GPI-Tree Search: Algorithms for Decision-time Planning with the General Policy Improvement Theorem

Louis Bagot, Lynn D'eer, Steven Latré, Tom De Schepper, Kevin Mets
Department of Computer Science, University of Antwerp – imec, Sint-Pietersvliet 7
2000 Antwerp, Belgium
firstname.lastname@uantwerpen.be

ABSTRACT

In Reinforcement Learning, Unsupervised Skill Discovery tackles the learning of several policies for downstream task transfer. Once these skills are learnt, the question of how best to use and combine them remains an open problem. The General Policy Improvement Theorem (GPI) creates a policy stronger than any individual skill by selecting the highest-valued policy at each timestep. However, the GPI policy is unable to mix and combine the skills at decision time to formulate stronger plans. In this paper, we propose to adopt a model-based setting in order to make such planning possible, and formally show that a forward search improves on the GPI policy and any shallower searches under some approximation term. We argue for decision-time planning, and design a family of algorithms, *GPI-Tree Search Algorithms*, to use Monte Carlo Tree Search (MCTS) with GPI. These algorithms foster the skills and Q-value priors of the GPI framework to guide and improve the search. Our quantitative experiments show that the resulting policies are much stronger than the GPI policy alone, while our qualitative results provide a good intuitive understanding of how each method works and of the possible design choices that can be made.

1 INTRODUCTION

In Reinforcement Learning (RL, Sutton and Barto [28]), an *agent* interacts with an *environment*, performing an *action* to gather *rewards* in a sequential task, using *states* as input. Within RL, Transfer Learning [31] aims to extract and re-use knowledge from previously learnt tasks for new environments. For example, strong visual navigation is critical for any downstream embodied agent task.

To capitalize on this transfer learning setting, Unsupervised Skill Discovery (USD) aims to learn a set of policies[1] to be used for downstream tasks. USD methods generally try to create policies that reach key states [18, 22] or that maximize skill diversity through a mutual information objective [7, 10]. In a similar line of work, Successor Features (SFs, Barreto et al. [4, 5]) have been proposed to efficiently learn Q-values of policies over a range of tasks. The authors used this property as motivation to introduce the General Policy Improvement Theorem (GPI) to instantly obtain a combined policy stronger than any individual skill. In order to achieve this, the GPI policy simply selects the highest-valued action, according to the skill Q-values, at each time-step. This is visualized on Figure 1: the GPI policy follows the skills when they directly lead to the goal. However, if they do not lead to the goal, their Q-values are null and the GPI policy cannot make an informed decision (represented in red states without arrows). It is clear, though, that a *combination* of the skills could lead to the goal from the upper left quadrant, and that only a few steps are required to join the high-value corridor in the rest of the state-space. In other words, **while the GPI policy improves on the skills, it is unable to plan and combine them to achieve a higher return**. This observation forms the intuitive basis for this paper.

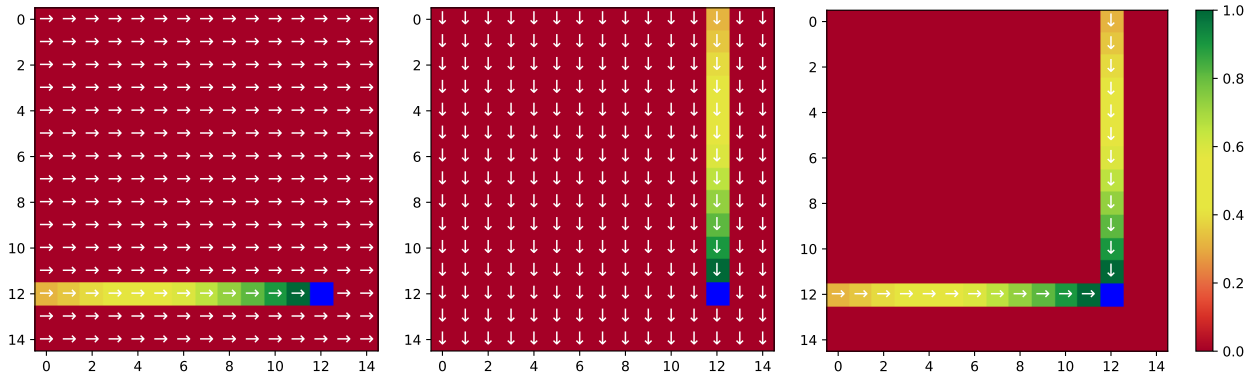
In order to allow the GPI to combine skills, it would need to simulate a few steps of action or skill usage to probe where they lead. We therefore turn to Model-Based Reinforcement Learning (MBRL) to allow such simulations. MBRL either assumes an oracle or learns a model of the environment's dynamics, which allows to learn from the model instead of direct interactions [11, 12, 27]. We refer to learning a policy or value function directly from the model as"backwards planning". In contrast,"decision-time planning" (terms from Sutton and Barto [28]) focuses on the current state s and unrolls the model from there to find the best action to execute, without explicit learning. In the last years, the most notable algorithms in decision-time planning have used the Monte-Carlo Tree Search algorithm [6], leading to major breakthroughs in the field of RL [24, 25].

