISSN: 1476-4687. DOI: `10.1038/s41586-020-03157-9`.

[10] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *arXiv:1703.03400 [cs]* (July 18, 2017). arXiv: `1703.03400`.

[11] Hado van Hasselt, A. Rupam Mahmood, and Richard S. Sutton. "Off-policy TD(λ) with a true online equivalence". In: *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*. UAI'14. Arlington, Virginia, USA: AUAI Press, July 23, 2014, pp. 330–339. ISBN: 978-0-9749039-1-0.

[12] Hado van Hasselt et al. "Deep Reinforcement Learning and the Deadly Triad". In: *arXiv:1812.02648 [cs]* (Dec. 6, 2018). arXiv: `1812.02648`.

[13] Hado van Hasselt et al. "Expected Eligibility Traces". In: *arXiv:2007.01839 [cs, stat]* (Feb. 8, 2021). arXiv: `2007.01839`.

[14] Ayush Jain and Doina Precup. "Eligibility Traces for Options". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '18. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, July 9, 2018, pp. 1008–1016.

[15] Andrew Levy et al. "Learning Multi-Level Hierarchies with Hindsight". In: International Conference on Learning Representations. Sept. 27, 2018.

[16] Long-Ji Lin. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching". In: *Machine Learning* 8.3 (May 1, 1992), pp. 293–321. ISSN: 1573-0565. DOI: `10.1023/A:1022628806385`.

[17] Chen Ma et al. "Universal Successor Features for Transfer Reinforcement Learning". In: *arXiv:2001.04025 [cs, stat]* (Jan. 4, 2020). arXiv: `2001.04025`.

[18] Rémi Munos et al. "Safe and Efficient Off-Policy Reinforcement Learning". In: *arXiv:1606.02647 [cs, stat]* (Nov. 7, 2016). arXiv: `1606.02647`.

[19] Ofir Nachum et al. “Data-Efficient Hierarchical Reinforcement Learning”. In: *arXiv:1805.08296 [cs, stat]* (Oct. 5, 2018). arXiv: `1805.08296`.

[20] Jing Peng and Ronald J. Williams. “Incremental multi-step Q-learning”. In: *Machine Learning* 22.1 (Mar. 1, 1996), pp. 283–290. ISSN: 1573-0565. DOI: `10.1007/BF00114731`.

[21] Silviu Pitis. “Source Traces for Temporal Difference Learning”. In: *arXiv:1902.02907 [cs, stat]* (Feb. 7, 2019). arXiv: `1902.02907`.

[22] Doina Precup, Richard S. Sutton, and Satinder P. Singh. “Eligibility Traces for Off-Policy Policy Evaluation”. In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML ’00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., June 29, 2000, pp. 759–766. ISBN: 978-1-55860-707-1.

[23] Tom Schaul et al. “Universal Value Function Approximators”. In: *International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 1938-7228. PMLR, June 1, 2015, pp. 1312–1320.

[24] Harm van Seijen et al. “True Online Temporal-Difference Learning”. In: *arXiv:1512.04087 [cs]* (Sept. 8, 2016). arXiv: `1512.04087`.

[25] Harm van Seijen et al. “A Theoretical and Empirical Analysis of Expected Sarsa”. In: *Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning: ADPRL* (2009).

[26] Richard S. Sutton. “Learning to predict by the methods of temporal differences”. In: *Machine Learning* 3.1 (Aug. 1988), pp. 9–44. ISSN: 0885-6125, 1573-0565. DOI: `10.1007/BF00115009`.

[27] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction*. second edition. Cambridge, Massachusetts: Bradford Books, Nov. 13, 2018. 552 pp. ISBN: 978-0-262-03924-6.

[28] Richard S. Sutton, Doina Precup, and Satinder Singh. “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: *Artificial Intelligence* 112.1 (Aug. 1, 1999), pp. 181–211. ISSN: 0004-3702. DOI: `10.1016/S0004-3702(99)00052-1`.

[29] Ahmed Touati et al. “Convergent Tree Backup and Retrace with Function Approximation”. In: *arXiv:1705.09322 [cs]* (Oct. 22, 2018). arXiv: `1705.09322`.

[30] Vivek Veeriah et al. “Discovery of Options via Meta-Learned Subgoals”. In: *arXiv:2102.06741 [cs]* (Feb. 12, 2021). arXiv: `2102.06741`.

[31] Alexander Sasha Vezhnevets et al. “FeUdal Networks for Hierarchical Reinforcement Learning”. In: *arXiv:1703.01161 [cs]* (Mar. 6, 2017). arXiv: `1703.01161`.

[32] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning”. In: *Machine Learning* 8.3 (May 1, 1992), pp. 279–292. ISSN: 1573-0565. DOI: `10.1007/BF00992698`.

