HI-VAL: Iterative Learning of Hierarchical Value Functions for Policy Generation

Roberto Capobianco[1], Francesco Riccio[1], Daniele Nardi[1]

[1] Department of Computer, Control, and Management Engineering,
Sapienza University of Rome,
via Ariosto 25, Rome, 00185, Italy
lastname@diag.uniroma1.it

Abstract. Task decomposition is effective in various applications where the global complexity of a problem makes planning and decision-making too demanding. This is true, for example, in high-dimensional robotics domains, where (1) unpredictabilities and modeling limitations typically prevent the manual specification of robust behaviors, and (2) learning an action policy is challenging due to the curse of dimensionality. In this work, we borrow the concept of Hierarchical Task Networks (HTNs) to decompose the learning procedure, and we exploit Upper Confidence Tree (UCT) search to introduce HI-VAL, a novel iterative algorithm for hierarchical optimistic planning with learned value functions. To obtain better generalization and generate policies, HI-VAL simultaneously learns and uses action values. These are used to formalize constraints within the search space and to reduce the dimensionality of the problem. We evaluate our algorithm both on a fetching task using a simulated 7-DOF KUKA light weight arm and, on a pick and delivery task with a Pioneer robot.

1 Introduction

Generating effective action policies is impractical in various applications that are characterized by large state spaces and require strong generalization capabilities. Many techniques tackle the curse-of-dimensionality problem by using expert demonstrations to initialize agents' behaviors and guide the learning process. However, this is not feasible when a reduced number of examples is available, and a direct mapping between the expert's and the agent's action space is difficult to obtain (e.g. highly redundant robots). While generalization is typically achieved by means of function approximation (e.g., using neural networks), solely relying on this and excluding prior knowledge can be inefficient, slow and even dangerous in multiple applications, such as robotics. Hence, Monte-Carlo tree search algorithms, and in particular the Upper Confidence Tree (UCT) algorithm [13], are widely used to exploit prior knowledge [21] in exploring search space. Nonetheless, they show limitations in generalizing among related states.

We build upon these state-of-the-art techniques to directly learn action values from experience, and accordingly learn a policy by using UCT with focused exploration. To this end, we introduce HI-VAL, a novel iterative algorithm for learning hierarchical value functions that are used to (1) capture multi-layered action semantics, (2) generate policies by scaffolding the acquired knowledge, and (3) guide the exploration of the

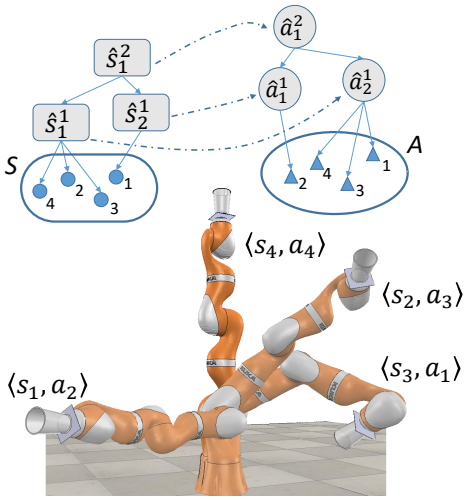

Fig. 1. HI-VAL generates high-level representations of actions, that are used to improve the exploration of the search space. In this figure, we show the action hierarchy generated for a fetching task using a redundant KUKA light weight arm.

state space. HI-VAL improves the UCT algorithm and builds upon concepts from previous literature, such as Hierarchical Task Networks (HTNs) [8], semi-MDPs [23] and MAX-Q decompositions [7], to decompose the learning procedure and to generate both action abstractions and search space constraints. The action hierarchy formalized by HI-VAL is learned iteratively by evaluating state-actions pairs generated by UCT after each episode. Fig. 1 shows an example of such a hierarchy where states and actions are associated at different layers of abstraction. HI-VAL assigns states and actions to different clusters s_l^c and a_l^c by evaluating the similarity of successor states that the agent can reach, by applying the actions in a_l^c in the states contained in s_l^c. Intuitively, similar successor states have similar reward values and can be evaluated altogether when exploring the search space. Different layers provide different granularity of action semantics (the higher the more coarse) and help the learning process to evaluate states hierarchically. HI-VAL runs UCT to explore the environment by sampling the joint distribution of rewards and state-action pairs. Each sample is continuously aggregated into a dataset, that is used to estimate – by means of Q-learning – the value function Q^λ of each layer λ in the hierarchy. Specifically, at each layer, Monte Carlo search is ran for a subset of actions that are evaluated according to their Q-value, thus driving the node-expansion phase during episode simulation.

