Numeric 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 inform reinforcement learning agents about the reward structure of the environment and often drastically speed up the learning process. However, reward machines only accept Boolean features such as `robot-reached-gold`. Consequently, many inherently numeric tasks cannot profit from the guidance offered by reward machines. To address this gap, we aim to extend reward machines with numeric features such as `distance-to-gold`. For this, we present two types of reward machines: *numeric–Boolean* and *numeric*. In a *numeric–Boolean* reward machine, `distance-to-gold` is emulated by two Boolean features `distance-to-gold-decreased` and `robot-reached-gold`. In a *numeric* reward machine, `distance-to-gold` is used directly alongside the Boolean feature `robot-reached-gold`. We compare our new approaches to a baseline reward machine in the Craft domain, where the numeric feature is the agent-to-target distance. We use cross-product Q-learning, Q-learning with counter-factual experiences, and the options framework for learning. Our experimental results show that our new approaches significantly outperform the baseline approach. Extending reward machines with numeric features opens up new possibilities of using reward machines in inherently numeric tasks.

1 Introduction

Reinforcement learning (RL) is the process of learning an optimal policy by interacting with an environment (Sutton and Barto 2018). The agent receives rewards from the environment based on its actions. The goal is to learn a policy that maximises the expected accumulated reward over time. The reward function is crucial for the agent to learn an optimal policy. However, designing it is often challenging and time-consuming.

Several methods have been developed to address this challenge. For example, instead of defining a full reward function, one can specify requirements for the agent's behaviour in logic formulas, called *specifications* (Jothimurugan 2023; Krasowski et al. 2023). The literature describes various specification languages and approaches for compiling specifications to rewards. Jothimurugan et al. (2021) designed a compositional learning approach, called DIRL, for translating the supplied specifications into an abstract graph on which high-level planning is then performed for model-based RL. Furthermore, Illanes et al. (2020) introduced high-level symbolic action models. Then, they used automated planning for guiding hierarchical RL (HRL) techniques. Another approach for encoding the required RL agent behaviour is to use temporal logic, for example, truncated linear temporal logic (Li, Vasile, and Belta 2017) and signal temporal logic (STL) (Balakrishnan and Deshmukh 2019).

In the aforementioned studies, the use of specifications for guiding the RL agent has shown a remarkable performance in comparison with entirely manual reward design. In addition, techniques such as automatic reward shaping (Ng, Harada, and Russell 1999) are used to reshape the reward function such that an optimal policy is found faster. However, these approaches cannot handle non-Markovian rewards (Skalse and Abate 2023) that naturally arise in sparse-reward or partially observable environments (Kazemi et al. 2022).

To handle non-Markovian rewards, automaton-based approaches have been developed for high-level guidance of the RL agent. For example, Jothimurugan, Alur, and Bastani (2019) designed a specification-to-reward compiler, called SPECRL, with a task monitor, which is an automaton that keeps track of completed tasks. Later, Icarte et al. (2022) introduced reward machines (RMs) that are automata representing the environment on a high level. RMs reflect sub-goals that the agent is supposed to achieve on the way to the main goal. RL algorithms with access to an RM can simulate experiences at all RM states using only a single interaction with the environment. This can drastically speed up learning by making it more sample-efficient. RMs not only handle non-Markovian tasks but are also advantageous over plain reward functions and STL-specified reward functions (Unniyankal et al. 2023).

However, RMs accept only Boolean features and thus cannot be used in inherently numeric tasks. Our work intends to address this gap by extending RMs with numeric features. We build upon the work by Icarte et al. (2022) and use their code for our experiments (Icarte 2021). We introduce two types of RMs: *numeric–Boolean* and *numeric*. To illustrate them with an example, consider a numeric feature d that represents the distance to the target. In a *numeric–Boolean* RM, d is emulated by two

Boolean features $\downarrow d$ and d=0. The former signals whether d decreases, and the latter signals whether the RL agent has reached the target. The agent receives a positive reward when one of these features becomes true. In a *numeric* RM, d is used directly together with the Boolean feature d=0, and the agent is rewarded with $-d$ after each action. We compare our proposed RMs with the original *Boolean* RM developed by Icarte et al. (2022).

