Task Variability in Autonomous Robots: Offline Learning for Online Performance

Majd Hawasly and **Subramanian Ramamoorthy**[1]

Abstract. A problem faced by autonomous robots is that of achieving quick, efficient operation in unseen variations of their tasks after experiencing a subset of these variations sampled offline at training time. We model the task variability in terms of a family of MDPs differing in transition dynamics and reward processes. In the case when it is not possible to experiment in the new world, e.g., in real-time situations, a policy for novel instances may be defined by averaging over the policies of the offline instances. This would be suboptimal in the general case, and for this we propose an alternate model that draws on the methodology of hierarchical reinforcement learning, wherein we learn partial policies for partial goals (subtasks) in the offline MDPs, in the form of options, and we treat solving a novel MDP as one of sequential composition of partial policies. Our procedure utilises a modified version of option interruption for control switching where the interruption signal is acquired from offline experience. We also show that desirable performance advantages can be attained in situations where the task can be decomposed into concurrent subtasks, allowing us to devise an alternate control structure that emphasises flexible switching and concurrent use of policy fragments. We demonstrate the utility of these ideas using example grid-world domains with variability in task.

1 Introduction

In this paper we address the autonomous agent's problem of performing a specific task in a world drawn from a family of related worlds with *no opportunity to experiment afresh*. Modelling these as MDPs, we consider that all the processes have the same state-action space, but differ in dynamics and/or reward processes. The agent, typically a robot with limited knowledge of context, may have no knowledge of the *type* of the new MDP; but it is asked to perform the task *as well as possible* and in *real-time*, without further experimentation in this new world.

1.1 Motivation

A robot is considered autonomous when it is capable of achieving relatively sophisticated tasks in changing worlds and over an extended deployment time. A characteristic of these changing worlds is that they are arbitrarily rich and may continuously change. In practice, the robot has only limited time in any newly-assigned task to act efficiently. This *real-time* requirement is due to the change in the world or the expiry of the task.

Consider, as an example, the case of a self-driving car in an urban area. Even though the task is structured and probably well known, the vehicle has to interact with the unmodelled dynamics emerging from the existence of other vehicles and pedestrians. The agent would have been trained in many different situations, but given the size and the richness of the problem, no specific instance would ever happen twice. The vehicle should achieve its task, of navigating toward a goal location, in real-time under the possibly unseen dynamics, and with no chance to repeat the same interaction again as learning algorithms usually require. The issue is more pronounced when the task is inherently unstructured, like the navigation problem for field robotics, or in the domain of disaster response.

1.2 Rationale

The question we are tackling then is: *how should the* offline *experience be utilised to enable the robot to survive the* online*, real-time task?* Here, utilisation is interpreted as determining which behaviours, representations and control structures should be learnt and how they should be used afterwards.

A standard reinforcement learning approach to deal with the variability might be to treat it as uncertainty generated by a big, latent stochastic process. That is, to consider the family of MDPs as one giant MDP. Then, a policy may be learnt for that stochastic process if given enough time and experience. This form of learning *across* the set of MDPs experienced in the offline phase would yield a suboptimal *averaging policy*.

In the control theory literature, where the goal is typically to handle disturbances and dynamics abnormalities by devising large basins of attraction, it is known that it is very hard to find suitably robust large-basin controllers for many complex but realistic systems. So, a collection of controllers is devised instead, with each controller specialised to stabilise a certain context, which then can be sequenced in a way that achieves the desired robustness in the complete task [3, 21]. Composing controllers in robotics has been an issue of interest recently, and, besides control, the question of reasoning about generic robot capabilities to generate viable plans that adhere to task specification has been investigated, e.g., using symbolic reasoning [2].

Here, we take a similar approach but posing the question using reinforcement learning. In [3], Burridge et al. employ a backchaining mechanism on a set of hand-designed feedback controllers with overlapping domains of attraction and goal sets, creating a hierarchy of 'funnels', to produce robust trajectories that lead to the goal starting from a large applicability domain (specifically, the union of individual controllers' domains of attraction). Here, we learn a set of policy fragments from the offline instances, we organise these policies in an appropriate control hierarchy, and we employ a switching paradigm that promotes reactivity to the environment 'feedback', ex-

[1] School of Informatics, The University of Edinburgh, 10 Crichton Street, Edinburgh, United Kingdom, EH8 9AB, email: m.hawasly@sms.ed.ac.uk

tracted from the changes caused by the unmodelled dynamics.

1.3 Approach

Reinforcement Learning (RL) is the classical technique for learning from online interaction [18] giving automatic guarantees for stationary environments, but careful thought and design effort are needed to make it work properly in situations with levels of change that cannot be described as slight parameter drifts. Besides, RL methods require many training episodes to achieve good performance in any task, which is problematic with our real-time 'one-shot' requirement. Model-based methods can achieve acceptable performance faster than model-free methods but need a good model of the environment, which is a 'moving target' that we assume we cannot identify fully. Hence, our focus is on the problem of structuring the RL problem so as to exploit the structure in the world and the task to achieve the stated requirements.

We propose to decompose the task into a collection of subtasks, then learn policies for these subtasks from the offline instances. We claim that factoring the variability into these components is beneficial to the quality of the produced policies. After that, we learn to compose these subtasks for novel instances, producing a policy that takes into account the new instance through an indirect feedback mechanism.

We argue that decomposing the task into subtasks, in the spirit of hierarchical reinforcement learning [1], and learning component-wise policies from the experienced MDPs may be beneficial to online performance. The intuition comes from the benefits of decomposition as a standard approach to learning in intricately complex stochastic systems, such as the process that generates all the variation in our problem, by factoring the variation to multiple subtasks and how they are put together, reducing the blurring effect induced by policy averaging. Our proposed method applies to any domain where the task has internal structure that supports such a decomposition, and many practical domains of interests satisfy this requirement.

