Reinforcement Learning for Long-Horizon Unordered Tasks: From Boolean to Coupled Reward Machines

Kristina Levina[1,2], **Nikolaos Pappas**[1], **Athanasios Karapantelakis**[2], **Aneta Vulgarakis Feljan**[2], **Jendrik Seipp**[1]
[1]Linköping University, Linköping, Sweden
[2]Ericsson Research, Stockholm, Sweden
{kristina.levina, nikolaos.pappas, jendrik.seipp}@liu.se
{aneta.vulgarakis, athanasios.karapantelakis}@ericsson.com

Abstract

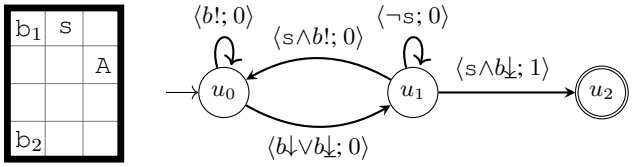
Reward machines (RMs) inform reinforcement learning agents about the reward structure of the environment, enabling support for non-Markovian tasks and improving sample efficiency. However, learning with RMs is ill-suited for long-horizon problems where subtasks can be completed in any order. In such cases, the amount of information to learn increases exponentially with the number of unordered subtasks. We address this issue by introducing three generalisations of RMs: (1) *Numeric* RMs allow users to express complex tasks in a compact form. (2) In *Agenda* RMs, states are associated with an agenda that tracks the remaining subtasks to complete. (3) *Coupled* RMs have coupled states associated with each subtask in the agenda. In addition, we introduce QCoRM, a new task-decomposition Q-learning-based algorithm that leverages coupled RMs and preserves global optimality guarantees in tabular settings. Our experiments across four domains—featuring both discrete and continuous action and state spaces—demonstrate that QCoRM scales better than baseline algorithms for long-horizon problems with unordered subtasks.

1 Introduction

Reinforcement learning (RL) agents learn policies—mappings from states to actions—by interacting with an environment and receiving reward-based feedback (Sutton and Barto 2018). Designing effective reward functions to guide learning is often difficult and time-consuming (Eschmann 2021). Moreover, learning complex, long-horizon tasks is challenging because rewards are often sparse and delayed (Dulac-Arnold et al. 2021). Task-decomposition methods mitigate these challenges by breaking down complex tasks into simpler subtasks (Dietterich 2000).

One class of task-decomposition methods leverages logical formulae, referred to as *specifications*, to express the desired behaviour of the agent (Krasowski et al. 2023). There are various specification languages and methods for compiling specifications into rewards (Camacho et al. 2019).

Another approach is to specify tasks in abstract graphs. Icarte et al. (2022) introduced Boolean reward machines (RMs)—automata in which states encode task progression. Augmenting environment states with RM states enables support for non-Markovian tasks. RMs have been shown to outperform plain reward functions and specifications in many settings (Neary et al. 2021; Unniyankal et al. 2023).



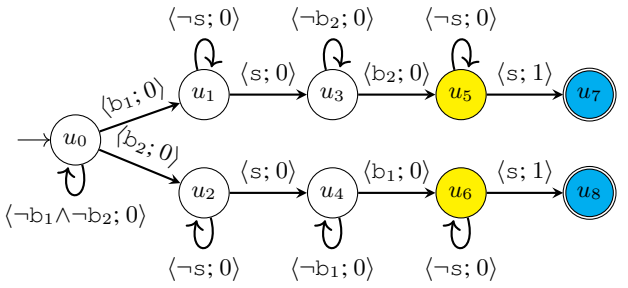
(a) (b)

Figure 1: (a) Example Delivery instance with agent A, station s, and two boxes $\mathtt{b}_1$ and $\mathtt{b}_2$. (b) Numeric reward machine (RM) for the Delivery domain. A discrete numeric variable counts the number of uncollected boxes and is mapped to numeric feature b. Here, $b{\downarrow}$ is true when a box is collected, $b{\underline{\downarrow}}$ is true when all boxes are collected, and $b!$ is true when not all boxes are collected. Boolean feature s is true when the agent arrives at the station. The agent receives a reward of 1 upon delivering all boxes to the station and 0 otherwise.

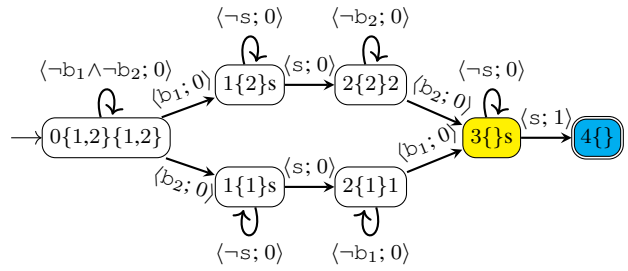
Although existing RM-aware algorithms improve sample efficiency, complex, long-horizon tasks remain challenging—especially when subtasks can be completed in any order. In such cases, the number of RM states grows exponentially with the number of subtasks: one state per subset of completed items, yielding 2^N states for N subtasks. Consequently, the agent must explore up to $N!$ distinct completion orderings to discover an optimal one. The resulting exponential state space makes learning difficult, and modelling an RM with so many states becomes tedious and error-prone.

We propose three generalisations of RMs that address the challenges mentioned above: *numeric*, *agenda*, and *coupled* RMs. To illustrate them, we introduce a running example based on a variant of the Delivery domain—a deterministic finite-grid world with four-connected cells, some containing boxes. The agent must collect all boxes and deliver them to a designated station cell. Entering a box cell automatically collects the box, and the agent cannot accidentally drop it, but only one box can be carried at a time. Figure 1a illustrates an example environment with two boxes.

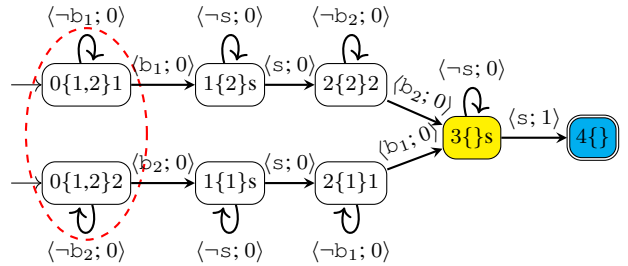
First, we introduce numeric features to RMs to simplify RM design for the user. For compact modelling, unordered subtasks are encapsulated into a discrete numeric variable, which is then mapped to a numeric feature. For our running example, Figure 1b shows a numeric RM for two boxes.



(a) Boolean RM.



(b) Agenda RM.



State	0{1,2}1	0{1,2}2	1{1}s	1{2}s	2{1}1	2{2}2	3{}s	4{}
η	12	10	6	9	2	8	1	0

(c) Coupled RM. The red dashed line indicates that the agent is in states $0\{1,2\}1$ and $0\{1,2\}2$ concurrently. The η values were found using Q-learning with the coupled RM (QCoRM).

Figure 2: RMs for the Delivery task with two boxes shown in Figure 1a. Features $\mathtt{b}_1$ and $\mathtt{b}_2$ become true when the agent collects boxes 1 and 2, respectively. Symmetric states are shown in yellow and blue.

Numeric variable w with domain $\mathcal{D}_w = \{0, 1, 2\}$ counts the number of uncollected boxes. However, the agent cannot directly use this RM for learning because RM states u_0 and u_1 are visited twice—once for each collected box. As a solution, the given numeric RM can be translated to a Boolean RM (Figure 2a). However, Boolean RMs scale poorly: the number of states grows exponentially with the number of boxes, and the agent must learn exponential information.