In summary, our main contribution is the introduction of numeric features into RMs. We show empirically using the Craft domain (Andreas, Klein, and Levine 2017) that *numeric–Boolean*-RM and *numeric*-RM-based methods outperform *Boolean*-RM-based methods in terms of learning speed. RL agents can now leverage RMs in inherently numeric tasks, such as energy optimisation (Gao et al. 2022). This expansion of the RM applicability can further promote research in RMs.

2 Background

We begin by providing background information on RL and RMs. For more details on the topics, please 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) (Sutton and Barto 2018). An MDP is a 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 available to the agent, $p : S \times A \to S$ is a transition function, $r : S \times A \times S \to \mathbb{R}$ is a reward function, and $\gamma \in (0, 1]$ is a discount factor. A policy $\pi(a|s)$ is a mapping from the state space S to the action space A, that is, $\pi : S \to A$. A trajectory τ is a sequence of states, actions, and rewards $\langle s_0, a_0, r_1, s_1, a_1, ..., r_T, s_T\rangle$ that describes the agent's interaction with the environment up to a given time step T (Skalse and Abate 2023). The trajectory return function is

$$g(\tau) = \sum_{t=0}^{T-1} \gamma^t r_{t+1}. \tag{1}$$

At state s_t, the agent executes action a_t according to the probability distribution $\pi(a_t|s_t)$. The result of action a_t is the transition 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 reward r_{t+1}. The process repeats either until episode termination or until reaching a terminal state. The objective is to choose a trajectory τ such that the trajectory return $g(\tau)$ is maximised. For this, the agent needs to discover an optimal policy π^* from interactions with the environment. The optimal policy $\pi^*(a|s_t = s)$ for all $s \in S$ corresponds to the maximum expected return $\mathbb{E}_{\pi^*}[\sum_{k=0}^{T-1} \gamma^k r_{t+k+1}|s_t = s]$.

The Q-function $q^\pi(s, a)$ in RL quantifies the expected return the agent can achieve by taking a specific action a in a given state s and following a policy π thereafter. Formally,

$$q^\pi(s, a) = \mathbb{E}_\pi[\sum_{k=0}^{T-1} \gamma^k r_{t+k+1}|s_t = s, a_t = a]. \tag{2}$$

For an optimal policy π^*, $q^* = q^{\pi^*}$. Every optimal policy π^* satisfies the Bellman optimality equation

$$q^*(s, a) = \sum_{s'\in S} p(s'|s, a)(r(s, a, s') + \gamma \max_{a'\in A} q^*(s', a')) \tag{3}$$

for all $a \in A$ and $s \in S$. The policy is optimal if the agent selects an action a greedily with respect to $q^*(s, a)$. Therefore, the RL task could be solved via solving Eq. 3 if we knew the transition probability p for every state of the environment.

Tabular Q-learning is an algorithm that does not need knowledge of the transition probability p for estimating optimal $q^*(s, a)$ for all $a \in A$ and $s \in S$ (Watkins and Dayan 1992). The algorithm estimates Q-values from interacting with the environment. The estimated values $Q(s, a)$ 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:

$$Q_{i+1}(s, a) \xleftarrow{\alpha} r(s, a, s') + \gamma \max_{a'} Q_i(s', a'), \tag{4}$$

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

Tabular Q-learning is a classic RL algorithm known for its simplicity and effectiveness in solving problems with discrete state and action spaces with relatively low dimensionality. This is also the case for the problem used in this work (see Sect. 3). To handle more complex problems with continuous or high-dimensional state spaces, deep-learning-based variants have been developed (Sutton and Barto 2018).