One of the most interesting properties of the GPI theorem is that it builds a strong policy without any additional learning, which allows for quick transfer. In order to maintain this while improving on the GPI policy, we propose to use a model to perform decision-time planning. Our contributions are the following:

- We propose to improve on the GPI through model-based RL, and decision-time planning in particular. We formally show (Theorem 3.1) that such a search improves on the GPI policy in the max norm, under some approximation term.
- We propose to leverage the skills and Q-value priors of the GPI framework to guide a Monte-Carlo Tree Search towards the most relevant nodes.
- We develop a family of algorithms, GPI-Tree Search, instantiating these ideas through different angles, and we provide quantitative and qualitative results to understand their differences and performances.

[1]"skills" and"policies" will be used interchangeably to refer to the set of behaviors learnt with USD

Proc. of the Adaptive and Learning Agents Workshop (ALA 2023), Cruz, Hayes, Wang, Yates (eds.), May 9-10, 2023, Online, https://ala2023.github.io/. 2023.



(a) $\pi_1(s) =$ **right** $\forall s$ **(b)** $\pi_2(s) =$ **down** $\forall s$ **(c) GPI policy over** $\{\pi_1, \pi_2\}$

Figure 1: Two skills and the associated GPI policy for an open, sparse-reward gridworld environment. The GPI policy is unable to reach the goal (blue) from the upper left quadrant of the world through a combination of the `right` and `down` skills.

2 BACKGROUND

Reinforcement Learning. In RL, interactions are generally formalized with a *Markov Decision Process* (MDP) $M = (\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$, respectively representing the state space, action space, reward space, dynamics function and discount factor. The dynamics function $p(s', r \mid s, a)$ dictates the next state and reward, and therefore generates the *reward function* $r(s, a) = \sum_{s',r} p(s', r \mid s, a)\, r$.

Unsupervised Skill Discovery. USD refers to the learning of a set of policies $\{\pi_1, \pi_2, \cdots\}$ (which can be infinite). The objective is to combine these skills to maximize reward. The learning is often broken down into building task-agnostic skills at first, followed by a transfer learning phase where they are combined to maximize the reward function.

General Policy Improvement. The GPI Theorem [5] provides an efficient way to use a finite skill set. It makes the assumption that we have access to the action-value functions of the skills, Q^{π_j}, to build a stronger policy:

$$\pi^0(s) \in \arg_a \max_{a,j} Q^{\pi_j}(s, a)$$

$$\pi^0 \geqslant \pi_j \; \forall j.$$

For convenience, we define $Q^0(s, a) = \max_j Q^{\pi_j}(s, a)$ and $\pi^0(s) \in \arg\max_a Q^0(s, a)$. Note that this is not the action-value function of π^0: indeed, $Q^{\pi^0} \geqslant Q^0$ according to the GPI theorem itself. Intuitively, if $Q^0(s, a) = 0 = Q_j^{\pi}(s, a) \; \forall s, a, j$, we might still have $Q^{\pi} > 0$ (for example at state (11,11) of Figure 1 where the GPI might randomly take actions `right` or `down`).

We call the assumptions of the GPI theorem –i.e., the skill set and knowledge of their value functions– the *GPI framework*. In this paper, we build onto those assumptions to build stronger policies.

Monte Carlo Tree Search. MCTS [6] is a decision-time planning algorithm. Starting from the current state s_0 as a root node, the algorithm builds a tree of possible outcomes, branching out at the chosen actions. More precisely, each node state s keeps track of variables $Q(s, a)$, an estimate of its state-action value function, and $N(s, a)$, the amount of visits of each actions. It then alternates the following 4 steps: (i) *selection*, climbing down the tree from the root node to a leaf s_l according to a *tree policy* π^{tree}; (ii) *expansion*, creating a new node at the leaf s_l; (iii) *simulation*, generating a return G_l from that leaf, using the model by unrolling a *rollout policy* π^{rollout}; (iv) *backup*, updating Q and N for all the nodes up the tree with the obtained return G for the action taken. This process is repeated for a certain amount of *rollouts* N. The tree policy π^{tree} is most often the Upper-Confidence Bound bandit algorithm (UCB; Agrawal [1], Katehakis and Robbins [19]) or one of its variants, while the rollout policy can either be random or any policy prior with fast inference time. MCTS was designed for deterministic environments, we also follow this setting.

The Options framework . Options [30] refer to temporally extended actions. In Hierarchical RL, the task is broken down into smaller and easier sub-tasks to be solved by lower-level controllers. An option formalizes a lower-level controller: it is defined as a triple $(\mathcal{I}, \pi, \beta)$, where $\mathcal{I} \subset \mathcal{S}$ is the option's initiation set, i.e., states in which the option can initiate; π is the option's policy; and $\beta : \mathcal{S} \to [0, 1]$ is the option's termination condition. A higher-level controller can then call options as if they were actions, substantially cutting down the problem depth.

3 GENERAL ALGORITHMIC FRAMEWORK

One of the main benefits of the GPI theorem is the ability to generate a stronger policy at inference, without additional learning. We therefore aim to improve on the GPI policy using a forward search at inference and without learning. For this reason, we focus our efforts on decision-time planning.

