

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

Kristina Levina<sup>1,2</sup>, Nikolaos Pappas<sup>1</sup>, Athanasios Karapantelakis<sup>2</sup>, Aneta Vulgarakis Feljan<sup>2</sup>,  
Jendrik Seipp<sup>1</sup>

<sup>1</sup>Linköping University, Linköping, Sweden

<sup>2</sup>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. This is particularly advantageous for complex non-Markovian tasks because agents with access to RMs can learn more efficiently from fewer samples. However, learning with RMs is ill-suited for long-horizon problems in which a set of subtasks can be executed in any order. In such cases, the amount of information to learn increases exponentially with the number of unordered subtasks. In this work, we address this limitation 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. Furthermore, we introduce a new compositional learning algorithm that leverages coupled RMs: *Q*-learning with coupled RMs (CoRM). Our experiments show that CoRM scales better than state-of-the-art RM algorithms for long-horizon problems with unordered subtasks.

## 1 Introduction

Reinforcement learning (RL) is the process of learning how to act through interaction with an environment and receiving reward-based feedback (Sutton and Barto 2018). The goal is to learn a policy that maximises the expected cumulative reward. Designing effective reward functions is critical but often difficult and time-consuming (Eschmann 2021).

To address this challenge, one approach is to specify the requirements for the agent’s behaviour in logic formulae, referred to as *specifications* (Jothimurugan 2023; Krasowski et al. 2023). Various specification languages and methods for compiling specifications into rewards have been developed (Li, Vasile, and Belta 2017; Jothimurugan, Alur, and Bastani 2019; Balakrishnan and Deshmukh 2019). However, a single specification language cannot satisfy the diversity of reward structures, and each requires a specialised *Q*-learning algorithm (Camacho et al. 2019).

### 1.1 Reward Machines

Another approach is to specify tasks in abstract graphs. Icarte et al. (2022) introduced Boolean reward machines (RMs)—automata with states associated with task progression. RMs

This is a preprint version of the paper.

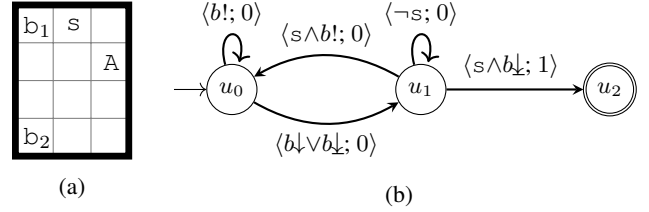


Figure 1: (a) Example Delivery instance with agent  $A$ , station  $s$ , and two boxes  $b_1$  and  $b_2$ . The thick black lines depict walls. (b) Numeric reward machine (RM) for the Delivery domain. Discrete numeric variable counts the number of uncollected boxes and is mapped to numeric feature  $b$ . Here,  $b \downarrow$  is true when a box is collected but some boxes remain uncollected,  $b \perp$  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.

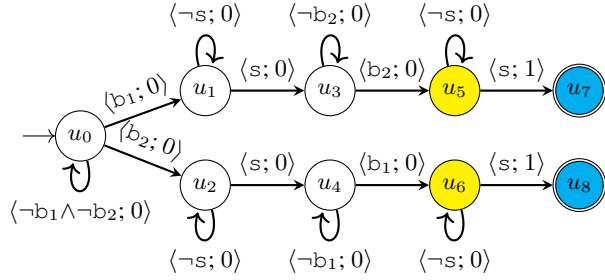
support non-Markovian tasks and outperform plain reward functions and specifications in many settings (Unniyankal et al. 2023). However, existing algorithms do not fully exploit the information contained in RMs, making complex long-horizon tasks challenging to solve.

Additional problems arise if tasks involve unordered subtasks. In such cases, the number of RM states grows exponentially with the number of subtasks because the agent must discover an optimal completion order. For example, collecting  $N$  boxes in any order requires tracking  $N!$  permutations of delivery sequences. As a result, the agent must learn from an exponentially large space, and modelling an RM with so many states becomes tedious and time-consuming.

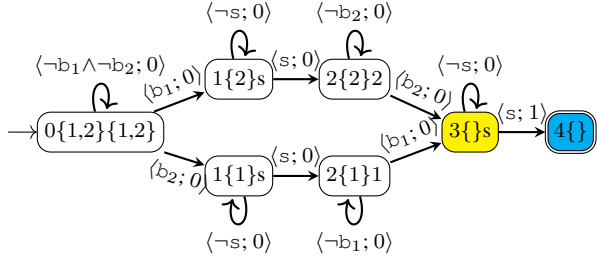
### 1.2 Reward Machines for Unordered Subtasks

We propose three generalisations of RMs that address the challenges mentioned above: *numeric*, *agenda*, and *coupled* RMs. Each RM is a stepping stone towards the next.

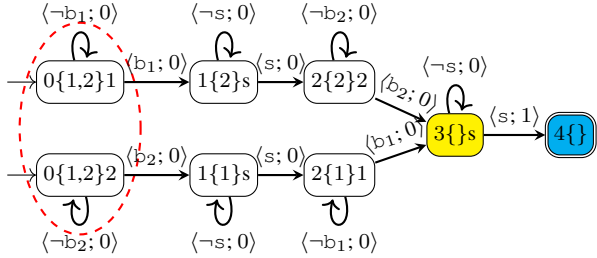
To put this in context, we introduce a running example based on a variant of the Delivery domain. Our Delivery domain is a deterministic finite-grid world with four-connected cells, some containing boxes. The agent should collect all boxes and deliver them to a designated station cell. Upon entering a box cell, the agent automatically picks the box up but can carry only one box at a time. The world contains no



(a) Boolean RM.



(b) Agenda RM.



State	$0\{1,2\}1$	$0\{1,2\}2$	$1\{1\}s$	$1\{2\}s$	$2\{1\}1$	$2\{2\}2$	$3\{s\}s$	$4\{\}$
$\eta$	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  $\eta$  values were found after the execution of compositional  $Q$ -learning.