We require that the robot has sufficient offline training time to learn in a few samples of the MDP family. In practice, this needs not be a separate phase, but all the experience accumulated in past interactions can be included to handle the new instance (cf. lifelong learning [22]). In the training phase, the robot may develop a set of *capabilities* - generic, reusable controllers that target relevant subtasks across the family of experienced worlds. To interpret these in the language of Hierarchical Reinforcement Learning (HRL), one may consider them to be options [19]: temporally-abstracted actions. If the task supports a set of variation-persistent subtasks, we can utilise the offline phase to develop a hierarchical model describing the task. A key aspect of our proposed approach is that through such a decomposition, a hierarchical model of offline-developed capabilities might outperform the alternative of an averaging policy learnt across the set of unstructured MDPs. In particular, we will show that this enables the agent to 'jump start' in terms of performance in a novel instance, without having to learn afresh.

A caveat that must be associated with any usage of hierarchical models in RL is that the best policy that can be achieved, in the general case, may be suboptimal. The notion of optimality in HRL is known as *hierarchical optimality* which refers to the goodness of policies in the hierarchy-induced policy subspace [5]. Nonetheless, the true optimality can be approached by modifications to the hierarchy or its induced control [7, 5].

One such modification in the options framework is known as interrupting options [19], which is a means that plays on option termination conditions to improve global performance in a specific MDP. The termination condition of an option is not restricted anymore to reaching a terminal state, but also in cases when a better option at some intermediate step can be invoked. This would loosen the constraints on the policy space, allowing policies that are chains of subtrajectories rather than chains of complete option-generated trajectories. To generate the interruption signal, the knowledge of all option values at all states in the specific MDP is usually assumed. This requirement is strong if the world is not known *a priori*. We propose that this can be relaxed by using values that are not immediately from the current instance, but rather statistics from the offline MDPs.

Another improvement can come from the hierarchical decomposition itself. Usually, subtasks are chosen to represent different objectives that the agent may need to achieve while seeking its goal. This decomposition does not often produce truly 'orthogonal' subtasks. The resulting policies, in many cases, share the state-action space and may have overlapping or non-compatible reward signals. Nonetheless, the hierarchical model eventually handles them as if they are truly independent, just as the standard RL framework handles primitive actions. To tackle that, we propose to include in the hierarchy, along with the original subtasks, a collection of *composed subtasks*: policies that achieve the goals of some subtasks concurrently, optimising their overlap.

In this paper, we will use the options framework to build a hierarchy to organise a set of policies (and compositions) learnt from extended offline experience, and plan with a reactive interruption mechanism allowing flexible sequencing in response to changes in the environment. Also, we relax the decomposition boundaries induced by the hierarchy by learning to achieve multiple subtasks at once for subtasks that support concurrency, and propose an alternate control hierarchy around that. We demonstrate these techniques using two gridworld tasks: a navigation task with changing, unpredictable wind, and a resource gathering task with partial observability and adversaries.

2 Setup

2.1 Markov decision process

We assume that the task of the robot can be modelled as a discrete-time Markov decision process (MDP). An MDP m is the tuple (S, A, T, R), where S is a finite state space, A is a finite action space, $T : S \times A \times S \to [0, 1]$ is the (stationary) dynamics of the world, and $R : S \times A \times S \to \mathbf{R}$ is the (stationary) reward process that encodes the goal of the task.

A (Markov) policy for an MDP is a (stochastic) mapping from states to actions, $\pi : S \times A \to [0, 1]$, and the optimal policy π^* is the policy that maximises expected cumulative reward. The cumulative reward is summarised using a state-action value function: $Q^\pi(s, a) = \mathbf{E}^\pi\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s, a\}$ for the future rewards $\{r_t\}$ and the discounting factor γ. The optimal action value function is $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ for all pairs (s, a).

2.2 Family of MDPs

We model the variability in a task using a family of related Markov decision processes $\mathcal{M}$ with the same goal. The family shares the state-action space $S \times A$, but the dynamics $T_i : S \times A \times S \to [0, 1]$ and the reward process $R_i : S \times A \times S \to \mathbf{R}$ may be different for each MDP in the family $m_i \in \mathcal{M}$. We assume that the variability model, formally defined as $\{\mathcal{T}, \mathcal{R}\}$ where $T_i = \mathcal{T}(i)$ and

$R_i = \mathcal{R}(i)$, to be *unknown* to the agent. The agent 'samples' from the variability model during the offline learning phase.

2.3 Semi-Markov decision process

In a semi-Markov decision process (SMDP), actions are allowed to extend over multiple time steps. This makes reasoning about time explicit in learning and planning. This kind of process emerges when dealing with temporally extended actions in MDPs, e.g. in hierarchical reinforcement learning.

2.4 Options and Interruption

The framework of options is one approach to hierarchical reinforcement learning [1, 19] using temporally-extended actions. An option is the three-tuple $\langle I, \pi, \beta \rangle$: $I \subseteq S$ is the initiation state set where the agent is allowed to invoke the option, π is a (Markov or semi-Markov) policy which be followed when the option is invoked, and β is a termination probability distribution over the state space, which encodes stochastically the termination condition of the option. Options are flexible objects that generalise primitive actions which can be considered as (trivial) 1-step options for planning purposes.

For a set of options O, an option switching policy $\mu : S \times O \to [0, 1]$ is a (stochastic) map from states to options. It is proven that sequencing a set of (Markov or semi-Markov) options from O defined over an MDP gives a well-defined semi-Markov decision process (SMDP), allowing planning and learning of μ via similar approaches to planning and learning in MDPs [19]. Q^μ is the option value function for μ.