[33] Christopher John Cornish Hellaby Watkins. “Learning From Delayed Rewards”. PhD thesis. Cambridge, UK, Jan. 1, 1989.

[34] Zheng Wen et al. “On Efficiency in Hierarchical Reinforcement Learning”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6708–6718.

A. Supplementary Results

In the following sections we outline additional results that complement our analyses but did not directly fit into the main narrative. Each section enumerates one distinct additional result.

A.1 Comparison to Original Algorithm

In order to compare our algorithm extensions to the original framework as outlined by Levy et al. [15], we empirically compared the 1-step algorithms either with or without action space bounding and a binary or penalizing reward function. We tested all our methods on the same benchmarking environments as illustrated in Figure 4.2. We refer to the unbounded action space agents as $\mathcal{A} \equiv \mathcal{S}$ and the bounded action spaces as $\mathcal{A} \subseteq \mathcal{S}$. The binary reward function is denoted as the indicator $\mathbf{r}_{t+1} = \mathbf{1}_{S_{t+1}}$ where the alternative, penalizing, reward function is given by $\mathbf{r}' = 1 - \mathbf{r}$. Other parameters are given as $\alpha = 1, \gamma = 0.95, \epsilon_{k=0} = 0.25, \epsilon_{k>0} = 0$ where we made use of stationary transition filtering and greedy goal termination during evaluation (not training).

Figure A.1 provides the resulting data from our algorithm comparison. The x-labels indicate the number of hierarchies $k \in \{2, 3\}$ that we tested, and the y-axis provides the number of steps needed to reach the goal for the current greedy environment policy averaged over the first 50 training episodes and shown as a distribution by the violin shapes over 100 random seeds. We see that on all environments that the penalizing reward function ([15]) induces significantly stronger variance and worse expected performance compared to the binary reward function (ours). Furthermore, the action-space bounding further decreases this variance and improves expected control performance (ours). However, for the binary reward function there is little to no difference for the impact of the bounded action space.

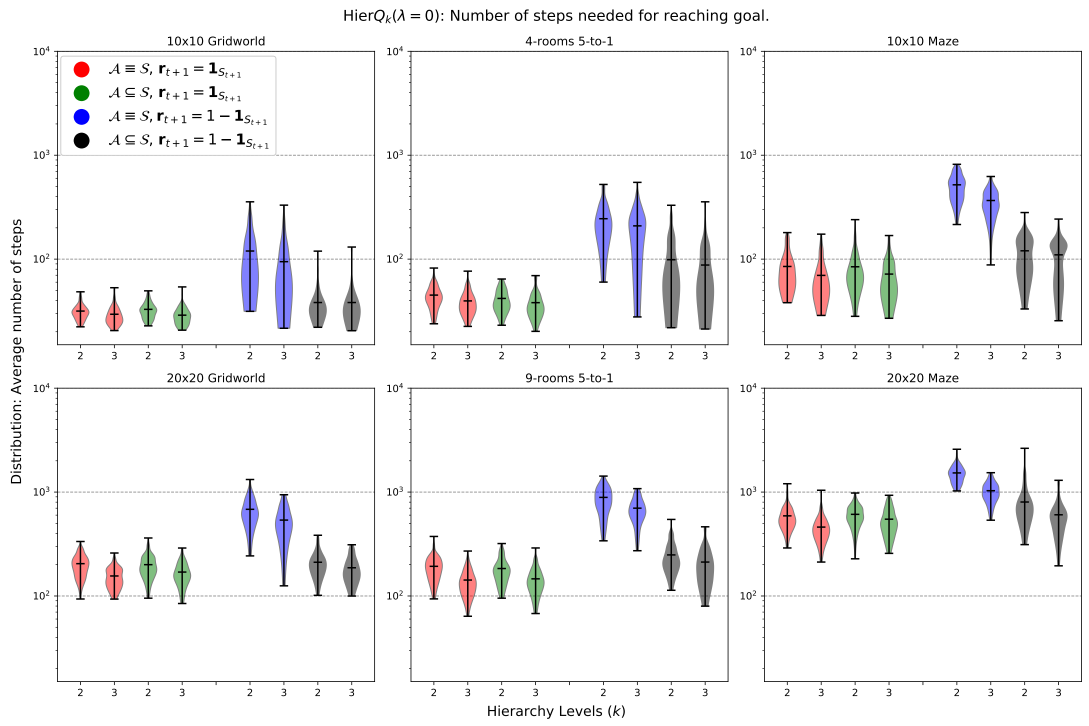

Figure A.1: Comparison of our framework for Hier$Q_k(\lambda = 0)$ to the canonical method of Levy et al. [15]. The x-axis shows the number of hierarchy levels k, split by the reward function and as indicated in the legend. The y-axis shows the aggregate number of steps to reach the environment goal of the greedy policy over the first 50 training episodes, the violinplots illustrate their distribution over 100 random seeds.