3.1 Q priors

We propose to use the Q^0 assumed by the GPI framework to guide a MCTS tree search. We explain this idea in this section.

Since the goal of the MCTS simulation step is to provide a return estimate G_l at the leaf node, one can note that if we had a value-function prior $\hat{V}$, we could simply use $G_l = \hat{V}(s_l)$ as a backup value and bypass the simulation. This essentially borrows from

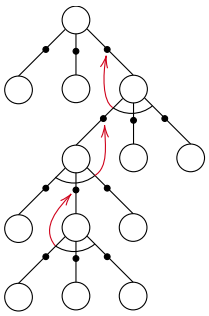


Figure 2: Maximum value backup: the backup value is $\max_a Q(s,a)$ rather than the Monte-Carlo return. The max operator is indicated by the arc over actions, backed up in red arrows to the parent state-action.

Temporal-Difference Learning [26] and was already used in prior work on MCTS to bypass the simulation step [25]. One can see this as turning the"rollout algorithm" that is MCTS into a"heuristic search" [28].

However, if we assume an action-value prior $\hat{Q}$ instead of $\hat{V}$, we can also use it to guide the UCB tree policy. The idea is very similar to using the policy as a prior in the pUCT algorithm used in the AlphaZero family of methods [23, 25]. To understand how, note that the UCB formula is

$$\pi^{\text{tree}} = \arg\max_a \left(Q(a) + cU(a)\right) \tag{1}$$

$$U(a) = \sqrt{\frac{\ln \sum_b N(b)}{N(a)}} \tag{2}$$

A prior $\hat{Q}$ would provide an estimate of Q before any sample is observed, allowing to directly dive into actions that the prior deems superior. This idea is explored in Heuristic MCTS [9] and later SAVE [13], where the authors choose to include this prior at the creation of the leaf node s_l: $Q_0(s_l,a) = \hat{Q}(s_l,a)$, $N_0(s_l,a) = 1$. In other words, the method acts as if the prior was a single observed return for this state-action pair.

We propose to use $\hat{Q} = Q^0$ as a prior action-value function to guide MCTS within the GPI framework. Note that in this scenario, $\pi^{tree}\ |_{c=0} = \pi^0$; in other words, the UCB policy with Q^0 priors collapses to the GPI policy without exploration (or additional observations to change the values).

3.2 Maximum value backup

In the original MCTS [6], the author explores different backup methods for MCTS. The max operator is dismissed for the following reasons:"When the number of moves is high, and the number of simulations is low, move estimates are noisy. So, instead of being really the best move, it is likely that the move with the best value is simply the most lucky move. Backing up the maximum evaluation overestimates the best move, and generates a great amount of instability in the search." This idea aligns with concepts explored in Double Q Learning [15] – the max operator propagates overestimations. However, in the context of MCTS with Q-value priors, the issues of number of moves and simulations is entirely deflected to the quality of the Q estimates. Since the GPI assumes knowledge of the Q^{π_j} with a small error, we opt for usage of the max operator, visualized in Figure 2. This step yet again draws us closer to a heuristic search. This operator allows us to benefit from the guarantees of Theorem 3.1 that we now introduce.

3.3 Improvement guarantees of an exhaustive search

In this section we present our main theoretical contribution: the k-step GPI Theorem, which justifies the usage of a search over the GPI:

Theorem 3.1 (k-step GPI). *Let $\{\pi_1, \cdots, \pi_D\}$ be a set of policies with value functions Q^{π_j}. Let $\tilde{Q}^{\pi_j}$ be their approximations such that $\left|Q^{\pi_j}(s,a) - \tilde{Q}^{\pi_j}(s,a)\right| \leqslant \epsilon$ for all state-action pairs s,a and policies j.*
Define $\Delta\tilde{Q} \doteq \max_j \max_{s,a_1,a_2} \left|\tilde{Q}^{\pi_j}(s,a_1) - \tilde{Q}^{\pi_j}(s,a_2)\right|$.
Let $Q^0(s,a) = \max_j \tilde{Q}^{\pi_j}(s,a)$ be the 0-step GPI Q-prior. Let $\pi^0(s) \in \arg\max_a Q^0(s,a)$ be the 0-step GPI policy. Let

$$\begin{aligned} Q^k(s,a) &= T^{\pi^{k-1}} Q^{k-1}(s,a) \\ &= \sum_{s',r} p(s',r \mid s,a) \left[r + \gamma \max_{a'} Q^{k-1}(s',a')\right] \end{aligned}$$

be the k-step GPI Q-prior, with T^π the Bellman operator for π. In other words, Q^k is the k-step greedy search over Q^0. Let $\pi^k(s) \in \arg\max_a Q^k(s,a)$ be the k-step GPI policy. Then, for search depth k,

$$\left\|Q^* - Q^{\pi^k}\right\|_\infty \leqslant \frac{2\gamma^k}{1-\gamma}\left(\gamma\Delta\tilde{Q} + (1+\gamma)\,\epsilon\right)$$

with $\|\cdot\|_\infty \doteq \max_{s,a} |\cdot|$.