Normally, an option selected by μ at some state continues to run until its termination condition is satisfied. On the other hand, option interruption is the process of switching control from the running option before its normal termination if its value at the current state s_t is inferior to the expected value of the policy at s_t: $Q^\mu(s_t, o) < \sum_{q \in O} \mu(s_t, q) Q^\mu(s_t, q)$. This is a kind of non-hierarchical execution in HRL, and it allows for performance improvement over the SMDP policy.

3 Handling task variability

3.1 Averaging policy

For a set of sampled MDPs $\mathcal{M}' \subseteq \mathcal{M}$, the *mean MDP* $\bar{m}$ is the process that has the same state-action space $S \times A$ as the members of $\mathcal{M}$ but has the dynamics and reward processes $\{\bar{T}, \bar{R}\} = \mathbf{E}_{\mathcal{M}'}\{\mathcal{T}, \mathcal{R}\}$.

The *averaging policy* $\bar{\pi}$ is the optimal policy for the mean MDP $\bar{m}$. Note that the value function of this policy averages sample returns generated from the models of sampled members in the family. If the variability in $\mathcal{M}$ is extensive, the performance of $\bar{\pi}$ would be poor in general. Intuitively, This is due to the policy trying to choose actions that suit the full spectrum of dynamics and rewards, and thus ending up with actions that are conservative for many of the task instances. Note that the specific averaging policy that will emerge from training on some $\mathcal{M}'$ will depend on the nature of the sampled MDPs as well as the sampled trajectories in them.

3.2 Capabilities: offline options

A *capability* is a skill that targets a specific subtask that occur frequently in the worlds that an agent has experienced. We model that subtask as an MDP that possibly lives in a subspace of $\mathcal{M}$'s state-action space, and which has a localised dynamics and a special reward function.

A capability can be described as an option $\langle I, \pi, \beta \rangle$, with I and β specifying the subtask (where the capability is viable, and when to stop it), and π being the policy learnt for the subtask across the set of offline instances that have been experienced. For this, we call a capability an *offline option*.

Because it is a smaller problem, we argue, as we show later in the experiments, that a capability will be less affected by variability compared to the full task. Intuitively, the initiation set $I \subseteq S$ and the policy π will limit the generated trajectories to a subset of all trajectories, which will reduce the dynamics variability effect on the option. Also, the capability's reward targets, by definition, a more persistent component of the complete task reward, making it less susceptible to reward variability.

Defining capabilities as options allows us to interpret the capability switching strategy as as an option switching policy μ that can be learnt over the SMDP induced from the set of offline options. Remember that in our case, however, the agent is unable to learn the switching policy μ for the online instance directly, due to the real-time requirement. If an averaging switching policy $\bar{\mu}$ is to be learnt to achieve the task from offline experience, this would only produce a hierarchically-optimal policy for the mean MDP, which will be sub-optimal to the averaging policy $\bar{\pi}$ in the general case. To improve on that, we propose to incorporate a notion of option interruption into the process. This is described in the next section.

3.3 Offline interruption

Employing interruption in the options framework not only improves performance through non-hierarchical execution of hierarchical policies, but also adds an element of 'reactivity' in the control structure by making it sensitive to the state of interaction. This appears to be useful for handling unknown situations, but the interruption condition in the original concept [19] requires knowledge of all option values in the desired instance, which we do not have. We propose a modified notion of interruption that depends on values extracted form the offline experience.

Definition 1 (Offline interruption)**.** *A running option o in the unknown MDP m may be* offline-interrupted *at state s_t if the maximum value of o at that state in all instances under the averaging option policy, $\hat{Q}^{\bar{\mu}}(s_t, o) = \max_{m \in \mathcal{M}} Q^{m,\bar{\mu}}(s_t, o)$, is strictly less than the averaging value of interrupting o and selecting a new option according to the policy $\bar{\mu}$: $\hat{Q}^{\bar{\mu}}(s_t, o) < V^{\bar{\mu}}(s_t) = \sum_q \bar{\mu}(s_t, q) Q^{\bar{\mu}}(s_t, q)$.*

The intuition is that the choices of $\bar{\mu}$ would be conservative and 'safe' across the set of experienced MDPs, and following $\bar{\mu}$ would be suboptimal in general. When the best seen value of the running option goes below that stable safety threshold upon reaching some state, it is reasonable to follow the safe choice. Because the agent only observes a subset of all instances $\mathcal{M}' \subseteq \mathcal{M}$, we estimate the maximum value function from the seen instances: $\hat{Q}^{\bar{\mu}}(s_t, o) \approx \max_{m \in \mathcal{M}'} Q^{m,\bar{\mu}}(s_t, o)$. Then, there implicitly lies an assumption that the real value of the option at the current instance would not be higher than what have been seen so far in the offline instances.

Following the original theorem of option interruption [19], we give a condition for offline interruption soundness in the next theorem.

Theorem 1 (Offline interruption)**.** *For a family of MDPs $\mathcal{M}$, a set of options O defined over the family, and an averaging switching* *policy $\bar{\mu} : S \times O \to [0, 1]$, define a new set of option O' with one-to-one mapping between the two option sets except for termination conditions which are defined as follows: $\beta = \beta'$ for all states but the ones in which $\hat{Q}^{\bar{\mu}}(s, o) < V^{\bar{\mu}}(s)$ – that is, the maximum value at s for an option o is less than the expected value of that state under the averaging policy – then we* may *make the termination condition $\beta'(s) = 1$. Let $\bar{\mu}'$ be the policy over O', $\bar{\mu} = \bar{\mu}'$. If the averaging policy is* pessimistic *with respect to values in an instance $m \in \mathcal{M}$, then $\bar{\mu}'$ is no worse than $\bar{\mu}$: $V^{m,\bar{\mu}'}(s) \geq V^{m,\bar{\mu}}(s)$ for all $s \in S$.*

