# Adaptive Skill Acquisition in Hierarchical Reinforcement Learning

Juraj Holas and Igor Farkaš

Faculty of Mathematics, Physics and Informatics*
Comenius University in Bratislava, Slovakia
{juraj.holas,igor.farkas}@fmph.uniba.sk

**Abstract.** Reinforcement learning has become an established class of powerful machine learning methods operating online on sequential tasks by direct interaction with an environment instead of processing precollected training datasets. At the same time, the nature of many tasks with an inner hierarchical structure has evoked interest in hierarchical RL approaches that introduced the two-level decomposition directly into computational models. These methods are usually composed of lower-level controllers – *skills* – providing simple behaviors, and a high-level controller which uses the skills to solve the overall task. Skill discovery and acquisition remain principal challenges in hierarchical RL, and most of the relevant works have focused on resolving this issue by using pretrained skills, fixed during the main learning process, which may lead to suboptimal solutions. We propose a universal pluggable framework of *Adaptive Skill Acquisition* (ASA), aimed to augment existing solutions by trying to achieve optimality. ASA can observe the high-level controller during its training and identify skills that it lacks to successfully learn the task. These missing skills are subsequently trained and integrated into the hierarchy, enabling better performance of the overall architecture. As we show in the pilot maze-type experiments, the identification of missing skills performs reasonably well, and embedding such skills into the hierarchy may significantly improve the performance of an overall model.

**Keywords:** Hierarchical reinforcement learning · skill acquisition · adaptive model

## 1 Introduction

As an approach inspired by the knowledge acquisition process in humans and other animals, the Reinforcement Learning (RL) has gained significant interest in both research studies and practical applications. Despite the impressive progress in traditional or 'flat' RL in the recent years, these approaches still struggle to solve tasks involving several layers of abstraction. To render such problems tractable, a promising approach of *Hierarchical Reinforcement Learning* (HRL) was introduced in [22].

* Supported by grant 1/0796/18 from Slovak Grant Agency for Science (VEGA)

Introduction of hierarchy into the RL framework has the ability to accelerate the learning process in sequential decision-making tasks, as the agents on different levels of hierarchy can decompose the problems at hand into smaller subproblems. The common underlying feature of the ongoing research in HRL field is the usage of skills – actions that are temporally extended in time – forming an implicit or explicit hierarchy within the task. The top-level task represents the original problem at hand and is solved by RL (the core Markov decision process, MDP), while the lower level may be fixed, pre-trained, or solved by separate RL themselves (sub-MDPs).

In the simplest scenario, the set of skills can be hand-crafted and trained manually, as a series of independent RL agents [17, 22]. Other methods use the pre-training phase to explore the nature of the environment and determine a useful set of skills to be trained [6, 7, 9, 13–15]. Moving towards more universal solutions, we may observe HRL algorithms that are able to (semi-)automatically learn hierarchies [1, 9, 16, 19, 23]. These algorithms often construct and learn the hierarchy one level at a time in a bottom-up fashion, hence the higher-level policies are trained only when the lower-level ones have been fixed.

However, having the skill set fixed before training the higher level of a hierarchy can considerably limit the final performance, as discussed in [11]. First, the skills are trained either using a surrogate reward signal, or by the deconstruction of the state space in a pre-training phase, which leads to useful, yet not necessarily optimal set of skills. Subsequently, the top-level agent using these skills as actions may not be able to optimally learn the overall task. As an example, we can imagine a walking robot tasked to navigate through the maze, and provide it with two skills: walk forward and turn left, but no 'turn right' skill. The top-level controller can still learn a strategy to solve the maze with given skills, however it will be clearly suboptimal for cases where the robot should have turned right. This principle of *optimality under given hierarchy* must be accounted for when designing HRL architectures [22].