The proof is derived from k-step Value Iteration. Through the exponentially decaying γ, Theorem 3.1 guarantees that a k-step look-ahead greedy search over Q^0 improves on the (0-step) GPI and any search of lower depth than k, under the max norm and some approximation term. In the context of decision-time planning with MCTS and a UCB tree policy, we can guarantee that all actions are visited at least once at the root node, with minor constraints on the UCB c hyperparameter and the range in Q-values.

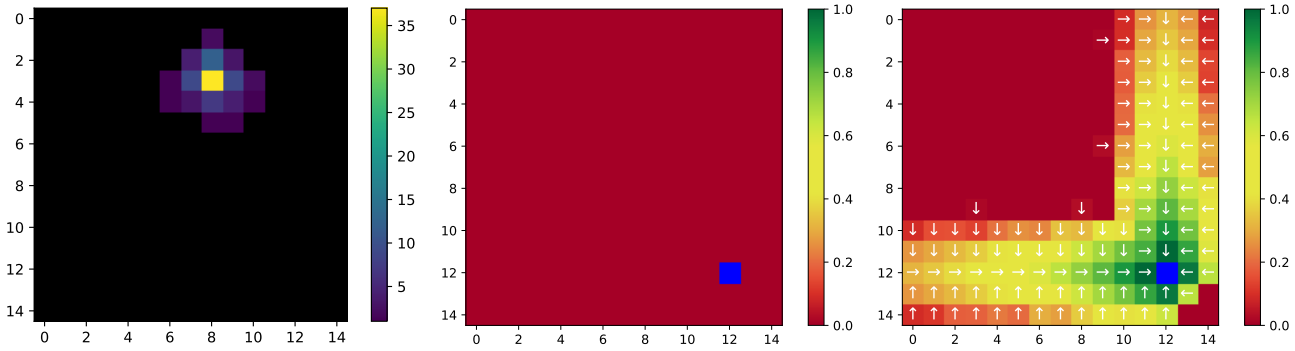
4 GPI-TREE SEARCH ALGORITHMS

We now turn to the design of algorithms for decision-time planning using the GPI framework. As far as we are aware, no method exists in the literature to search in these conditions, even for model-based RL as a whole. We call *GPI-Tree Search Algorithms* (GPI-TS) any method that make use of the Q^0 or π_j policy priors assumed by the GPI framework to enhance MCTS, as detailed in Section 3.

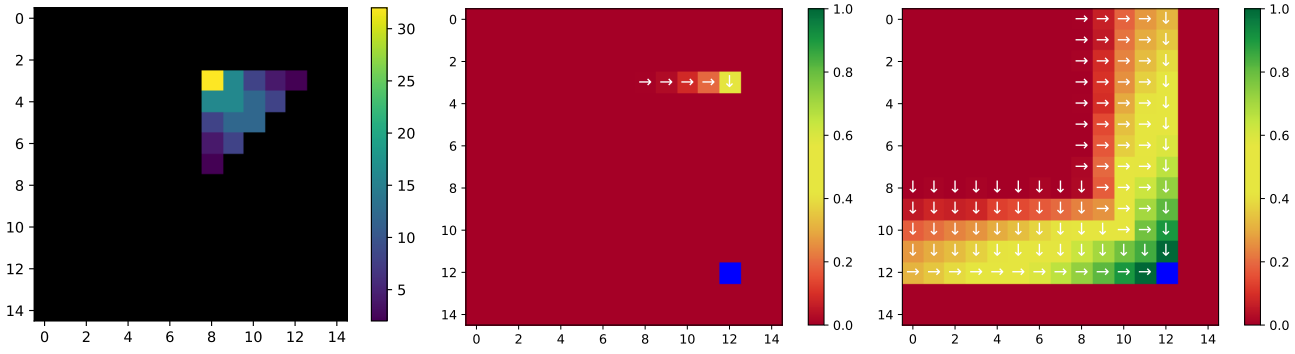
We will be directly showcasing the algorithms along with experiments on Figure 3, as the open gridworld allows a good intuitive understanding of each method. For this purpose, we start by clarifying this environment.

4.1 Environment, policies and visualization

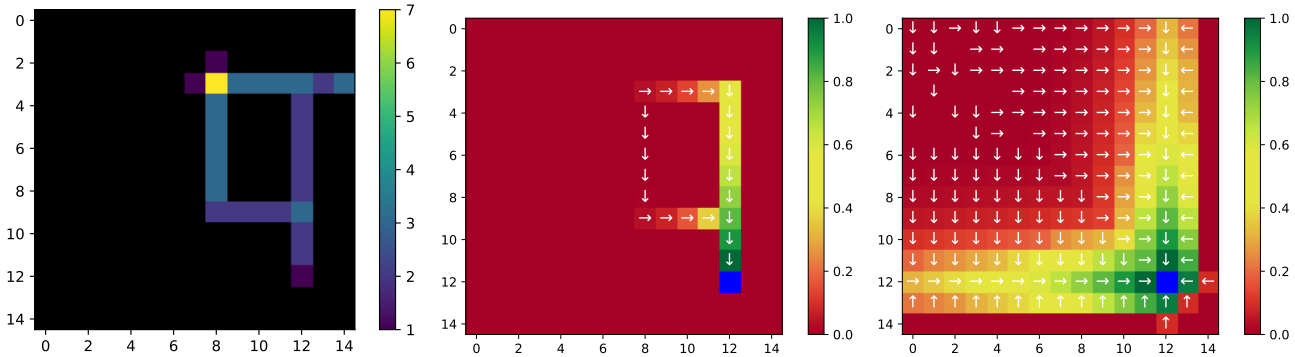
We use an open griworld of 15×15 with a terminal reward of 1 at the goal state (12, 12) (origin in the upper left at 0, 0). The actions are up, down, `right`, `left`, without any form of noise. We pick our policy