Proof. We follow the proof in [19]. In some arbitrary MDP m, for $V^{m,\bar{\mu}'}(s) \geq V^{m,\bar{\mu}}(s)$ to be true for arbitrary s, it is enough to show that following μ' from s then continuing with μ is no worse than following μ all the time. So, it is enough to show that: $r_s^{o'} + \sum_{s'} p_{ss'}^{o'} V^{m,\bar{\mu}}(s') \geq r_s^o + \sum_{s'} p_{ss'}^o V^{m,\bar{\mu}}(s')$. The two sides are equal for any history ss' that is not interrupted (because the policies are the same in that case), therefore we shall only consider the expected value under interrupted histories. That is, we would like to prove that: $E\{r + \gamma^k V^{m,\bar{\mu}}(s')\} \geq E\{\beta(s')[r + \gamma^k V^{m,\bar{\mu}}(s')] + (1 - \beta(s'))[r + \gamma^k Q^{m,\bar{\mu}}(s', o)]\}$, for the interrupted histories ss' under o' followed by o. This is true if $Q^{m,\bar{\mu}}(s', o) \leq V^{m,\bar{\mu}}(s')$, i.e. the return from the interrupted option o at interruption state s' *in the specific instance* m is less than the return of $\bar{\mu}$ in m.

We know that $Q^{m,\bar{\mu}}(s', o) \leq \hat{Q}^{\bar{\mu}}(s', o)$ (by the definition of $\hat{Q}^{\bar{\mu}}$), and that $\hat{Q}^{\bar{\mu}}(s', o) < V^{\bar{\mu}}(s')$ for all interruption states s' (by the definition of offline interruption). For that, the proof will be completed when $V^{\bar{\mu}}(s') \leq V^{m,\bar{\mu}}(s'), \forall s'$, i.e. the averaging policy pessimistically underestimates the values of the interruption states in the instance m. □

Although the premise would not be true always for any instance $m \in \mathcal{M}$, the method is still able to produce good results empirically. Algorithm 1 gives a simple procedure for decision making with offline options and offline interruption.

Algorithm 1 Decision making with Offline Interruption

```
Require: $O$: option set; $\bar{\mu}$: averaging option policy; $Q^{\bar{\mu}}$: value
    function of $\bar{\mu}$ over $O$; $\hat{Q}^{\bar{\mu}}$: maximum values of the option set
    $O$ under $\bar{\mu}$; $s_t$: current state.
 1: $o_{run} \leftarrow \phi$.
 2: for every time step $t$ do
 3:   if $o_{run}$ is empty then
 4:     $o_{run} \leftarrow \arg\max_{o \in \mathcal{O}} Q^{\bar{\mu}}(s_t, o)$.
 5:   else
 6:     if $\hat{Q}^{\bar{\mu}}(s_t, o_{run}) < \sum_q \bar{\mu}(s_t, q) Q^{\bar{\mu}}(s_t, q)$ then
 7:       $o \sim \bar{\mu}(s_t, .)$
 8:       $o_{run} \leftarrow o$.
 9:     end if
10:   end if
11:   $a_t \sim \pi^{o_{run}}(s_t, .)$.
12: end for
```

In the algorithm, $\bar{\mu}(s, .)$ is a probability distribution over options, corresponding to the option switching averaging policy, and $\pi^o(s, .)$ is a probability distribution over other options or primitive actions, corresponding to option o's policy.

3.3.1 Example - Windy gridworld

The aim of this experiment is to test offline options and offline interruption. A gridworld of 5×5 cells has an obstacle with two exits (Figure 1). Wind blows immediately before the exits. It has an unknown but fixed direction in any instance, and it blows strong gusts with an unknown but fixed probability throughout the episode, pushing the agent one cell at a time. The goal of the agent, starting from a cell in the the leftmost column (marked with 'S'), is to pass one of the exits to the right side (marked with 'G'). The agent gets -1 penalty for every action taken until the goal is reached or the episode elapsed (100 time steps). Moving towards a wall does not change the location of the agent, but it will cost it the usual penalty.

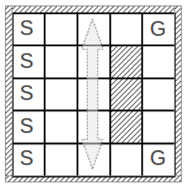

Figure 1. The Windy gridworld. The agent starts randomly in one of the cells marked with 'S' and is tasked with reaching any of the cells marked with 'G'. The arrow indicates the locations and possible directions of wind.

Each instance of this MDP family is characterised with two parameters (p, dir). $p \in [0, 1]$ is the probability with which the wind will succeed in changing the position of the agent, while $dir \in \{North, South\}$ is the wind direction. The agent might be pushed one cell in the direction dir with the probability p while in the windy cells. Figure 1 shows the setup.

The agent experiences many instances of this family in the offline phase, and learns an averaging policy across all these instances. This is a policy over the primitive actions that reaches the goal via any of the two exits. At the same time, the agent is made to learn two options with handpicked goals, one for reaching the goal through each of the exits. The agent learns an averaging option switching policy as well, and estimates the maximum option values from these training instances using a Monte-Carlo learning method.

Figure 2 shows the result of 5 runs of a 10000-episode testing phase.

In each test episode, the agent is given 100 time steps in a new instance (p, dir), in which both the flat averaging policy and the offline-interrupted option policy are evaluated. The performance criterion is the accumulated reward in the episode (ranging from -100 to 0). We report in Figure 2 the difference in performance between the two methods, sorted in ascending order to ease interpretation.

As the figure shows, the offline-interrupted option policy is no worse than the flat averaging policy in almost 80% of all episodes. This can be justified by the ability of the leant partial-policies to capture delicate details about the interaction (e.g. the consistent correlation of the wind direction in the windy cells), in contrast to the flat averaging policy. Also, the interruption mechanism allowed for active intervention in the control process (sensed through the change in state) that put that knowledge into play. The results look similar if the setup is not identical in the learning and testing phases, e.g. having only *southerly* winds in the test phase.