To solve these problems, we label each RM state u with $\langle d, T, x\rangle$, where d is the distance from the initial state in the RM to u; T is an *agenda*, a set of remaining subtasks to be completed in any order; and x is the current objective. For conciseness, we abbreviate $\langle d, \{\tau_1, \tau_2, \dots\}, x\rangle$ with $d\{\tau_1, \tau_2, \dots\}x$. States with identical labels are symmetric—they share a single Q-function—and can be merged. For example, in the agenda RM in Figure 2b, yellow state $3\{\}$s corresponds to states u_5 and u_6 in the Boolean RM. State reduction accelerates learning by decreasing the number of distinct paths the agent must explore in the RM. Moreover, associating states with an agenda T allows us to split states by the subtasks in T, yielding a *coupled* RM (Figure 2c). For example, if $T = \{\tau_1,\tau_2\}$, the state with current objective $x = T$ can be split into two coupled states: $d\{\tau_1,\tau_2\}\tau_1$ and $d\{\tau_1,\tau_2\}\tau_2$. Conceptually, the agent occupies all coupled states concurrently and may transition from any of them.

We exploit the structure of coupled RMs in a new task-decomposition Q-learning-based algorithm: QCoRM. In QCoRM, the agent learns low-level policies for completing individual subtasks along with a high-level policy over coupled RM states to decide completion order. The algorithm leverages the fact that each state in a coupled RM always corresponds to a single subtask. For the high-level policy, we treat coupled RM states as multi-armed bandits. As the decision metric, we use the expected number of steps $\eta(u)$ from each RM state u to a goal state. The high-level policy coordinates low-level learning by shaping the reward signals using the episode length, preserving global optimality in tabular settings. In our running example, the QCoRM agent learns low-level policies to locate box 1, box 2, and the station corresponding to the right parts of the RM-state labels: 1, 2, and s, respectively. In coupled states $0\{1,2\}1$ and $0\{1,2\}2$, the agent decides to transition from $0\{1,2\}2$ because its η value is 10, which is lower than 12 for state $0\{1,2\}1$.

While we focus on unordered subtasks, QCoRM can reduce the number of low-level policies to learn in sequential tasks. For example, to reach $\mathtt{a}$, $\mathtt{b}$, and $\mathtt{a}$ in order (RM states u_0, u_1, and u_2), the QCoRM agent learns two reusable policies to reach $\mathtt{a}$ and $\mathtt{b}$. In contrast, the Boolean-RM agent learns three policies, one per RM state u_0, u_1, and u_2.

2 Background

We begin by providing background information on RL and RMs. For more details on these topics, we refer to Sutton and Barto (2018) and Icarte et al. (2022), respectively.

2.1 Reinforcement Learning

Single-agent RL tasks are formalised via Markov decision processes (MDPs), defined by the tuple $M = \langle S, s_0, A, p, r, \gamma\rangle$, where S is a finite set of environment states, $s_0 \in S$ is an initial state, A is a finite set of actions, $p : S \times A \to \Delta(S)$ is a transition probability function, $\Delta(S)$ is the probability simplex over S, $r : S \times A \times S \to \mathbb{R}$ is a reward function, and $\gamma \in (0, 1)$ is a discount factor. In state s_t, the agent performs action a_t according to policy $\pi(a_t|s_t)$, transitions to state s_{t+1} with probability $p(s_{t+1}|s_t, a_t)$, and receives reward r_{t+1}. The process repeats until episode termination. The objective is to find an optimal policy $\pi^*(a_t|s_t = s)$ for all $s \in S$ to maximise the expected return $\mathbb{E}_{\pi^*}[\sum_{k=0}^{K-1} \gamma^k r_{t+k+1}|s_t = s]$, where K is the episode length.

The Q-function $q^\pi(s, a)$ quantifies the expected return the RL agent can achieve by taking a specific action a in a given

state s and following a policy π thereafter. Formally,

$$q^{\pi}(s,a) = \mathbb{E}_{\pi}[\sum_{k=0}^{K-1} \gamma^k r_{t+k+1} | s_t = s, a_t = a]. \quad (1)$$

For an optimal policy π^*, $q^* = q^{\pi^*}$.

Tabular Q-learning is an off-policy algorithm that estimates $q^*(s,a)$ without knowledge of the transition function p (Watkins and Dayan 1992). The Q-values are learnt through environment interaction and are guaranteed to converge to $q^*(s,a)$ under infinite exploration of all state–action pairs. The algorithm initialises a Q-table randomly for all (s,a). The $Q(s,a)$ values are then updated at each iteration i:

$$Q_{i+1}(s,a) \overset{\alpha}{\leftarrow} r(s,a,s') + \gamma \max_{a'} Q_i(s',a'), \quad (2)$$

where α is a learning rate. The notation $x \overset{\alpha}{\leftarrow} y$ is expanded as $x \overset{\alpha}{\leftarrow} x + \alpha(y - x)$.

For continuous states and actions, DDQN (van Hasselt, Guez, and Silver 2016) and TD3 (Fujimoto, van Hoof, and Meger 2018) extend Q-learning, though without global optimality guarantees (Mnih et al. 2015).

2.2 Boolean Reward Machines

A Boolean RM is a finite-state automaton that encapsulates the reward structure of the environment. Transitions between RM states specify the rewards received by the agent.

Definition 1 (Boolean reward machine)**.** *A* Boolean *RM is a tuple $\mathcal{R}^{Bool}_{\mathcal{P}SA} = \langle U, u_0, F, \delta_u, \delta_r \rangle$ given a set of propositional symbols $\mathcal{P}$, a set of environment states S, and a set of actions A. In the tuple, U is a finite set of states, u_0 is an initial state, F is a finite set of terminal states, δ_u is a state-transition function such that $\delta_u : U \times 2^{\mathcal{P}} \to U \cup F$, and δ_r is a state-reward function such that $\delta_r : U \to [S \times A \times S \to \mathbb{R}]$.*

Following Icarte et al. (2022), a labelling function $L : S \times A \times S \to 2^{\mathcal{P}}$ maps each environment experience (s,a,s') to the set of true propositions; δ_u then selects the abstract successor RM state; and δ_r assigns the reward. Intuitively, an MDP with RMs (MDPRM) is an MDP defined over the cross-product $\tilde{S} = S \times (U \cup F)$: an MDPRM is a tuple $M_{\mathcal{R}} = \langle \tilde{S}, \tilde{s}_0, \tilde{A}, \tilde{p}, \tilde{r}, \tilde{\gamma} \rangle$, where $\tilde{s}_0 \in \tilde{S}$ is an initial state; $\tilde{A} = A$; state-transition function $\tilde{p}(\langle s', u' \rangle \mid \langle s, u \rangle, a)$ is $p(s'|s,a)$ if $u' = \delta_u(u, L(s,a,s'))$ and $u \in U$, $p(s'|s,a)$ if $u' = u$ and $u \in F$, and 0 otherwise; state-reward function $\tilde{r}(\langle s, u \rangle, a, \langle s', u' \rangle)$ is $\delta_r(u)(s,a,s')$ if $u \notin F$ and 0 otherwise; and $\tilde{\gamma} = \gamma$ is a discount factor. The task formulation with respect to MDPRM is Markovian. Thus, Q-learning (among other RL methods) can be used to solve it. Optimal-solution guarantees of RL algorithms for MDPRMs are the same as for regular MDPs.

