Using a Learned Policy Basis to Optimally Solve Reward Machines

Guillermo Infante*, **Anders Jonsson, Vicenç, Gómez**
AI&ML group, Universitat Pompeu Fabra
Barcelona, Spain
{guillermo.infante,vicen.gomez,anders.jonsson}@upf.edu,

David Kuric*, **Herke van Hoof**
AMLab, University of Amsterdam
Amsterdam, Netherlands
{d.kuric,h.c.vanhoof}@uva.nl

Abstract

Conventional reinforcement learning methods can successfully solve a wide range of sequential decision problems. However, learning policies that can generalize predictably across multiple tasks in a setting with non-Markovian reward specifications is a challenging problem. We propose to use successor features to learn a policy basis so that each subpolicy in it solves a well-defined subproblem. In a task described by a reward machine that involves the same set of subproblems, the combination of these subpolicies can then be used to generate an optimal solution without additional learning. In contrast to other methods that combine subpolicies via planning, our method asymptotically attains global optimality, even in stochastic environments.[1]

Introduction

Autonomous agents that interact with an environment usually face tasks that comprise complex, entangled behaviors over long horizons. Conventional reinforcement learning (RL) methods have successfully addressed this. However, in cases when the agent is meant to perform several tasks across similar environments, training a policy for every task separately can be time-consuming and requires a lot of data. In such cases, the agent can utilize a method that has built-in generalization capabilities. One such method relies on the assumption that reward functions of these tasks can be decomposed into a linear combination of successor features (SF) [Barreto et al., 2017]. When a new task is presented, it is possible to combine previously learned policies and their successor features to solve a new task. While combining such policies is guaranteed to be an improvement over any previously learned policy, it may not necessarily be optimal. However, as shown by Alegre et al. [2022], one can leverage recent advancements in multi-objective RL to learn a set of policies that constitutes a policy basis to retrieve an optimal policy for any linear combination of successor features.

While traditional RL methods rely on Markovian reward functions, defining a task using such a function can be challenging and sometimes impossible [Whitehead and Lin, 1995]. In scenarios where expressing the reward function in Markovian terms is not feasible, there has been a growing interest in alternative methods for task specification in recent years [Toro Icarte et al., 2018a, Camacho et al.,

*Authors contributed equally.

[1]A previous version of the paper was accepted at *ICAPS 2024*. URL: https://arxiv.org/pdf/2403.15301.

17th European Workshop on Reinforcement Learning (ewrl 2024).

2019]. In our work we focus on developing a method that utilizes the generalization capabilities of SF in a setting with non-Markovian reward functions.

Prior techniques for such settings have been proposed in contexts where a set of propositional symbols $\mathcal{P}$ enables the definition of high-level tasks using logic [Vaezipoor et al., 2021, Toro Icarte et al., 2019] or reward machines (RM) [Toro Icarte et al., 2018a]. Like in hierarchical RL, they are often based on decomposing tasks into subtasks and solving each subtask independently [Dietterich, 2000, Sutton et al., 1999]. However, combining optimal solutions for subtasks may potentially result in a suboptimal overall policy. This is referred to as recursive optimality [Dietterich, 2000] or myopic policy [Vaezipoor et al., 2021].

To alleviate this issue, one can consider methods that condition the policy or the value function on the specification of the whole task [Schaul et al., 2015] and such approaches were recently also proposed for tasks with non-Markovian reward functions [Vaezipoor et al., 2021]. However, the methods that specify the whole task usually rely on a blackbox neural network for planning when determining which subgoal to reach next. This makes it hard to interpret the plan to solve the task and although they show promising results in practice, it is unclear whether and when these approaches will generalize to a new task.

Instead, our work aims to use task decomposition without sacrificing global optimality to achieve predictable generalization. The method we propose learns a set of *local* policies in subtasks such that their combination forms a *globally optimal* policy for a large collection of problems described with RMs. A new policy that solves any new task can then be created, without additional learning, by planning on a given RM instance. Our contributions are:

- We propose to use successor features to learn a policy basis that is suitable for planning in stochastic domains.
- We develop a planning framework that uses such policy bases for zero-shot generalization to complex temporal tasks described by an arbitrary reward machine.
- We prove that if the policies in this basis are optimal, our framework produces a globally optimal solution even in stochastic domains.

Background and Notation

Given a finite set $\mathcal{X}$, let $\Delta(\mathcal{X}) = \{p \in \mathbb{R}^{\mathcal{X}} : \sum_x p(x) = 1, p(x) \geq 0 \ (\forall x)\}$ denote the probability simplex on $\mathcal{X}$. Given a probability distribution $q \in \Delta(\mathcal{X})$, let $\text{supp}(q) = \{x \in \mathcal{X} : q(x) > 0\} \subseteq \mathcal{X}$ denote the support of q. We abuse notation and let $\Delta(d)$ represent the (standard) simplex in $\mathbb{R}^d$.

Reinforcement Learning RL problems commonly assume an underlying Markov Decision Process (MDP). We define an MDP as the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{E}, \mathcal{A}, \mathcal{R}, \mathbb{P}_0, \mathbb{P}, \gamma \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{E}$ is the set of exit states (or terminal states), $\mathcal{A}$ is the action space, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, $\mathbb{P}_0 \in \Delta(\mathcal{S})$ is the probability distribution of initial states, $\mathbb{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition probability function and $0 \leq \gamma < 1$ is the discount factor. The set of exit states $\mathcal{E}$ induces a set of terminal transitions $T = (\mathcal{S} \setminus \mathcal{E}) \times \mathcal{A} \times \mathcal{E}$.

The learning agent interacts in an episodic manner with the environment following a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. At each timestep, the agent observes a state s_t, chooses the action $a_t \sim \pi(s_t)$, transitions to a new state $s_{t+1} \sim \mathbb{P}(\cdot|s_t, a_t)$ and receives a reward $\mathcal{R}(s_t, a_t, s_{t+1})$. The episode ends when the agent observes a terminal transition $(s_t, a_t, s_{t+1}) \in T$ and a new episode starts with initial state $s_0 \sim \mathbb{P}_0(\cdot)$.

The goal of the agent is to find an optimal policy π^* that maximizes the expected discounted return, for any state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$,

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} \mathcal{R}(S_i, A_i, S_{i+1}) \,\middle|\, S_t = s, A_t = a \right]. \tag{1}$$

Hence, an optimal policy is $\pi^*(s) \in \arg\max_a \max_\pi Q^\pi(s, a)$ for all states $s \in \mathcal{S}$, with ties broken arbitrarily. The action value function defined in Equation (1) satisfies the recursive Bellman equation

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim \mathbb{P}(\cdot|s,a)} \Big[\mathcal{R}(s, a, s') + \gamma V^\pi(s') \Big], \tag{2}$$

for any $(s, a) \in \mathcal{S} \times \mathcal{A}$. The state value function is obtained by averaging the action value function over the actions, $V^{\pi}(s) = \mathbb{E}_{a \sim \pi(s)}\left[Q^{\pi}(s, a)\right] \ \forall s \in \mathcal{S}$. Throughout the paper we use Q^{*} and V^{*} to refer to the optimal, respectively, action and state value functions.