3.4 Composition-based hierarchy

Although decomposition of a task is beneficial to manage variability and improve performance in an unknown world, the hierarchy does limit the producible policies to a policy subspace spanned by

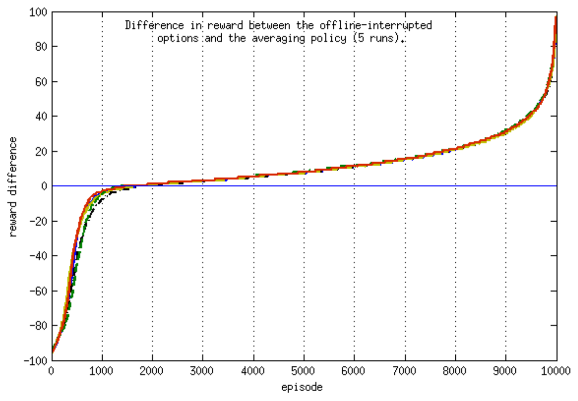

Figure 2. Difference in performance between the offline-interrupted option policy and the averaging policy in 5 runs. Values above zero are episodes where the offline interruption outperforms the averaging policy. Note that episodes are sorted in the figure by performance to facilitate interpretation.

sequences of (sub-)trajectories generated by the (interrupted) option policies. This is a known problem of hierarchical reinforcement learning that resulted in adopting various optimality criteria by HRL methods [5]. We propose next a representation for tasks that admit concurrent execution of subtasks.

3.4.1 Subtask composition

In robotic applications, subtasks are rarely fully independent in the way HRL frameworks handle them. That is, it is common that option policies share state-action space and may have conflicting or interacting goals. We propose to learn options that target the goals of more than one subtask at once, and we call these *composed subtasks*. This can be interpreted as a controlled kind of concurrency for policy synthesis.

Definition 2 (Composed subtask)**.** *If the domain of applicability of two offline options happen to intersect in a non-trivial way, a new option can be defined for that intersection where both the corresponding subtasks are tenable. For o_1 and o_2 being offline options with domains $S_1 \times A_1$ and $S_2 \times A_2$ respectively, a composed subtask $o_{1,2}$ is a capability (of order 2) defined for the state space $S_1 \cup S_2$ and the action space $A_1 \cup A_2$ that optimises the goals of the two subtasks simultaneously. For the composed subtask $o_{1,2}$, the subtasks o_1 and o_2 are called* components.

Because $o_{1,2}$ subsumes o_1 and o_2 in domain and reward, an action taken by a component subtask policy is an action of the composed subtask from a learning point of view. This leads to that every learning update of a component policy also triggers an update for its parent, composed subtask. Thus, almost all learning in the hierarchy can happen *off-policy* while learning the components at the leaves, and very little additional learning effort is ever needed.

Composition is not limited to order 2. Three subtasks may be composed to produce a higher-order subtask, but if the components are *pairwise-composable*. The composition relation defines a tree hierarchy with the primitive options at the leaves, and more complex subtasks at higher levels.

3.4.2 Model description

Composed subtasks can be used as any other option for the purpose of policy synthesis, along with their offline values and interruption. This will produce richer policies that are closer to true optimality. However, this ‘unstructured’ approach ignores the natural subsumption of these subtasks. We would like to see the composed subtask given the priority over its components when it is able to achieve their joint objectives. That is, an appropriate component should take charge only when the composed subtask is not tenable, but handles control immediately when it is. The generic options framework does not provide us with this flexibility.

To emphasise that, we propose a control hierarchy for concurrent, rather than sequential, tasks that we impose over our offline option implementation. In the *composition-based hierarchy*, control always starts with the highest-order composed subtask (which achieves the goals of all the subtasks underneath). Note that the policy of this subtask is the full task averaging policy. This process is allowed to run as long as it is able to achieve its objectives. Otherwise, it is interrupted and control is moved to an appropriate component. The component, which may be composed from other subtasks as well, is run in a similar fashion. The trick is that the status of the parent subtask is continuously checked, and when it is ready to run again, the running component is interrupted and the parent subtask regains control. In short, control is moved up and down the tree hierarchy, from the more general to the more specific and vice versa, in response to changes in the world captured by the offline interruption function. This can be seen as a special kind of *polling execution* of non-hierarchical execution of hierarchical policies [5].

3.4.3 Learning a model of subtask attainability

The mechanism of offline interruption is one way to learn control switching from seen instances, but it is not by any means the only one. Any function that has the quality of predicting the viability of subtasks can be considered a generalisation to the interruption mechanism. This, for example, is important if the variability in rewards is high in a way that makes relying on upper bounds less useful. *Subtask robustness* is one example of these generalisations.

Robustness is an offline estimate of goal attainability that does not depend immediately on the rewards accumulated by the different policies. Rather, it abstracts away from values and ask explicitly about the success or failure of a subtask. It utilises a level of aspiration as a threshold to control when to switch out from one policy.

One simple realisation of robustness can be in the form of a state-action value function, e.g. acquired using Q-learning. The value $\mathbf{R}(o, s, a)$ estimates the expectation that the option o will be able to achieve its goals from state s and action a. Switching can be controlled through a threshold τ that represents the level of reliability/safety the agent requires. Whenever the option fails to deliver robustness at least as high as the threshold, e.g. due to unobservable task parameters, it is deemed unsafe to use and suspended. This is a trade-off between performance and safety, as robustness prefers a policy that is safe when the world unexpectedly changes, but cannot in general guarantee performance. The robustness threshold τ can be used as a tunable parameter to control the risk attitude of the agent.

Note, however, that the notion of robustness should not be restricted to simple scalar value functions, but can have other forms that reflect the viability of capabilities at states in different settings. This may include, for example, a Pareto efficiency measure with a Pareto frontier as a switching threshold in multi-objective capabilities, or the utility of correlated equilibria with a minimax solution threshold in interactive, multi-agent setting.