2.2 Reward Machines

An RM is a finite-state machine that encapsulates an abstract description of the environment. An RM specifies the reward the agent gets when it transitions between two abstract states in the RM (possibly via a self-loop). Owing to their state-based design, RMs allow to encode non-Markovian rewards.

Formally, an RM is a tuple $R_{\mathcal{P}SA} = \langle U, u_0, F, \delta_u, \delta_r\rangle$ 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 $\delta_r : U \to [S \times A \times S \to \mathbb{R}]$.

At each time step t, given an environment experience $\langle s, a, s'\rangle$, a labeling function $L : S \times A \times S \to 2^{\mathcal{P}}$ assigns truth values to propositions. At each step, the set of propositions that are true in the current environment state is sent to the RM. The state-transition function then decides which abstract successor state is reached. The reward function to be used for the underlying MDP is chosen by the state-reward function.

If it is sufficient to return a real-valued reward rather than a reward function, the RM definition can be simplified. A simple RM is a tuple $R_{\mathcal{P}} = \langle U, u_0, F, \delta_u, \delta_r\rangle$ given a set of propositional symbols $\mathcal{P}$. In the tuple, U, u_0, F, and δ_u are defined similarly as in $R_{\mathcal{P}SA}$. However, the state-reward function is now $\delta_r : U \times 2^{\mathcal{P}} \to \mathbb{R}$. In this work, we focus on simple RMs.

Intuitively, an MDP with an RM (MDPRM) is an MDP defined over the cross-product $S \times U$. That is, an MDPRM is a tuple $M_R = \langle \tilde{S}, \tilde{s}_0, \tilde{A}, \tilde{p}, \tilde{r}, \tilde{\gamma} \rangle$, 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, and $\tilde{\gamma} = \gamma$ is a discount factor. The task formulation with respect to MDPRM is Markovian. Consequently, tabular Q-learning (among other RL methods) can be used to solve it. Optimal-solution guarantees of an RL algorithm for MDPRMs are the same as for regular MDPs (Icarte et al. 2022).

In their work, Icarte et al. (2022) provided three algorithms for Q-learning for an MDPRM. The first algorithm is a baseline for solving MDPRM using tabular Q-learning (QRM). It keeps track of the current RM state u and learns Q-values $Q(\langle s, u \rangle, a)$ over the cross-product of the environment and RM states. As this algorithm does not use the reward structure encoded in an RM, Icarte et al. also proposed Q-learning with counterfactual experiences (CRM) and hierarchical RL for RMs (HRM).

In CRM, learning happens in the same way as in the baseline algorithm, but counterfactual reasoning is used to generate multiple experiences for a single environment interaction. Specifically, the reward acquisition at any abstract state $\overline{u} \in U$ for a fixed environment state s is decoupled from the direct interactions with the environment. Therefore, the agent with experience $\langle s, a, s' \rangle$ can simulate synthetic experiences for each RM state without visiting it. These synthetic experiences are called *counterfactual* experiences. Their use in learning makes the algorithms more sample-efficient.

In HRM, the problem is decomposed into sub-problems called *options*, each representing transitions between RM states. Each option $Y(u, u_t)$ corresponds to the transition in the RM between states u and u_t. The agent learns two policies. A high-level policy learns to select among the available options to collect the highest reward. It does so by learning Q-values $Q^H(s, u, u_t)$ that select a state u_t to reach from a given state u. A low-level policy based on $Q_{u,u_t}(s, a)$ learns to act within an option $Y(u, u_t)$. Noteworthy, synthetic experiences are used for learning $Q_{u,u_t}(s, a)$ for each option.