Successor Features [Dayan, 1993, Barreto et al., 2017] introduce a widely used RL representation framework that assumes the reward function is linearly expressible with respect to a feature vector,

$$\mathcal{R}^{\mathbf{w}}(s, a, s') = \mathbf{w}^{\intercal} \boldsymbol{\phi}(s, a, s'). \tag{3}$$

Here, $\boldsymbol{\phi} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}^{d}$ maps transitions to feature vectors and $\mathbf{w} \in \mathbb{R}^{d}$ is a weight vector. Every weight vector $\mathbf{w}$ induces a different reward function and, thus, a task. The SF vector of a state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ under a policy π is the expected discounted sum of future feature vectors:

$$\boldsymbol{\psi}^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\phi}(S_i, A_i, S_{i+1}) \,\middle|\, S_t = s, A_t = a\right]. \tag{4}$$

The action value function for a state-action pair (s, a) under policy π can be efficiently represented using the SF vector. Due to the linearity of the reward function, the weight vector can be decoupled from the Bellman recursion. Following the definition of Equations (1) and (3), the action value function in the SF framework can be rewritten as

$$\begin{aligned} Q^{\pi}_{\mathbf{w}}(s, a) &= \mathbb{E}_{\pi}\left[\sum_{i=t}^{\infty} \gamma^{i-t} \mathbf{w}^{\intercal} \boldsymbol{\phi}(S_i, A_i, S_{i+1}) \,\middle|\, S_t = s, A_t = a\right] \\ &= \mathbf{w}^{\intercal} \mathbb{E}_{\pi}\left[\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\phi}(S_i, A_i, S_{i+1}) \,\middle|\, S_t = s, A_t = a\right] \\ &= \mathbf{w}^{\intercal} \boldsymbol{\psi}^{\pi}(s, a). \end{aligned} \tag{5}$$

The SF representation leads to *generalized policy evaluation* (GPE) over multiple tasks [Barreto et al., 2020], and similarly, to *generalized policy improvement* (GPI) to obtain new better policies [Barreto et al., 2017].

A family of MDPs is defined as the set of MDPs that share all the components, except the reward function. This set is formally defined as

$$\mathcal{M}^{\boldsymbol{\phi}} \equiv \{\langle \mathcal{S}, \mathcal{E}, \mathcal{A}, \mathcal{R}_{\mathbf{w}}, \mathbb{P}_0, \mathbb{P}, \gamma \rangle \mid \mathcal{R}_{\mathbf{w}} = \mathbf{w}^{\intercal} \boldsymbol{\phi}, \forall \mathbf{w} \in \mathbb{R}^{d}\}.$$

Transfer learning on families of MDPs is possible thanks to GPI. Given a set of policies Π, learned on the same family $\mathcal{M}^{\boldsymbol{\phi}}$, for which their respective SF representations have been computed, and a new task $\mathbf{w}' \in \mathbb{R}^{d}$, a GPI policy π_{GPI} for any $s \in \mathcal{S}$ is derived as

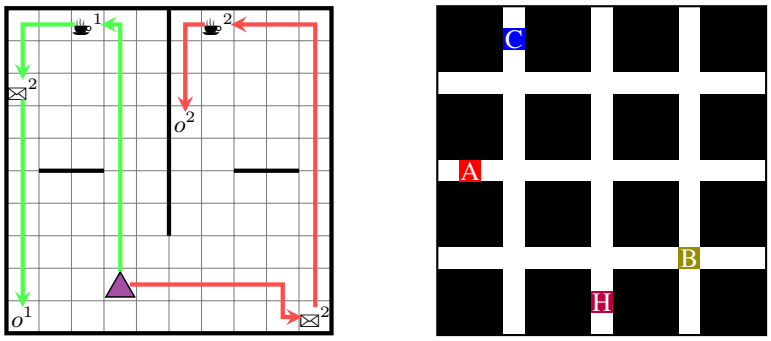
$$\pi_{\text{GPI}}(s) \in \arg\max_{a \in \mathcal{A}} \max_{\pi \in \Pi} Q^{\pi}_{\mathbf{w}'}(s, a). \tag{6}$$

However, there is no guarantee of optimality for $\mathbf{w}'$. A fundamental question to solve the so-called *optimal policy transfer learning problem* is which policies should be included in the set of policies Π so an optimal policy for any weight vector $\mathbf{w} \in \mathbb{R}^{d}$ can be obtained with GPI.

Convex Coverage Set of Policies The recent work of Alegre et al. [2022] solves the optimal policy transfer learning problem. They draw the connection between the SF transfer learning problem and multi-objective RL. The pivotal fact is that the SF representation in Equation (4) can be interpreted as a multidimensional value function and the construction of the aforementioned set of policies Π can be cast as a multi-objective optimization problem.

Consequently, the optimistic linear support (OLS) algorithm is extended with successor features in order to learn a set of policies that constitutes a *convex coverage set* (CCS) [Roijers et al., 2015]. Their main result is the SFOLS algorithm (see Appendix B for a full, technical description) in which a set Π_{CCS} is built incrementally by adding (new) policies to such a set, until convergence. The set Π_{CCS} contains all non-dominated policies in terms of their multi-objective value functions, where the dominance relation is defined over scalarized values $V^{\pi}_{\mathbf{w}} = \mathbb{E}_{S_0 \sim \mathbb{P}_0}\left[V^{\pi}_{\mathbf{w}}(S_0)\right]$, and is characterized as

$$\begin{aligned} \Pi_{\text{CCS}} &= \{\pi \mid \exists \mathbf{w} \text{ s.t. } \forall \boldsymbol{\psi}^{\pi'}, \ \mathbf{w}^{\intercal} \boldsymbol{\psi}^{\pi} \geq \mathbf{w}^{\intercal} \boldsymbol{\psi}^{\pi'}\} \\ &= \{\pi \mid \exists \mathbf{w} \text{ s.t. } \forall \pi', \ V^{\pi}_{\mathbf{w}} \geq V^{\pi'}_{\mathbf{w}}\}. \end{aligned} \tag{7}$$



(a) (b)

Figure 1: Depiction of the Office (a) and Delivery (b) environments. In (a) $\mathcal{P} = \{☕, ✉, o\}$ and $\mathcal{E} = \{☕^1, ☕^2, ✉^1, ✉^2, o^1, o^2\}$. In (b), $\mathcal{E} = \{A, B, C, H\}$ and $\mathcal{P} = \{A, B, C, H, ■\}$.

In every iteration k, SFOLS proposes a new weight vector $\mathbf{w}^k \in \Delta(d)$ for which an optimal policy (and its corresponding SF representation) is learned and added to Π_{CCS}. It is sufficient to consider weights in $\Delta(d)$ to learn a complete Π_{CCS}. The output of SFOLS is both Π_{CCS} and the SF representation $\boldsymbol{\psi}^\pi$ for every $\pi \in \Pi_{\text{CCS}}$.

Intuitively, all policies in Π_{CCS} are optimal in at least one task $\mathbf{w} \in \Delta(d)$. The set Π_{CCS} is combined with GPI, see Equation (6), and upon convergence, for any (new) given task $\mathbf{w}' \in \mathbb{R}^d$ an optimal policy can be identified [Alegre et al., 2022, cf. Theorem 2].

Propositional Logic We assume that environments are endowed with a set of high-level, boolean-valued propositional symbols $\mathcal{P}$. Without loss of generality, we assume for now that these symbols are observed when the agent transitions into some exit state $\varepsilon \in \mathcal{E}$ of the low-level MDP $\mathcal{M}^\phi$. This may not be necessarily the case, as the agent could observe further symbols in non-exit states (e.g. a symbol representing an obstacle). Every transition $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ induces some propositional valuation (assignment of truth values) $2^\mathcal{P}$. Such a valuation depends solely on the new state s' and occurs under a mapping $\mathcal{O} : \mathcal{S} \to 2^\mathcal{P}$ that is known to the agent. This implies that the agent can associate valuations $2^\mathcal{P}$ to transitions any time. Propositional symbols are assumed to be mutually exclusive which means that the agent cannot observe two symbols evaluated true in the same transition. A valuation Γ is said to satisfy a propositional symbol p, formally $\Gamma \vDash p$, if p is true in Γ.