Icarte et al. (2022) introduced Q-learning algorithms for MDPRMs. One is Q-learning with RMs (QRM) over the cross-product $S \times U$. To exploit the RM reward structure, Icarte et al. (2022) proposed QRM with counterfactual reasoning (CRM) and a sub-optimal options-based hierarchical RL algorithm—HRM. In CRM, synthetic experiences are generated for each RM state per environment interaction.

3 Three New Types of Reward Machines

We now formally introduce numeric, agenda, and coupled RMs and compare them to Boolean RMs.

3.1 Numeric Reward Machines

Numeric RMs compactly model changes in numeric variables, simplifying the RM design process, as shown in the Delivery example (Figure 1b). In this example, the numeric variable counts uncollected boxes—remaining subtasks. More generally, numeric variables can encode any discrete numeric information, such as distance to a goal or resource usage. Numeric RMs cannot support continuous numeric variables, though, because the induced Boolean RM would have infinitely many states. To ensure that the induced Boolean RM is finite, we make the following assumption.

Assumption 1. *All numeric variables in numeric RMs are discrete and have finite bounds.*

We introduce numeric features inspired by qualitative numeric planning (Srivastava et al. 2011), where a numeric feature signals whether a discrete numeric variable increases or decreases after applying an action. In our setting, numeric variables give a sense of task progression to the agent, so we restrict them to decrease only.

Each numeric variable $w \in W$ is lower-bounded by a value w_ℓ that specifies the goal value for that variable. We map each w to a numeric feature p_w with domain $\mathcal{D}_{p_w} = \{\downarrow, \downarrow_{\!\!-}, !\}$. Value $p_w\downarrow$ indicates that w has decreased since the previous observation but remains above w_ℓ. Value $p_w\downarrow_{\!\!-}$ indicates that w has reached w_ℓ. Value $p_w!$ indicates that w has neither decreased nor is equal to w_ℓ.

Proposition 1. *Domain $\mathcal{D}_{p_w}$ fully captures possible changes between the current and previous values of w: w_t and w_{t-1}.*

Proof. Since w cannot increase and is lower-bounded by w_ℓ, there are four possible relations between w_t, w_{t-1}, and w_ℓ: (1) $w_{t-1} > w_t > w_\ell$ maps to $p_w\downarrow$; (2) $w_{t-1} > w_t = w_\ell$ and (3) $w_{t-1} = w_t = w_\ell$ map to $p_w\downarrow_{\!\!-}$; and (4) $w_{t-1} = w_t > w_\ell$ maps to $p_w!$. The mapping is thus complete. □

Now, we have all the ingredients to define numeric RMs.

Definition 2 (Numeric reward machine)**.** *A* numeric RM *is a tuple $\mathcal{R}^{num}_{\mathcal{P}SA} = \langle U, u_0, F, \delta_u, \delta_r \rangle$ given sets of Boolean and numeric features $\mathcal{P} = \mathcal{P}_{Bool} \cup \mathcal{P}_{num}$, environment states S, and actions A. U, u_0, F, and δ_r are defined as in a Boolean RM, and $\delta_u : U \times 2^{\mathcal{P}_{Bool}} \times 3^{\mathcal{P}_{num}} \to U \cup F$ is a state-transition function. Each numeric feature is associated with a discrete numeric variable $w \in W$ with a finite domain $\mathcal{D}_w$.*

Features encode observable environment events. A numeric RM accepts both Boolean and numeric features. Boolean features are propositional symbols.

Assumption 2. *For any Boolean feature p_B and state s, the agent has access to $[\![p_B]\!]$, where $[\![]\!]$ is the Iverson bracket.*

This common assumption ensures that the agent's traversal through the RM aligns with actual environment conditions. Likewise, we assume that the agent has access to the values

of each numeric feature $p_w \in \mathcal{D}_{p_w}$. That is, the agent can observe whether w has decreased since the previous observation and whether the goal w_ℓ has been reached.

Numeric RMs offer a compact way to specify tasks, reducing the modelling effort. They are particularly useful for tasks in which subtasks can be completed in any order and the agent must learn an optimal ordering. However, learning with numeric RMs has two challenges. First, numeric RMs do not provide any task decomposition or intermediate guidance, which is an essential property of RMs. Second, a policy cannot be learnt directly from a numeric RM because a single RM state may correspond to multiple stages of task progression. For instance, in the example numeric RM (Figure 1b), the agent can occupy only two RM states before completing the task, regardless of the number of boxes to deliver. Specifically, different trajectory segments—corresponding to different numbers of undelivered boxes—all map to the same numeric-RM state u_0. This would cause the agent to repeatedly overwrite learnt information for u_0.

To address these issues, numeric RMs must be translated to Boolean RMs by unfolding their states with respect to the domains of numeric variables $\mathcal{D}_w$.

3.2 From Numeric to Boolean Reward Machines

To address the agent's inability to learn from a user-provided numeric RM, it is translated to a Boolean RM under the following assumptions.

Assumption 3. *Numeric variables in a numeric RM have finite domains that correspond to the collections of different tasks given to the agent.*

To enable the functionality of subtask completion in any order, we specifically assume the following:

Assumption 4. *Sets of tasks encapsulated in numeric variables can be completed in any order.*

Encapsulation of *ordered* tasks into a *numeric* variable is a useful extension to be realised in future work. For now, we assume that each subtask in the collection of ordered subtasks is assigned a Boolean feature. Next, we restrict our numeric RMs as follows:

Assumption 5. *A numeric RM is formulated such that only one task for each feature p_w can be completed per step in the environment and, upon its completion, $p_w{\downarrow}$ or $p_w{\downarrow\!\!\downarrow}$ signals to transition to the next RM state.*

If the assumptions are met, a translation algorithm takes a numeric RM $\mathcal{R}^{\text{num}}_{\mathcal{P}SA}$ along with the numeric-variable domains $\mathcal{D}_w$ as input and outputs a Boolean RM $\mathcal{R}^{\text{Bool}}_{\mathcal{P}SA}$. The translation algorithm unfolds the RM states with respect to $\mathcal{D}_w$ following the transitions in $\mathcal{R}^{\text{num}}_{\mathcal{P}SA}$. In the translated $\mathcal{R}^{\text{Bool}}_{\mathcal{P}SA}$, each state corresponds to a collection of subtasks to perform next for each possible completion order. As a result, the $\mathcal{R}^{\text{Bool}}_{\mathcal{P}SA}$ representation is exponential in the number of subtasks and can contain symmetric states corresponding to the same Q-values. However, reducing symmetric states requires associating the RM states with the additional information about uncompleted subtasks and completion orders.

Another limitation of Boolean RMs is that CRM is only partially applicable when completed subtasks are removed from the environment, as counterfactual experiences for their associated RM states become incorrect. For example, in the toy task depicted in Figure 1a, collecting box 1 transitions the agent to u_1 in the Boolean RM (Figure 2a) and clears its cell, so future visits no longer make Boolean feature $\mathtt{b}_1$ true. Consequently, counterfactual experiences for states u_0 and u_4 will be erroneous, and counterfactual reasoning can only be applied to RM states reachable after u_1.