3.4.4 Algorithm

Algorithm 2 shows a simple procedure for action selection by hierarchical search in a composition-based hierarchy $\mathcal{K}$. The running option o is used as long as it is able to produce satisfactory behaviour as defined by its robustness function (line 6). If the robustness drops below the threshold τ, o is suspended and one of its components are selected using a suitable metric, e.g. robustness (line 10). The new capability is used similarly (line 11) until the robustness of the parent is recovered above its threshold (line 3).

Algorithm 2 HierarchicalSearch

```
Require: $\mathcal{K}$: task hierarchy; $o_{run}$: running subtask; $p$: parent subtask; $\tau$: robustness threshold; $s_t$: current state.
 1: $a_p \sim \pi^p(s_t, .)$.
 2: if $\mathbf{R}(p, s_t, a_p) > \tau$ then
 3:     return $a_p$.
 4: else
 5:     $a \sim \pi^{o_{run}}(s_t, .)$.
 6:     if $o_{run}$ is primitive option or $\mathbf{R}(o_{run}, s_t, a) > \tau$ then
 7:         return $a$.
 8:     else
 9:         $O' \leftarrow$ components of $o_{run}$ in $\mathcal{K}$.
10:         $q^* \leftarrow \arg\max_{q \in O'} \mathbf{R}(q, s_t, a)$.
11:         return HierarchicalSearch($\mathcal{K}$, $q^*$, $o_{run}$, $\tau$, $s_t$).
12:     end if
13: end if
```

The first call to the algorithm takes the root of the hierarchy as the running subtask. $\pi^o(s, .)$ is a probability distribution over primitive actions related to the policy of option o at state s. The function $\mathbf{R}(o, s, a)$ returns the robustness of state-action pair (s, a) for the option o. Finally, the function $\mathbf{HierarchicalSearch}$ is a recursive call for the procedure itself.

3.5 Example - Wargus resource gathering task

Following the setup in [14], we implemented Wargus resource gathering task. On a gridworld of 32×32 cells, an agent has to collect items while avoiding an adversarial patrol agent. Agents can move in the eight compass directions, they can only see objects less than 8 cells away (but bearing is always observed), and the adversary can shoot for a maximum range of 5 cells. The objective of the agent is to collect as many items as possible in a specific duration. These items appear in the world one at a time and stay in place until picked by the agent, triggering a new item to appear in a random spot. The patrol agent navigates in a random walk most of the time, but when it sees the agent it uses the shortest path to it, then shoots once in range.

The variability in this task comes from the random (and, most often, unobservable) location of items, and the random (and, most often, unobservable) location of the adversary. A specific assignment of these two parameters produces one MDP from the possible family.

3.5.1 Experimental setup

No relearning is allowed for our agent, and hence every instance of the world is tried by the agent only once. We mix the training and testing phases in this experiment, such that new instances use all the knowledge gathered in all the previous episodes, an approach related to the notion of lifelong learning [22].

For this, the agent starts with *no prior knowledge* of any sort and learns everything (averaging policy, options, switching policies, robustness) from scratch.

An episode starts with a random positioning of the two agents and a single item, and it only ends when the agent is destroyed or when the episode elapses. Performance is measured by the number of items the agent manages to gather throughout the episode. A set of 30 episodes followed by a set of 5 nominal test episodes is called a trial. The test episodes have fixed parameters, but the agent is not allowed to learn in them. This is in order to test the improvement in performance as more experience is gathered. We report the scores achieved in the nominal test episodes.

The experiment is run for 500 trials, and repeated 5 times with the final scores averaged and smoothed.

3.5.2 Methods

We compare the composition-based hierarchy with 3 other methods: a simple averaging policy, offline options, and offline options with interruption:

- Averaging policy: Q-learning over the 8 primitive actions, trained across all the experienced instances.
- Offline options: three options, and their switching policy, are learnt. The options are: `ToGoal (TG)` for navigating towards the item, `FromEnemy (FE)` for navigating away from the patrol agent, and `TG+FE`, the composition of the two. These options are designed to start anywhere on the grid, and terminate upon the occurrence of an event such as seeing the adversary or losing sight of the goal item. The option policies and the switching policy are averaging policies over the experienced instances.
- Offline options with interruption: we added to the previous implementation an offline interruption mechanism that uses the offline values of the option policy. That is, the policy may switch between the three options based on their maximal historical values, rather than having to wait until normal termination.
- Composition-based hierarchy: the hierarchy in Figure 3 is implemented using the same options as above and the following robustness function: the robustness values for the `TG` is acquired using Q-learning with the reward $+1$ if an action leads the agent to where the goal is reachable (seen), -1 only when it leaves the reachability area, and 0 otherwise. For `FE`, a penalty of -1 is given as long as the agent is within the range of sight of the opponent, $+1$ once when it escapes it, and 0 otherwise. The composed subtask `TG+FE` learns using the sum of the two rewards. The robustness threshold is chosen empirically to be 2 for all capabilities.

3.5.3 Results

We compare the performance of the four methods for 500 trials. The results are shown in Figure 4.