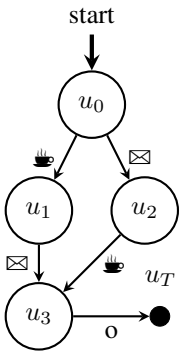


Figure 2

Reward Machines Task instructions can be specified via a reward machine. These are tuples $\mathcal{F} = \langle \mathcal{U}, u_0, \mathcal{T}, L, \delta \rangle$ where $\mathcal{U}$ is the finite set of states, $u_0 \in \mathcal{U}$ is the initial state, $\mathcal{T}$ is the set of terminal states with $\mathcal{U} \cap \mathcal{T} = \emptyset$, $L : \mathcal{U} \times (\mathcal{U} \cup \mathcal{T}) \to 2^\mathcal{P}$ is a labeling function that maps RM states transitions to truth values for the propositions and $\delta : \mathcal{U} \cup \mathcal{T} \to \mathbb{R}$ is a high-level reward function. Each transition among RM states (u, u') defines a subgoal. The agent has to observe some propositional valuation $L(u, u')$ in order to achieve it and RM states can only be connected by a subgoal. E.g. in Figure 2, the RM state u_0 has two outgoing subgoals: getting mail (labeled as ✉) and getting coffee (labeled as ☕). Non-existing transitions (u, u') get mapped to $L(u, u') = \perp$. The reward function δ gives a reward larger than 0 only to terminal states. In other words, such a reward function is $\delta(u) = 0 \; \forall u \in \mathcal{U}$ and $\delta(\mathbf{t}) > 0 \; \forall \mathbf{t} \in \mathcal{T}$.

Using Successor Features to Solve non-Markovian Reward Specifications

We focus on the setting in which a low-level MDP is equipped with a reward structure like in Equation (3). We let the low-level be represented by a family of MDPs $\mathcal{M}^\phi$, where each weight vector $\mathbf{w} \in \mathbb{R}^d$ specifies a low-level task. The agent receives high-level task specifications in the more flexible form of a RM which permits the specification of non-Markovian reward structures. The

Algorithm 1: SF-RM-VI

Input: Low-level MDP $\mathcal{M}^{\phi}$, task specification $\mathcal{F}$

Obtain Π_{CCS} on $\mathcal{M}^{\phi}$.
Initially $\mathbf{w}^0(u) = \mathbf{0} \in \mathbb{R}^{|\mathcal{E}|} \ \ \forall u \in \mathcal{U}$.
while not done **do**
    for $u \in \mathcal{U}$ **do**
        Update each $\mathbf{w}_j^{k+1}(u)$ with Equation (11).
return $\{\mathbf{w}^*(u) \ \forall u \in \mathcal{U}\}$

combination of a low-level family of MDPs and a RM gives rise to a *product MDP* $\mathcal{M}' = \mathcal{F} \times \mathcal{M}^{\phi}$ that satisfies the Markov property, and where the state space is augmented to be $\mathcal{U} \times \mathcal{S}$.

A product MDP $\mathcal{M}'$ is a well-defined MDP. The agent now follows a policy $\mu : \mathcal{U} \times \mathcal{S} \to \Delta(\mathcal{A})$, that depends both on the RM state and the underlying MDP state. $\mathcal{M}'$ can be solved with conventional RL methods by finding an optimal policy μ^* that maximizes

$$Q^{\mu}(u, s, a) = \mathbb{E}_{\mu}\left[\sum_{i=t}^{\infty} \gamma^{i-t} \mathcal{R}(U_i, S_i, A_i, U_{i+1}, S_{i+1}) \,\middle|\, U_t = u, S_t = s, A_t = a\right].$$

This is, however, impractical since policies should be retrained every time a new high-level task is specified. Exploiting the problem structure is essential for tractable learning, where components can be reused for new task specifications. The special reward structure of the low-level MDPs and our particular choice of feature vectors, introduced below, allow us to define an algorithm able to achieve a solution by simply planning in the space of augmented exit states $\mathcal{U} \times \mathcal{E}$. This inherently makes obtaining an optimal policy more efficient than solving the whole product MDP, as we reduce the number of states on which it is necessary to compute the value function.

The agent may have to perform the same subtask at different moments of the plan or in different RM instances. We aim to provide agents with a collection of base behaviors that can be combined to retrieve the optimal behavior for the whole task.

In line with the previous reasoning, we introduce a two-step algorithm in which the agent first learns a Π_{CCS} (a set of policies that constitute a CCS) on a well-specified representation of the environment. Then these subpolicies are used to solve efficiently any RM specification on the propositional symbols of the environment. In what follows, we motivate the design of the feature vectors, explain our high-level dynamic programming algorithm and prove that it achieves the optimal solution.

Feature vectors Our approach works with a very specific set of feature vectors that we proceed to explain. For a family of MDPs $\mathcal{M}^{\phi}$, the feature map is $\boldsymbol{\phi} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^{|\mathcal{E}|}$. Each feature component $\boldsymbol{\phi}_j$ is associated with an exit state $\varepsilon_j \in \mathcal{E} = \{\varepsilon_1, \ldots, \varepsilon_{|\mathcal{E}|}\}$. Such vectors are built as follows. At terminal transitions $(s, a, \varepsilon_i) \in T$, $\boldsymbol{\phi}_j = 1$ when $j = i$ and $\boldsymbol{\phi}_j = 0$ when $j \neq i$. For non-terminal transitions, $\boldsymbol{\phi}(s, a, s') = \mathbf{0}$, implying that the reward is 0 and that the SF representation in Equation (4) of each policy consists of a discounted distribution over the exit states. This indicates how likely it is to reach each exit state following such a policy. We can further extend this representation to consider undesirable symbols (such as obstacles). In this case, it is enough to add one component per extra symbol to the feature vector, which is defined as $\boldsymbol{\phi}_j = -1$ where j is the component associated with such extra symbol and $\boldsymbol{\phi}_i = 0$ when $i \neq j$. Furthermore, we require that $\mathcal{E} \subset \text{supp}(\mathbb{P}_0)$ so the value functions at exit states are well-defined.

Example In the office domain depicted in Figure 1a, the propositional symbols are $\mathcal{P} = \{$☕, ✉, o$\}$ while the exit states $\mathcal{E} = \{$☕^{1}, ☕^{2}, ✉^{1}, ✉^{2}, o^1, $o^2\}$. Consequently, the same propositional symbol is satisfied at different exit locations, this is $\mathcal{O}($☕$^1) \vDash$ ☕ and $\mathcal{O}($☕$^2) \vDash$ ☕. In this case, $\boldsymbol{\phi}(s, a, s') \in \mathbb{R}^6$, is defined as the zero vector in $\mathbb{R}^6$ for every $s' \in \mathcal{S} \setminus \mathcal{E}$ and gets the corresponding vector component equal to 1 when $s' \in \mathcal{E}$. Figure 2 shows the RM specification for the composite task in this domain, note that RMs use symbols in $\mathcal{P}$ to define the subgoals. The interpretation of this RM in natural language is 'get coffee and mail in any order, and then go to an office'.