3.3 Agenda Reward Machines

We handle the problems above by associating the RM states with an *agenda*—the remaining subtasks to complete.

Definition 3 (Agenda reward machine)**.** *An agenda RM is a Boolean RM with numeric variables $w \in W$ with domains $\mathcal{D}_w = \mathcal{T}$ enumerating unordered subtasks and a labelling function $\Xi : U \cup F \to \{\langle d, T, x\rangle \mid T \in 2^{\mathcal{T}}\}$, where $\mathcal{T}$ is the set of all subtasks to complete, $T \subseteq \mathcal{T}$ is the set of remaining subtasks, d is the RM state depth, and x is the current objective. The depth d of RM state u is the distance from the initial state of the RM to u.*

We construct a power set $2^{\mathcal{T}}$ from all subtasks assuming that the agent needs to find an optimal completion order. In $\langle d, T, x\rangle$, x can be either T or a Boolean feature which must become true in the environment to trigger the transition to the next RM state. If $x = T$, then, upon completion of any subtask from T, the agent transitions to the next RM state.

This labelling scheme aligns with the RL principle of learning from information contained in future states (see the Q-learning update in (2)). By tracking remaining subtasks rather than completed ones, the RM maintains the correct direction of information flow for value backpropagation. Furthermore, we track the RM state depth so that symmetric states get identical labels and can be merged.

Proposition 2. *The labelling scheme in an agenda RM uniquely identifies RM states with respect to task progression.*

Proof. Assume, for the sake of contradiction, that two RM states have identical labels $\langle d, T, x\rangle$ but correspond to different stages of task progression. However, task progression is completely determined by the remaining subtasks T, current objective x, and RM-state depth d. If it were different for the same T and x, then the d values would be different. For example, in task $\mathtt{a}{\to}\mathtt{b}{\to}\mathtt{a}$, $d = 0$ for the first $\mathtt{a}$, and $d = 2$ for the second $\mathtt{a}$. Therefore, RM states cannot have equal labels at different task-progression stages. Conversely, if two states have identical task progression, then by definition they share the same T, d, and x. Thus, their labels coincide. □

We translate the given numeric RM into an agenda RM using a similar algorithm to that of numeric-to-Boolean-RM translation. The translation differs only in the labelling scheme. While the agenda RM already provides computational advantages via state reduction, converting it to a coupled RM yields even more benefits.

3.4 Coupled Reward Machines

In a coupled RM, agenda-RM states are split by subtasks in the current objective $T = \{\tau_1, \tau_2, \ldots, \tau_N\}$.

Definition 4 (Coupled reward machine)**.** *For task set $T = \{\tau_1, \tau_2, \ldots, \tau_N\}$, a* coupled RM *is an agenda RM where states labelled with $\langle d, T, T\rangle$ are split into N states $\langle d, T, \tau_1\rangle$, $\langle d, T, \tau_2\rangle$, …, $\langle d, T, \tau_N\rangle$. The split states remain coupled such that the agent is in all of them concurrently.*

Algorithmically, the agent occupies all coupled-RM states simultaneously while being in a single environment state. Moreover, as each coupled-RM state corresponds to exactly one subtask, the agent can learn low-level policies for completing individual subtasks. The agent has access to the RM label and can therefore identify which subtask to perform.

4 Q-learning with Coupled Reward Machines

We introduce Q-learning with coupled RMs (QCoRM), a new task-decomposition algorithm in which the agent learns low-level policies for completing individual subtasks and a high-level policy for determining their completion order.

4.1 Low-Level Policies

Each state in a coupled RM is associated with exactly one subtask, allowing the agent to learn a separate low-level policy for each subtask in $U' = \mathcal{T} \cup B$, where $\mathcal{T}$ is the complete agenda and B is the set of all Boolean objectives. Consequently, the state space of the corresponding MDPRM can be constructed as $S \times U'$ using the set of all subtasks U' rather than the full set of RM states U. The amount of low-level information to learn thus increases only linearly with $|U'|$. In contrast, Boolean RMs require the full state space $S \times U$, forcing the agent to learn information exponential in the number of unordered subtasks.

As the agent occupies all coupled states $\langle d, T, \tau_1\rangle$, $\langle d, T, \tau_2\rangle, \ldots, \langle d, T, \tau_N\rangle$ at once, it can perform Q-learning updates in parallel. As long as the agent remains in the current set of coupled RM states, the Q-values for each subtask τ_k, $k = 1, \ldots, N$ are updated concurrently:

$$Q_{i+1}(\langle s, \tau_k\rangle, a) \xleftarrow{\alpha} r(\langle s, \tau_k\rangle, a, \langle s', \tau_k\rangle) + \gamma \max_{a' \in A} Q_i(\langle s', \tau_k\rangle, a'), \tag{3}$$

where α is the learning rate, γ is the discount factor, and $r(\langle s, \tau_k\rangle, a, \langle s', \tau_k\rangle)$ is the immediate reward.

When the agent completes subtask τ_ℓ and transitions to the next RM state, the Q-values associated with the remaining subtasks $T \setminus \tau_\ell$ are updated according to (3). The update for RM state $\langle d, T, \tau_\ell\rangle$ is, however,

$$Q_{i+1}(\langle s, \tau_\ell\rangle, a) \xleftarrow{\alpha} R(K, \tau), \tag{4}$$

where $K^{\tau_\ell} > 0$ is the τ_ℓ execution duration in timesteps and $R(K, \tau)$ is the final reward for τ_ℓ completion.

Theorem 3. *The final reward that preserves global optimality guarantees of tabular Q-learning for any subtask τ is*

$$R(K, \tau) = \begin{cases} \check{R} & \text{if } \delta K^\tau = 0, \\ \gamma^{\delta K^\tau + 1}\check{R} + \frac{\gamma^{1-K^\tau} - \gamma^{\delta K^\tau}}{1-\gamma} r_{\min} & \text{otherwise}, \end{cases} \tag{5}$$

where K^τ is the τ execution duration in timesteps, $\check{R} \in \mathbb{R}_{>0}$ is the final reward, $\delta K^\tau = \max\{\bar{K}^\tau_{op} - K^\tau, K - \bar{K}_{mi}\}$, K is the episode length, $\bar{K}_{mi}$ is the optimal minimum expected episode length, $\bar{K}^\tau_{op}$ is the optimal expected duration of τ under a globally optimal policy, and $r_{\min} \leq 0$ is the minimum immediate reward.

Proof. Part 1: Agent prefers globally optimal policies under given $\bar{K}_{op}$ and $\bar{K}^\tau_{op}$. Consider an arbitrary subtask τ that is part of some complete policy for the entire task. The discounted return for completing τ with any *locally sub-optimal* policy should exceed that of any *locally optimal* policy if the former contributes to a *globally optimal* solution. Let π^τ_1 be a locally optimal but globally sub-optimal policy with the expected local duration $\mathbb{E}[K^\tau_1] = \bar{K}^\tau_1$ and induced episode length $\bar{K}_1 > \bar{K}_{\text{op}}$. Let π^τ_{op} be a locally sub-optimal but globally optimal policy with the expected local duration $\bar{K}^\tau_{\text{op}}$ and induced episode length $\bar{K}_{\text{op}}$. By assumption, $\bar{K}^\tau_{\text{op}} > \bar{K}^\tau_1$.