3 Numeric Reward Machines

In this section, we present three RMs that progress from Boolean features to numeric features. The first RM, *Boolean*, is the original RM that uses Boolean features. The second RM, *numeric–Boolean*, emulates numeric features using Boolean features. The third RM, *numeric*, uses numeric features directly. Finally, we analyse and compare the theoretical guarantees of Q-learning-based methods with the three introduced RMs with regards to the path lengths

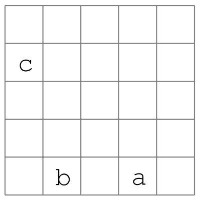


Figure 1: Example grid map inspired by the Craft domain (Andreas, Klein, and Levine 2017). It contains objects a, b, and c.

$$\begin{tikzcd}
u_a \arrow[loop above, "\langle \neg\, \mathtt{a}; 0\rangle"] \arrow[r, "\langle \mathtt{a}; 0\rangle"] & u_b \arrow[loop above, "\langle \neg\, \mathtt{b}; 0\rangle"] \arrow[r, "\langle \mathtt{b}; 0\rangle"] & u_c \arrow[loop above, "\langle \neg\, \mathtt{c}; 0\rangle"] \arrow[r, "\langle \mathtt{c}; 1\rangle"] & u
\end{tikzcd}$$

Figure 2: *Boolean* reward machine with Boolean features a, b, and c for a sequential task a−b−c.

they find in the Craft domain. We begin by describing the Craft domain (Andreas, Klein, and Levine 2017), which serves as a running example and is the basis for our experimental comparisons.

3.1 Environment

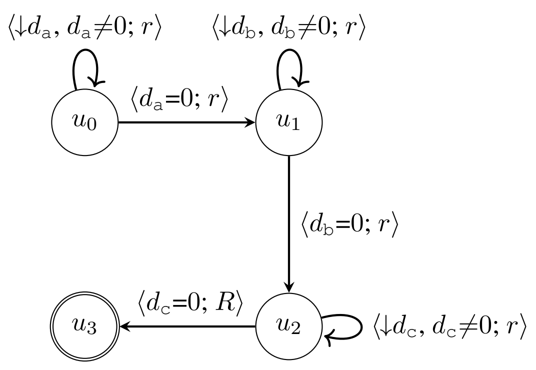
We use environments inspired by the Craft domain (Andreas, Klein, and Levine 2017). It is a finite-grid world with objects of different types. The RL agent is instructed to visit some objects. The world contains no obstacles besides the surrounding walls, and its dimensions are fixed. We show an example environment with three objects in Fig. 1.

3.2 *Boolean* Reward Machines

The concept of *Boolean* RMs was introduced by Icarte et al. (2022). In *Boolean* RMs, the features are propositions $p \in \mathcal{P}$ that can become true in the environment given experience $\langle s, a, s' \rangle$. Figure 2 shows an example of such a *Boolean* RM for a sequential task a−b−c, where the agent has to visit three objects of types a, b, and c, in order. The agent receives reward r after taking a transition labeled with $\langle \{P\}; r \rangle$ in the depicted RM, where a set of propositions $P \subseteq \mathcal{P}$ is currently true in the environment.

To address the common problem of reward sparsity, one typically uses potential-based reward shaping. Ng, Harada, and Russell (1999) proved that, for any given MDP $\langle S, s_0, A, r, p, \gamma \rangle$ and a potential function $\Phi : S \to \mathbb{R}$, the reward function r can be changed to $r'(s, a, s') = r(s, a, s') + \gamma\Phi(s') - \Phi(s)$ without changing the set of optimal policies. Like Icarte et al. (2022), we compute the potential function Φ using value iteration over RM states, treating the RM as an MDP.