In this work, we aim at demonstrating that Q-values can be learned hierarchically to influence exploration, and to represent action semantics at different levels of abstraction, thus linking learning techniques to low-level agent controls. Our main contributions consist in (1) a novel integration of Monte-Carlo tree search, hierarchical planning and Q-learning, that enables good performance with selective state exploration and improved generalization capabilities, in (2) a two-sided extension of TD-search, that not

only executes on multiple hierarchy layers, but also constructs upper confidence bounds on the value functions – and selects actions optimistically with respect to those – and in (3) a reduction of the curse-of-dimensionality that is obtained by means of focused exploration. We evaluated the HI-VAL performance in two different scenarios, an "object-fetching" task with a 7-DOF KUKA light weight arm and, a "pick and delivery" task in a simple environment with a Pioneer robot, where the agent has to collect an item and delivering it reduction in the number of states explored – which makes the method more suitable in robotic applications – and, the ability of HI-VAL to represent action semantics through its hierarchy in order to boost the learning process.

2 Related Work

Policy learning is widely adopted to generate practical behaviors in several applications. This is true for complex domains, such as robotics [4], where unstructured environments and uncertain dynamics through handcrafted policies – that typically fail or must be refined [14]. Although designing effective policies is impractical in most of these scenarios, and learning techniques are typically demanding and time consuming [11] for problems with large state spaces. The computational demand can be alleviated by initializing a policy with expert demonstrations, that restrict the learning process to a promising hypothesis space [12, 20]. However, to apply imitation learning in complex domains, a large dataset of good-quality expert demonstrations is generally required, that can be efficiently mapped to the agent's action space. Unfortunately, this is not always possible due to the lack of (1) domain experts, (2) practical ways of providing demonstrations, and (3) action mappings from experts to agents (e.g. hyper-redundant robots).

To overcome these difficulties, we propose an approach that does not require expert demonstrations to initialize agent behaviors, and decomposes the learning procedure to generate action abstractions and search space constraints. In literature, multiple authors exploit the notions of skills and semi Markov Decision Processes (semi-MDPs), and define hierarchical representations such as *options* [23] and MAX-Q decompositions [7]. Unfortunately, applications of these methods in complex domains like robotics are limited, and prior knowledge has to be enforced in the learning process by means of expert demonstrations. In fact, although hierarchical learning and value function approximations techniques have been adopted in several applications, state-of-the-art techniques still show considerable margin of improvement. For example, [19] provide a better policy generalization by exploiting the concept of Generalized Value Functions, to improve value function approximation. In a different settings, [6] use expert demonstrations to learn high-level tasks as a combination of action-primitives. Unfortunately, these approaches only learn specific hierarchical structures, that poorly generalize and cannot profit from the expressiveness of value functions. Similarly, [22] apply hierarchical learning to sequences of motion primitives on a pick-and-place task with a hyper-redundant robotic arm. [15] initialize skill trees from human demonstrations, improving them over time. However, their representations use expert demonstrations and do not represent action on higher levels of abstractions. Conversely, [10] apply hierarchical policy learning to solve a 2-DOF stand-up task for a robotic arm. They exploit

Q-learning and actor-critic methods to learn both task decompositions and local trajectories that solve specific sub-goals. Alternatively, [9] and [2] formalize action hierarchies to represent actions at different levels of abstraction. However, these procedures are not easily scalable to higher dimensionality problems.

Motivated by our discussion, we extend our previous work [17, 16] to formalize action hierarchy by introducing HI-VAL, an iterative algorithm that learns hierarchical value functions to drive the policy search, and that achieves good generalization with a focused state exploration without the aid of expert demonstrations. Specifically, we enable HI-VAL to (1) learn a hierarchical value function directly from experience and (2) simultaneously use learned values during exploration, to generate a competitive policy. As in Hierarchical Task Networks, value functions are used to plan both over compound and primitive action spaces, whereas compound actions have specific implementations that depend on the state of the environment. Like [21], we exploit TD (temporal difference) methods to learn action values during a Monte-Carlo tree search. In this way, on the one hand, we are able to learn Q-values, and on the other hand, we adopt Q-values to support decision-theoretic planning for generalization and exploration at multiple levels of abstraction. Differently from [21], in fact, not only we improve our model by preserving the selective search of Monte-Carlo algorithms when bootstrapping is ongoing, but we also generate action and state abstractions. Specifically, HI-VAL extends TD-search by constructing upper confidence bounds on a hierarchy of value functions, and by selecting optimistically with respect to those. As the experimental evaluation shows in both scenarios, HI-VAL generates competitive policies – with the additional benefit of a reduction in (1) number of simulations (or expanded node), and (2) exploration of the search space – that alleviates the curse-of-dimensionality.