Research has also been performed on training the whole hierarchy at a time, especially in continuous state and action spaces. A number of papers on this topic are restricted for use with only Universal-MDP (which, despite its name, is a subset of MDP), limited only to spatial tasks [11, 16, 23]. Only a little work has been done on training the entire hierarchy in the continuous environment of a general MDP [12]. All of these approaches, however, rely on building a specialised hierarchical structure of abstract agents, and on using a tailored algorithm to train such a structure, which is not transferable to any other infrastructure. Hence, adapting their work into an existing model in order to improve its efficiency may not be feasible, and the complete replacement of the existing model may be necessary.

In this paper, we present a new approach called *Adaptive Skill Acquisition* in Hierarchical Reinforcement Learning, or ASA-HRL. It represents a pluggable component able to dynamically construct and integrate missing skills into any hierarchy of agents. The vast majority of HRL approaches fix the skill set prior to training the higher level of hierarchy, which, as discussed, can lead to im-

perfections within the hierarchy. We focus on the most common case of such imperfections – a useful skill that was not identified during lower-level training, and hence is missing from the skill set. While the top-level agent is being trained, ASA can automatically identify these missing skills, train them, and incorporate them into the hierarchy. The top-level agent then resumes its training with this enriched hierarchy, and can solve the core-MDP more efficiently.

The ASA approach is composed of three key steps: First, the missing skill is identified by self-observation of the top agent, by gathering statistics about the sequences of skills it invokes. Second, the identified skill is trained by traditional RL methods. Third and final step is to integrate the newly trained skill into the overall hierarchy, after which the training of top-level controller can continue.

## 2 Preliminaries

We define a *Markov Decision Process* (MDP) as a tuple $\langle S, A, P, p_0, R, \gamma \rangle$, where $S$ is a set of states; $A$ is a set of actions; $P : S \times A \times S \to [0, 1]$ is a probability distribution describing the state transition; $p_0 : S \to [0, 1]$ is a probability distribution of the initial state, $R : S \times A \to \mathbb{R}$ is a (possibly non-deterministic) reward function; $\gamma \in (0, 1]$ is a reward discount factor. Traditional ('flat') RL aims to find an optimal policy $\pi : S \times A \to [0, 1]$ that optimizes the expected overall discounted return $G(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)]$, where $\mathbb{E}_\pi$ denotes the expected value if an agent follows a policy $\pi$.

In Hierarchical RL, we do not optimize a single policy $\pi$, but rather a set of policies on two (or more) levels. We have a set of skills – low-level policies $\pi^L_1, \ldots, \pi^L_n$ that act using the original actions: $\pi^L_i(s_t) = a_t$. On top of them we have a manager – high-level policy $\pi^H$. Its purpose is to decide which skill will be used in a given situation, and thus its high-level actions $a^H_t$ are in fact invocations of skill policies: $\pi^H(s_t) = a^H_t \in \{\pi^L_1, \ldots, \pi^L_n\}$ . In our paper we study variable-length sequences of such skill invocations, which we denote by $\delta = [a^H_t, \ldots, a^H_{t+k}]$.

## 3 Related work

Building agents that can learn hierarchical policies is a longstanding problem in Reinforcement Learning. One of the most general approaches to define temporally extended hierarchies is the *Options framework* [22], upon which most of other research is built. However, most HRL approaches only work in discrete domains [2, 4, 7, 13–15, 22], and only recent ones can be applied to high-dimensional continuous domains [1, 6, 9, 10], as we do in this work.

Numerous methods use a different reward signal for training the lower level, either in a form of a low-level target produced by a high-level agent [2, 11, 16], or by constructing a surrogate reward for each skill which may lead to more versatile skills [6, 23]. Similarly to those, we also construct a specialised reward signal for each missing skill, so that each new skill fills in the specific gap in the skill set.

While the vast majority of research field focuses on strictly two-level hierarchies [1, 2, 6, 7, 13, 14, 9, 10, 15, 22, 23], some authors opted for more dynamical multi-level hierarchies, essentially forming a tree of skills [4]. Despite having tested ASA only on two levels, we designed it in a way that it could be in principle deployed on any level of general tree-like hierarchy.