The idea of automatic reward shaping is to change the reward function such that the policy is easier to learn. To this end, intermediate rewards are given to the agent as it approaches the target. However, these intermediate rewards do not carry any interpretable meaning. They are



$\langle{\downarrow}d_{\mathtt{a}}, d_{\mathtt{a}}{\neq}0; r\rangle$
$\langle{\downarrow}d_{\mathtt{b}}, d_{\mathtt{b}}{\neq}0; r\rangle$
u_0
$\langle d_{\mathtt{a}}{=}0; r\rangle$
u_1
$\langle d_{\mathtt{b}}{=}0; r\rangle$
$\langle d_{\mathtt{c}}{=}0; R\rangle$
u_3
u_2
$\langle{\downarrow}d_{\mathtt{c}}, d_{\mathtt{c}}{\neq}0; r\rangle$

Figure 3: *Numeric–Boolean* reward machine with Boolean features ${\downarrow}d_{\mathtt{a}}, {\downarrow}d_{\mathtt{b}}, {\downarrow}d_{\mathtt{c}}$, $d_{\mathtt{a}}{=}0$, $d_{\mathtt{b}}{=}0$, and $d_{\mathtt{c}}{=}0$ for a sequential task `a-b-c`.

only tweaked for finding the optimal policy fast. To obtain a method that preserves the meaning of rewards, we propose to use numeric features to shape the reward function.

3.3 *Numeric–Boolean* Reward Machines

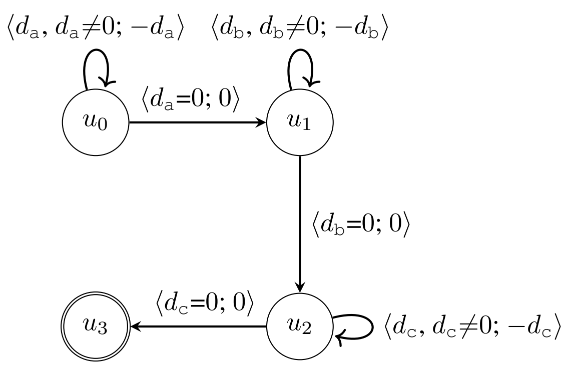
The first type of RM that we introduce in this paper is a *numeric–Boolean* RM. This concept builds directly on *Boolean* RMs. Instead of automatic reward shaping, we introduce a numeric feature $d_{\mathtt{a}}$—distance between agent `A` and target `a`. We translate $d_{\mathtt{a}}$ to two Boolean features ${\downarrow}d_{\mathtt{a}}$ and $d_{\mathtt{a}}{=}0$. Feature ${\downarrow}d_{\mathtt{a}}$ becomes true if the distance between the agent and object `a` decreases. Feature $d_{\mathtt{a}}{=}0$ becomes true when the agent reaches object `a`. The agent is rewarded with a fixed reward $r > 0$ when ${\downarrow}d_{\mathtt{a}}$ becomes true. When $d_{\mathtt{a}}{=}0$ becomes true, the agent is rewarded with $R > 0$. An example RM is shown in Fig. 3.

In a *Boolean* RM with automatic reward shaping, the agent is rewarded for each experience $\langle s, a, s'\rangle$ by an automatically generated positive reward (Icarte et al. 2022). In contrast, in a *numeric–Boolean* RM, the agent is rewarded by a constant reward r or R for a part of experiences $\langle s, a, s'\rangle$ when ${\downarrow}d_{\mathtt{a}}$ or $d_{\mathtt{a}}{=}0$ becomes true, respectively. In all other cases, the agent receives a reward of 0. In this way, the agent is positively reinforced to approach the target. Exposing the *numeric–Boolean*-RM structure to the agent in CRM and HRM has the potential to boost learning similar to the original approach.

Ideally, the agent should receive a reward that depends on $d_{\mathtt{a}}$. In this way, the agent will be meaningfully guided by each experience $\langle s, a, s'\rangle$. To realise this, we introduce *numeric* RMs.

3.4 *Numeric* Reward Machines

The second new type of RM is a *numeric* RM. Here, we use the numeric feature $d_{\mathtt{a}}$ along with the Boolean feature $d_{\mathtt{a}}{=}0$ directly in the RM. In this RM, the Boolean feature $d_{\mathtt{a}}{=}0$ governs the transition between the RM states, and the numeric feature $d_{\mathtt{a}}$ is used for rewarding the agent with $-d_{\mathtt{a}}$. In this way, the agent is rewarded for each experience by the negative distance to the target. The reward becomes 0 upon arrival at the target. Rewarding the agent is therefore controlled by the real conditions in the environment. An example *numeric* RM is shown in Fig. 4.