Figure 2: RMs for the Delivery domain with two boxes shown in Figure 1a. Features  $b_1$  and  $b_2$  become true when the agent picks up box 1 and box 2, respectively. Symmetric states are shown in yellow and blue.

obstacles besides the surrounding walls, and its size is fixed. 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 single discrete numeric variable, which is then mapped to a numeric feature. For our running example, Figure 1b shows a numeric RM for delivering two boxes. Numeric variable  $w$  counts the number of uncollected boxes and has domain  $\mathcal{D}_w = \{0, 1, 2\}$ . However, the agent cannot directly use this RM for learning because RM states  $u_0$  and  $u_1$  represent different task progression steps and 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 their states grows exponentially with

the number of boxes. Furthermore, the agent needs to learn exponentially large information because it discovers both low- and high-level policies through trial and error.

To solve these problems, we associate each RM state  $u$  with an *agenda*,  $T$ , which is the set of subtasks to be completed in any order. To this end, we label  $u$  with  $\langle d, T, x \rangle$ , where  $d$  is the distance from the initial state in the RM to  $u$ ,  $T$  is agenda, and  $x$  is current objective. Importantly, states with the same label correspond to one  $Q$ -value function and are thus symmetric and can be reduced to one state. For example, in an agenda RM in Figure 2b, yellow state  $3\{s\}s$  corresponds to states  $u_5$  and  $u_6$  in the Boolean RM. State reduction speeds up learning because the agent traverses a lower number of paths in the RM.

Moreover, associating states with  $T$  allows us to split states by subtasks in  $T$ . The result is a *coupled* RM. For example, if  $T = \{\tau_1, \tau_2\}$ , the state with current objective  $x = T$  can be split to two states labelled  $d\{\tau_1, \tau_2\}\tau_1$  and  $d\{\tau_1, \tau_2\}\tau_2$ . The split states remain coupled for the agent to visit them concurrently. The agent can transition from any coupled state. Figure 2c shows a coupled RM for the running example.

To leverage coupled RMs, we introduce a new compositional learning algorithm:  $Q$ -learning with coupled RMs (CoRM). At the low level, the CoRM agent learns  $Q$ -values per subtask because each coupled RM state encodes a single objective  $x$ . To determine the completion order, we store the fewest number of steps  $\eta_u$  observed from each RM state  $u$  to a goal state. The high-level controller then selects the subtask associated with the coupled state that has minimum  $\eta_u$  to complete next. To ensure that low-level policies are learnt such that solutions are globally optimal, subtask completions receive a final reward that depends on the episode length relative to the best observed length. During exploration, the agent covers all possible completion orders.

In our running example, the CoRM 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  $\eta$  value is 10, which is lower than 12 of state  $0\{1,2\}1$ .

**Contributions.** First, we define numeric, agenda, and coupled RMs. Numeric RMs allow compact task descriptions, agenda RMs associate state with agendas, and coupled RMs allow for parallel and independent learning. Our second contribution is a compositional learning algorithm— $Q$ -learning with coupled RMs (CoRM). CoRM can learn an optimal policy for complex unordered non-Markovian tasks. While we focus on unordered subtasks, CoRM is general, and its application to sequential tasks can reduce the number of low-level policies to learn. For example, for a sequential task to reach  $a$ ,  $b$ , and  $a$  in order (RM states  $u_0$ ,  $u_1$ , and  $u_2$ ), the CoRM agent learns two reusable policies to reach  $a$  and  $b$ . In contrast, the Boolean-RM agent learns three policies associated with all RM states  $u_0$ ,  $u_1$ , 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 generally 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,  $p : S \times A \rightarrow S$  is a transition function,  $r : S \times A \times S \rightarrow \mathbb{R}$  is a reward function, and  $\gamma \in (0, 1)$  is a discount factor. A policy  $\pi(a|s)$  maps the state space  $S$  to the action space  $A$ .

In state  $s_t$ , the agent performs the action  $a_t$  according to policy  $\pi(a_t|s_t)$  and transitions to state  $s_{t+1}$  according to the probability distribution  $p(s_{t+1}|s_t, a_t)$ . After transitioning to  $s_{t+1}$ , the agent receives a reward  $r_{t+1}$ . The process repeats until episode termination or reaching a goal state. 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  $\pi$  thereafter. Formally,

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

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

Tabular  $Q$ -learning is an algorithm that estimates  $q^*(s, a)$  without requiring knowledge of the transition function  $p$  (Watkins and Dayan 1992). The  $Q$ -values are learnt through interaction with the environment and are guaranteed to converge to  $q^*(s, a)$  if the agent visits all environment states  $s \in S$  infinitely often and takes all possible actions from  $s$  infinitely many times. First, a  $Q$ -table is initialised randomly for each  $s \in S$  and  $a \in A$ . The  $Q(s, a)$  values are then updated at each iteration  $i$ :

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

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

Tabular  $Q$ -learning is effective in solving problems with discrete state and action spaces with relatively low dimensionality. To handle problems with continuous or high-dimensional state spaces, solutions based on deep learning (DL) have been developed (Sutton and Barto 2018). For example, double deep  $Q$ -learning (DDQN) (Van Hasselt, Guez, and Silver 2016) is one solution that reduces overestimation bias in deep  $Q$ -learning by decoupling action selection and evaluation. Noteworthy, DL solutions no longer guarantee convergence to a globally optimal policy (Mnih et al. 2015).

## 2.2 Boolean Reward Machines

An RM is a finite-state machine that encapsulates a reward structure of the environment. An RM specifies the reward the agent gets when it transitions between two abstract states in the RM. The formal definition follows.

**Definition 1** (Boolean reward machine). A Boolean RM is a tuple  $\mathcal{R}_{\mathcal{P}SA}^{\text{Bool}} = \langle U, u_0, F, \delta_u, \delta_r \rangle$  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,  $\delta_u$  is a state-transition function such that  $\delta_u : U \times 2^{\mathcal{P}} \rightarrow U \cup F$ , and  $\delta_r$  is a state-reward function such that  $\delta_r : U \rightarrow [S \times A \times S \rightarrow \mathbb{R}]$ .

At each time step, the RM receives the set of propositions that are true in the current environment state. The transition function then selects the abstract successor state, and the reward function assigns the corresponding reward.