As the results show, using the composition-based hierarchy is superior to the other three methods. It exploits the composed capability much more than the other methods, specifically in every state where the subgoals are not in conflict, producing the score difference. Notice that the composition-based hierarchy is approximating the optimal averaging policy, hence it cannot beat the averaging policy's Q-learning asymptotically if given enough time and experience. Still,

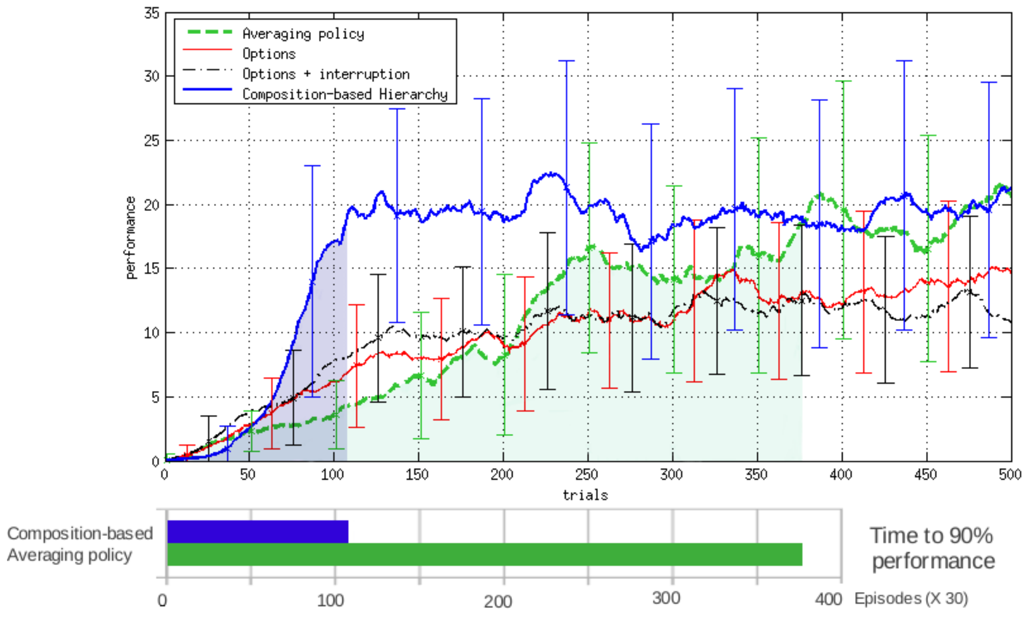

Figure 4. Performance in 500-trial Wargus resource gathering experiment. The curves show the average item count in the test phases of each trial. The curves are means of 5 repetitions, and the error bars show the standard deviation. The composition-based hierarchy (solid - blue) outperforms the other implementations in this task with a clear margin, and achieves the performance level of the averaging policy (green - dashed) in less than third the time. The bars underneath the plot show the time needed by these two methods to achieve 90% to their maximum performance.

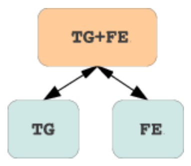

Figure 3. Composition-based hierarchy of the Wargus resource gathering experiment. The bottom nodes represent the subtasks of seeking the resource and escaping the adversary, respectively. The upper node is the composition of the two, with the goal of achieving both objectives at once (here, the full task). The arrows connecting the nodes suggest how control flows flexibly up and down the hierarchy in response to changes in the environment.

the performance head-start is evident as well as the persistent difference in performance for the first half of the experiment. The averaging policy approach needs around 15000 episodes to catch up. The shaded regions in the figure show the time required by the two methods to achieve 90% of their final performance. While the averaging policy needed more than 350 trials, the composition-based hierarchy achieved that in around 100 trials.

Our method needed more time than the other methods in the first few episodes to ascend performance. This can be justified by the need to learn a robustness function in addition to learning the option policies as required by the other methods as well. Remember that these policies and functions are to be readily learnt in the offline phase, preparing the robot to perform immediately in the real world.

4 Related work

The problem of dealing with task variations is of fundamental interest within autonomous robotics, and autonomous agent design in a more general setting. This problem has been studied from many different angles, with tools and techniques being developed to address aspects of the full problem.

One such thread, of relevance to the discussion in this paper, is transfer for reinforcement learning [20]. Techniques for transfer learning attempt to use an existing policy in a source task, typically as an exploration policy to bias learning in a novel target task, with the hope of achieving learning speed-up. When transfer works well, it is because of exploitation of structural properties of tasks which allows for reuse. This is the high-level goal of our approach as well. However, at the algorithmic level, in contrast to transfer as a bias that speeds up learning over numerous trials, we seek a policy that may be immediately applied in a novel instance (in some cases, as a sophisticated initialisation to a final learning step which allows for convergence to a true optimum, but we do not handle this in this paper). Another difference is that we aim to repeatedly apply that transferred policy in many different novel, unknown worlds, while the usual assumption in transfer is that there is a single (and usually known) target task. This is the motivation behind our definition of the problem in terms of policy fragments that are sequenced in different ways to achieve a novel policy.

Multi-task reinforcement learning (MTRL) [20] is a specific branch of the general problem of reinforcement learning transfer, in that it explicitly targets the problem of variation within a family of tasks. Typically, in the MTRL setting, tasks share the same state-action space, and the aim is to learn a policy from sampled task instances that appropriately utilises this experience of multiple contexts, for example, yielding an Average Value Function [13] which is similar to our averaging policy. Some MTRL methods (e.g. [12]) assume the observability of the *type* of the task, while our assumption that the agent is oblivious to that is more practically plausible. Others allow the agent many trials in the new world (e.g. [6]), while we require the agent to act promptly without time to learn afresh. Bayesian methods (e.g. [23]) assume explicit knowledge of the distribution describing the MDP family, whereas we do not.

In this paper, we assumed that the capability goals are chosen by the designer. However, learning these could have also been considered (cf. learning structure and subgoal discovery in HRL, e.g. [11, 17, 9, 8]).

Learning termination conditions for pre-acquired options is discussed in [4]. They use gradient descent procedure on a special functional encoding of the termination condition to learn the optimal switching points under certain requirements, not necessarily maximising accumulated reward. We aim for a more general and less constrained approach for interruption, using offline values and robustness functions.

Finally, Concurrency and composition in HRL is a problem with some history within the literature, e.g., concurrent ALisp [10] and concurrent options [15]. In [16], an explicit approach to composing policies is given. We alternatively opted for allowing the agent to learn the composed subtasks in the offline training phase, as it learns the other options. We believe this to be the better approach to capture the intricacies of different domains and subtask models in order to be able to deal with the problem of offline learning for improved online performance.