Algorithm The solution to a RM task specification implies solving a product MDP $\mathcal{M}' = \mathcal{F} \times \mathcal{M}^{\phi}$. Since we have the CCS, the optimal Q-function can be represented by a weight vector $\mathbf{w}^*(u)$ for

each RM state u:

$$Q^*_{\mathbf{w}}(u,s,a) = \max_{\pi \in \Pi_{\text{CCS}}} \mathbf{w}^*(u)^\intercal \boldsymbol{\psi}^\pi(s,a). \tag{8}$$

for all $(u,s,a) \in \mathcal{U} \times \mathcal{S} \times \mathcal{A}$. Here, $\mathbf{w}^*_j(u)$ indicates the optimal value of exit state $\varepsilon_j \in \mathcal{E}$ for RM state u. Then an optimal policy is defined as

$$\mu^*_{\mathbf{w}}(u,s) \in \arg\max_{a \in \mathcal{A}} Q^*_{\mathbf{w}}(u,s,a) \; \forall (s,u) \in \mathcal{U} \times \mathcal{S}. \tag{9}$$

Therefore, we observe that finding the optimal weight vectors $\mathbf{w}^*(u), \forall u \in \mathcal{U}$ is enough for retrieving the optimal action value function of the product MDP $\mathcal{M}'$ and, thus, an optimal policy. Using GPI, we can obtain this vector using a dynamic-programming approach similar to value iteration:

$$\mathbf{w}^{k+1}_j(u) = \max_a Q^*_{\mathbf{w}}\big(\tau(u, \mathcal{O}(\varepsilon_j)), a\big) \tag{10}$$

$$= \max_{a,\pi} \mathbf{w}^k \big(\tau(u, \mathcal{O}(\varepsilon_j))\big)^\intercal \boldsymbol{\psi}^\pi(\varepsilon_j, a), \tag{11}$$

where $\tau(u, \mathcal{O}(\varepsilon)) \in \mathcal{U}$ is the RM state that results from achieving the valuation $\mathcal{O}(\varepsilon)$ in u. We know that $\mathbf{w}^k_j(u) = \delta(\mathbf{t})$ if $\tau(u, \mathcal{O}(j)) = \mathbf{t}$, by definition. As a result, we propose SF-RM-VI (see Algorithm 1) to extract an optimal policy for a product MDP. As $k \to \infty$, SF-RM-VI converges to the optimal set of weight vectors $\{\mathbf{w}^*(u)\}_{u \in \mathcal{U}}$ and, hence, to the optimal value function in Equation (8). We provide a proof of optimality for SF-RM-VI in Appendix A.

Experiments

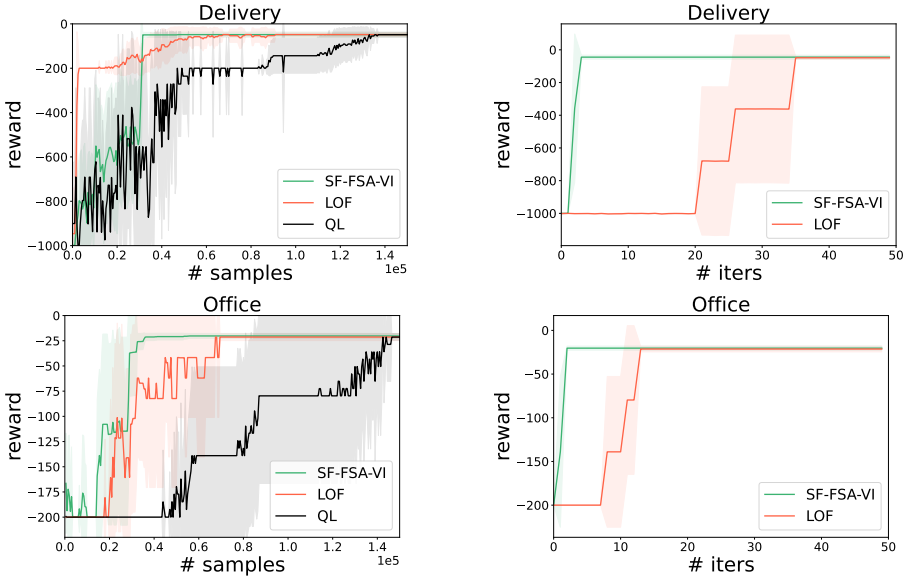


Figure 3: Experimental results for learning (Delivery, top-left and Office, bottom-left) and compositionality (Delivery, top-right and Office, bottom-right). Results show the average performance and standard deviation over the three tasks and 5 seeds per task.

We test SF-RM-VI in two complex discrete environments.[2] At test time, we change the reward to -1 for non-terminal states and use the cumulative reward as the performance metric. We report two types of results. First, we are interested in observing the performance of the derived optimal policy, in Equation (9), during the learning phase. For this, we fully retrain the high-level policy (lines 2-6 in Algorithm 1) every several interactions with the environment as Π_{CCS} is being learned. Second, once the base behaviors are learned (in other words, a complete convex coverage set Π_{CCS} has been computed), we measure how many planning iterations SF-RM-VI needs to converge to an optimal solution for different task specifications. In both cases, we compare against existing baselines.

[2]Code available in the following URL:https://github.com/guillermoim/sf-fsa-vi.

Environments and tasks We use the Delivery domain [Araki et al., 2021] and a modified version of the Office domain [Toro Icarte et al., 2018a] as testbeds for our algorithm. Both environments are depicted in Figure 1 and present a propositional vocabulary that is rich enough to build complex tasks. In the Delivery domain there is a single low-level state associated with each of the propositional symbols, implying that $\mathcal{E} = \{A, B, C, H\}$ and $\mathcal{P} = \{A, B, C, H, \blacksquare\}$. The feature vectors are consistent with our design choice. For terminal transitions, the feature vector is a one-hot encoding of the terminal states. There exist obstacle states (represented by black squares that raise the propositional symbol $\blacksquare$) that share a feature which equals -1 upon entering, and the associated weight is 1000, corresponding to a large negative reward. For regular grid cells (in white) $\boldsymbol{\phi}(s, a, s') = \mathbf{0} \in \mathbb{R}^5$. The Office domain is more complex since there are three propositional symbols $\mathcal{P} = \{☕, ✉, o\}$ which can be satisfied at different locations, namely $\mathcal{E} = \{☕^1, ☕^2, ✉^1, ✉^1, o^1, o^2\}$. Here, there are no obstacle states and $\boldsymbol{\phi}(s, a, s') = \mathbf{0} \in \mathbb{R}^6$ for non-terminal transitions.

For each of the environments we define three different tasks: sequential, disjunction and composite (all described in Appendix C). The sequential task is meant to show how our algorithm can indeed be effectively used to plan over long horizons, when the other two tasks show the ability of our method to optimally compose the base subpolicies in complex settings. In natural language, the tasks in the Delivery domain correspond to: "go to A, then B, then C and finally H" (sequential), "go to A or B, then C and finally H" (disjunction) and "go to A and B in any order, then B, then C and finally H" (composite). The agent has to complete the tasks by avoiding obstacles. The counterpart of these tasks in the Office environment are: "get a coffee, then pick up mail and then go to an office" (sequential), "get coffee or mail, and then go to an office" (disjunction) and "get coffee and mail in any order, and then go to an office" (composite). Our agent never learns how to solve these tasks, but rather learns the set of subpolicies that constitutes the CCS. At test time, we provide the agent with the RM task specification, extract a high-level optimal policy and test its performance on solving the task.

Baselines In the literature, we find the most similar approach to ours is the Logical Options Framework (LOF) [Araki et al., 2021]. We thus use LOF and flat Q-learning [Watkins and Dayan, 1992] on the product MDP as baselines. LOF trains one option per exit state, which are trained simultaneously using intra-option learning, and then uses a high-level value iteration algorithm to train a meta-policy that decides which option to execute in each of the MDP states. On the other hand, Q-learning learns the action value function in the flat product MDP, from which it extracts the policy. Under certain conditions, flat Q-learning converges to the optimal value function but, especially for longer tasks, it may take a large number of samples. Additionally, it is trained for a specific task, so it is not able to generalize to other task specifications. For LOF, we followed the implementation details prescribed by the authors.