A slight parallel to our approach can be found in Skill-chaining [9], a two-level HRL method that incrementally chains options backwards from the end goal state to the start state. Similarly to us, they also construct new skills based on their starting and target region. However, their algorithm is well suitable for near-linearly organized tasks, but should fail to achieve reasonable results if the environment features high branching factor, or if it is not bounded at all.

Closer to our work is the algorithm [21], which also allows for enrichment of the skill set by supplementing it with new ones. However, they require a hand-defined curriculum of tasks supported by pre-defined stochastic grammar from which the new skills are generated, which poses a fixed limit for capabilities of new skills. Our key advantage relative to their approach is that ASA is not bounded by a predefined set of possible skills, and thus can create a new skill that was not considered in advance.

## 4 Our approach

We propose an approach named *Adaptive Skill Acquisition* (ASA) in hierarchical reinforcement learning. The novelty of this approach lies in an additional augmentation of existing pre-trained skills according to the needs of the agent.

In its core, ASA is an algorithm that enables the agent to *add new skills* if needed, in the midst of learning the core task. If an agent was presented with a set of pre-trained skills (as in [1, 6, 7, 9, 13, 14, 17, 19, 22]), it is possible that this set would not contain all skills necessary to solve the core task. In such case, ASA could dynamically add new ones that would cover the missing functionality.

The hierarchical architecture needed for ASA is based on the common features of almost all relevant research: a HRL system consisting of two layers[1], where the top layer learns the core task and the bottom layer employs a fixed number of pre-trained policies. These loose constraints enable the usage in various algorithms. In terms of MDP complexity, ASA is aimed to work in both discrete and continuous state spaces, as well as continuous action spaces in the lower level (the action space of a high-level agent is inherently discrete). The knowledge of the model, needed for UMDP (as in [11, 16, 23]), or other specificities are not presumed by ASA either. We thoroughly focus on working with sparse-reward environments which, though being much harder to solve, offer a greater research potential.

One of our core goals was to develop ASA as an independent pluggable component that can be adapted into any existing HRL model to enrich its capabilities without extensive re-work. The Adaptive Skill Acquisition approach consists of

[1] ASA can be deployed on multiple levels of a multi-level hierarchy.

three key steps: identification, training, and integration; as presented in following sections. The identification and creation of a new skill can be adapted with virtually zero implementation changes. For the skill's integration into HRL hierarchy, we implemented several options that can be directly used or customized, and users can also easily create new ones that fit their specific architecture.

### 4.1 Identification of a missing skill

During learning the core task, the agent must be able to recognize the need for another skill. By means of self-observation, the agent will try to identify potentially sub-optimal sequences of high-level actions (skill invocations) that tend to occur significantly more often. Such sequences hint at a regularity in the core-MDP that was not discovered by the original skill building process, and is only modelled using the reoccurring sequence of pre-defined skills. This sequence will serve as a candidate for training a new skill, capable of solving the subtask in a more efficient way.

We denote $\delta = [a_t^H, \ldots, a_{t+k}^H]$ a sequence of high-level actions taken between timesteps $t$ and $t+k$. Throughout the learning process, we collect the counts $C(\delta)$ for observed sequences, limiting the sequence length to a reasonable threshold. As mentioned, most frequent sequences are suitable candidates for detected inefficiencies. However, short sequences naturally have a tendency to appear more frequently in the data, hence we cannot simply take the sequence $\delta$ with greatest $C(\delta)$. Instead, we analytically compute a null-hypothesis count:

$$C_H(\delta) = (T - |\delta| + 1) \prod_{i=1}^{|\delta|} p(\delta_i)$$

where $T$ denotes the length of the whole roll-out. The quantity $C_H(\delta)$ thus describes how many occurrences of the sequence $\delta$ are expected under a random 'null' policy. We use the empirical probability distribution of invoked skills for $p(\delta_i)$. The overall score (frequency) of the sequence is then computed as a ratio of two counts: $f(\delta) = C(\delta)/C_H(\delta)$. Every sequence with $f(\delta) > 1$ occurs more often than it would do under the null policy. Hence, the higher $f$-score the sequence gains, the more likely it is to execute the repeated non-optimal steps. By ranking the sequences using $f$-score, we can identify possible candidates for a new skill.