If the final reward was equal to constant $\check{R}$, the agent would prefer π^τ_1 over π^τ_{op}, as π^τ_1 is locally optimal and hence yields higher expected return (1): $\check{Q}^\tau_1 > \check{Q}^\tau_{\text{op}}$ ($\mathbb{E}_{\pi^\tau_1}[\check{R}\gamma^{K^\tau_1 - 1} + \sum_{k_1=0}^{K^\tau_1 - 2} \gamma^{k_1} r^\tau_{k_1+1}] > \mathbb{E}_{\pi^\tau_{\text{op}}}[\check{R}\gamma^{K^\tau_{\text{op}} - 1} + \sum_{k_{\text{op}}=0}^{K^\tau_{\text{op}} - 2} \gamma^{k_{\text{op}}} r^\tau_{k_{\text{op}}+1}]$). For readability, we omit time index t. To prevent convergence to a globally sub-optimal policy, the final reward depends on K^τ_1, and the returns become $Q^\tau_1 = \mathbb{E}_{\pi^\tau_1}[R(K^\tau_1)\gamma^{K^\tau_1 - 1} + \sum_{k_1=0}^{K^\tau_1 - 2} \gamma^{k_1} r^\tau_{k_1+1}]$ and $Q^\tau_{\text{op}} = \check{Q}^\tau_{\text{op}}$. To achieve $Q^\tau_{\text{op}} > Q^\tau_1$, the reward must therefore satisfy $R(\bar{K}^\tau_1) < \check{R} - \gamma^{1-\bar{K}^\tau_1}(\check{Q}^\tau_1 - \check{Q}^\tau_{\text{op}})$ for global optimality.

Because in many non-cumulative tasks, such as navigation or control, the agent typically receives negative or zero intermediate rewards to avoid non-optimal behaviours (Cui and Yu 2023), we assume that immediate rewards for all subtasks are bounded between $r_{\min} \leq 0$ and $r_{\max} \leq 0$. We can then bound the return difference as follows: $\check{Q}^\tau_1 - \check{Q}^\tau_{\text{op}} \leq \check{R}(\gamma^{\bar{K}^\tau_1 - 1} - \gamma^{\bar{K}^\tau_{\text{op}} - 1}) - \frac{1-\gamma^{\bar{K}^\tau_{\text{op}} - 1}}{1-\gamma} r_{\min}$. Rearranging the terms yields the bound on the reward $R(\bar{K}^\tau_1) < \gamma^{\bar{K}^\tau_{\text{op}} - \bar{K}^\tau_1}\check{R} + \frac{\gamma^{1-\bar{K}^\tau_1} - \gamma^{\bar{K}^\tau_{\text{op}} - \bar{K}^\tau_1}}{1-\gamma} r_{\min}$. Since $\gamma^{\bar{K}^\tau_{\text{op}} - \bar{K}^\tau_1} > \gamma^{\max\{\bar{K}^\tau_{\text{op}} - \bar{K}^\tau_1, \bar{K}_1 - \bar{K}_{\text{mi}}\} + 1}$, as $\gamma < 1$, expression (5) satisfies the bound above for any local duration $K^\tau < \bar{K}^\tau_{\text{op}}$ or any realised episode length $K > \bar{K}_{\text{mi}}$. In the limit of Q-learning, realised episode lengths concentrate on their expected values under each policy, so the return ordering holds almost surely. The result is valid for stationary $\bar{K}_{\text{mi}}$ and $\bar{K}^\tau_{\text{op}}$, but they are updated $\bar{K}_{\text{mi}} \xleftarrow{\alpha_K} K$ and $\bar{K}^\tau_{\text{op}} \xleftarrow{\alpha_K} K^\tau$.

Part 2: $\bar{K}_{mi}$ and $\bar{K}^\tau_{op}$ converge to $\bar{K}^*_{mi}$ and $\bar{K}^{*\tau}_{op}$, respectively. To push the optimal episode length to minimum, $\bar{K}_{\text{mi}}$ is updated when $K \leq \bar{K}_{\text{mi}}$ during exploitation, so the sequence $\{\bar{K}_{\text{mi}}\}$ is non-increasing and bounded below by $\bar{K}^*_{\text{mi}} > 0$. At each fixed $\bar{K}_{\text{mi}}$ and $\bar{K}^\tau_{\text{op}}$, the reward (5) is stationary; by Watkins and Dayan (1992), Q-learning contracts to the optimal Q^*-values for each low-level policy. By Part 1, each converged low-level policy achieves $\delta K^\tau = 0$, so the reward reduces to the constant $\check{R}$, independent of episode length. With the reward fixed at $\check{R}$, the Bellman contraction with $\gamma < 1$ independently minimises each subtask duration K^τ toward $\bar{K}^{*\tau}_{\text{op}}$, since $\gamma^{K^\tau - 1}\check{R}$ is strictly decreasing in K^τ; the

global episode length is thus collectively reduced, decrementing $\bar{K}_{\text{mi}}$ if a shorter episode exists. Since no episode shorter than $\bar{K}^*_{\text{mi}}$ exists, the sequence stabilises at $\bar{K}^*_{\text{mi}}$, the reward permanently fixes at the optimal level, and the globally optimal Q^*-values follow. The local durations $\bar{K}^{*\tau}_{\text{op}}$ are then fixed and aligned with the optimal episode. Therefore, the reward (5) preserves global optimality for any K^τ and τ. □

Experiences with transitions between distinct RM states are stored in a dedicated buffer during the episode and are replayed once the episode terminates and the episode length along with the low-level-policy durations are known. Whenever $K \leq \bar{K}_{\text{mi}}$, $\bar{K}_{\text{mi}}$ is updated by incremental averaging, so that the estimates track the minimum episode length. The global episode-length penalty is needed so that each local policy is aware of the global context and is not greedily optimised for the local duration.

4.2 High-Level Policy

The high-level policy selects the next subtask from the remaining set $T = \{\tau_1, \ldots, \tau_N\}$ by treating the available choices as arms in a stochastic multi-armed bandit (Sutton and Barto 2018). Because costs change as low-level policies improve (non-stationarity), we maintain exponentially recency-weighted estimates via a constant step-size incremental update. Let $\eta(u)$ be the estimated number of steps from each RM state u to a goal. After each episode e, for every RM state $u_e \in U_e$ visited during that episode, we update $\eta(u_e) \xleftarrow{\alpha_\eta} K_{u_e \to \text{goal}}$, where α_η is a constant learning rate and $K_{u_e \to \text{goal}}$ is the observed number of steps from u_e to the goal. We store $\eta(u)$ in a lookup table with $|U|$ entries, which can grow with the agenda structure, but each update is a single scalar operation. From the set of coupled states, the agent selects the transition with the lowest cost as the next subtask. Once the agent converges to an optimal solution, the stored $\eta(u)$ values equal the optimal steps-to-go, and the high-level policy yields a trajectory of length $\bar{K}_{\text{op}}$.

To force exploring all transitions in the RM, we introduce an exploration probability $\xi \in (0, 1)$. During exploration, the agent randomly selects RM transitions with a bias towards unused transitions. The experiences stored in the temporary buffer (4) are only used for learning when the agent exploits, i.e., when it selects the best transition based on the lowest $\eta(\langle d, T, \tau_k \rangle), k = 1, 2, \ldots, N$. Therefore, the low-level policies are learnt to align with the best high-level policy.