Intuitively, an MDP with RMs (MDPRM) is an MDP defined over the cross-product  $S \times U$ . That is, an MDPRM is a tuple  $M_{\mathcal{R}} = \langle \tilde{S}, \tilde{s}_0, \tilde{A}, \tilde{p}, \tilde{r}, \tilde{\gamma} \rangle$ , where  $\tilde{S}$  is a set of states  $S \times U$ ,  $\tilde{s}_0 \in \tilde{S}$  is an initial state,  $\tilde{A} = A$  is a set of actions available to the agent,  $\tilde{p}(\langle s', u' \rangle | \langle s, u \rangle, a)$  is a state-transition function that now depends on both  $u$  and  $s$ ,  $\tilde{r}(\langle s, u \rangle, a, \langle s', u' \rangle) = \delta_r(u, L(s, a, s'))$  is a state-reward function with  $L(s, a, s')$  being a labeling function, and  $\tilde{\gamma} = \gamma$  is a discount factor. The task formulation with respect to MDPRM is Markovian. Consequently,  $Q$ -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).

Icarte et al. (2022) introduced algorithms for  $Q$ -learning for MDPRMs. One algorithm is  $Q$ -learning with RMs (QRM) over the cross-product  $S \times U$ . To exploit the reward structure encoded in an RM, Icarte et al. (2022) proposed QRM with counterfactual reasoning (CRM). In CRM, synthetic experiences are generated for each RM state per a single interaction with the environment.

## 3 Three New Types of Reward Machines

In this section, we formally introduce numeric, agenda, and coupled RMs and explain their relation to Boolean RMs.

### 3.1 Numeric Reward Machines

As shown in the Delivery task example (Figure 1), changes in numeric variables can be compactly modelled inside a numeric RM. In the example, the numeric variable counts the number of completed subtasks, but numeric variables can be used to model any discrete numeric information, such as the distance to a goal or resource usage. Numeric RMs cannot support continuous numeric variables because the number of states in the induced Boolean RM would be infinite. One might want to try learning a policy from a numeric RM directly, but numeric RMs do not provide any task decomposition, which is the essential property of RMs. For instance, in the numeric RM in Figure 1b, the agent can only be in two different RM states before solving the task, regardless of the number of boxes to deliver. To ensure a finite state space in the resulting Boolean RM, we assume the following.

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

We introduce numeric features by taking inspiration from 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 case, numeric variables give a sense of task progression to the agent, so we only allow *discrete* numeric variables. Each numeric variable  $w \in W$  is lower-bounded at some value  $w_\ell$

that indicates a goal for the agent. Each  $w$  is then mapped to a numeric feature  $p_w$  with domain  $\mathcal{D}_{p_w} = \{\downarrow, \perp, !\}$ . Value  $p_w\downarrow$  indicates that  $w$  has decreased between the current and previous observations in the environment but remains above  $w_\ell$ . Value  $p_w\perp$  indicates that  $w$  has reached  $w_\ell$ . Value  $p_w!$  indicates that  $w$  has neither decreased nor is equal to  $w_\ell$ .

**Theorem 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.* Considering that  $w$  is lower-bounded at  $w_\ell$  and cannot increase from there, there are five possible combinations of the relations between  $w_t$ ,  $w_{t-1}$ , and  $w_\ell$ : (1)  $w_{t-1} > w_t > w_\ell$ , (2)  $w_{t-1} > w_t = w_\ell$ , (3)  $w_{t-1} = w_t = w_\ell$ , (4)  $w_{t-1} = w_t > w_\ell$ , and (5)  $w_\ell < w_{t-1} < w_t$ . Case (1) is mapped to  $p_w\downarrow$ . Cases (2) and (3) are mapped to  $p_w\perp$ . Finally, cases (4) and (5) are mapped to  $p_w!$ . This concludes the proof.  $\square$

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

**Definition 2** (Numeric reward machine). *A numeric RM is a tuple  $\mathcal{R}_{\mathcal{P}SA}^{\text{num}} = \langle U, u_0, F, \delta_u, \delta_r \rangle$  given sets of Boolean and numeric features  $\mathcal{P} = \mathcal{P}_{\text{Bool}} \cup \mathcal{P}_{\text{num}}$ , environment states  $S$ , and actions  $A$ .  $U$ ,  $u_0$ ,  $F$ , and  $\delta_r$  are defined as in a Boolean RM, and  $\delta_u : U \times 2^{\mathcal{P}_{\text{Bool}}} \times 3^{\mathcal{P}_{\text{num}}} \rightarrow U \cup F$  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 environment events that the agent can detect. A numeric RM accepts both Boolean and numeric features. Boolean features are propositional symbols, as in the RM definition by Icarte et al. (2022).

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

This common assumption is necessary to align the agent’s traversal through the RM with the real conditions in the environment. We also assume that the agent has access to the values in  $\mathcal{D}_{p_w}$ . That is, the agent can test whether  $w$  has decreased between the previous and current observations in the environment and whether the goal is achieved.

Numeric RMs should be unrolled with respect to the domains of numeric variables  $\mathcal{D}_w$  because they cannot be used for learning directly. One reason is the lack of intermediate guidance. Another is that a single numeric-RM state can correspond to different parts of the agent’s trajectory. For example, in Figure 1b,  $N$  different parts of the agent’s trajectory corresponding to the number of undelivered boxes map to one numeric-RM state—state  $u_0$ . This would cause the agent to repeatedly alter the learnt information for state  $u_0$ , depending on the remaining subtasks.

Numeric RMs give us a compact specification of a task, reducing the modelling effort. In particular, tasks where subtasks need to be completed in any order and the agent needs to learn an optimal completion order are easily encoded in numeric RMs. Next, numeric RMs should be translated automatically to an RM that the agent can use for learning.