We implemented a customized data structure based on lexicographic trees to count the sequences in time-efficient way, and to retrieve our desired statistics. This way, the computational increase to the overall learning process was shrunk to a marginal level.

After having identified the sequence as a candidate for a new skill, we need to define the MDP problem that it represents. As the RL process is guided solely by the reward signal, we especially need to formulate a reward signal that will lead it. We adapted the approach by [9] of having the starting and target regions for each skill. As the sequences are being populated for the computation of $f(\delta)$, we also record the starting and ending states of each sequence. Thus, during training

of the new skill based on sequence $\delta$, the agent is spawned in one of $\delta$'s starting states, and is rewarded only upon reaching the corresponding target region. We intentionally train a skill using such sparse-reward environment, as we want to avoid any engineered bias caused by more complicated reward shaping.

### 4.2 Training of a new skill

Next we can initiate the training process of a new skill. Using standard RL methods, we need to train a controller to solve MDP $\langle S, A, P, p'_0, R', \gamma \rangle$, where $S$, $A$, $P$, and $\gamma$ are given by the environment, while $p'_0$ and $R'$ were constructed during the skill identification. After complete MDP specification, the problem simplifies into a standard task of reinforcement learning. We employed TRPO [20] or Natural Policy Gradient [8] algorithm for training the new low-level agent.

### 4.3 Integration of a new skill

After the new skill is ready, we can integrate it into an existing HRL architecture. This is the only step that is inherently approach-specific, i.e. it might need to be adjusted when ASA will be used in different architectures. Nevertheless, we focused on the most common implementation and created several options that can be directly employed.

All relevant recent approaches operate on a policy-optimization scheme stemming from actor–critic architecture, where both the action-selecting *actor* and the state-evaluating *critic* are implemented as neural networks. The architecture of an actor can vary from a simple multi-layer perceptron to convolutional neural network or recurrent models such as LSTM. However, its output always directly represents the action to be executed, either as a real-valued vector of actions (in continuous-action environments), or one-hot-encoded index of an action (i.e. classification in discrete-action environments).

The high-level actor in HRL architectures chooses from a discrete set of $n$ actions (skills), forming a neural classifier. Adding an $(n+1)$-th skill to the architecture essentially means extending the output vector of the actor network by one extra unit. The new output unit, of course, needs to have some initialised weights (and a bias, if used).

We tested several initialisation schemes to prepare weights for the new output unit, both uninformed (Table 1) and informed (Table 2):
**Uninformed schemes** construct the new weight vector in a random manner, not utilising any additional knowledge collected during training.
**Informed schemes** utilize the collected data in order to bootstrap the training of the agent, while keeping the computational complexity minimal. During the initialization we considered the existing weights of the output layer $[w_1, \ldots, w_n]$ for current $n$ skills, the sequence of skill invocations $\delta = [a^H_t, \ldots, a^H_{t+k}]$, and the states $s_t$ that occurred at the beginning of $\delta$ in different roll-outs. In the four informed schemes, the new weights vector is computed as a linear combination of the old weights: $w_{n+1} = \sum_{i=1}^{n} c_i w_i$. The coefficients of this combination are normalized to the unit sum: $\sum_{i=1}^{n} c_i = 1$.