5 Experiments

We integrate QCoRM with DDQN and TD3 by adding a dedicated replay buffer that stores valuable experiences corresponding to the transitions between different RM states. After each episode, these stored transitions are replayed to train the network using the same update procedure as above.

Our implementation builds on the code by Icarte et al. (2022) and Stable-Baselines3 (Raffin et al. 2021). For evaluation, we use two tabular domains, Delivery (our running example) and Office (Icarte et al. 2022), and two continuous domains, Water (Karpathy 2015) and MuJoCo HalfCheetah (Todorov, Erez, and Tassa 2012). The agent's initial position is fixed throughout the experiments. Experiments run on Intel Xeon Gold 6130 CPUs, using a single core with 16 GiB of memory for the tabular domains and 32 GiB for the continuous domains. Each algorithm run is limited to 10^6 steps and 40 hours. We report median performance across 10 seeds, with the shaded regions on plots representing the 25th and 75th percentiles.

We compare QCoRM against state-of-the-art RM methods (CRM, HRM, and QRM) and DiRL (Jothimurugan et al. 2021), a compositional learning approach from specifications. We use the DiRL code from the authors' repository and translate agenda RMs into abstract reachability graphs required by DiRL. For fair comparison, we use the same low-level policy learners for DiRL and QCoRM.

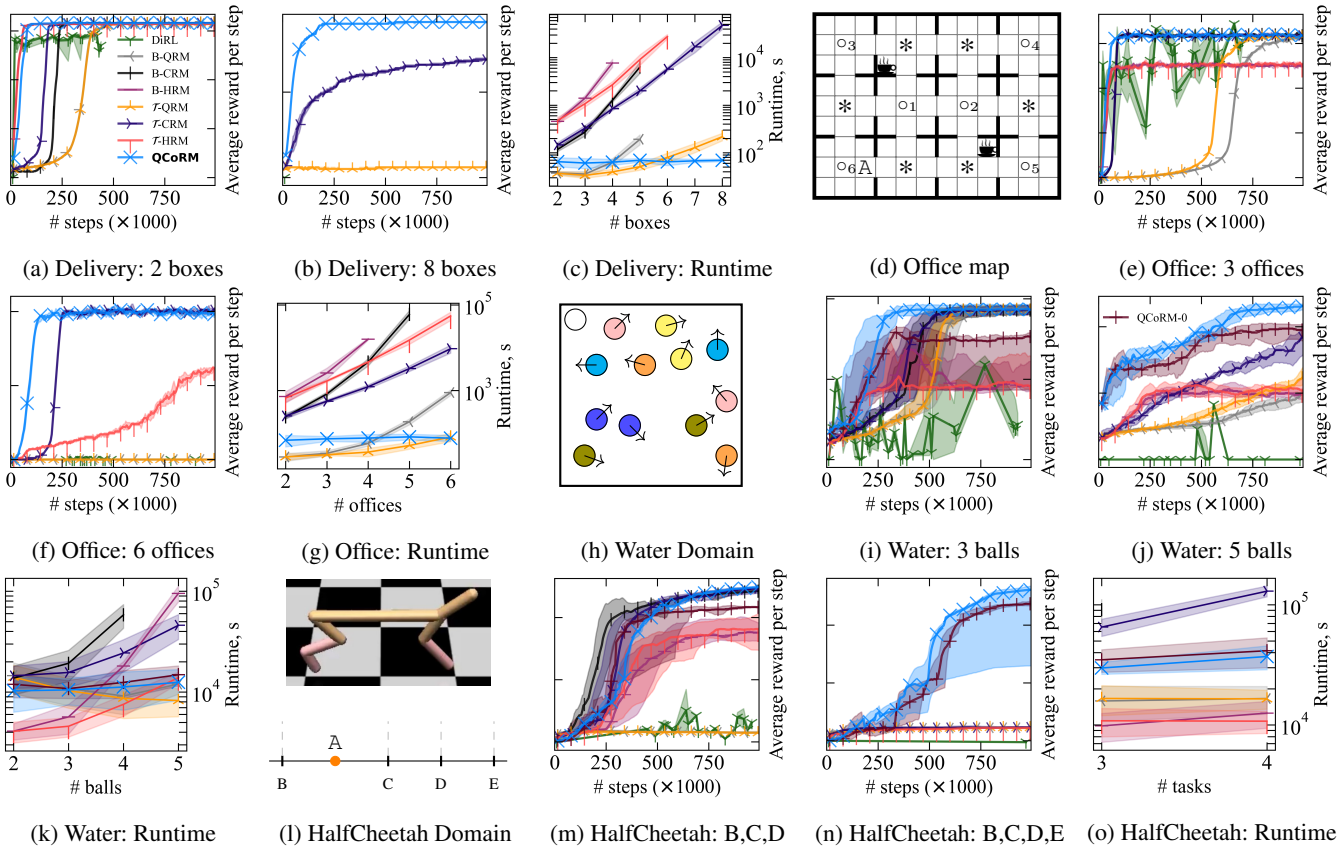
We adopt a sparse reward setting, where the agent receives a positive reward only after task completion and zero or negative reward otherwise, because giving positive intermediate rewards can lead to sub-optimal policies (Cui and Yu 2023). In particular, for the tabular and Water domains, the agent receives a reward of 1 after task completion and 0 otherwise. For HalfCheetah, the agent receives a reward of 1 000 upon completing each subtask and a control penalty otherwise.

Parameters. For tabular Q-learning, the learning rate $\alpha = 10^{-5}$, discount factor $\gamma = 0.9$, and exploration parameter $\epsilon = 0.1$. For DDQN, $\alpha = 10^{-5}$, $\gamma = 0.9$, soft update parameter $\tau = 0.1$, and ϵ decays linearly from 1 to 0.1 in increments of 0.01. The replay buffer and batch sizes are 50K and 32 for QRM and are scaled up by $|U'|$ for QCoRM and $|U|$ for HRM and CRM. The RM-exploration parameter ξ decays from 1 to 0.1 in increments of 0.001. For TD3, $\alpha = 3 \times 10^{-4}$, $\gamma = 0.99$, $\tau = 0.005$, buffer size is 10^6, and batch size is 256. All neural networks have three layers with 512 hidden units each; training frequency and gradient steps are 1. The episode-length learning rate α_K is 0.005 and 0.05 for tabular and continuous domains, respectively; and $\alpha_\eta = 0.005$ for all domains. All other parameters use the default setting.

5.1 Tabular Domains

Our **Delivery** tasks are seven 10×10 grids with 2–8 randomly placed boxes that are removed upon collection. Figures 3a and 3b show experimental results for two and eight boxes, respectively. Although HRM converges fastest for two boxes, it fails to converge for eight boxes. QCoRM converges to an optimal policy for two and eight boxes, followed by the CRM methods. DiRL converges to a sub-optimal policy for two boxes and fails to converge for eight. Notably, QCoRM's performance does not deteriorate after the removal of collected boxes. In contrast, the CRM methods simulate experiences only for reachable RM states, so fewer experiences are available as the agent traverses the RM. Convergence is faster for CRM with the agenda RMs than with the Boolean RMs because more states can be identified as reachable due to the more informative labelling scheme. Boolean RMs for tasks with more than five boxes exceed available memory.