### 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.*

In addition, to enable the functionality of subtask completion in any order, we specifically assume the following:

**Assumption 4.** *Collections 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  $p_w\perp$  signals to transition to a next RM state.*

If the assumptions are met, a translation algorithm (Supplementary Material) takes a numeric RM  $\mathcal{R}_{\mathcal{P}SA}^{\text{num}}$  along with the numeric-variable domains  $\mathcal{D}_w$  as input and outputs a Boolean RM  $\mathcal{R}_{\mathcal{P}SA}^{\text{Bool}}$ . The translation algorithm unrolls the RM states with respect to  $\mathcal{D}_w$  following the given transitions in  $\mathcal{R}_{\mathcal{P}SA}^{\text{num}}$ . In the translated  $\mathcal{R}_{\mathcal{P}SA}^{\text{Bool}}$ , each state corresponds to a collection of subtasks to perform next for each possible completion order. As a result, the  $\mathcal{R}_{\mathcal{P}SA}^{\text{Bool}}$  representation is exponential in the number of subtasks. Moreover,  $\mathcal{R}_{\mathcal{P}SA}^{\text{Bool}}$  can contain symmetric states corresponding to the same  $Q$ -function.

However, reducing symmetric states is difficult without associating the RM states with the additional information about the uncompleted subtasks and completion orders.

Another problem with Boolean RMs is the partial use of CRM in domains where completed subtasks are removed from the environment. Once a subtask is completed, the agent will simulate incorrect experiences for RM states linked to it. For example, for the toy task depicted in Figure 1a, the agent will transition to  $u_1$  in the Boolean RM (Figure 2a) upon collecting box 1. The cell containing  $b_1$  will be cleared, and future visits to this cell will not make Boolean feature  $b_1$  true. Consequently, counterfactual experiences for states  $u_0$  and  $u_4$  will be erroneous. Therefore, counterfactual reasoning can only be used for future RM states with respect to  $u_1$ .

### 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  $\Xi : U \cup F \rightarrow \{\langle d, T, x \rangle \mid T \in \mathcal{P}(\mathcal{T})\}$ , where  $\mathcal{T}$  is the set of all subtasks to complete,  $T$  is a subset of the remaining subtasks to complete,  $d$  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 sometimes abbreviate the full label  $\langle d, \{\tau_1, \tau_2, \dots\}, x \rangle$  with  $d\{\tau_1, \tau_2, \dots\}x$  in our example RMs. We construct a power set  $\mathcal{P}(\mathcal{T})$  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 based on the information received from the next or future states. For example, see the  $Q$ -learning update in Eq. 2. Thus, tracking remaining subtasks, rather than completed ones, ensures proper backpropagation of information. Furthermore, we track the RM state depth so that symmetric states get the same labels and can be merged.

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

*Proof.* Assume, for the sake of contradiction, that there exist two states with the same labels  $\langle d, T, x \rangle$  but different task progression. The task progression is different if either (1) the remaining subtasks,  $T$ , are different; (2) the current objective,  $x$ , is different; or (3) the task progression stage is different for the same  $T$  and  $x$ . This can happen if the agent performs sequential subtasks. For example, in task  $a \rightarrow b \rightarrow a$ , the current objectives of  $a$  have same  $T = \{\}$  because there are no unordered subtasks. However, in this case, these RM states will be distinguished by  $d$ . Therefore, RM states with different task progression cannot have equal labels. Assume there are two states with different labels  $\langle d_1, T_1, x_1 \rangle$  and  $\langle d_2, T_2, x_2 \rangle$  but same task progression. However, the task progression is completely defined by  $d$ ,  $T$ , and  $x$ . Thus,  $d_1 = d_2$ ,  $T_1 = T_2$ , and  $x_1 = x_2$ . This completes the proof.  $\square$

We translate the given numeric RM to an agenda RM using a similar algorithm to that of numeric-to-Boolean-RM translation. The translation differs only in the labelling scheme. The agenda RM already provides computational advantages over plain Boolean RMs due to state reduction. However, more advantages are gained after splitting the agenda-RM states with  $x = T$ .

### 3.4 Coupled Reward Machines

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

**Definition 4** (Coupled reward machine). A coupled RM is an agenda RM where states labelled with  $\langle d, T, T \rangle$  are split into  $N$  states  $\langle d, T, \tau_1 \rangle, \langle d, T, \tau_2 \rangle, \dots, \langle d, T, \tau_N \rangle$ . The split states remain coupled so that the agent is at all of them concurrently.

Algorithmically, the agent is in all coupled RM states simultaneously. This is possible because the task component encoded in the RM is decoupled from the environment states that the agent directly observes. As a result, the agent can exist in several RM states at once while being in one environment state.

With this formulation, the agent can directly learn low-level policies how to complete subtasks. In particular, each coupled-RM state corresponds to only one subtask—one low-level policy. The agent has access to the RM label and hence can identify which subtask is to be performed. We explain how to perform  $Q$ -learning updates for coupled states in the next section.

## 4 $Q$ -learning with Coupled RMs

We introduce a new algorithm,  $Q$ -learning with coupled RMs (CoRM), where the agent learns two types of policies: low-level policies to complete each subtask and a high-level policy to decide the completion order.

First, we explain how **low-level policies** are learnt by the CoRM agent. The agent learns a low-level policy for each subtask in  $U' = \mathcal{T} \cup B$  in the context of the problem. Here,  $\mathcal{T}$  is the complete agenda, and  $B$  is the set of all Boolean objectives. Such low-level policies are possible to learn because each state in a coupled RM is associated with exactly one subtask. For example, state  $\langle d, T, \tau_k \rangle$ ,  $k = 1, 2, \dots, N$ , is associated with one subtask  $\tau_k \in T$ . Hence, the state space  $S \times U'$  in the corresponding MDPRM can be constructed using the set of all subtasks  $U'$  instead of the entire set of RM states  $U$ . Consequently, the amount of low-level information to learn increases only linearly with  $|U'|$ . In contrast, when defining the state space in the MDPRM as  $S \times U$ , the agent has to learn information that is exponential in the number of unordered subtasks.

Next, we discuss the learning process of low-level policies. Since the agent exists simultaneously in all coupled states  $\langle d, T, \tau_1 \rangle, \langle d, T, \tau_2 \rangle, \dots, \langle d, T, \tau_N \rangle$ , it can perform  $Q$ -learning updates in parallel. As long as the agent does not transition to the next RM state, the information about how to complete each subtask  $\tau_k$  is updated concurrently:

$$Q(\langle s, \tau_k \rangle, a) \leftarrow^\alpha r(\langle s, \tau_k \rangle, a, \langle s', \tau_k \rangle) + \gamma \max_{a' \in A} Q(\langle s', \tau_k \rangle, a'), \quad (3)$$

where  $\alpha$  is a learning rate,  $\gamma$  is a discount factor, and  $r(\langle s, \tau_k \rangle, a, \langle s', \tau_k \rangle)$  is an immediate reward.