3 HI-VAL

Formulation. HI-VAL is an iterative algorithm that, at each iteration i, generates a new policy π_i which improves π_{i-1} [17]. To obtain an improved π_i, our algorithm leverages (1) data aggregation [18], and (2) Upper Confidence Bounds for Trees (UCT) [13], a variant of Monte-Carlo Tree Search that adopts an upper confidence bound strategy – UCB1 [3] – for balancing between exploration and exploitation on the tree. To describe HI-VAL we adopt the Markov Decision Process (MDP) notation, in which the decision-making problem is represented as a tuple $MDP = (\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$, where $\mathcal{S}$ is the set of discrete states of the environment, $\mathcal{A}$ represents the set of discrete actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a stochastic transition function that models the probabilities of transitioning from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ when taking action $a \in \mathcal{A}$, $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, and γ is a discount factor in $[0, 1)$.

Function Approximation. HI-VAL addresses the generalization problem by relying on previous literature [5]. We choose to approximate the Q function using probability densities in the form of a mixture of K Gaussians (i.e., Gaussian Mixture Models – GMMs). We integrate the approach in [1] with a data aggregation [18] procedure, where a dataset of samples is iteratively collected and aggregated. Specifically, at each iteration i, the

values Q^i are determined according to the Q-learning update rule

$$Q^i(s,a) = \hat{Q}'(s,a) + \alpha(r + \gamma \max_{a'} \hat{Q}'(s',a') - \hat{Q}'(s,a)), \tag{1}$$

where α is the learning rate, $\hat{Q}'$ is the function approximation learned at previous iteration, and $Q^0(s,a_j) = 0$. As we discuss later, the function approximation $\hat{Q}$ is learned over an aggregated dataset $\mathcal{D}^{0:i} = \{\cup \mathcal{D}^d | d = 0 \dots i\}$.

3.1 Exploration and Sample Collection

At every step h, for $h = 1 \dots H$, UCT simulates the execution of all the actions $\mathcal{A}_L \subseteq \mathcal{A}$ that are "admissible" in s_h, as detailed in next section. Specifically, each simulation executes an action $a \in \mathcal{A}_L$, followed by K roll-outs, that run an ϵ-greedy policy based on π_{i-1} until a terminal state is reached. The best action a_h is selected according to

$$\begin{aligned} e &= C \cdot \sqrt{\frac{\log(\sum_a \eta(s_h,a))}{\eta(s_h,a)}} \\ a_h^* &= \arg\max_a \hat{Q}(s_h,a) + e, \end{aligned} \tag{2}$$

where C is a constant that multiplies and controls the exploration term e, and $\eta(s_h,a)$ is the number of occurrences of a in s_h. Since we assume a discrete state space $\mathcal{S}$, for continuous problems we define a similarity operator that informs the algorithm whether the difference of two states is smaller than a given threshold ξ – thus discretizing the space.
During each roll-out:

- a dataset $\mathcal{D}^i$ of samples $x = (s, a, Q^i(s,a))$ is collected to improve our estimate $\hat{Q}$, as detailed in previous section;
- HI-VAL uses UCT as an expert and collects a dataset $\mathcal{D}_{\pi,uct}$ of H samples $x = (s_h, a_h^*, s_h')$ that are selected by the tree search
- similarly to DAgger [18], $\mathcal{D}_{\pi,uct}$ is aggregated into a dataset $\mathcal{D}_{\pi,i} = \mathcal{D}_{\pi,uct} \cup \mathcal{D}_{\pi,i-1}$.

When H UCT steps are run, $\mathcal{D}_{\pi,i}$ is used to generate hierarchy clusters – as detailed in the next section – and learn a new policy π_i. Our algorithm uses a discovery process that is supported by the UCT search and policy.

3.2 Hierarchical Action Selection

The hierarchical model adopted in HI-VAL builds upon the concepts of *High-Level Actions (HLAs)* and *Reachable Sets (RSs)* in HTNs [8].

- HLAs are defined recursively as a sequence of action primitives and/or other HLAs. When a HLA is composed by only primitives, such sequence is called "implementation".

– RSs model preconditions and effects of HLAs. They are defined as the union of possible states reachable by the the different implementations of HLAs. A RS is bounded by a pessimistic RS^- and optimistic RS^+ set. RS^- represents the set of states that are reached independently from the chosen implementation, while RS^+ represents the set of states reached by all the possible implementations. Reachable sets describe interesting properties: (1) if RS^- intersects the set of goal states, then a sequence of admissible actions leading to the goal is found; (2) if, instead, RS^+ intersects the set of goal states, a plan exists but its implementations do not reach the goal yet.