$\langle d_{\mathtt{a}}, d_{\mathtt{a}}{\neq}0; -d_{\mathtt{a}}\rangle$
$\langle d_{\mathtt{b}}, d_{\mathtt{b}}{\neq}0; -d_{\mathtt{b}}\rangle$
u_0
$\langle d_{\mathtt{a}}{=}0; 0\rangle$
u_1
$\langle d_{\mathtt{b}}{=}0; 0\rangle$
$\langle d_{\mathtt{c}}{=}0; 0\rangle$
u_3
u_2
$\langle d_{\mathtt{c}}, d_{\mathtt{c}}{\neq}0; -d_{\mathtt{c}}\rangle$

Figure 4: *Numeric* reward machine with numeric features $d_{\mathtt{a}}$, $d_{\mathtt{b}}$, and $d_{\mathtt{c}}$ and Boolean features $d_{\mathtt{a}}{=}0$, $d_{\mathtt{b}}{=}0$, and $d_{\mathtt{c}}{=}0$ for a sequential task `a-b-c`.

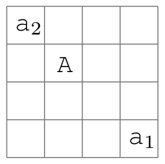


Figure 5: Agent `A` can choose to approach either $\mathtt{a}_1$ or $\mathtt{a}_2$.

Noteworthy, as the RM structure is still exposed to the agent via the Boolean feature $d_{\mathtt{a}}{=}0$, learning can be boosted by CRM or HRM.

3.5 Shortest-Path Guarantees

Next, we investigate whether the discovered policies by a Q-learning-based method correspond to the shortest paths between the agent and the targets for the proposed RMs. Ideally, we would like our RMs to have this property to avoid tweaking the reward values in the RM for each new task. For this analysis, we use an example map, shown in Fig. 5, with two targets $\mathtt{a}_1$ and $\mathtt{a}_2$, both of type `a`. The agent is instructed to visit one of the targets and can freely choose whether to approach $\mathtt{a}_1$ or $\mathtt{a}_2$. We choose this example as an extreme case, where the agent approaches exactly one of the targets with every move. This can "confuse" the agent, as it receives a positive reward for any move, and thus might lead to finding sub-optimal paths. In addition, we consider a simplified scenario with only one target `a` on the map (at any location).

Since the agent performs one move action at each time step, the path that it takes to either target is optimal if the number of episode steps T is minimum. To check whether this is the case, we dissect the trajectory return functions (Eq. 1) with respect to T for the introduced RMs. In all our calculations, we assume $\gamma < 1$.

***Boolean* Reward Machines** The agent receives reward R when it arrives at a terminal state and zero otherwise, i.e.,

$$r_t = \begin{cases} R, & t = T \\ 0, & t \neq T. \end{cases} \tag{5}$$

Therefore, the return for any trajectory is

$$g(T) = \gamma^{T-1} R. \tag{6}$$

As the agent seeks to maximise the accumulated reward, the lower the episode length T, the higher the return $g(T)$. Thus, *Boolean* RMs guarantee to find shortest paths. This holds for both the example map with two targets and the simplified scenario with one target.

***Numeric–Boolean* Reward Machines** A reward $r > 0$ is given for decreasing the distance towards any target. Consequently, for the example map in Fig. 5, where the targets are in the corners of the map, the agent receives a positive reward for any move, i.e.,

$$r_t = \begin{cases} R, & t = T \\ r, & t \neq T. \end{cases} \tag{7}$$

The return for any trajectory is $g(T) = \gamma^{T-1}R + r\sum_{t=0}^{T-2}\gamma^t$. The geometric-series solution (Zorich and Paniagua 2016) yields

$$g(T) = \gamma^{T-1}R + r\frac{1-\gamma^{T-1}}{1-\gamma}. \tag{8}$$

The second term in this equation increases with the number of steps T. Therefore, the resulting path length might not be minimum.