(a) GPI-Tree Search



(b) GPI-Constrained Tree Search



(c) GPI-Option Search

Figure 3: GPI-TS Algorithms: [left] the states visited by the search from a given root node at $(3, 8)$. [middle] The resulting MCTS Qs of the nodes visited during the search. [right] The search applied to the entire state space, with a visualized greedy policy. States with uniform $Q(s, \cdot) = 0$ priors are shown with an empty greedy policy, to highlight the lack of a preferred action. All experiments are made with $N = 30$ rollouts.

set: two skills that, respectively, repeatedly use actions `right`, and down: $\{\pi_1(s) = \texttt{right}, \pi_2(s) = \texttt{down}\ \forall s\}$. In Figure 1 (left, middle), we visualize them with their value functions, in other words the Q^{π_j} assumed known by GPI framework. On the right of Figure 1, we visualize Q^0 and π^0 only for states where $Q^0 > 0$. We leave the policy blank otherwise (it should be random over all actions). We do this instead of computing Q^{π^0} in order to highlight what information is available at inference time, and the actual decision of the method. From here onward, all plots similar to Figure 1 will use this visualization choice.

4.2 GPI-Tree Search

We start instantiating the GPI-TS algorithms with an implementation of MCTS with Q^0 prior as described in Section 2 and 3.1, with maximum value backups. We do not make explicit use of the skills $a \sim \pi_j$ here, but only their state-action value Q^{π_j} through the

Algorithm 1 GPI-Tree Search

Input: Rollouts N, root state s_0

Function `GPI-TS`(N, s_0):
  $Q(s_0, a) \leftarrow Q^0(s_0, a), \forall a$
  $N(s_0, a) \leftarrow 1, \forall a$
  $node \leftarrow s_0$
  for $n \in 1 \cdots N$ **do**
    $node, a \leftarrow$ `Selection`($node$)
    $G \leftarrow$ `Expansion`($node, a$)
    `Backup`($node, G, a$)
  end for
  return $Q(s_0, \cdot)$
End Function

Function `Selection`($node$):
  $a \leftarrow \pi^{tree}(node)$ ▷ Eq (1)
  if $node.children[a]$ **is not None then**
    $node \leftarrow node.children[a]$
    $a \leftarrow$ `Selection`($node$)
  end if
  return $node, a$
End Function

Function `Expansion`($node$, a):
  $s', r \leftarrow p(node, a)$
  $Q(s', a') \leftarrow Q^0(s', a'), \forall a'$ ▷ Section 3.1
  $N(s', a') \leftarrow 1, \forall a'$
  $node.children[a] \leftarrow s'$
  return $\max_{a'} Q(s', a')$ ▷ Section 3.2
End Function

Function `Backup`($node$, G, a):
  $N(node, a) + \leftarrow 1$
  $Q(node, a) + \leftarrow (G - Q(node, a)) / N(node, a)$
  $G \leftarrow \max_a Q(node, a)$ ▷ Section 3.2
  if $node.parent$ **is not None then**
    $node, a = node.parent$
    `Backup`($node, G, a$)
  end if
End Function

prior Q^0. The algorithm is described in Algorithm 1. As discussed in Section 3.3, this policy is guaranteed to improve over the GPI policy. The main intuitive benefit of this algorithm over GPI is that the search enables to recover high-value states according to Q^0 from parts of the state-action space we had no good prior information about. This is made clear in Figure 3a: we can see that the search explores the space around the root node. Applying this search over the entire state space[2] results in the algorithm finding how to join the GPI policy from all nearby states.

4.3 GPI-Constrained Tree Search

While GPI-TS makes efficient use of the Q^0 prior assumed by the GPI framework, we do not benefit from the skill set that also comes with it. For example, in the gridworld environment we study, it is clear that the `right` and `down` actions of our skill set are generally to be preferred, as the goal is in the lower right corner.