Table 1: Uninformed initialization schemes tested in ASA model.

| # | Description |
|---|---|
| 1 | **Random initialisation:** As a baseline, we used randomized initialisation of both weights and bias, using the original weight initializator of the network. Mean and variation were adjusted to match the mean and variation of weights for previous $n$ skills. |
| 2 | **Random with bias boost:** Initialize weights randomly, again adjusting the mean and variation. However, bias was made significantly greater in order to intentionally increase activation of $(n+1)$-th unit. This should encourage natural exploration of new skill by the high-level RL agent. |

Table 2: Informed initialization schemes tested in ASA model.

| # | Description | Coefficients |
|---|---|---|
| 3 | **Initial states' skills:** Combine the old weights w.r.t. probability of choosing the old skill in $s_t$. | $c_i = P(a_t^H = \pi_i^L \mid s_t)$ |
| 4 | **$\delta$'s skills:** Combine the old weights w.r.t. frequency of using the old skill in $\delta$. | $c_i \propto \sum_l^k [a_{t+l}^H = \pi_i^L]$ where $a_{t+l}^H \in \delta$ |
| 5 | **Smoothed $\delta$'s skills:** New weights are computed as an exponentially smoothed average of skills used in $\delta$ (the first skill of $\delta$ has a greatest weight on average, the last one has the lowest). | $c_i \propto \sum_l^k \gamma^l [a_{t+l}^H = \pi^{L_i}]$ where $a_{t+l}^H \in \delta$ |
| 6 | **$\delta$'s first skill:** Initialize weights by copying weights of skill that was used as first step of $\delta$. Small noise is added to prevent the two skills from being chosen identically. | no coefficients, $w_{n+1} = w_{a_t^H} + \epsilon$ |

All these schemes (except for random initialisation) were designed with a goal to increase the probability of choosing the newly trained skill in situations it has been trained for. This subsequently enhances the exploration of new skill, helping the top-level agent to adopt it quickly.

## 5 Experiments

We designed three experiments (sec. 5.3) to answer the following questions:

- How much does adding a new skill help in training an overall task?
- How well does ASA identify and formulate a new skill?
- How do different skill integration schemes affect efficiency in an overall task?

### 5.1 Environment and task specification

We evaluated our approach on a continuous task of navigating through various mazes. We constructed a set of six different mazes, altering both the structure

and difficulty of the maps, as shown in Fig. 1. In each episode, a map is chosen from this set randomly, and the agent is placed at its beginning. An episode ends upon successful reaching of the goal point (depicted by a green sphere), or after 100 high-level steps, yielding an unsuccessful run.

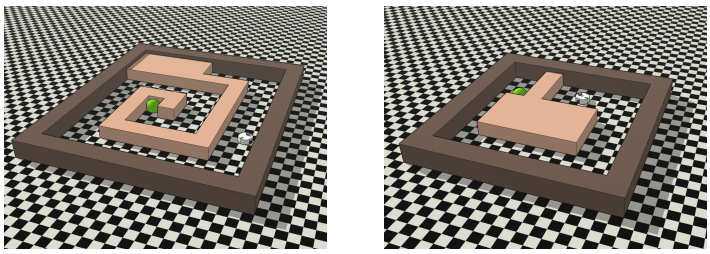

Fig. 1: Examples of mazes the agent had to navigate through. The green sphere designates the target position.

The agent is a simulated vacuum-cleaner-like robot, with continuous activation on both wheels. Robot's only observation is a LIDAR-like sensor, that shows a presence of objects and their distance in all directions, with a limited range radius (three times the robot's size). The agent does *not* have any information about its orientation (compass), or which maze it has been placed into.

As our primary goal is to focus on sparse-reward environments, the agent is given the reward of $+1$ only upon reaching the goal point. All other actions are uniformly penalised with $-0.05$ reward per step, giving the agent no information about its progress. Thus we operate in a randomised, continuous state- and action-space environment with sparse rewards, yielding a reasonably challenging task to face.

### 5.2 Implementation and training description

The HRL agent is constructed of a two-level hierarchy of trained agents. Initially it is given an imperfect set of locomotion skills, including ones to efficiently move a larger distance forward/backward, or turn left, but not a skill to turn right. Both the low-level and high-level controllers are trained using either Natural Policy Gradient [8] or TRPO [20], depending on experiment setup. The high-level policy is updated each *iteration* – a batch of 5000 high-level steps ($\approx$ 80 episodes, on average). We use multi-layer perceptrons for both actor and critic networks, as well as trained policy-baseline function. The ASA project was implemented with a great help of *RL-Lab* [5] and *Garage* [3] frameworks. It is organised as modular and pluggable component that can be deployed into further works.