Results

Learning Empirical results for learning are shown in Figure 3 (top-left and bottom-left). The plots reflect how the different methods (ours, LOF and flat Q-learning) perform at solving a RM task specification during the learning phase. In the case of SF-RM-VI and LOF, the learning phase corresponds to obtaining the low level subpolicies for Π_{CCS} and the options for LOF. Results are averaged over the three tasks (sequential, disjunction and composite) previously described for each environment. Each data point in the plots represent the cumulative reward obtained by a fully retrained policy with the current status of Π_{CCS} and options. In both environments, SF-RM-VI is the first to reach optimal performance. There exist, however, some differences between LOF and SF-RM-VI. LOF trains all options simultaneously with intra-option learning. This means that every transition (s_t, a_t, s_{t+1}) is used to update all options' value functions and policies. The learning of a Π_{CCS}, on the other hand, is done sequentially. A fixed sample budget per subpolicy is set prior to learning, which can be seen as a hyperparameter. We use a total of $8 \cdot 10^3$ samples per subpolicy in both environments. A experience replay buffer is used to speed up the learning of the policy basis Π_{CCS}. Both options and the SF representation of subpolicies are learned using Q-learning. Due to the incremental nature, at the beginning of the learning process there might not be enough policies in the basis Π_{CCS} to construct a feasible solution. This is clearly observed in the Delivery domain (Figure 3, top left), where at the early stages of the interaction, SF-RM-VI achieves very low cumulative reward due to failing at delivering a solution. It is not until there are enough subpolicies in the basis that Algorithm 1 attains a policy that solves the problem, which eventually converges to an optimal policy. Similarly, LOF converges to an optimal policy albeit takes slightly longer to learn. In the more complex Office environment, results follow the same pattern. However, this environment breaks one of the of LOF

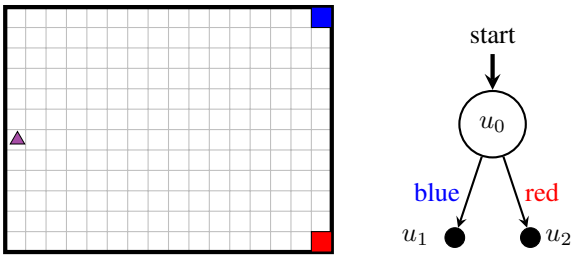


Figure 4: Double Slit environment (left) and RM task specification to reach either goal locations blue or red.

requirements for optimality: to have a single exit state associated with each propositional predicate. In this problem, for each predicate there exist two exit states that can satisfy them. This makes LOF prone to converge to suboptimal solutions while SF-RM-VI attains optimality. This is the case for the composite task, where LOF is short-sighted and returns a longer path (in red, Figure 1a) in contrast to ours that retrieves the optimal solution (in green, Figure 1a). This means that SF-RM-VI is more flexible in the task specification. In this environment, our algorithm also converges faster with a more obvious gap with respect to LOF. In any case, learning subpolicies or options is faster than learning directly on the flat product MDP, as flat Q-learning takes the longest to converge.

Planning Figure 3 top-right and bottom-right show how fast SF-RM-VI and LOF can plan for an optimal solution. Results are again averaged for the three tasks for each environment. Here, a complete policy basis Π_{CCS} has been previously computed, as well as the option's optimal policies. In LOF, the cost of each iteration of value iteration is $|\mathcal{U}| \times |\mathcal{S}| \times |\mathcal{K}|$, where $\mathcal{K}$ is the set of options, while for the Algorithm 1 we propose it is $|\mathcal{U}| \times |\mathcal{E}| \times |\Pi_{\text{CCS}}|$. By definition, the number of options is equivalent to the number of exit states $|\mathcal{K}| = |\mathcal{E}|$, so a single iteration of SF-RM-VI is more efficient than LOF whenever $|\Pi_{\text{CCS}}| \ll |\mathcal{S}|$. In our experiments, the sizes of the CCS are 15 and 12 for the Delivery and Office domains, respectively, while the sizes of the state spaces are of 225 and 121. Therefore, since our algorithm needs fewer, shorter iterations during planning, it outperforms LOF in terms of planning speed in both domains when composing the global solution. This can be observed in the plots for both environments.

Policy basis over options In deterministic environments, it is sufficient to learn the subpolicies associated with the extrema weights (i.e. those subpolicies that reach each of the exit states individually) to find a globally optimal policy via planning. In such cases, it may not be necessary to learn a full CCS. That is why approaches that use the options framework such as LOF traditionally define one option per subgoal. However, there are scenarios in which these approaches will not find an optimal policy. This is the case for most stochastic environments. For example, consider the very simple domain of Double Slit in and the RM task specification in Figure 4. In this environment, there are two exit states $\mathcal{E} = \{\text{blue}, \text{red}\}$. The agent starts in the leftmost column and middle row. At every timestep, the agent chooses an action among $\{\text{UP}, \text{RIGHT}, \text{DOWN}\}$ and is pushed one column to the right in addition to moving in the chosen direction, except in the last column. If the agent chooses RIGHT, it moves an extra column to the right. At every timestep there is a random wind that can blow the agent away up to three positions in the vertical direction. The RM task specification represents a task in which the agent is indifferent between achieving either of the goal states. Since the RIGHT action brings the agent closer to both goals, the optimal behavior in this case is to commit to either goal as late as possible. In this setting, methods that use one policy per subgoal, such as LOF, train two policies to reach both goals. This means that the agent has to commit to one of the goals from the very beginning, which hurts the performance as it has to make up for the consequences of the random noise. On the other hand, the CCS used by SF-RM-VI will contain an additional policy that is indifferent between two goals. This leads to a performance gap as our approach achieves an average accumulated reward of -19.7 ± 3.65 and LOF -22.70 ± 5.72.

Related Work

One of the key distinctions in our research compared to prior studies is the optimality of the final solution. As noted by Dietterich [2000], hierarchical methods usually have the capability to achieve hierarchical, recursive, or global optimality. The challenge that often arises when subtask policies are trained in isolation is that the combination of these locally optimal policies does not lead to a globally optimal policy but a recursively [Dayan and Hinton, 1992] or hierarchically optimal policy [Sutton et al., 1999, Mann et al., 2015, Araki et al., 2021]. To tackle this challenge, our approach relies on acquiring a set of low-level policies for each subtask and employing planning to identify the optimal combination of low-level policies when solving a particular task. By learning the CCS with OLS [Roijers et al., 2014] in combination with high-level planning our approach ensures that globally optimal policy is found. In this regard, the work of Alegre et al. [2022] is of particular interest as it was the first work that used OLS and successor features [Barreto et al., 2017] for optimal policy transfer learning. However, this method has only been applied in a setting with Markovian reward function and has not been used with non-Markovian task specifications or high-level planning.

On the other hand, many recent approaches proposed to use high-level task specifications in the form of LTL [Toro Icarte et al., 2018b, Kuo et al., 2020, Vaezipoor et al., 2021, Jothimurugan et al., 2021], or similar formal language specifications [Toro Icarte et al., 2019, Camacho et al., 2019, Araki et al., 2021, Toro Icarte et al., 2022] to learn policies. However, the majority of the methods in this area are designed for single-task solutions, with only several focusing on acquiring a set of policies that is capable of addressing multiple tasks [Toro Icarte et al., 2018b, León et al., 2020, Kuo et al., 2020, Araki et al., 2021, Vaezipoor et al., 2021]. But, in contrast to our approach, they do not guarantee optimality of the solution.