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

$$Q(\langle s, \tau_\ell \rangle, a) \leftarrow^\alpha R(K), \quad (4)$$

where  $K > 0$  is the episode length and  $R(K)$  is a final reward given to the agent for performing subtask  $\tau_\ell$ . We store the experiences containing the information about the transitions between RM states in a temporary buffer. These experiences are used to perform learning updates after the episode terminates and the episode length  $K$  becomes known. In this way, the agent learns the low-level policies while considering their place in the task completion order and the context of their neighboring subtasks. Without this global context expressed in the episode-length signal, the low-level policies would be learnt in isolation and converge to locally optimal but globally sub-optimal behaviours. For example, in the continuous Water domain (Karpathy 2015), the agent would not learn to slow down before completing subtask  $\tau_\ell$  to be optimally set up for the next subtask.

Next, we define  $R(K)$  such that optimality guarantees in  $Q$ -learning are preserved. We start by considering an arbitrary subtask  $x$  that is part of some complete policy for the entire task. The discounted return for completing  $x$  with any locally sub-optimal policy should be higher than the one obtained

with any locally optimal policy if the former results in a globally optimal solution. Let local policy  $\pi_1^x$  (with length  $K_1^x$ ) be locally optimal but globally sub-optimal with total episode length  $K_1 > K^{\min}$ , where  $K^{\min}$  is the shortest episode length experienced thus far. Let local policy  $\pi_2^x$  (with length  $K_2^x$ ) be locally sub-optimal but globally optimal with  $K_2 = K^{\min}$ . By assumption,  $K_2^x > K_1^x$ .

If  $R(K)$  would not depend on  $K$  and would be equal to a constant reward,  $\tilde{R} \in \mathbb{R}$ , then, the agent would prefer  $\pi_1^x$  with return  $\tilde{Q}_1^x$  over  $\pi_2^x$  with return  $\tilde{Q}_2^x$  because  $\pi_1^x$  is locally optimal, i.e.  $\tilde{Q}_1^x > \tilde{Q}_2^x$ . Without loss of generality, we set  $\tilde{R} = 1$ . The discounted returns (Eq. 1) in this case are

$$\tilde{Q}_1^x = \gamma^{K_1^x-1} + \sum_{k=0}^{K_1^x-2} \gamma^{k_1} r_{k_1+1}^x, \quad (5)$$

$$\tilde{Q}_2^x = \gamma^{K_2^x-1} + \sum_{k_2=0}^{K_2^x-2} \gamma^{k_2} r_{k_2+1}^x, \quad (6)$$

where  $r_{k_i+1}^x$  is the immediate reward received at time step  $t + k_i + 1$  when following policy  $\pi_i^x$ . In the equations, we omit time index  $t$  for readability.

To prevent this undesired behaviour, joint optimisation with Eq. 4 is applied; the discounted returns are

$$Q_1^x = \gamma^{K_1^x-1} R(K_1) + \sum_{k_1=0}^{K_1^x-2} \gamma^{k_1} r_{k_1+1}^x, \quad (7)$$

$$Q_2^x = \gamma^{K_2^x-1} + \sum_{k_2=0}^{K_2^x-2} \gamma^{k_2} r_{k_2+1}^x = \tilde{Q}_2^x \quad (8)$$

Herein, we set  $R(K^{\min}) = 1$  for consistency with Eqs. 5 and 6. We need to choose  $R(K_1)$  such that  $Q_2^x > Q_1^x$ . First, we rewrite Eq. 7 using Eq. 5:  $Q_1^x = \tilde{Q}_1^x + \gamma^{K_1^x-1} R(K_1) - \gamma^{K_1^x-1}$ . The return difference (Eqs. 7 and 8) is then

$$Q_1^x - Q_2^x = \tilde{Q}_1^x - \tilde{Q}_2^x + \gamma^{K_1^x-1} R(K_1) - \gamma^{K_1^x-1} < 0.$$

Thus, to preserve global optimality, the reward  $R(K_1)$  must satisfy

$$R(K_1) < 1 - \gamma^{1-K_1^x} (\tilde{Q}_1^x - \tilde{Q}_2^x). \quad (9)$$

Substituting Eqs. 5 and 6 into the right-hand side of this inequality will help us find a suitable definition for  $R(K_1)$ . Before doing this, we find the maximum allowed return difference  $\tilde{Q}_1^x - \tilde{Q}_2^x$ , which will tighten the upper bound for  $R(K_1)$ . Assuming that all immediate rewards are bounded between  $r_{\min}$  and  $r_{\max}$ , we have  $\tilde{Q}_1^x - \tilde{Q}_2^x \leq \gamma^{K_1^x-1} - \gamma^{K_2^x-1} + \frac{1-\gamma^{K_1^x-1}}{1-\gamma} r_{\max} - \frac{1-\gamma^{K_2^x-1}}{1-\gamma} r_{\min}$ . Rearranging the terms,  $R(K_1) < \gamma^{K_2^x-K_1^x} - \frac{\gamma^{1-K_1^x}}{1-\gamma} (r_{\max} - r_{\min} - \gamma^{K_1^x-1} (r_{\max} - \gamma^{K_2^x-K_1^x} r_{\min}))$ . Herein,  $\gamma^{K_1^x-1} > 0$  and  $\frac{\gamma^{1-K_1^x}}{1-\gamma} > 0$  because  $\gamma \in (0, 1)$  by definition. Next,  $r_{\max} \geq \gamma^{K_2^x-K_1^x} r_{\min}$  because  $K_2^x > K_1^x$  by assumption and, hence,  $\gamma^{K_2^x-K_1^x} < 1$ . We can tighten the bound further and conclude that

$$R(K_1) < \gamma^{K_2^x-K_1^x} + \frac{\gamma^{1-K_1^x}}{1-\gamma} (r_{\max} - r_{\min}). \quad (10)$$

To simplify the reward expression, we relate episode lengths at different scales—for any subtask  $x$ , the difference between the current and best episode lengths ( $\Delta K_1 = K_1 - K^{\min}$ ) must be at least as large as the difference in steps needed to complete that subtask from the fewest number of steps ( $K_2^x - K_1^x$ ). This is true because initially we assumed that each local policy must be part of some complete policy. Formally,  $\Delta K_1 \geq K_2^x - K_1^x$ . Therefore,  $R(K_1) = \gamma^{\Delta K_1+1} + \frac{\gamma^{1-K_1^x}}{1-\gamma} (r_{\max} - r_{\min})$  is a valid reward function that satisfies Eq. 10 and preserves optimality guarantees in  $Q$ -learning.