### 5.3 Results

**Overall performance:** In experiment 1 we compared the performance of agents with engaged ASA and those without it, to appreciate the gain in performance

caused by our approach. The evaluation metric is Average discounted reward, i.e. $G(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)]$. As the agent is penalised for each step it takes, greater reward essentially indicates faster routes to the goal point. The results from each experiment were averaged over 8 trials with different random seeds.

Our approach significantly outperformed the baseline agent in various conditions. Fig. 2a depicts the usage of full-stack approach, i.e. all three key steps (decision, training, integration) were performed by ASA. We trained the model either using TRPO or an older NPG algorithm, to demonstrate its usability with different methods. In Fig. 2b we forced the initialisation of a new skill in different iterations, leaving the skill formulation, training, and integration to ASA. As can be seen, regardless of the initiation point, we were able to identify the missing skill that helps to increase the efficiency.

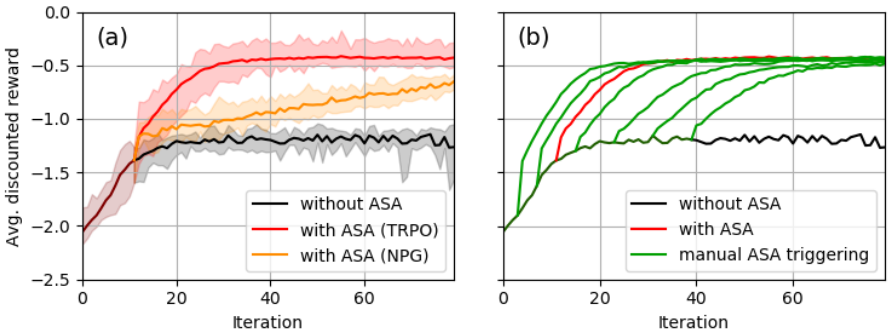


Fig. 2: Results after using ASA to train and integrate a new skill. New skill was added (a) according to ASA computation, or (b) at other times overriding ASA's decision.

**Quality of skill identification:** In experiment 2, we wanted to falsify the hypothesis that *any* added skill may cause an improvement. We manually designed an unprofitable skill policy, and let ASA integrate it into agent's hierarchy. As shown in Fig. 3, the performance with a bad skill initially dropped as the high-level agent tried to explore it, and eventually leveled out with the original performance when the agent learned to ignore it. This *lower bound* on a skill quality shows that ASA's identification of the missing skill is useful.

On the other side of the spectrum, we also wanted to state an *upper bound* on a skill quality to see how much room there is for ASA improvement. For this purpose we created an ideal skill that was missing in the hierarchy, and let ASA integrate it. As we saw in fig. 2, the skill produced by ASA increased the average reward from $-1.24$ to $-0.43$. The ideal skill, however, could only raise it by a small amount to $-0.32$, as seen in Fig. 4. Such a result implies that the identification of a skill performed by ASA is sufficiently effective.

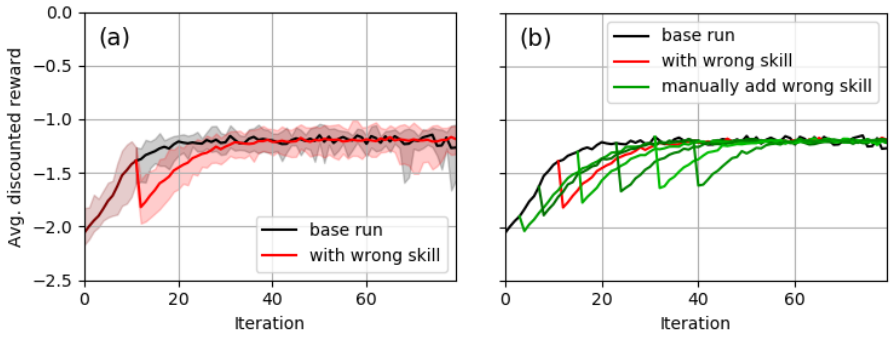


Fig. 3: Integration of a new skill that is intentionally badly manually designed, (a) at time of ASA decision, or (b) at other times.