From these works, our approach is the most similar to the Logical Options Framework [Araki et al., 2021]. The main difference is that LOF trains a single policy for each subgoal, resulting in a set of learned policies that is either smaller than or equal to the set acquired through SF-RM-VI. While employing one policy per subgoal proves sufficient for obtaining a globally optimal policy through planning in deterministic environments [Wen et al., 2020], this may not hold true in stochastic environments, as our experiments demonstrate. In such instances, the policies generated by LOF are hierarchically optimal but fall short of global optimality.

Two notable examples from aforementioned works on multi-task learning with formal language specifications are the works of Toro Icarte et al. [2018b] and Vaezipoor et al. [2021]. The former struggles with generalizing to unseen tasks, because it uses LTL progression to determine which subtasks need to be learned to solve given tasks. The Q-functions that are subsequently learned for each LTL subtask will therefore not be useful for a new task if its subtasks were not part of the training set. Such limitation does not apply to the latter as it instead encodes the remaining LTL task specification using a neural network and conditions the policy on this LTL embedding. While this approach may be more adaptable to tasks with numerous propositions or subgoals, it risks generating suboptimal policies as it relies solely on the neural network to select the next proposition to achieve, without incorporating planning. Additionally, since the planning is implicitly done by the neural network, the policy is less interpretable than when explicit planning is used.

The method we propose can be viewed as a method for composing value functions through successor features, akin to previously proposed approaches for composition of value functions and policies [van Niekerk et al., 2019, Barreto et al., 2019, Nangue Tasse et al., 2020, Infante et al., 2022]. Lastly, since our approach uses the values of exit states for planning it is also related to planning with exit profiles [Wen et al., 2020]. The CCS that we propose to use as a policy basis in our work can be seen as a collection of policies that are optimal for all possible exit profiles.

Discussion and Conclusion

In this work, we address the problem of finding optimal behavior for new non-Markovian goal specifications in known environments. To do so, we introduce a novel approach that uses successor features to learn a policy basis, that can subsequently be used to solve any unseen task specified by a RM with the set of given predicates $\mathcal{P}$ by planning. SF-RM-VI is the first algorithm that can provably generalize to such new task specification without sacrificing optimality in both deterministic and stochastic environments.

The experiments show that SF-RM-VI offers several advantages over previous methods. First, due to the use of SF, it allows for faster composition of the high-level value function since it drastically reduces the number of states to plan on. Secondly, thanks to using a CCS over a set of options, SF-RM-VI achieves optimality even in stochastic environments (as shown in the Double Slit example). Lastly, we do not require that there exists a single exit state per predicate which permits more flexible task specification while at the same time allowing deployment in more complex environments.

A limitation of our approach could be the need to construct a full CCS if one wants to attain global optimality. While the construction of CCS is not time-comsuming for environments with several exit states presented in our work, the computation cost of finding the full CCS could become too large for environments with many exit states. In such case one could instead learn a partial CCS at the cost of a bounded decrease in performance [Alegre et al., 2022] or consider splitting the environment into smaller parts with fewer exit states. While our experiments only considered discrete environments, SF-RM-VI should also be applicable in continuous environments with minor adjustments. These include: using an contiguous set of states instead of a single exit state and using reward shaping to facilitate learning in sparse reward setting.

Acknowledgments and Disclosure of Funding

Anders Jonsson is partially supported by AGAUR SGR and Spanish grant PID2019-108141GB-I00. This publication is part of the action CNS2022-136178 financed by MCIN/AEI/10.13039/501100011033 and by the European Union Next Generation EU/PRTR. David Kuric acknowledges travel support from the ELISE Network funded by European Union's Horizon 2020 research and innovation programme (GA No 951847).

References

Lucas Nunes Alegre, Ana Bazzan, and Bruno C Da Silva. Optimistic linear support and successor features as a basis for optimal policy transfer. In *International Conference on Machine Learning*, pages 394–413. PMLR, 2022.

Brandon Araki, Xiao Li, Kiran Vodrahalli, Jonathan DeCastro, Micah Fry, and Daniela Rus. The logical options framework. In *International Conference on Machine Learning*, pages 307–317. PMLR, 2021.

André Barreto, Will Dabney, Rémi Munos, Jonathan J. Hunt, Tom Schaul, David Silver, and Hado van Hasselt. Successor features for transfer in reinforcement learning. In *Neural Information Processing Systems 30*, pages 4055–4065, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/350db081a661525235354dd3e19b8c05-Abstract.html.

André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Jonathan J. Hunt, Shibl Mourad, David Silver, and Doina Precup. The option keyboard: Combining skills in reinforcement learning. In *Neural Information Processing Systems*, pages 13031–13041, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/251c5ffd6b62cc21c446c963c76cf214-Abstract.html.

André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117 (48):30079–30087, 2020.

Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6065–6073, 7 2019. doi: 10.24963/ijcai.2019/840. URL https://doi.org/10.24963/ijcai.2019/840.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993. doi: 10.1162/neco.1993.5.4.613.

Peter Dayan and Geoffrey E Hinton. Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 5, 1992. URL https://proceedings.neurips.cc/paper/1992/hash/d14220ee66aeec73c49038385428ec4c-Abstract.html.

T. G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, November 2000. ISSN 1076-9757. doi: 10.1613/jair.639. URL https://www.jair.org/index.php/jair/article/view/10266.

Guillermo Infante, Anders Jonsson, and Vicenç Gómez. Globally optimal hierarchical reinforcement learning for linearly-solvable markov decision processes. In *AAAI Conference on Artificial Intelligence*, volume 36, pages 6970–6977, Jun. 2022. doi: 10.1609/aaai.v36i6.20655. URL https://ojs.aaai.org/index.php/AAAI/article/view/20655.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional Reinforcement Learning from Logical Specifications. In *Advances in Neural Information Processing Systems*, volume 34, pages 10026–10039, 2021. URL https://papers.nips.cc/paper_files/paper/2021/hash/531db99cb00833bcd414459069dc7387-Abstract.html.

Yen-Ling Kuo, Boris Katz, and Andrei Barbu. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of ltl formulas. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5604–5610, 2020.

Borja G. León, Murray Shanahan, and Francesco Belardinelli. Systematic generalisation through task temporal logic and deep reinforcement learning. *CoRR*, abs/2006.08767, 2020. URL https://arxiv.org/abs/2006.08767.

Timothy A Mann, Shie Mannor, and Doina Precup. Approximate value iteration with temporally extended actions. *Journal of Artificial Intelligence Research*, 53:375–438, 2015.

Geraud Nangue Tasse, Steven James, and Benjamin Rosman. A Boolean Task Algebra for Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 9497–9507, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6ba3af5d7b2790e73f0de32e5c8c1798-Abstract.html.

Diederik M Roijers, Shimon Whiteson, and Frans A Oliehoek. Linear support for multi-objective coordination graphs. In *International Conference on Autonomous Agents & Multiagent Systems*, volume 2, pages 1297–1304, 2014.

Diederik Marijn Roijers, Shimon Whiteson, and Frans A Oliehoek. Computing convex coverage sets for faster multi-objective coordination. *Journal of Artificial Intelligence Research*, 52:399–443, 2015.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schaul15.html.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, volume 80, pages 2107–2116. PMLR, 10–15 Jul 2018a. URL https://proceedings.mlr.press/v80/icarte18a.html.

Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Teaching multiple tasks to an rl agent using ltl. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 452–461, 2018b.

Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. Learning Reward Machines for Partially Observable Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://papers.nips.cc/paper_files/paper/2019/hash/532435c44bec236b471a47a88d63513d-Abstract.html.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *Journal of Artificial Intelligence Research*, 73:173–208, January 2022. ISSN 1076-9757. doi: 10.1613/jair.1.12440. URL https://www.jair.org/index.php/jair/article/view/12440.

Pashootan Vaezipoor, Andrew C. Li, Rodrigo A. Toro Icarte, and Sheila A. Mcilraith. LTL2Action: Generalizing LTL Instructions for Multi-Task RL. In *International Conference on Machine Learning*, pages 10497–10508, July 2021. URL https://proceedings.mlr.press/v139/vaezipoor21a.html.

Benjamin van Niekerk, Steven D. James, Adam Christopher Earle, and Benjamin Rosman. Composing value functions in reinforcement learning. In *International Conference on Machine Learning*, volume 97, pages 6401–6409, 2019. URL http://proceedings.mlr.press/v97/van-niekerk19a.html.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

Zheng Wen, Doina Precup, Morteza Ibrahimi, Andre Barreto, Benjamin Van Roy, and Satinder Singh. On Efficiency in Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 6708–6718, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/4a5cfa9281924139db466a8a19291aff-Abstract.html.

Steven D Whitehead and Long-Ji Lin. Reinforcement learning of non-markov decision processes. *Artificial intelligence*, 73(1-2):271–306, 1995.

Tom Zahavy, Andre Barreto, Daniel J Mankowitz, Shaobo Hou, Brendan O'Donoghue, Iurii Kemaev, and Satinder Singh. Discovering a set of policies for the worst case reward. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=PUkhWz65dy5.

A Proof of Optimality

We first restate the following theorem from Alegre et al. [2022].

Theorem 1 (Alegre, Bazzan, and Silva, 2022)**.** *Let Π be a set of policies such that the set of their expected SFs, $\Psi = \{\boldsymbol{\psi}^{\pi}\}_{\pi \in \Pi}$, constitutes a CCS. Then, given any weight vector $\mathbf{w} \in \mathbb{R}^d$, the GPI policy $\pi^{GPI}_{\mathbf{w}}(s) \in \arg\max_{a \in A} \max_{\pi \in \Pi} Q^{\pi}_{\mathbf{w}}(s,a)$ is optimal with respect to $\mathbf{w}$: $V^{GPI}_{\mathbf{w}} = V^{*}_{\mathbf{w}}$.*

Applied to our setting, once the set of policies Π_{CCS} and associated SFs have been computed, we can define an arbitrary vector $\mathbf{w}$ of rewards on the exit states, and use the CCS to obtain an optimal policy $\mu^{*}_{\mathbf{w}}$ and an optimal value function $V^{*}_{\mathbf{w}}$ without learning. We can then use composition by setting the reward of the exit states equal to the optimal value.

We aim to show that for each augmented state $(u, s) \in \mathcal{U} \times \mathcal{S}$, the value function output by our algorithm equals the optimal value of (u, s) in the product MDP $\mathcal{M}' = \mathcal{F} \times \mathcal{M}^{\phi}$, i.e. that $V_{\mathbf{w}(u)}(s) = V^{*}_{\mathcal{M}'}(u, s)$. To do so, it is sufficient to show that the weight vectors $\{\mathbf{w}(u)\}_{u \in \mathcal{U}}$ are optimal.

Each element of $\mathbf{w}(u)$ is recursively defined as $\mathbf{w}_j(u) = V_{\mathbf{w}(\tau(u, \mathcal{O}(\varepsilon_j)))}(\varepsilon_j)$. If all weight vectors are optimal, it holds that $V_{\mathbf{w}(\tau(u,\mathcal{O}(\varepsilon_j)))}(\varepsilon_j) = V^{*}_{\mathcal{M}'}(\mathbf{w}(\tau(u, \mathcal{O}(\varepsilon_j))), \varepsilon_j)$ for each such exit state. Due to the above theorem, the value function $V_{\mathbf{w}(u)}$ is optimal for $\mathbf{w}(u)$. Due to composition that follows GPE and GPI, this means that the value of each internal state s is optimal, i.e. that $V_{\mathbf{w}(u)}(s) = V^{*}_{\mathcal{M}'}(u, s)$.

It remains to show that the weight vectors $\{\mathbf{w}(u)\}_{u \in \mathcal{U}}$ returned by the algorithm are indeed optimal. To do so it is sufficient to focus on the set of augmented exit states $\mathcal{U} \times \mathcal{E}$. We can state a set of optimality equations on the weight vectors as follows:

$$\begin{aligned} \mathbf{w}^{*}_{j}(u) &= V_{\mathbf{w}^{*}(\tau(u,\mathcal{O}(\varepsilon)))}(\varepsilon_j) = \max_{a} Q^{*}(\tau(u, \mathcal{O}(\varepsilon)), \varepsilon_j, a) \\ &= \max_{a} \max_{\pi} \boldsymbol{\psi}^{\pi}(\varepsilon_j, a)^{\intercal} \mathbf{w}^{*}(\tau(u, \mathcal{O}(\varepsilon))), \end{aligned}$$

where $\boldsymbol{\psi}^{\pi}(\varepsilon_j, a) = \sum_{s'} \mathbb{P}(s'|\varepsilon_j, a) \boldsymbol{\psi}^{\pi}(\varepsilon_j, a, s')$. Our termination condition implies that all subtasks take at least one time step to complete, and due to the discount factor γ, we have $\|\boldsymbol{\psi}(\varepsilon_j, a)\|_1 < 1$. Hence the update rule in Equation (11) is a contraction and converges to the set of optimal weight vectors due to the Contraction Mapping Theorem.

B Computation of a Π_{CCS}

The successor features (SF) extension of the optimistic linear support (OLS) algorithm (SFOLS, Alegre et al. [2022]) that is used in our work to compute Π_{CCS} is fully described in Algorithm 2.

It utilizes the value of the *set max policy* (SMP, Zahavy et al. [2021]), which is a commonly used, weaker approach for transfer learning in multi-objective RL. For a given weight vector $\mathbf{w}$ and a set of policies Π, it is defined as:

$$\pi^{\text{SMP}}_{\mathbf{w}}(s) = \pi'(s), \text{where } \pi' = \arg\max_{\pi \in \Pi} V^{\pi}_{\mathbf{w}}$$

and the value of this policy is $V^{\text{SMP}}_{\mathbf{w}} = \max_{\pi \in \Pi} V^{\pi}_{\mathbf{w}}$.

The algorithm constructs Π_{CCS} incrementally. Starting with the extremum points of weights simplex $\mathcal{C}^d$, it sequentially processes weights from its weight priority queue Q. In each iteration, an (optimal) policy and its successor feature representation are found for the selected weight $\mathbf{w}$. If the successor features of this policy are different from the successor features of policies currently in the Π_{CCS}, the new policy is added to the Π_{CCS} and the weight priority queue Q is adjusted. This adjustment has two steps. Firstly, the weights for which the new policy performs better than all current policies are removed and secondly, the new corner weights found with Algorithm 4 are added to Q with corresponding priority computed with Algorithm 3.