5 Conclusion

The main motivation for this paper is the problem of designing an autonomous robot or agent that is capable of quickly and efficiently solving a variety of different problems, drawn from some family defining the domain. This family of problems represents many real world effects, including incompleteness of knowledge arising from the arbitrary richness of the environment (e.g., factors outside the model that do have additional dynamics), or continual change (such as due to other agents in the environment). This way of phrasing the problems differs from the more traditional question of obtaining an optimal policy for a stochastic environment, although the issue is increasingly being considered in many different communities such as under the heading of transfer and multi-task learning.

We have presented a novel approach to policy design and learning, wherein we learn subtasks that make sense across the entire domain (for multiple task/environment settings) and associate with them models such as for interruption. This allows us to define novel policies in terms of compositions of policy fragments that are learnt offline. Our approach builds on existing methodologies such as hierarchical RL with options, but modifies them to address the more general problem identified above. With this, we demonstrate that we are able to achieve superior performance in an online setting, benefiting from problem structuring.

ACKNOWLEDGEMENTS

This work has taken place in the Robust Autonomy and Decisions group within the School of Informatics. Research of the RAD Group is supported by the UK Engineering and Physical Sciences Research Council (grant number EP/H012338/1) and the European Commission (TOMSY Grant Agreement 270436, under FP7-ICT-2009.2.1 Call 6).

REFERENCES

[1] A.G. Barto and S. Mahadevan, 'Recent advances in hierarchical reinforcement learning', *Discrete Event Dynamic Systems*, **13**(4), 341–379, (2003).

[2] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G.J. Pappas, 'Symbolic planning and control of robot motion [grand challenges of robotics]', *Robotics & Automation Magazine, IEEE*, **14**(1), 61–70, (2007).

[3] R.R. Burridge, A.A. Rizzi, and D.E. Koditschek, 'Sequential composition of dynamically dexterous robot behaviors', *The International Journal of Robotics Research*, **18**(6), 534–555, (1999).

[4] G. Comanici and D. Precup, 'Optimal policy switching algorithms for reinforcement learning', in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 709–714. International Foundation for Autonomous Agents and Multiagent Systems, (2010).

[5] T.G. Dietterich, 'Hierarchical reinforcement learning with the maxq value function decomposition', *Journal of Artificial Intelligence Research*, **13**(1), (1999).

[6] F. Fernández and M. Veloso, 'Probabilistic policy reuse in a reinforcement learning agent', in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 720–727. ACM, (2006).

[7] L.P. Kaelbling, 'Hierarchical learning in stochastic domains: Preliminary results', in *Proceedings of the Tenth International Conference on Machine Learning*, volume 951, pp. 167–173. Citeseer, (1993).

[8] S. Kazemitabar and H. Beigy, 'Automatic discovery of subgoals in reinforcement learning using strongly connected components', *Advances in Neuro-Information Processing*, 829–834, (2009).

[9] S. Mannor, I. Menache, A. Hoze, and U. Klein, 'Dynamic abstraction in reinforcement learning via clustering', in *Proceedings of the twenty-first international conference on Machine learning*, p. 71. ACM, (2004).

[10] B. Marthi, S. Russell, D. Latham, and C. Guestrin, 'Concurrent hierarchical reinforcement learning', in *Proceedings of the national conference on artificial intelligence*, volume 20, p. 1652. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, (2005).

[11] A. McGovern and A.G. Barto, 'Automatic discovery of subgoals in reinforcement learning using diverse density', *Computer Science Department Faculty Publication Series*, 8, (2001).

[12] N. Mehta, S. Natarajan, P. Tadepalli, and A. Fern, 'Transfer in variable-reward hierarchical reinforcement learning', *Machine Learning*, **73**(3), 289–312, (2008).

[13] T.J. Perkins and D. Precup, 'Using options for knowledge transfer in reinforcement learning', *University of Massachusetts, Amherst, MA, USA, Tech. Rep*, (1999).

[14] M. Ponsen, M. Taylor, and K. Tuyls, 'Abstraction and generalization in reinforcement learning: A summary and framework', *Adaptive and Learning Agents*, 1–32, (2010).

[15] K. Rohanimanesh and S. Mahadevan, 'Decision-theoretic planning with concurrent temporally extended actions', in *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pp. 472–479. Morgan Kaufmann Publishers Inc., (2001).

[16] K. Rohanimanesh and S. Mahadevan, 'Coarticulation: An approach for generating concurrent plans in markov decision processes', in *Proceedings of the 22nd international conference on Machine learning*, pp. 720–727. ACM, (2005).

[17] Ö. Şimşek and A.G. Barto, 'Using relative novelty to identify useful temporal abstractions in reinforcement learning', in *Proceedings of the twenty-first international conference on Machine learning*, p. 95. ACM, (2004).

[18] R.S. Sutton and A.G. Barto, *Reinforcement learning: An introduction*, volume 1, Cambridge Univ Press, 1998.

[19] R.S. Sutton, D. Precup, and S. Singh, 'Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning', *Artificial intelligence*, **112**(1), 181–211, (1999).

[20] M.E. Taylor and P. Stone, 'Transfer learning for reinforcement learning domains: A survey', *The Journal of Machine Learning Research*, **10**, 1633–1685, (2009).

[21] R.L. Tedrake et al., 'Lqr-trees: Feedback motion planning on sparse randomized trees', (2009).

[22] S. Thrun and T.M. Mitchell, 'Lifelong robot learning', *Robotics and autonomous systems*, **15**(1-2), 25–46, (1995).

[23] A. Wilson, A. Fern, S. Ray, and P. Tadepalli, 'Multi-task reinforcement learning: a hierarchical bayesian approach', in *Proceedings of the 24th international conference on Machine learning*, pp. 1015–1022. ACM, (2007).