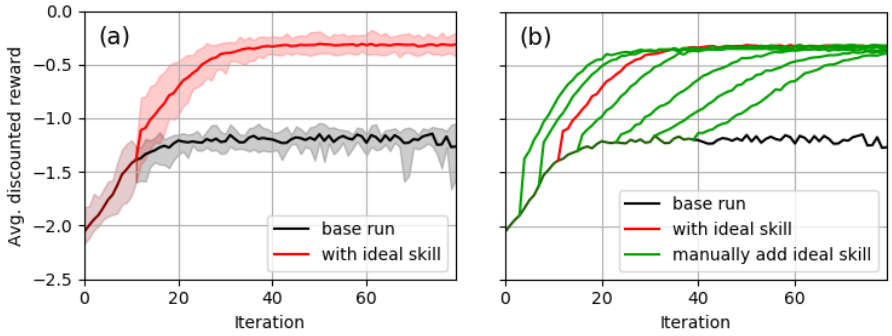


Fig. 4: Integration of an ideal new skill that was designed manually to optimally enrich the skill set, (a) at time of ASA decision, or (b) at other times.

**Integration schemes:** Next, in experiment 3 we aimed to compare the efficiency of skill integration schemes 1 to 6 described in sec. 4.3. With the exception of random initialisation (scheme #1), all of these schemes were designed to boost the exploration of the new skill, and thus also the overall performance with an added skill. As shown in Fig. 5, usage of different integration schemes before (a) or at time when ASA decided to add new skill (b) had negligible effect on the reward gain or learning speed.

The only, yet still not very significant difference in learning speed can be observed between uninformed (#1,2) and informed (#4–6) schemes if we add a new skill much later, i.e. after the training of the imperfect hierarchy converged to a stable solution (Fig. 5c). By observing this phenomenon we found that the later we add a new skill, the more notable this difference is. We attribute this effect to the naturally decreased exploration in later stages of training, which is not supported by uninformed schemes in contrast to the informed ones.

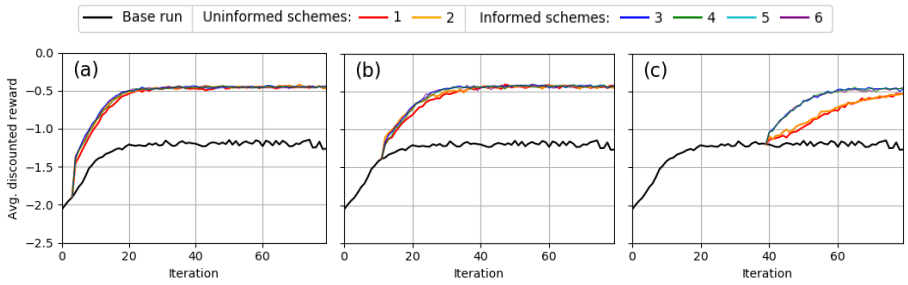


Fig. 5: Effect of using different integration schemes during the skill integration (a) in $4^{\text{th}}$ iteration; (b) in $12^{\text{th}}$ iteration (ASA decision); (c) in $40^{\text{th}}$ iteration.

## 6 Conclusion and future work

In this paper, we examined the option of adding new skills to (possibly suboptimal) HRL hierarchies. We proposed a framework of *Adaptive Skill Acquisition* that is able to dynamically detect that a skill is missing, formulate the task, train, and integrate new skill into any existing hierarchical architecture. Our results in continuous sparse-reward environments confirm that ASA can successfully recognize what skill is missing and that its integration can significantly improve the overall performance. However, we found that none of our proposed skill integration schemes outperformed the random one, even though all of them worked reasonably well. In future work, we plan to extend our pilot results by employing multiple RL tasks and more hierarchical architectures, confirm the intended reusability of our approach. We would also like to investigate whether employing pseudo-rehearsal [18] into integration scheme may increase the adaptation of a new skill.