A.2 Algorithm Analysis: Additional Ablations

This section shows additional ablation results for Chapter 4. For the propagation balancing experiment, for both multistep HierQ_k and naive hierarchical SARSA, we show the experimental results when utilizing stationary transition filtering for training the hierarchical policies in Figure A.2. Then, for the performance ablations, we show the additional results from Figure 4.5 for the exact same experimental setup *with* greedy goal termination during policy evaluation in Figure A.3.

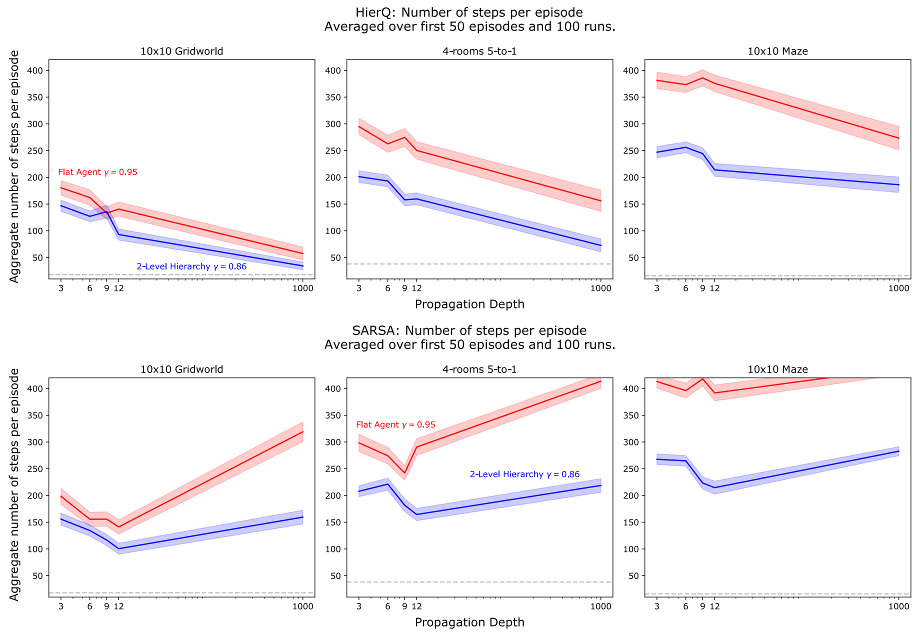

Figure A.2: Results for the flat Vs. hierarchical reward propagation depth balancing experiment, as shown in Figure 4.4, with filtering of stationary transitions during training.

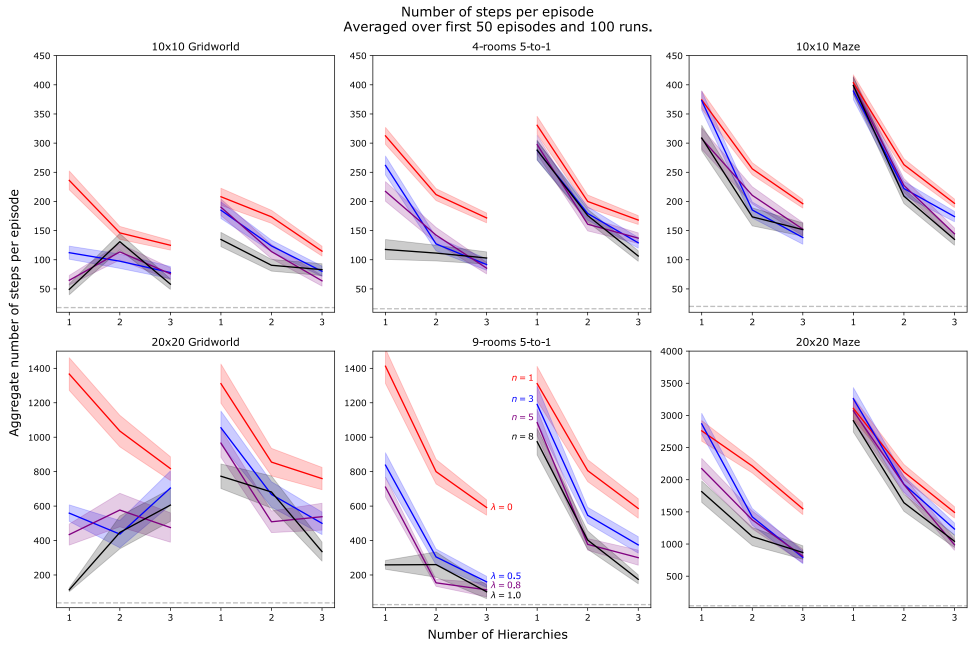

Figure A.3: Results for the hierarchy ablations, as shown in Figure 4.5, with greedy goal termination.