The **Office** domain (Icarte et al. 2022), shown in Figure 3d, features a grid world with walls, doors, and decorations. The agent is given five progressively more complex tasks: deliver coffee to 2–6 offices in any order. The agent fails the episode upon stepping on a decoration. Figures 3e and 3f



(a) Delivery: 2 boxes (b) Delivery: 8 boxes (c) Delivery: Runtime (d) Office map (e) Office: 3 offices

(f) Office: 6 offices (g) Office: Runtime (h) Water Domain (i) Water: 3 balls (j) Water: 5 balls

(k) Water: Runtime (l) HalfCheetah Domain (m) HalfCheetah: B,C,D (n) HalfCheetah: B,C,D,E (o) HalfCheetah: Runtime

Figure 3: Results of QCoRM (ours) and baselines, Q-learning (QRM), QRM with counterfactual reasoning (CRM), hierarchical HRM, and DiRL. Boolean RMs (B-QRM, B-CRM, B-HRM) and our agenda RMs ($\mathcal{T}$-QRM, $\mathcal{T}$-CRM, and $\mathcal{T}$-HRM) were tested for Delivery (a,b), Office (e,f), Water (i,j), and HalfCheetah (m,n). As DiRL does not allow setting the number of environment interactions in advance and provides a complete evaluation only after training finishes, we cannot plot DiRL's learning curve. Instead, we run DiRL with an increasing number of iterations and mark its final performance as a single point per run against the total environment interactions consumed—summing both low-level policy training and high-level policy construction. QCoRM-0 stands for QCoRM without the joint optimisation from (5). (c,g,k,o) Time (in seconds) to run 10^6 steps in relation to the number of unordered objectives. (d) Office environment with agent A, offices $\circ_i, i = 1, 2, \ldots, 6$, decorations marked by stars, and coffee machines. (h) Water environment with the agent (white ball) and balls of different colours with arrows indicating their velocities. (l) HalfCheetah environment with agent A (orange dot) and objectives B, C, D, and E along the horizontal axis.

show the results for delivering coffee to three and six offices, respectively. QRM-based methods converge slowest. DiRL converges for three coffees but fails for six. HRM converges to sub-optimal policies. As in Delivery, QCoRM outperforms CRM-based methods. In contrast, CRM with both the agenda and Boolean RMs converge at similar rates because offices are not removed from the map after the agent's visit, keeping all the RM states reachable for experience simulation. We obtained similar results for different numbers of offices.

5.2 Continuous Domains and Ablation Study

The **Water** domain (Karpathy 2015) is a two-dimensional arena populated by coloured balls that move at fixed speeds and rebound off walls (Figure 3h). The agent, represented by a white ball, adjusts its velocity along four cardinal directions via discrete control. This domain is challenging because the order in which balls are touched affects the low-level policies for reaching each individual ball. For example, the agent may need to decelerate before touching one ball to optimally position itself for the next. As ball positions and velocities are continuous, we use DDQN. We randomly generate maps with six pairs of differently coloured balls and define four progressively difficult tasks to touch 2–5 balls of specified colours in any order without touching other balls. Figures 3i and 3j show the results for three and five balls, respectively. QCoRM again converges fastest, followed by CRM, then HRM (sub-optimal), then QRM, and finally DiRL (sub-optimal). We observe similar results across all Water instances.

We conduct an **ablation study** to empirically demonstrate that the reward function (5) in QCoRM promotes convergence to more optimal policies. To this end, we remove the joint optimisation from (5) and give a reward of 1 upon each subtask completion (agent QCoRM-0). The results in Figures 3i and 3j show that the agent converges to sub-optimal

policies without the joint optimisation, as expected, because low-level policies are not shaped by the global episode-length signal. Thus, the gap between QCoRM and QCoRM-0 isolates the contribution of reward shaping, while the gap between QCoRM-0 and the CRM baselines reflects the structural advantage of the coupled RM representation.

The **HalfCheetah** domain (Todorov, Erez, and Tassa 2012) involves controlling a two-dimensional bipedal robot to reach specific positions along a horizontal axis (Figure 3l). We define two tasks where the agent must reach three (B, C, D) or four (B, C, D, E) positions in any order. We use TD3 for continuous control. Because QCoRM delays learning from valuable experiences stored in its dedicated replay buffer, it initially converges more slowly than the CRM-based methods, as we observe in the task to reach three positions (Figure 3m). In the previously discussed domains, this was not an issue because of zero intermediate rewards. In HalfCheetah, however, the agent receives dense negative feedback, making early learning from all experiences more critical. Nevertheless, reaching four positions (Figure 3n) can only be done by QCoRM, proving its superior scalability. QRM methods and DiRL fail on both tasks. Notably, QCoRM without joint optimisation learns to reach the leftmost position B without decelerating, resulting in sub-optimal overall solutions.

5.3 Runtimes

We compare the runtimes of the tested algorithms against task size in Figures 3c, 3g, 3k, and 3o. In tabular domains, QRM runtimes scale exponentially due to increasing Q-table sizes. In continuous domains, however, QRM runtimes do not change much because the neural network sizes are independent of the task size. The plots show that QCoRM runtimes scale linearly because the number of low-level simulated experiences grows linearly with the number of subtasks. In contrast, both HRM and CRM runtimes exhibit exponential growth because the number of simulated experiences is proportional to the number of RM states. Notably, the exponential blowup in the number of Boolean RM states causes B-CRM and B-HRM to exceed time limits in large tasks. While the CRM- and HRM-based methods are therefore unsuitable for large tasks, QCoRM handles them efficiently.

6 Related Work

Beyond the RMs discussed in our paper, a separate line of work uses automata in RL for task decomposition. In these approaches, the automaton structure is inferred from experience rather than provided by the user. First, Furelos-Blanco et al. (2021) introduced ISA—a method for discovering subgoal automata with binary rewards (0 or 1) from traces. Next, Furelos-Blanco et al. (2023) introduced hierarchies of RMs (HRMs), which are learnt from traces. Compared to the RMs in our work, HRMs only support labelling edges with real-valued rewards rather than reward functions. Moreover, unlike in QCoRM, the global policy learnt with HRMs is not guaranteed to be optimal because each automaton is learnt in isolation. More recently, Ardon et al. (2025) introduced first-order RMs (FORMs), where first-order logic is used for more compact task specifications. However, FORMs still support only real-valued rewards. Furthermore, in contrast to the RMs discussed in our paper, for any number of unordered subtasks, there is only one FORM state expressed by a universally quantified FO atom, thus providing no guidance for unordered-subtask completion.

Below, we compare our QCoRM algorithm with task-decomposition RL algorithms for tackling long-horizon tasks composed of unordered subtasks. RM-based RL belongs to a class of approaches in which tasks are decomposed into symbolically defined subtasks. By contrast, approaches that focus on discovering low-level skills or options (Bacon, Harb, and Precup 2017; Klissarov and Machado 2023) or do not accept symbolic specifications as input (Nachum et al. 2018; Mendez, van Seijen, and Eaton 2022) are outside the scope of this comparison but could be complementary to RMs.

Learning a policy for each subtask is well-established in the literature (Mohan, Zhang, and Lindauer 2024). Some notable works include options (Sutton and Barto 2018), policy sketches (Andreas, Klein, and Levine 2017), taskable RL (Illanes et al. 2020), and hierarchical policies (Drexler, Seipp, and Geffner 2023). These approaches typically optimise Markovian returns and do not, by default, provide semantics for non-Markovian rewards.