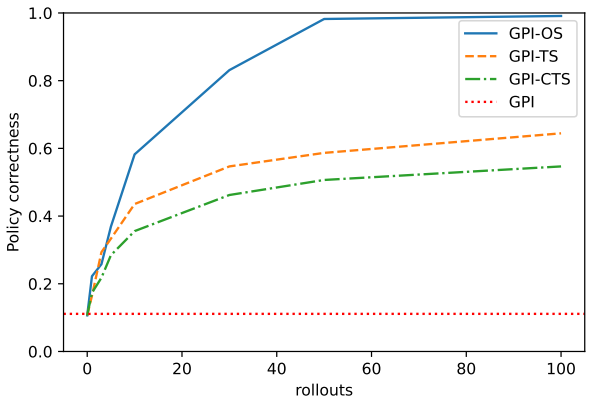
In general, we can make the assumption that the skills assumed by the GPI framework are of relevance to the task. We take advantage of this to constrain the search to only the actions the skills would take: $\mathcal{A}^- = \{a \mid \exists j : \pi_j(a \mid s) > 0\}$. We call this algorithm GPI-Contrained Tree Search (GPI-CTS). It enables a deeper search over the skills' state occupancy distributions from the root node. This is made clear in Figure 3b: following our skills, the search is concentrated in the lower right direction, where it eventually finds a state of high value according to the Q^0 prior and joins up with the GPI policy. The resulting policy over the whole state space can find the correct actions sooner than GPI-TS in the upper left corner of the space. It is clear however that it is completely unable to reach the goal from the lower and right parts of the space, due to the skill set chosen. We note that since the action space was restricted, the assumptions from Theorem 3.1 no longer apply and GPI-CTS is not guaranteed to improve over the GPI policy. It can be relevant in practice though, depending on the skill set chosen, for example if we have prior knowledge on bad actions.

[2]No information is carried from one state to another – we simply called Algorithm 1 from each state to obtain output Q^0 and report on the map $\max_a Q^0$ for all states, and its highest-valued action.

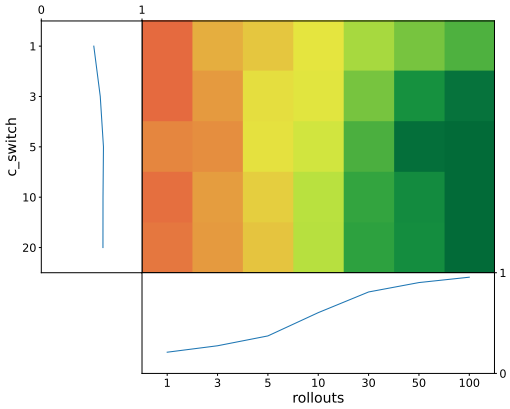
4.4 GPI-Option Search

Skills as temporal abstractions. We start from the intuition that skills are generally better at exploration than a random policy – especially if learnt with USD. **We therefore want to be able to follow the skills for several steps during our search.** While GPI-CTS can make use of the skills through action priors, it is still unable to foster the temporal-abstraction potential that skills inherently possess. Indeed, skills can be used to break down tasks and drastically cut the depth of the MDP. This is the general idea of Hierarchical Reinforcement Learning [16]. One framework to study and exploit such properties is through options [30], already introduced in Section 2.

Searching over Options. We turn to options to benefit from such temporal abstractions – in other words, we want to use options to allow the searching agent to call the skills π_j over several steps. We define options $o_j \in O, o_j = (\mathcal{S}, \pi_j, \beta_j)$: the initiation set is the whole state space $\mathcal{S}$, and follow policy π_j. We define the termination condition following recommendations from the options framework:"we can compare the value of continuing our option to the value of interrupting it and selecting a new option. If the latter is more highly valued, then why not interrupt the option and allow the switch?" [30]. This sounds ideal, but is rarely applied in practice, as the value function of all options is generally unknown. However, the GPI framework assumes these to be known (generally through Successor Features), so we can simply apply this idea directly.



(a) Comparing GPI-TS algorithms for varying rollout values



(b) Hyperparameter sensitivity of GPI-OS

Figure 4: Hyperparameter studies

More concretely, we add our skills as additional actions to the original set, $\mathcal{A}^+ = \mathcal{A} \bigcup \mathcal{O}$. We then define stopping our skill o_j with $\beta_j(s) = \mathbf{1}_{\exists a \in \mathcal{A}^+ | Q^0(s,a) > Q^0(s,o_j)}$ with $\mathbf{1}$ being the indicator function. We expand all visited states during skills and count them as rollouts, for memory and complexity fairness with the other algorithms. We initialize the option values with prior $Q_0(s, o_j) = \max_a Q^{\pi_j}(s, a)$. In order to prevent a single skill from using up all the budget, we add a time-constraint: a skill can not last longer than c_{switch} steps. The general inspiration for this algorithm, as well as the name of the variable, come from Explore Options [2, 3]. Since a skill takes primitive actions, we backup the values of the primitive actions taken during the skill; at the node the skill was chosen, we backup both the option and corresponding action's values. This algorithm is referred to as GPI-Option Search (GPI-OS). Note that since we have only extended the action set, GPI-OS preserves the theoretical improvement guarantees from Theorem 3.1.