Using similar concepts, we allow HI-VAL to evaluate actions at multiple levels of abstraction. Specifically, a hierarchy of actions is obtained using an *agglomerative clustering* algorithm, which is ran on the set of next states $\{s'\}$ present in the $\mathcal{D}_{\pi,i}$. Our key assumption is that similar s' encode information about actions with similar effects and thus can be clusterized altogether. Particulary, we refer to $\mathcal{H}$ as the set of layers in the action hierarchy generated by the clustering algorithm. The clustering routine organizes $\{s'\}$ in clusters $\hat{s}'$ over a predefined number of layers, which are then tranferred into the action space, generating a set of action clusters $\hat{a}$ organized along the same structure. Such a mapping is realized by evaluating each element contained in the $\hat{s}'$ clusters and backpropagated to the original dataset $\mathcal{D}_{\pi,i}$ in order to retrieve the set of actions generating the transitions to the $\{s'\}$ states. In fact, dataset elements are a tuple of three components encoding a transition from the current state s to the next state s' by means of the action a. Fig. 1 shows a simplistic example of such hierarchies.

In our algorithm, each action cluster $\hat{a}$ corresponds to a HLA in the layer $\lambda \in \mathcal{H}$, and each layer has an associated Q^λ function, approximated as $\hat{Q}^\lambda$. The result of choosing an action $\hat{a}$ consists of selecting a cluster of lower level actions with a similar expectation to reach a desired cluster $\hat{s}'$. Noticeably, such a model intrinsically connects to the concept of reachable sets. Clusters $\hat{s}'$ are in fact an approximation of optimistic sets RS^+, and they evaluate actions $\hat{a}$ that lead to more rewarding states altogether.

HI-VAL uses each Q^λ in s to select the set $\mathcal{A}_L$ of "admissible" actions for UCT. Intuitively, a primitive action a is admissible in s if, for each layer λ, a belongs to the cluster $\hat{a}^\lambda$ selected according to Q^λ. More formally:

$$\mathcal{A}_L = \bigcup_{a \in \mathcal{A}} \left\{ a \mid \forall \lambda \in \mathcal{H} : a \in \hat{a}^\lambda \right\} \tag{3}$$

$$\hat{a}^\lambda = \arg\max_{\hat{a}} Q^\lambda(s, \hat{a}) + \delta^\lambda_{s,\hat{a}} \tag{4}$$

$$\delta^\lambda_{s,\hat{a}} \sim \mathcal{N}(0, \sigma^2(Q^\lambda | s, \hat{a})),$$

where σ^2 is the standard deviation of the regression approximation [1].

Through $\delta^\lambda_{s,\hat{a}}$, the prediction error for each action abstraction is captured, leading to a more directed exploration of the action hierarchy. Action primitives are finally chosen and executed by UCT according to the lowest-level $\hat{Q}$, as detailed in previous section. To obtain a less biased exploration and avoid value function over-fitting, inadmissible actions are anyway expanded and selected by UCT with a 30% probability.

Algorithm 1: HI-VAL

Input: $\mathcal{D}_0$ dataset of random state action pairs $\{(s, a, s')\}$.
Output: π_N policy learned after N iterations of the algorithm.
Data: $\mathcal{A}$ set of primitive actions, N number of iterations of the algorithm, Δ initial state distribution, H UCT horizon, K ϵ-greedy roll-outs, T policy execution timesteps, $\mathcal{H}$ set of layers.

```
begin
    Initialize Q̂^0 to predict 0.
    Train classifier π_0 on D_{π,0}.
    for i = 1 to N do
        s_0 ← random state from Δ
        for t = 1 to T do
            Get state s_t by executing π_{i−1}(s_{t−1}).
            D_uct ← UCT(H, s_t)
            D_{π,i} ← D_{π,i} ∪ D_uct
            ŝ, ŝ' ←agglomerativeClustering(D_{π,i})
            â ← mapping(ŝ')
            foreach λ ∈ H do
                // update estimated Q-values
                R̄(ŝ, â) ← 1/|(ŝ,â)| Σ_{s∈ŝ,a*∈â} R(s, a*)
                D^{λ,i} ← getSamples(D_uct, R̄(ŝ, â))
                D^{λ,0:i} ← D^{λ,0:i} ∪ D^{λ,i}
                Train Q̂^{λ,i}(ŝ, â) on D^{λ,0:i}
            end
        end
        Train classifier π_i on D_{π,i}
    end
    return π_N
end
```