## References

1. Bacon, P.L., Harb, J., Precup, D.: The option-critic architecture. In: AAAI Conference on Artificial Intelligence (2017)
2. Bakker, B., Schmidhuber, J.: Hierarchical reinforcement learning with subpolicies specializing for learned subgoals. In: International Conference on Neural Networks and Computational Intelligence. pp. 125–130 (2004)
3. Garage contributors: Garage: A toolkit for reproducible reinforcement learning research. https://github.com/rlworkgroup/garage (2019)
4. Dietterich, T.G.: Hierarchical reinforcement learning with the maxq value function decomposition. Journal of Artificial Intelligence Research **13**(1), 227–303 (2000)
5. Duan, Y., Chen, X., Houthooft, R., Schulman, J., Abbeel, P.: Benchmarking deep reinforcement learning for continuous control. In: International Conference on Machine Learning. pp. 1329–1338 (2016)
6. Florensa, C., Duan, Y., Abbeel, P.: Stochastic neural networks for hierarchical reinforcement learning. In: International Conference on Learning Representations (2017)

7. Goel, S., Huber, M.: Subgoal discovery for hierarchical reinforcement learning using learned policies. In: Florida AI Research Society Conference. pp. 346–350 (2003)
8. Kakade, S.M.: A natural policy gradient. In: Advances in Neural Information Processing Systems. pp. 1531–1538 (2002)
9. Konidaris, G., Barto, A.G.: Skill discovery in continuous reinforcement learning domains using skill chaining. In: Advances in Neural Information Processing Systems. pp. 1015–1023 (2009)
10. Kulkarni, T.D., Narasimhan, K., Saeedi, A., Tenenbaum, J.: Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In: Advances in Neural Information Processing Systems. pp. 3675–3683 (2016)
11. Levy, A., Konidaris, G., Platt, R., Saenko, K.: Learning multi-level hierarchies with hindsight. In: International Conference on Learning Representations (2019)
12. Li, A.C., Florensa, C., Clavera, I., Abbeel, P.: Sub-policy adaptation for hierarchical reinforcement learning. In: International Conference on Learning Representations (2020)
13. McGovern, A., Barto, A.G.: Automatic discovery of subgoals in reinforcement learning using diverse density. In: International Conference on Machine Learning. vol. 1, pp. 361–368 (2001)
14. McGovern, E.A., Barto, A.G.: Autonomous discovery of temporal abstractions from interaction with an environment. Ph.D. thesis, University of Massachusetts at Amherst (2002)
15. Menache, I., Mannor, S., Shimkin, N.: Q-cut—dynamic discovery of sub-goals in reinforcement learning. In: European Conference on Machine Learning. pp. 295–306 (2002)
16. Nachum, O., Gu, S.S., Lee, H., Levine, S.: Data-efficient hierarchical reinforcement learning. In: Advances in Neural Information Processing Systems. pp. 3303–3313 (2018)
17. Parr, R., Russell, S.J.: Reinforcement learning with hierarchies of machines. In: Advances in Neural Information Processing Systems. pp. 1043–1049 (1998)
18. Robins, A.: Catastrophic forgetting, rehearsal and pseudorehearsal. Connection Science **7**(2), 123–146 (1995)
19. Schmidhuber, J.: Learning to generate sub-goals for action sequences. In: Artificial neural networks. pp. 967–972 (1991)
20. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning. pp. 1889–1897 (2015)
21. Shu, T., Xiong, C., Socher, R.: Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. In: International Conference on Learning Representations (2018)
22. Sutton, R.S., Precup, D., Singh, S.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence **112**, 181–211 (1999)
23. Vezhnevets, A.S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., Kavukcuoglu, K.: Feudal networks for hierarchical reinforcement learning. In: International Conference on Machine Learning. pp. 3540–3549 (2017)