Assume that a shortest path uses T^* steps, resulting in the trajectory return

$$g(T^*) = \gamma^{T^*-1}R + r\frac{1-\gamma^{T^*-1}}{1-\gamma}. \tag{9}$$

If the agent takes two additional back-and-forth steps n times, the corresponding return is

$$g(T^*+2n) = \gamma^{T^*-1+2n}R + r\frac{1-\gamma^{T^*-1+2n}}{1-\gamma}. \tag{10}$$

If $g(T^*)$ is larger than $g(T^* + 2n)$, the resulting path is guaranteed to have the shortest length. The comparison yields

$$\begin{aligned} \gamma^{T^*-1}R + r\frac{1-\gamma^{T^*-1}}{1-\gamma} > \gamma^{T^*-1+2n}R + r\frac{1-\gamma^{T^*-1+2n}}{1-\gamma}; \\ \gamma^{T^*-1}R(1-\gamma^{2n}) > \frac{r\gamma^{T^*-1}}{1-\gamma}(1-\gamma^{2n}); \\ r < R(1-\gamma). \end{aligned} \tag{11}$$

Therefore, we find a shortest path if r is lower than $R(1-\gamma)$. For example, if $R = 1$ and $\gamma = 0.9$, then r needs to be lower than 0.1.

Next, we consider the simplified scenario with one target a. The agent gets a positive reward r if the distance to a decreases. The distance from the agent to the target at step t is written as d_t, and the change in distance from step t to $t+1$ is captured by $\Delta d = d_{t+1} - d_t$. If $\Delta d < 0$, then the agent approaches the target and receives reward r, i.e.,

$$r_t = \begin{cases} R, & \Delta d = 0, \\ r, & \Delta d < 0, \\ 0, & \Delta d > 0. \end{cases} \tag{12}$$

Now, if the agent takes $2n$ additional back-and-forth steps, the trajectory return is

$$\begin{aligned} g_o(T^*+2n) = \gamma^{T^*-1+2n}R + r\frac{1-\gamma^{T^*-1}}{1-\gamma} + \\ + r\gamma^{T^*-1}(\gamma + \gamma^3 + \dots + \gamma^{2n-1}), \end{aligned} \tag{13}$$

where the subscript o denotes that only one a is on the map. The final trajectory return is then rewritten using the geometric-series solution as follows:

$$\begin{aligned} g_o(T^*+2n) = \gamma^{T^*-1+2n}R + r\frac{1-\gamma^{T^*-1}}{1-\gamma} + \\ + r\gamma^{T^*-1}\frac{\gamma-\gamma^{2n+1}}{1-\gamma^2}. \end{aligned} \tag{14}$$

The comparison between $g_o(T^*)$ and $g_o(T^*+2n)$ yields

$$\begin{aligned} \gamma^{T^*-1}R > \gamma^{T^*-1+2n}R + r\gamma^{T^*-1}\frac{\gamma-\gamma^{2n+1}}{1-\gamma^2}; \\ r < R\frac{1-\gamma^2}{\gamma}. \end{aligned} \tag{15}$$

Here, r needs to be smaller than $R\frac{1-\gamma^2}{\gamma}$ to guarantee that we find a shortest path. For example, if $R = 1$ and $\gamma = 0.9$, then r needs to be smaller than 0.21.

We showed that rewards in *numeric–Boolean* RMs should be selected carefully. This solution dependence on the reward values is problematic and could be addressed in future work.

***Numeric* Reward Machines** For several targets on the map, the reward scheme of *numeric* RMs can be as follows: The agent is rewarded with $-d$, where d is the lowest distance to any target of the same type. We hypothesise that this modification preserves the guarantee to find shortest paths. However, we leave the proof of this conjecture for future work.