3.3 HI-VAL Algorithm

The goal of HI-VAL consists of iteratively updating each layer's value function approximation $\hat{Q}^{\lambda}$, to generate a policy π_i that maximizes the expected reward of the agent. The underlying insight of HI-VAL is that, while exploring the search space, collected state-action pairs are used at each iteration i to (1) update the approximated Q functions for refining the policy π_{i-1} into a policy π_i, and (2) use Q-values to influence UCT exploration in accordance with Eq. 3. The complete HI-VAL algorithm – described in Alg. 1 – for each iteration proceeds as follows:

1. **Roll-in.** The agent follows the previous policy π_{i-1} and generates a set of s_t states for T timesteps.
2. **UCT search.** For each of the generated states s_t, HI-VAL runs an UCT search with horizon H. At each step h, UCT simulates the execution of every "admissible" action in the set $\mathcal{A}_L$, computed according to Eq. 3. For each action $a \in \mathcal{A}_L$, a

simulation consists of the execution of a, followed by K ϵ-greedy roll-outs based on π_{i-1}, which are used to estimate the Q-values of each visited state. Finally, for each step, the best action a_h^* is (1) chosen according to Eq. 2 and (2) aggregated into a dataset D_{uct} together with s_h and s_{h+1}. It is worth remarking that a vanilla implementation of the UCT search evaluates all possible actions and explores a significant amount of states to generate an effective policy. Our approach, instead, leverages the hierarchical structure of $\mathcal{H}$ to generate a restricted subset of "admissible" actions, with high estimated Q-value. This efficiently reduces the exploration phase by guiding the algorithm to discard actions that are not expected to improve π_{i-1}.

3. **Hierarchical data aggregation and $\hat{Q}$ update.** After UCT, new data $D_{uct} = \{(s_{t+h}, a^*_{t+h}, s'_{t+h+1}) \mid h = 1 \dots H\}$ is available to be aggregated into a larger dataset $\mathcal{D}_{\pi,i}$. This dataset is used to generate clusters $\hat{s}$, $\hat{a}$, and $\hat{s'}$ in two steps: first, the sets of states $\{s\}$ and $\{s'\}$ in $D_{\pi,i}$ are separately clustered within λ layers; then, the hierarchy of next state clusters $\hat{s'}$ is transferred into the action space to generate the action clusters $\hat{a}$, each of them corresponding to high-level actions. In order to correctly update the $\hat{Q}^\lambda$ estimation for every $\hat{a}$, samples of the form $\mathbf{x}^\lambda = (\hat{s}, \hat{a}, Q^\lambda(s, \hat{a}))$ are generated for each layer λ, with $Q^\lambda(s, \hat{a})$ determined as in Eq. (1). Such samples are then (1) aggregated into a dataset $\mathcal{D}^{\lambda,0:i}$ and (2) used to improve the estimate $\hat{Q}^\lambda$, as described in previous sections. Specifically, $\hat{Q}^\lambda$ is updated for each $(\hat{s}, \hat{a})$ containing the state-action pairs $(s_{t+h}, a^*_{t+h}) \in \mathcal{D}_{uct}$, by using the averaged reward $R(s_{t+h}, a^*_{t+h})$ of the corresponding state-action pairs in the clusters $(\hat{s}, \hat{a})$.
4. **Training.** Once data aggregation has been performed, a new policy π_i is trained from the dataset $\mathcal{D}_{\pi,i}$ (e.g., using a classifier).

4 Experimental Evaluation

We evaluate our approach in generating a policy for executing a "fetching" task, and a "pick and delivery" task in a simple environment [7], with a reduced number of state-action pairs explored by UCT. We compare our results with a random-UCT and vanilla-UCT implementations, the TD-search [21] algorithm and different configurations of the HI-VAL action hierarchy. We will refer to as VAL as a basic implementation of the action hierarchy composed solely by a single layer of the primitive actions. Then, the random-UCT algorithm selects a random action at each h step of the Monte Carlo search, while in the vanilla-UCT all actions are considered "admissible" at each step. In all algorithms, we implement a shaped reward function that computes reward values at each visited state. We deploy these algorithms within a simulated environment on a 7-DOF KUKA light weight arm and, on a Pioneer robot.