Supporting history-dependent objectives generally requires state augmentation or an explicit automaton (Skalse and Abate 2023). In addition to RMs, some notable works include R-AVI (Jothimurugan, Bastani, and Alur 2021), DiRL (Jothimurugan et al. 2021), and LSTS (Shukla et al. 2024). Both DiRL and LSTS translate SPECTRL tasks into abstract graphs. A high-level policy for selecting transitions in the graph is then determined using Dijkstra's algorithm with costs based on the transition success probabilities between different state-space regions.

Likewise, QCoRM uses the RM structure to guide high-level decision making. However, QCoRM converges to optimal solutions via shaping low-level policies by the global episode-length signal. In contrast, DiRL's local policies optimise only for individual abstract transitions, so the composed high-level policy is not guaranteed to be globally optimal. Moreover, DiRL learns one low-level policy at a time, whereas QCoRM learns a low-level policy for each subtask in all coupled RM states in parallel. This distinction becomes crucial when tasks contain many unordered subtasks.

7 Conclusions

We introduce three generalisations of RMs to tackle long-horizon tasks with unordered subtasks. (1) Numeric RMs assist the user with compact task representation. (2) States in agenda RMs are associated with an agenda—the remaining subtasks to complete. (3) In coupled RMs, states are split by subtasks in the agenda, allowing for parallel learning of low-level policies and high-level decision-making on top. We develop a new task-decomposition algorithm for Q-learning with coupled RMs (QCoRM) that shows computational advantages over baseline methods in tabular and continuous domains. In particular, QCoRM scales well for long-horizon tasks with unordered subtasks, while preserving the optimality guarantees of tabular Q-learning. In future work, QCoRM could be extended beyond Q-learning-based methods.

8 Acknowledgments

This work was supported by Ericsson Research and the Wallenberg AI, Autonomous Systems, and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

Andreas, J.; Klein, D.; and Levine, S. 2017. Modular Multitask Reinforcement Learning with Policy Sketches. In *Proceedings of the 34th International Conference on Machine Learning*, 166–175. PMLR.

Ardon, L.; Furelos-Blanco, D.; Parac, R.; and Russo, A. 2025. FORM: Learning Expressive and Transferable First-Order Logic Reward Machines. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, 142–151.

Bacon, P.-L.; Harb, J.; and Precup, D. 2017. The Option-Critic Architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Camacho, A.; Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2019. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 19, 6065–6073.

Cui, W.; and Yu, W. 2023. Reinforcement Learning with Non-Cumulative Objective. *IEEE Transactions on Machine Learning in Communications and Networking*, 1: 124–137.

Dietterich, T. G. 2000. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13: 227–303.

Drexler, D.; Seipp, J.; and Geffner, H. 2023. Learning Hierarchical Policies by Iteratively Reducing the Width of Sketch Rules. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, 208–218.

Dulac-Arnold, G.; Levine, N.; Mankowitz, D. J.; Li, J.; Paduraru, C.; Gowal, S.; and Hester, T. 2021. Challenges of Real-World Reinforcement Learning: Definitions, Benchmarks and Analysis. *Machine Learning*, 110(9): 2419–2468.

Eschmann, J. 2021. Reward Function Design in Reinforcement Learning. *Reinforcement Learning Algorithms: Analysis and Applications*, 25–33.

Fujimoto, S.; van Hoof, H.; and Meger, D. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning*.

Furelos-Blanco, D.; Law, M.; Jonsson, A.; Broda, K.; and Russo, A. 2021. Induction and Exploitation of Subgoal Automata for Reinforcement Learning. *Journal of Artificial Intelligence Research*, 70: 1031–1116.

Furelos-Blanco, D.; Law, M.; Jonsson, A.; Broda, K.; and Russo, A. 2023. Hierarchies of Reward Machines. In *Proceedings of the 40th International Conference on Machine Learning*, 10494–10541. PMLR.

Icarte, R. T.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2022. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *Journal of Artificial Intelligence Research*, 73: 173–208.

Illanes, L.; Yan, X.; Icarte, R. T.; and McIlraith, S. A. 2020. Symbolic Plans as High-Level Instructions for Reinforcement Learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 540–550.

Jothimurugan, K.; Bansal, S.; Bastani, O.; and Alur, R. 2021. Compositional Reinforcement Learning from Logical Specifications. *Advances in Neural Information Processing Systems*, 34: 10026–10039.

Jothimurugan, K.; Bastani, O.; and Alur, R. 2021. Abstract Value Iteration for Hierarchical Reinforcement Learning. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, 1162–1170. PMLR.

Karpathy, A. 2015. REINFORCEjs: WaterWorld Demo. https://cs.stanford.edu/people/karpathy/reinforcejs/waterworld.html. Accessed: 2025-07-31.

Klissarov, M.; and Machado, M. C. 2023. Deep Laplacian-Based Options for Temporally-Extended Exploration. In *Proceedings of the 40th International Conference on Machine Learning*, 17198–17217. PMLR.

Krasowski, H.; Thumm, J.; Müller, M.; Schäfer, L.; Wang, X.; and Althoff, M. 2023. Provably Safe Reinforcement Learning: Conceptual Analysis, Survey, and Benchmarking. *Transactions on Machine Learning Research*.

Mendez, J. A.; van Seijen, H.; and Eaton, E. 2022. Modular Lifelong Reinforcement Learning via Neural Composition. In *Proceedings of the 10th International Conference on Learning Representations*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-Level Control through Deep Reinforcement Learning. *Nature*, 518(7540): 529–533.

Mohan, A.; Zhang, A.; and Lindauer, M. 2024. Structure in Deep Reinforcement Learning: A Survey and Open Problems. *Journal of Artificial Intelligence Research*, 79: 1167–1236.

Nachum, O.; Gu, S. S.; Lee, H.; and Levine, S. 2018. Data-Efficient Hierarchical Reinforcement Learning. *Advances in Neural Information Processing Systems*, 31.

Neary, C.; Xu, Z.; Wu, B.; and Topcu, U. 2021. Reward Machines for Cooperative Multi-Agent Reinforcement Learning. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems*, 934–942.

Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; and Dormann, N. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268): 1–8.

Shukla, Y.; Burman, T.; Kulkarni, A. N.; Wright, R.; Velasquez, A.; and Sinapov, J. 2024. Logical Specifications-Guided Dynamic Task Sampling for Reinforcement Learning Agents. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 532–540.

Skalse, J.; and Abate, A. 2023. On the Limitations of Markovian Rewards to Express Multi-Objective, Risk-Sensitive, and Modal Tasks. In *Proceedings of the 39th Conference on Uncertainty in Artificial Intelligence*, 1974–1984. PMLR.

Srivastava, S.; Zilberstein, S.; Immerman, N.; and Geffner, H. 2011. Qualitative Numeric Planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, 1010–1016.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. MuJoCo: A Physics Engine for Model-Based Control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. IEEE.

Unniyankal, H.; Belardinelli, F.; Ferrando, A.; and Malvone, V. 2023. RMLGym: A Formal Reward Machine Framework for Reinforcement Learning. In *WOA 2023: 24th Workshop From Objects to Agents*, CEUR Workshop Proceedings.

van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Watkins, C. J. C. H.; and Dayan, P. 1992. Q-Learning. *Machine Learning*, 8: 279–292.