The runtime complexity of the SFOLS algorithm depends heavily on the number of policies that have to be trained and considered for Π_{CCS}. It is the same as the number of corner weights that must be analyzed in the original OLS for which Roijers et al. [2015] provide a following bound:

$$O(\binom{|CCS|\lfloor\frac{|\mathcal{E}|+1}{2}\rfloor}{|CCS|-|\mathcal{E}|} + \binom{|CCS|\lfloor\frac{|\mathcal{E}|+2}{2}\rfloor}{|CCS|-|\mathcal{E}|}).$$

Algorithm 2: SFs Optimistic Linear Support (SFOLS)

Initialize: $\Pi_{\text{CCS}} \leftarrow \{\}, \Psi \leftarrow \{\}, \mathcal{W} \leftarrow \{\}, Q \leftarrow \{\}$

1: **for** each extremum weight vector $\mathbf{w}_e \in \mathcal{C}^d$ **do**
2: Add $\mathbf{w}_e$ to Q with max. priority
3: **repeat**
4: $\mathbf{w} \leftarrow$ pop weight with max. priority in Q
5: $\pi, \boldsymbol{\psi}^\pi \leftarrow$ Solve $\langle \mathcal{S}, \mathcal{E}, \mathcal{A}, \mathcal{R}_{\mathbf{w}}, \mathbb{P}_0, \mathbb{P}, \gamma \rangle$
6: Add $\mathbf{w}$ to $\mathcal{W}$
7: **if** $\boldsymbol{\psi}^\pi \notin \Psi$ **then**
8: Remove from Q all $\mathbf{w}'$ s.t. $\mathbf{w}'^\intercal \boldsymbol{\psi}^\pi > V^{\text{SMP}}_{\mathbf{w}'}$
9: $X \leftarrow$ CornerWeights($\boldsymbol{\psi}^\pi, \mathbf{w}, \Psi$)
10: Add Ψ^π to Ψ and π to Π_{CCS}
11: **for** $\mathbf{w}' \in X$ **do**
12: $\Delta(\mathbf{w}') \leftarrow$ EstimateImprovement($\mathbf{w}', \Psi$)
13: Add $\mathbf{w}'$ to Q with priority $\Delta(\mathbf{w}')$
14: **until** Q is empty
15: **return** Π_{CCS}, Ψ

Scaling to many objectives can thus be prohibitive but should be possible by sacrificing optimality and using ϵ-CCS [Alegre et al., 2022].

Algorithm 3: EstimateImprovement

Input: New weight vector $\mathbf{w}$, Ψ, $\mathcal{W}$

1: Let $\bar{V}^*_{\mathbf{w}'}$ be the optimistic upper bound on $V^*_{\mathbf{w}'}$ computed by the following linear program

$$\max \mathbf{w}^\intercal \boldsymbol{\psi}$$
$$\text{subject to } \mathbf{w}'^\intercal \boldsymbol{\psi} \leq V^{\text{SMP}}_{\mathbf{w}'} \quad \forall\, \mathbf{w}' \in \mathcal{W}$$

2: $\Delta(\mathbf{w}) \leftarrow \bar{V}^*_{\mathbf{w}'} - V^{\text{SMP}}_{\mathbf{w}'}$
3: **return** $\Delta(\mathbf{w})$

Algorithm 4: CornerWeights

Input: New SF vector $\boldsymbol{\psi}^\pi$, current weight vector $\mathbf{w}$, current set Ψ

1: Let $\mathcal{W}_{\text{del}}$ be the set of obsolete weights removed from Q in line 8 of Algorithm 2
2: Add $\mathbf{w}$ to $\mathcal{W}_{\text{del}}$
3: $\mathcal{V}_{\text{rel}} \leftarrow \{\boldsymbol{\psi}^\pi | \boldsymbol{\psi}^\pi \in \arg\max_{\boldsymbol{\psi}^\pi \in \Psi} \mathbf{w}'^\intercal \boldsymbol{\psi}^\pi\}$ for at least one $\mathbf{w}' \in \mathcal{W}_{\text{del}}$
4: $\mathcal{B}_{\text{rel}} \leftarrow$ the set of boundaries of the weight simplex $\mathcal{C}^d$ involved in any $\mathbf{w}' \in \mathcal{W}_{\text{del}}$
5: $X \leftarrow \{\}$
6: **for** each subset Y of $d-1$ elements from $\mathcal{V}_{\text{rel}} \cup \mathcal{B}_{\text{rel}}$ **do**
7: $\mathbf{w}_c \in \mathcal{C}^d$ where $\boldsymbol{\psi}^\pi$ intersects with boundaries in Y
8: Add $\mathbf{w}_c$ to $\mathcal{C}_c$
9: **return** X

C Task specifications

The RM specifications used in the experiments are fully described in Figure 5 (Office environment) and Figure 6 (Delivery environment). We divide the tasks into three types and their natural language interpretation is as follows:

- **Sequential** 'Get coffee, then get mail and then go to an office location' (Office domain, Figure 5a) and 'go A, then B, then C and then H' (Delivery domain, Figure 6a).
- **Disjunction** 'Get coffee OR get mail, then go to an office location' (Office domain, Figure 5b)) and 'go to A OR B, then to C, and then to H' (Delivery domain, Figure 6b).

- **Composite** 'Get coffee AND get mail in any order, then go to an office location' (office domain, Figure 5c)) 'go to A AND B in any order, then go to C, then H' (Delivery domain, Figure 6c).

Note that agents must satisfy such tasks in the least possible number of steps.

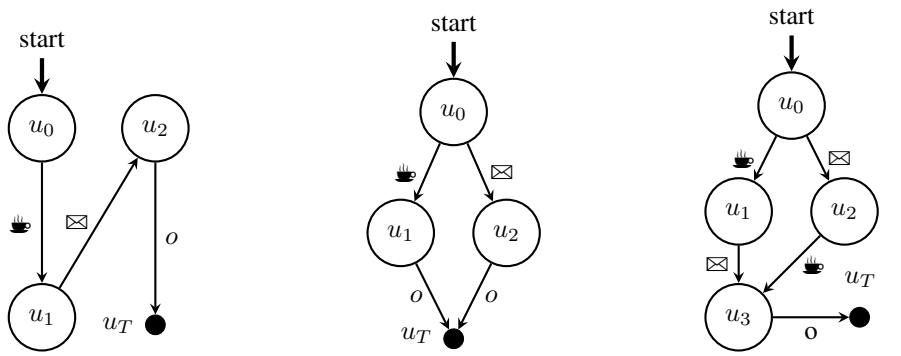


Figure 5: RMs for the Office domain (sequential (a), disjunction (b) and composite (c)) tasks.

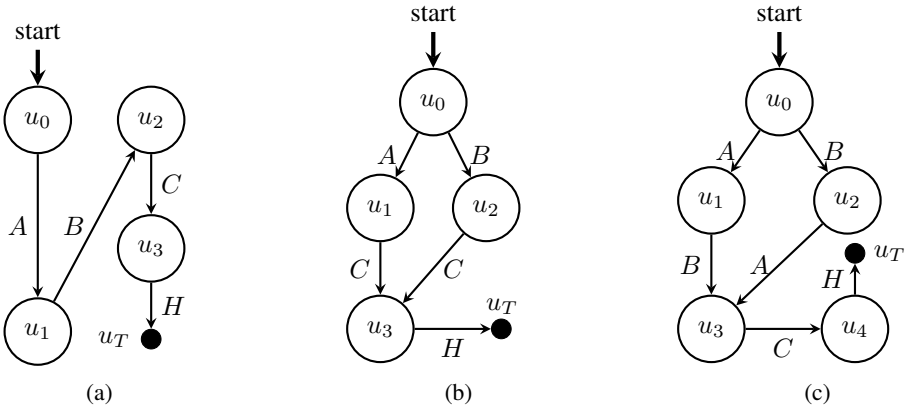


Figure 6: RMs for the Delivery domain (sequential (d), disjunction (e) and composite (f)) tasks. In this case, all non-terminal states also have a transition to a rejecting, terminal state u_r for which $\delta(u_r) = 1000$ that represents observing an obstacle ■.