Experiments have been conducted within the V-REP simulation environment, using a single Intel i7-5700HQ core, with CPU@2.70GHz and 16GB of RAM. For both the scenarios, the UCT search has been configured as follows: (1) search horizon $H = 4$; (2) exploration constant $C = 0.707$; (3) $K = 3$ roll-outs. The number of components of the GMMs is evaluated according to the BIC criterion, which has been tested using up to 6 Gaussians. Q-values are updated with a learning rate $\alpha = 0.1$, and a discounted

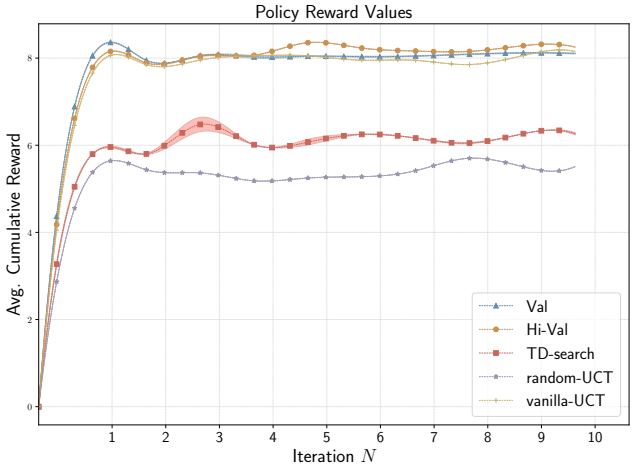

Fig. 2. Average cumulative reward obtained by the random-UCT, TD-search, vanilla-UCT, VAL and HI-VAL over 10 iterations.

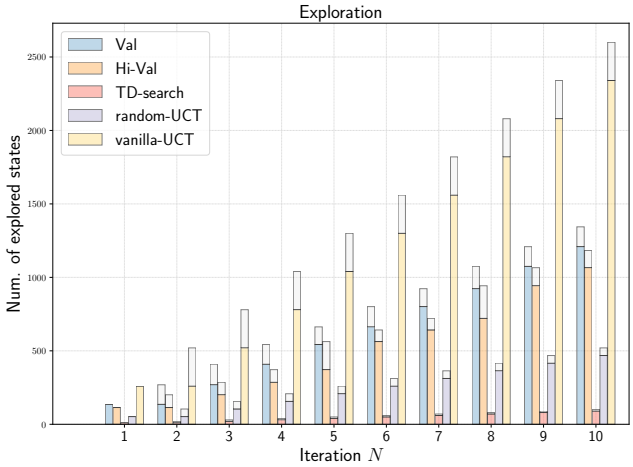

Fig. 3. Number of states expanded by the random-UCT, TD-search, vanilla-UCT, VAL and HI-VAL.

factor $\gamma = 0.8$. The algorithm is ran for $T = 15$ timesteps at each iteration. Moreover, stochastic actions are induced by randomizing the outcome of an action with a 5% probability.

4.1 Fetching task

In this scenario the state of the problem is represented as a 7-feature vector, where 3 components represent the distance of the robot end-effector to the target, 3 components

encode the distance to an obstacle introduced in the scene, and the last component is the angle difference between the end-effector and world axis Z. We include such component to bias the agent in planning to fetch an object with a preferable orientation. The reward function is a weighted sum of such components, and it is designed to promote states that are far from the obstacle, and close to the target position. Additionally, it penalizes states in which the end-effector does not point upwards, to simulate objects that have to be held with a preferred orientation (e.g. a glass full of water). The robot explores an action space composed by 13 actions: 6 translation actions to move the arm back and forth along the Cartesian axes, and 6 rotation actions to move the arm counter-/clockwise on the Roll, Pitch and Yaw angles. A *no-op* action is introduced to let the robot in its state. Fig. 2 and Fig. 3 illustrate obtained results by reporting the average cumulative reward and the number of explored states obtained during 10 iterations. In detail, reward values are averaged over 10 simulated fetching trials for each of the iterations and for each algorithm, the continuous lines represent average cumulative rewards while the line width their standard deviation. In the explored state plots, the gray top part of each bar highlights the amount of states expanded during i with respect to the total number of states explored until $i - 1$. While baseline algorithms perform worse in terms of obtained rewards (random-UCT, TD-search), only the vanilla-UCT shows results that are comparable to HI-VAL. However, the number of explored states of vanilla-UCT is significantly higher. Specifically, the naive implementation of UCT evaluates more than two times the number of states that HI-VAL ($\sim$55%). In fact, HI-VAL approximates the optimistic set of a HTN by evaluating only "admissible" actions that are expected to lead the search towards states with high reward. This is achieved by exploiting the action hierarchy updated at each iteration. Moreover, the results compare two different configurations of HI-VAL. The first, VAL, is organized as a single layer structure, where the number of clusters within the layer is equal to the number of primitive actions, while the latter configuration, HI-VAL, is organized over 2 layers where the first layer also contains the set of primitive actions and, the second layer groups actions in 5 clusters. Again HI-VAL further reduces the number of explored states and confirms that a hierarchical evaluation of the search space improves the learning process. In this scenario, we do not notice a significant improvement between HI-VAL and VAL since, differently from the "pick and delivery" task, the structure of "fetching" is not hierarchical. However, we aim at showing that increasing the number of layers in the representation, even when that is not needed, does not damage the obtained performance and, still, slightly decreases the number of visited states.