However, when the episode length difference  $\Delta K_1$  grows large, the term  $\gamma^{\Delta K_1+1}$  becomes very small due to exponential decay (since  $\gamma < 1$ ). If  $r_{\max} - r_{\min}$  is also small, the total reward approaches zero. In practice, this may slow down learning. To address this issue, we introduce a hyperparameter  $\Delta K^w$  that limits the penalty for deviating from the best episode length  $K^{\min}$ . The final reward function for any subtask  $x$  is then

$$R(K) = \begin{cases} 1 & \text{if } \Delta K = 0, \\ \gamma^{\Delta K+1} + R(K^x) & \text{if } 0 < \Delta K < \Delta K^w, \\ \gamma^{\Delta K^w+1} + R(K^x) & \text{otherwise,} \end{cases} \quad (11)$$

where  $R(K^x) = \frac{\gamma^{1-K_1^x}}{1-\gamma} (r_{\max} - r_{\min})$  and  $\Delta K = K - K^{\min}$ . The window  $\Delta K^w$  ensures that the reward does not decrease below  $\gamma^{\Delta K^w+1} + R(K^x)$  even if the episode length difference  $\Delta K$  increases further. It should be chosen large enough for the problem at hand to not break the optimality guarantees. The possible values of  $\Delta K^w$  are  $\{-1, 0\} \cup \mathbb{N}$ . The optimal solution corresponds to  $K^{\min}$  and thus to the highest reward of 1. If  $K \neq K^{\min}$ , the reward decreases from 1 to  $\gamma^{\Delta K^w+1}$  with increasing  $\Delta K$ . In this way, the agent learns to complete subtasks using the fewest number of steps. When  $\Delta K^w = -1$ , the agent receives a reward of 1 upon transitioning to the next RM state regardless of the episode length  $K$ , which is useful if the completion order does not affect the local policies.

From subtasks  $\tau_k, k = 1, 2, \dots, N$ , associated with coupled RM states  $\langle d, T, \tau_k \rangle$ , a **high-level policy** selects one subtask  $\tau_\ell$  to complete next. The decision is based on  $\eta_{\langle d, T, \tau_k \rangle}$ , the minimum number of steps from  $\langle d, T, \tau_k \rangle$  to a goal state experienced thus far. For each RM state  $u$ , we store  $\eta_u$  in an additional lookup table. Once the agent converges to an optimal solution, the stored  $\eta_u$  values correspond to a trajectory with total length  $K^{\min}$ .

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 (Eq. 4) are only used for learning when the agent exploits, i.e. when it selects the best transition based on the lowest  $\eta_{\langle d, T, \tau_k \rangle}, k = 1, 2, \dots, N$ . Therefore, the low-level policies are learnt to align with the best high-level policy.

We integrate CoRM with DDQN by adding a separate replay buffer that stores valuable experiences corresponding to the transitions between RM states. These experiences are used for training the neural network after each episode



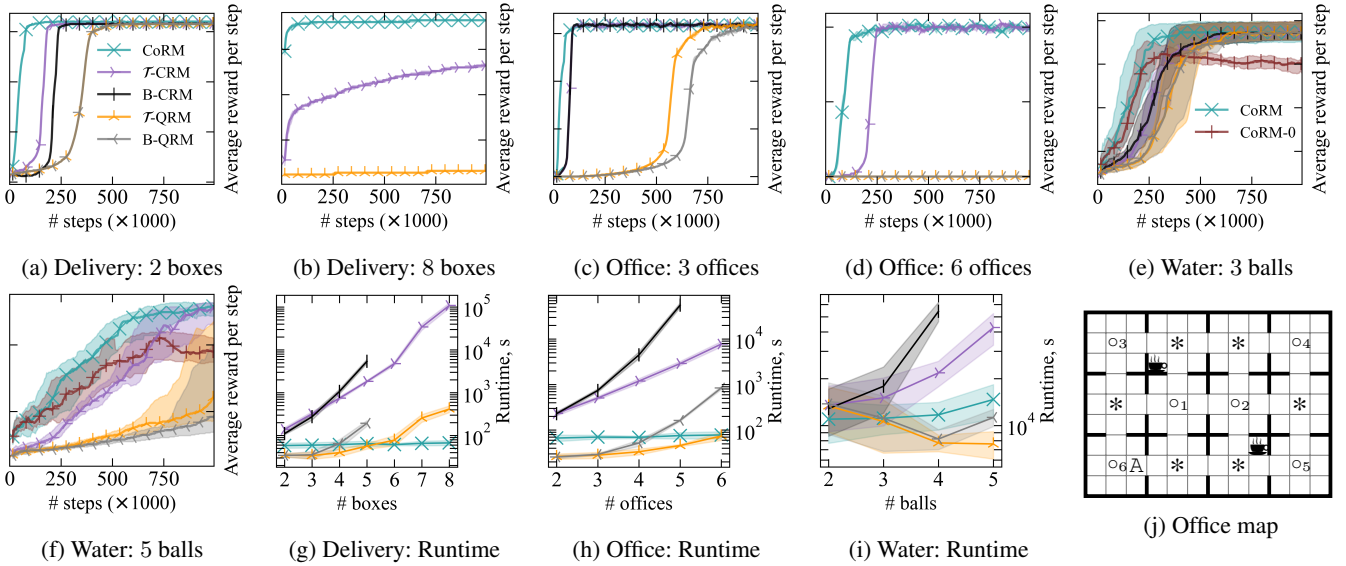


Figure 3: Results of CoRM (ours) and  $Q$ -learning with and without counterfactual reasoning (CRM and QRM, respectively) with Boolean RMs (B-CRM and B-QRM) and our agenda RMs ( $\mathcal{T}$ -CRM and  $\mathcal{T}$ -QRM) for the Delivery (a,b), Office (c,d), and Water (e,f) domains. CoRM-0 stands for CoRM without the joint optimisation from Eq. 11. (g–i) Time (in seconds) to run  $10^6$  steps in relation to the number of objectives. (j) Office environment used for experiments in (c,d) with agent A, offices  $o_i$ ,  $i = 1, 2, \dots, 6$ , decorations marked by stars, and coffee machines.

completion, following the same procedure described above.