Visualization. Figure 3c visualizes GPI-OS. We can see that the search is extended and much deeper in the direction of the skills (with $c_{switch} = 5$ here) than previous GPI-TS algorithms. This enables the algorithm to join the Q^0 priors and even find the original environment reward from a state where GPI-TS and GPI-CTS could not. As a consequence, the resulting Q^{GPIOS} map and policy covers a much greater part of the state-space. On the other hand, the deeper search of GPI-OS means it relies much more heavily on the quality of the model to mitigate compounding errors. We note that while the search can be quite deep in the general direction of the skills, GPI-OS is still unable to recover the full optimal policy from the lower right corner. While a higher number of rollouts would eventually fix this issue, this is impractical. This further justifies the search for better exploratory skills in Unsupervised Skill Discovery [8].

arg max *operator tie-breaking.* The initialization of Q_0 with $\max_a Q^{\pi_j}(s, a)$ for skills and $\max_j Q^{\pi_j}(s, a)$ for actions implies that several initial Q values will be shared in our extended action set, even in the case of function approximation. This leads to the very interesting problem of tie-breaking in the arg max operator during UCB: we can decide to favor skills, primitive actions, or none in particular. We empirically find that prioritizing primitive actions leads to a broader search, while prioritizing skills leads to a deeper search. We choose the latter in our experiments, as it emphasises the nature of the algorithm.

4.5 Comparison of algorithms and parameters

Policy correctness. We now provide a quantitative comparison of the algorithms to add to the intuitive understanding provided by Figure 3. In order to achieve this, we need a metric to evaluate the methods. Due to discounting, a distance to the optimal value function would favor algorithms that learn values close to the goal. Instead, we recall that the point of GPI is to improve on a set of policies, and use as metric the"policy correctness", i.e., the proportion of agreement between the obtained policy and the optimal policy, as a percentage of optimal actions over the state space. The main variable for our algorithms is the number of rollouts N: in Figure 3, we used $N = 30$. In Figure 4a we compare the algorithm's policy correctness as a function of rollouts. We observe that not only does GPI-OS perform better overall, it's also stronger as N grows. This is aligned with our intuition, since a higher amount of rollouts allow for a deeper search over skills. GPI-TS performs better than GPI-CTS here, mainly due to the shape of the environment and policy set. Indeed, there is a high amount of states below and on the right of the Q^0 prior, out of the reach of GPI-CTS.

GPI-OS hyperparameters. Another hyperparameter that was introduced is the skill time-limit c_{switch} in GPI-OS. This was added to prevent a single skill from claiming all the rollout budget, preventing a wide search. In Figure 4b we perform a study of the policy correctness of GPI-OS with varying rollouts N and c_{switch}. We can see that the algorithm is sensitive to the rollouts, but fairly insensitive to c_{switch}. Its value matters most for high-valued N, where it starts being beneficial to have higher c_{switch}; otherwise the budget is used up too quickly.

Algorithm	Improvement guarantees	Skill set usage	Model sensitivity
GPI-TS	**Yes**	None	**Mild**
GPI-CTS	No	Action constraints	**Mild**
GPI-OS	**Yes**	**Options**	High

Figure 5: Overview of pros (bold) and cons of the introduced GPI-TS algorithms

Overview of pros and cons of algorithms. We have designed three algorithms of the GPI-Tree Search family, and compared their qualitative and quantitative performances. We now summarize our findings in Table 5. GPI-TS and GPI-OS both search over all primitive actions, therefore benefitting from the improvement guarantees from Theorem 3.1. GPI-TS ignores the skills assumed in the GPI framework, while GPI-CTS and GPI-OS both make use of them, through action constraints and options respectively.

In this work, we have assumed a known model $p(s', r \mid s, a)$. We did not dive into the consequences of a model approximation. In general, modeling errors compound and make deeper searches more prone to errors – in our case, GPI-OS would therefore be more sensitive to the model than the other algorithms. We provide this intuition in Table 5 but leave the study of such modeling errors to future work. Without modeling errors in mind, we recommend usage of GPI-OS, as we have seen that the deep, directed search can greatly help performance and scales better with the rollout budget. GPI-CTS is to be preferred over GPI-TS only if there is prior information that the skills can provide valuable action constraints.

5 RELATED WORK

GPI and Successor Features. SFs and the GPI were introduced in Section 2. Due to its close ties with Unsupervized Skill Discovery, SFs and the GPI have made several appearances for skill learning and temporal abstractions [14, 21]. While connections have been shown between SFs and model-based RL [20], model-based applications of the GPI have, as far as we are aware, not been explored before, let alone forward planning.

MCTS and Skill search. MCTS was introduced by [6], and gained in popularity with the AlphaZero series [24, 25]. As far as we are aware, the usage of Q-value priors to guide the search was only explored in [13]. None of these works have direct relations with the GPI, SFs, or Hierarchical RL in general.

The idea to search over skills or options was raised with the option framework itself [30] and recently explored to learn reward-respecting policies [29], however, the model was used for more generic"planning", in the sense of building a value function directly through the model. In comparison, GPI-TS algorithms (as decision-time planning algorithms) focus on the current state and generate a stronger policy at inference time without learning an explicit value function.

An MCTS search over options was designed in Ilhan and Etaner-Uyar [17], however, this work only searched over skills, and not primitive actions, which loses the guarantees of Theorem 3.1. In addition, the search is not guided by any Q-value priors such as the one assumed by the GPI framework.