4.2 Pick and delivery task

Here the environment is represented as a 5x5 grid-world where the Pioneer has to collect an object at a random location and, carry it to an operator. The scenario resembles the one addressed by the "taxi-agent" in [7], however a comparison with max-Q would not be proper since our reward is implemented to be *shaped* and not *sparse*; and we implement our approach in a robotic context where the reduced number of samples and iterations are limiting constraints. Here, the state is a 9-feature vector where the first two components represent the position of the robot, the following two encode the current target of the robot (either the object station or the delivery one), the fifth component

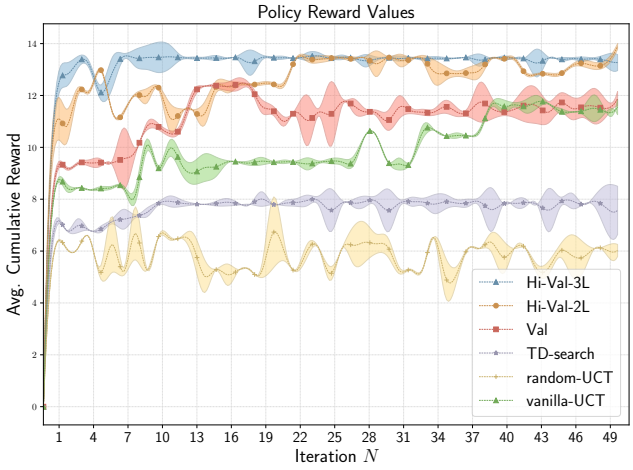

Fig. 4. Average cumulative reward obtained by the random-UCT, TD-search, vanilla-UCT, VAL, HI-VAL-2L and HI-VAL-3L over 49 iterations.

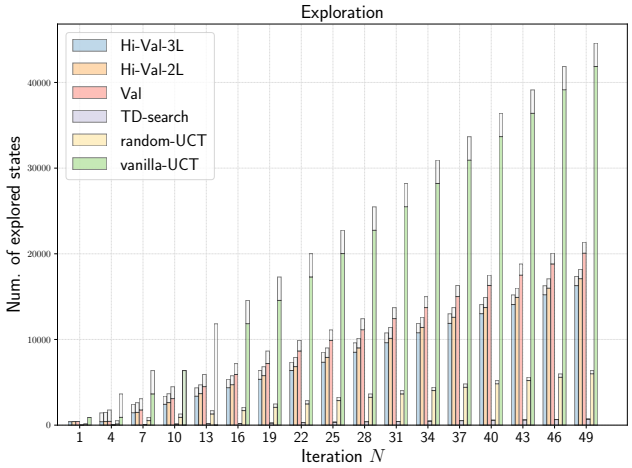

Fig. 5. Number of states expanded by the random-UCT, TD-search, vanilla-UCT, VAL, HI-VAL-2L and HI-VAL-3L over 49 iterations.

indicates whether the object is picked, and the last four components indicate whether there is an obstacle in one of the four possible directions (e.g. wall). The action space of the agent in composed by 6 actions, four to move through cells, and two to pick and drop the object. Moreover, we assume that a robot is helped to collect and drop the objects by an external operator. The reward function is a weighted sum of two components encoding the distance of the robot to the target object and the distance of the object to its delivery station. This task represents a more complex scenario due to temporal

constraints imposed by the status of the object, but as in the previous task, a similar analysis of the results can be observed. Fig. 4 and Fig. 5 illustrate the average cumulative reward and the number of explored states obtained during 49 iterations. Also in this case, vanilla-UCT has comparable reward values, but the number of explored states is still significantly higher than each configuration of HI-VAL. The action hierarchy of HI-VAL, in fact, improves the overall performance showing the best results with a 3 layered structure. Particularly in these complex scenarios – where ordering constraints exist and the task can be decomposed – HI-VAL performs better and confirms that a multi-layered representation of action semantics improves the exploration of the search space.