## 5 Experiments

Our implementation builds on the code by Icarte et al. (2022) and Stable-Baselines3 (Raffin et al. 2021). For our experiments, we use two tabular domains, Delivery (our running example) and Office (Icarte et al. 2022), and the continuous-space Water domain (Karpathy 2015). We run our experiments on Intel Xeon Gold 6130 CPUs, using a single core and at most 16 and 32 GiB of memory for the tabular and continuous domains, respectively. Each algorithm run is limited to  $10^6$  steps and 40 hours. We report the median performance across 10 seeds, with the shaded regions on the plots representing the 25th and 75th percentiles.

In Boolean and agenda RMs, the agent gets a reward of 1 only after completing the entire task and a reward of 0 otherwise. We choose this reward because supplying nonzero intermediate rewards can affect optimality guarantees of the learned policies (Cui and Yu 2023). For denser reward feedback, potential-based reward shaping can be performed on top while preserving optimality guarantees (Ng, Harada, and Russell 1999). Finally, we calculate the average reward per step.

For tabular  $Q$ -learning, we use the learning rate  $\alpha = 10^{-5}$ , action-space-exploration parameter  $\epsilon = 0.1$ , discounting rate  $\gamma = 0.9$ , and RM-exploration parameter  $\xi = 0.1$ . All  $Q$ -values are initialised to 1. For DDQN,  $\alpha = 10^{-5}$ ;  $\epsilon$  decays linearly from 1 to 0.1, decreasing by 0.01 per episode;  $\gamma = 0.9$ , soft update coefficient  $\tau = 0.1$ ; the training frequency is 1, and the gradient step is 1. The neural network has three layers with 512 hidden units each. This neural network size is needed for CRM to learn a single  $Q$ -function for all RM

states. The buffer and batch sizes are 50K and 32 for QRM. They are both scaled up by factors of  $|U'|$  and  $|U|$  for CoRM and CRM, respectively. We choose  $\xi = 0.1$  for all domains. In tabular domains, subtasks are independent of each other, and hence, we select an episode-length window  $\Delta K^w$  of  $-1$ . For the Water domain, we tested the  $\Delta K^w$  range from 0 to 100 with increments of 10 and selected  $\Delta K^w = 80$  for its smoothest convergence. All other parameters use the default setting in Stable-Baselines3 or the code by Icarte et al. (2022).

### 5.1 Delivery Domain

The Delivery instances are seven  $10 \times 10$  grids with 2–8 randomly placed boxes. The agent’s initial position is fixed throughout the experiments. Figures 3a and 3b show the results of executing CoRM and state-of-the-art RM algorithms with Boolean and agenda RMs for solving the Delivery problem with two and eight boxes, respectively. CoRM converges fastest, followed by the CRM-based methods. Notably, CoRM’s performance does not deteriorate after the removal of completed objectives. In contrast, as the CRM-based methods simulate experiences only for reachable RM states, fewer experiences are simulated as the agent traverses the RM. Convergence is faster for CRM with the agenda RMs than for CRM with the Boolean RMs because more states can be identified as reachable due to the more informative labelling scheme. Importantly, the large size of Boolean RMs for the tasks of delivering more than five packages leads to running out of memory. We obtained similar results in experiments with different numbers of boxes.

## 5.2 Office Domain

The Office domain (Icarte et al. 2022), shown in Figure 3j, features a grid world with walls, doors, and decorations to be avoided by the agent. We provide the agent with five different progressively more complex tasks: deliver coffee to 2–6 offices. The completion order is up to the agent. The Office domain is more complex because (1) the agent fails the episode upon stumbling on a decoration and (2) walls block paths. Figures 3c and 3d show the results for the tasks of delivering coffee to three and six offices, respectively. The convergence of QRM-based methods is slowest. Similar to the Delivery results, CoRM outperforms CRM-based methods. Differently, CRM with both the agenda and Boolean RMs converge at a similar rate because objectives (offices and coffee machines) are not removed from the map upon completion. As a result, all RM states remain reachable for simulating experiences from any RM state. We obtained similar results for different numbers of offices.

## 5.3 Water Domain and Ablation Study

The Water domain (Karpathy 2015) is a two-dimensional area populated with colored balls that move at fixed speeds and rebound off the walls. The agent, represented by a white ball, can adjust its velocity along four cardinal directions. This domain is complicated for the agent because the order in which balls are touched affects the low-level policies to touch each ball. For example, the agent may need to learn to slow down before touching a ball to be optimally set up for moving to the next ball. As both positions and velocities are continuous, we use DDQN for this domain. We randomly generate three maps with six pairs of differently coloured balls. The agent is instructed to complete four progressively difficult tasks to touch 2–5 balls of specified colours in any order without touching other balls. The results are averaged over the three maps. Figures 3e and 3f show the results for the tasks of touching three and five balls, respectively. CoRM again converges faster than the CRM-based methods followed by the QRM methods. We observe similar results for other task sizes.

We conduct a small ablation study for the Water domain to show that the given reward (Eq. 11) in CoRM results in converging to an optimal policy, only slightly affecting the convergence speed. To this end, we remove the joint optimisation from Eq. 11 and give the agent a reward of 1 upon each subtask completion. The results in Figures 3e and 3f show that the agent converges to sub-optimal policies without the joint optimisation, as expected. Judging by the plots, the convergence speed is unaffected within the confidence bounds.