6 CONCLUSION

We have proposed to use model-based RL, and in particular decision-time planning, to foster the benefits of the GPI within forward searches. We have provided max norm improvement guarantees on the GPI using such searches, for any depth and under approximation. We have introduced a family of algorithms to achieve this in the GPI framework using MCTS. These algorithms make use of the Q-value prior and skills assumed by the framework to efficiently guide the search. We have designed three algorithms within the family, and studied their properties both qualitatively and quantitatively. The strongest algorithm we introduce, GPI-Option Search, can search over both primitive actions and skills, using the Q-value priors of the GPI framework to trigger and stop skills. Since the experimental setup of this work is limited to tabular cases in order to provide intuition, future work could demonstrate the performance of GPI-Tree Search algorithms in function approximation scenarios for both the agent and model. It could also extend the proofs to tighten the k-step GPI optimality bounds, and provide some indications to the type of skills most adapted to being used in such searches.

ACKNOWLEDGMENTS

This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificile Intelligentie (AI) Vlaanderen" programme.

REFERENCES

[1] Rajeev Agrawal. 1995. Sample mean based index policies by o (log n) regret for the multi-armed bandit problem. *Advances in Applied Probability* 27, 4 (1995), 1054–1078.

[2] Louis Bagot, Kevin Mets, Tom De Schepper, and Steven Latré. 2022. Deep Learning of Intrinsically Motivated Options in the Arcade Learning Environment. *NeurIPS Workshop on Deep Reinforcement Learning* (2022).

[3] Louis Bagot, Kevin Mets, and Steven Latré. 2020. Learning intrinsically motivated options to stimulate policy exploration. *ICML Workshop on LifeLong Learning* (2020).

[4] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. 2018. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *International Conference on Machine Learning*. PMLR, 501–510.

[5] Andre Barreto, Will Dabney, Remi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. 2017. Successor Features for Transfer in Reinforcement Learning. In *Advances in Neural Information Processing Systems*, Vol. 30.

[6] Rémi Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*. Springer, 72–83.

[7] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. 2019. Diversity is All You Need: Learning Skills without a Reward Function. In *International Conference on Learning Representations*.

[8] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. 2021. The information geometry of unsupervised reinforcement learning. *arXiv preprint arXiv:2110.02719* (2021).

[9] Sylvain Gelly and David Silver. 2011. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence* 175, 11 (2011), 1856–1875.

[10] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. 2017. Variational intrinsic control. *ICLR Workshop* (2017).

[11] David Ha and Jürgen Schmidhuber. 2018. World models. *arXiv preprint arXiv:1803.10122* (2018).

[12] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. 2019. Dream to Control: Learning Behaviors by Latent Imagination. In *International Conference on Learning Representations*.

[13] Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Tobias Pfaff, Theophane Weber, Lars Buesing, and Peter W. Battaglia. 2020. Combining Q-Learning and Search with Amortized Value Estimates. In *International Conference on Learning Representations*.

[14] Steven Hansen, Will Dabney, Andre Barreto, David Warde-Farley, Tom Van de Wiele, and Volodymyr Mnih. 2020. Fast Task Inference with Variational Intrinsic Successor Features. In *International Conference on Learning Representations*.

[15] Hado Hasselt. 2010. Double Q-learning. *Advances in neural information processing systems* 23 (2010).

[16] Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. 2022. Hierarchical Reinforcement Learning: A Survey and Open Research Challenges. *Machine Learning and Knowledge Extraction* 4, 1 (2022), 172–221.

[17] Ercüment Ilhan and A Şima Etaner-Uyar. 2017. Monte Carlo tree search with temporal-difference learning for general video game playing. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 317–324.

[18] Yuu Jinnai, Jee Won Park, Marlos C Machado, and George Konidaris. 2019. Exploration in reinforcement learning with deep covering options. In *International Conference on Learning Representations*.

[19] Michael N Katehakis and Herbert Robbins. 1995. Sequential choice from several populations. *Proceedings of the National Academy of Sciences* 92, 19 (1995), 8584–8585.

[20] Lucas Lehnert and Michael L Littman. 2020. Successor Features Combine Elements of Model-Free and Model-based Reinforcement Learning. *J. Mach. Learn. Res.* 21 (2020), 196–1.

[21] Marlos C Machado, Andre Barreto, and Doina Precup. 2021. Temporal abstraction in reinforcement learning with the successor representation. *arXiv preprint arXiv:2110.05740* (2021).

[22] Marlos C Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. 2018. Eigenoption Discovery through the Deep Successor Representation. In *International Conference on Learning Representations*.

[23] Christopher D Rosin. 2011. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence* 61, 3 (2011), 203–230.

[24] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* 588, 7839 (2020), 604–609.

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

[26] Richard S Sutton. 1988. Learning to predict by the methods of temporal differences. *Machine learning* 3, 1 (1988), 9–44.

[27] Richard S Sutton. 1991. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin* 2, 4 (1991), 160–163.

[28] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.).

[29] Richard S Sutton, Marlos C Machado, G Zacharias Holland, David Szepesvari Finbarr Timbers, Brian Tanner, and Adam White. 2022. Reward-Respecting Subtasks for Model-Based Reinforcement Learning. *arXiv preprint arXiv:2202.03466* (2022).

[30] Richard S Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112, 1-2 (1999), 181–211.

[31] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. 2020. Transfer Learning in Deep Reinforcement Learning: A Survey. *CoRR* abs/2009.07888 (2020).