5 Conclusion

In this paper we introduced HI-VAL, an iterative learning algorithm of hierarchical value functions for policy generation. We discussed its key features and described how it improves search space exploration in order to generate efficient policies. The results of our experimental evaluation show the efficacy of HI-VAL in enabling the agent to learn a good policy by evaluating a significant lower number of states. In fact, HI-VAL can be used to solve different tasks in multiple domains. Finally, we are investigating different directions to further improve HI-VAL, such as a proper formulation for continuous problems. Moreover, we want to explore the possibility of transferring hierarchical value functions among learning agents in order to take advantage of abstract actions and their semantics in different tasks.

References

1. Agostini, A., Celaya, E.: Reinforcement learning with a gaussian mixture model. In: The 2010 International Joint Conference on Neural Networks. pp. 1–8 (July 2010)
2. Anand, A., Grover, A., Mausam, M., Singla, P.: Asap-uct: Abstraction of state-action pairs in uct. In: Proceedings of the 24th International Conference on Artificial Intelligence. pp. 1509–1515. IJCAI'15, AAAI Press (2015), http://dl.acm.org/citation.cfm?id=2832415.2832459
3. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning 47(2-3), 235–256 (May 2002)
4. Bagnell, J.A., Schneider, J.G.: Autonomous helicopter control using reinforcement learning policy search methods. In: 2001 IEEE International Conference on Robotics and Automation. vol. 2, pp. 1615–1620 vol.2 (2001)
5. Chowdhary, G., Liu, M., Grande, R., Walsh, T., How, J., Carin, L.: Off-policy reinforcement learning with gaussian processes. IEEE/CAA Journal of Automatica Sinica 1(3), 227–238 (2014)
6. Clair, A.S., Saldanha, C., Boteanu, A., Chernova, S.: Interactive hierarchical task learning via crowdsourcing for robot adaptability. In: Refereed workshop Planning for Human-Robot Interaction: Shared Autonomy and Collaborative Robotics at Robotics: Science and Systems, Ann Arbor, Michigan. RSS (2016)
7. Dietterich, T.G.: Hierarchical reinforcement learning with the maxq value function decomposition. Journal of Artificial Intelligence Research (JAIR) 13, 227–303 (2000)

8. Erol, K., Hendler, J., Nau, D.S.: Htn planning: Complexity and expressivity. In: the Twelfth National Conference on Artificial Intelligence (Vol. 2). pp. 1123–1128. AAAI'94, American Association for Artificial Intelligence, Menlo Park, CA, USA (1994), http://dl.acm.org/citation.cfm?id=199480.199459
9. Hostetler, J., Fern, A., Dietterich, T.G.: Sample-based tree search with fixed and adaptive state abstractions. J. Artif. Intell. Res. 60, 717–777 (2017), https://doi.org/10.1613/jair.5483
10. Jun, M., Kenji, D.: Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. Robotics and Autonomous Systems 36(1), 37 – 51 (2001)
11. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. International Journal of Robotics Research (July 2013)
12. Kober, J., Peters, J.R.: Policy search for motor primitives in robotics. In: Advances in neural information processing systems. pp. 849–856 (2009)
13. Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning, pp. 282–293. Springer Berlin Heidelberg, Berlin, Heidelberg (2006), https://doi.org/10.1007/11871842_29
14. Kohl, N., Stone, P.: Policy gradient reinforcement learning for fast quadrupedal locomotion. In: 2004 IEEE International Conference on Robotics and Automation. vol. 3, pp. 2619–2624 Vol.3 (April 2004)
15. Konidaris, G., Kuindersma, S., Grupen, R., Barto, A.: Robot learning from demonstration by constructing skill trees. International Journal of Robotics Research 31(3), 360–375 (Mar 2012)
16. Riccio, F., Capobianco, R., Nardi, D.: Dop: Deep optimistic planning with approximate value function evaluation. In: Proceedings of the 2018 International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. – (2018)
17. Riccio, F., Capobianco, R., Nardi, D.: Q-cp: Learning action values for cooperative planning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. – (2018)
18. Ross, S., Gordon, G.J., Bagnell, D.: A reduction of imitation learning and structured prediction to no-regret online learning. In: International Conference on Artificial Intelligence and Statistics. pp. 627–635 (2011)
19. Schaul, T., Ring, M.: Better generalization with forecasts. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence. pp. 1656–1662. IJCAI '13, AAAI Press (2013)
20. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–503 (2016)
21. Silver, D., Sutton, R.S., Müller, M.: Temporal-difference search in computer go. Machine learning 87(2), 183–219 (2012)
22. Stulp, F., Schaal, S.: Hierarchical reinforcement learning with movement primitives. In: 2011 IEEE-RAS International Conference on Humanoid Robots. pp. 231–238 (Oct 2011)
23. Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial intelligence 112(1-2), 181–211 (1999)