## 5.4 Runtimes

We compare the runtimes of the tested algorithms in Figures 3g–3i. Noteworthy, the QRM runtimes scale exponentially with the task size in tabular domains due to the increasing size of the  $Q$ -tables. However, in continuous domains, the QRM runtimes do not change significantly with the task size because the neural-network size does not depend on the task size. The plots demonstrate that the runtime scales linearly

with the number of RM states in CoRM because the number of simulated experiences grows linearly with the number of subtasks. In contrast, we observe an exponential blowup in CRM runtimes, where the number of simulated experiences is proportional to the number of RM states. Notably, the exponential blowup in the number of Boolean RM states lets the runtime of B-CRM scale exponentially in the task size, often leading to hitting the time limit. While the CRM-based methods are therefore unusable for large tasks, CoRM is able to handle them efficiently.

## 6 Related Work

The concept of learning a policy for each subtask is well-established in the literature (Mohan, Zhang, and Lindauer 2024). Notable works include options (Sutton and Barto 2018), policy sketches (Andreas, Klein, and Levine 2017), taskable RL (Illanes et al. 2020), hierarchical policies (Drexler, Seipp, and Geffner 2023), and other hierarchical and compositional approaches (Hu et al. 2023). In these frameworks, the order of subtask completion is either predefined or learnt by the agent. However, long-horizon tasks with unordered subtasks present significant scalability challenges for these approaches, particularly as the number of subtasks increases (Mendez, van Seijen, and Eaton 2022).

Some approaches address scalability by relaxing optimality guarantees. For example, the DiRL algorithm (Jothimurugan et al. 2021) translates tasks specified in SPECTRL (SPECifying Tasks for RL) into abstract graphs. A high-level policy for selecting transitions in the graph is determined using Dijkstra’s algorithm with costs based on the transition success probabilities between different state-space regions. As a result, the learnt completion order of subtasks can be sub-optimal.

Inspired by DiRL, we use the reward structure provided in an RM for high-level decision making in CoRM. As the agent freely traverses the RM during learning, CoRM converges to an optimal solution in deterministic environments.

In another related work, Shukla et al. (2024) specify a task with SPECTRL and then convert it to an abstract graph, similar to DiRL. In contrast to DiRL, the authors use sampling to bias learning towards more promising transitions in the graph, mitigating the scalability issue. The method is called Logical Specifications-guided Dynamic Task Sampling (LSTS).

Importantly, both DiRL and LSTS learn only one low-level policy at a time. CoRM learns a low-level policy for each subtask in coupled RM states in parallel. The closest setup would be a multi-agent environment where each agent is assigned a subtask. However, multi-agent RL has its own challenges (Canese et al. 2021).

In comparison to  $Q$ -learning with state-of-the-art RMs, CoRM has reduced state space, improved sample efficiency, and faster convergence. Furthermore, CRM and hierarchical  $Q$ -learning with RMs introduced by Icarte et al. (2022) do not fully exploit the reward structure specified in RMs. In contrast, CoRM exploits all the available high-level reward structure. Both CRM and CoRM find optimal completion orders.



## 7 Conclusions and Future Work

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 algorithm for  $Q$ -learning with coupled RMs (CoRM) that shows computational advantages over state-of-the-art RM algorithms. In particular, CoRM scales well for long-horizon tasks with unordered subtasks.

In future work, we plan to investigate the generalisation capabilities of coupled RMs across different domains. To extend the CoRM algorithm to stochastic environments, we aim to base its high-level policy on the *expected* number of steps to reach a goal state, rather than the shortest observed path. Finally, CoRM could be integrated with deep-deterministic-policy-gradient (Lillicrap et al. 2015) and soft-actor-critic (Stone, Talbert, and Eberle 2022) agents for control tasks.

## 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 Multi-task Reinforcement Learning with Policy Sketches. In *International conference on machine learning*, 166–175. PMLR.
- Balakrishnan, A.; and Deshmukh, J. V. 2019. Structured Reward Shaping using Signal Temporal Logic Specifications. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3481–3486. IEEE.
- 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 *IJCAI*, volume 19, 6065–6073.
- Canese, L.; Cardarilli, G. C.; Di Nunzio, L.; Fazzolari, R.; Giardino, D.; Re, M.; and Spanò, S. 2021. Multi-agent Reinforcement Learning: A Review of Challenges and Applications. *Applied Sciences*, 11(11): 4948.
- Cui, W.; and Yu, W. 2023. Reinforcement Learning with Non-cumulative Objective. *IEEE Transactions on Machine Learning in Communications and Networking*, 1: 124–137.
- 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.
- Eschmann, J. 2021. Reward Function Design in Reinforcement Learning. *Reinforcement Learning Algorithms: Analysis and Applications*, 25–33.
- Hu, W.; Wang, H.; He, M.; and Wang, N. 2023. Uncertainty-aware Hierarchical Reinforcement Learning for Long-horizon Tasks. *Applied Intelligence*, 53: 28555–28569.
- 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. 2023. *Specification-Guided Reinforcement Learning*. Ph.D. thesis, University of Pennsylvania.
- Jothimurugan, K.; Alur, R.; and Bastani, O. 2019. A Composable Specification Language for Reinforcement Learning Tasks. *Advances in Neural Information Processing Systems*, 32.
- 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.
- Karpathy, A. 2015. REINFORCEjs: WaterWorld demo. <https://cs.stanford.edu/people/karpathy/reinforcejs/waterworld.html>. Accessed: 2025-10-31.
- 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*.
- Li, X.; Vasile, C.-I.; and Belta, C. 2017. Reinforcement Learning with Temporal Logic Rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3834–3839. IEEE.
- Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous Control with Deep Reinforcement Learning. *arXiv preprint arXiv:1509.02971*.
- Mendez, J. A.; van Seijen, H.; and Eaton, E. 2022. Modular Lifelong Reinforcement Learning via Neural Composition. *arXiv preprint arXiv:2207.00429*.
- 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.
- Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping. In *ICML*, volume 99, 278–287. Citeseer.
- 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.
- 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.
- Stone, G. B.; Talbert, D. A.; and Eberle, W. 2022. A Survey of Scalable Reinforcement Learning. *Int. J. Intell. Comput. Res.*, 13: 1118–1124.
- Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT press.
- 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*.
- 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.; and Dayan, P. 1992. Q-learning. *Machine learning*, 8: 279–292.