For the case with a single target a, *numeric* RMs reward the agent with

$$r_t = -d, \tag{16}$$

where d is the number of steps to the closest target. The shortest-trajectory return is then

$$g_o(T^*) = \sum_{t=0}^{T^*-1}\gamma^t(t-T^*). \tag{17}$$

If the agent takes $2n$ additional back-and-forth steps just before reaching the target, the trajectory return is

$$\begin{aligned} g_o(T^*+2n) = \sum_{t=0}^{T^*-1}\gamma^t(t-T^*) - \\ -2\gamma^{T^*-1}\sum_{k=0}^{n}\gamma^{2k-2} - 1\gamma^{T^*-1}\sum_{k=0}^{n-1}\gamma^{2k+1}, \end{aligned} \tag{18}$$

where the last two series correspond to the rewards received at step "back" and "forth", respectively. The difference between $g_o(T^*)$ and $g_o(T^*+2n)$ will always be positive because the last two terms in $g_o(T^* + 2n)$ will always be negative. Therefore, *numeric* RMs guarantee shortest-path solutions regardless of the selected parameters for a map with one target. In future work, we plan to extend these proofs to sequential tasks, where the agent needs to visit multiple targets.

4 Experimental Evaluation

For our experimental evaluation, we choose a 41×41 grid, matching the grid size in the Craft domain tested by Icarte et al. (2022). We use two different object setups: (1) `1a1b1c` and (2) `2a2b2c`. Letters (`a`, `b`, and `c`) encode the object type, and the numeral in front of each letter (`1` and `2`) encodes how often the object type appears on the map. That is, in setup `1a1b1c`, three objects, `a`, `b`, and `c`, are randomly placed on the grid. Likewise, in setup `2a2b2c`, two pairs of objects `a`, `b`, and `c` are randomly placed on the grid. In the latter setup, the agent is expected to discover which objects of the same type are closest to its initial location. We use Manhattan distance as the distance metric in this work, i.e., the grid cells are four-connected. The initial agent position is fixed throughout the experiments. For each setup, we generate 10 random maps to check the variability of the results.

For a given map, the agent is instructed to visit objects sequentially in three different tasks: `a`, `a-b`, and `a-b-c`. In task `a`, the agent needs to visit only an object of type `a`. In task `a-b`, the agent needs to visit object types `a` and then `b`. Finally, in task `a-b-c`, the agent needs to visit object types `a`, `b`, and `c`, in order. We show results only for two maps (stemming from setups `1a1b1c` and `2a2b2c`) because the relative algorithm performance is similar on all generated maps of one setup. One of these maps, with setup `2a2b2c`, is provided in the appendix in Fig. A.1 for discussion purposes. The median results for all 10 maps are shown in the appendix in Fig. A.2.

For each map, we run each algorithm six times and measure the average reward per step. All rewards are normalised to be between 0 and 1. A reward of 1 corresponds to the optimal policy found via value iteration (Icarte et al. 2022). We report the median performance over the six runs and the 25th and 75th percentiles in Fig. 6 (barely visible for most algorithms).

We compare the following combinations of methods. First, the best-performing methods for Boolean features and tabular domains are CRM with reward shaping (crm-rs-bool) and HRM (hrm-bool) according to Icarte et al. (2022). Despite being faster than crm-rs-bool, hrm-bool tends to converge to sub-optimal policies. We use both methods as baselines in our experiments. For *numeric–Boolean* and *numeric* RMs, we use CRM, HRM, and Q-learning, abbreviated as crm-num-bool, hrm-num-bool, qrm-num-bool, crm-num, hrm-num, and qrm-num. The reason for including Q-learning together with *numeric–Boolean* and *numeric* RMs is to explore whether the knowledge of the RM structure boosts the agent's learning in CRM and HRM. The parameters in the Q-learning, CRM, and HRM methods are taken from Icarte et al. (2022).

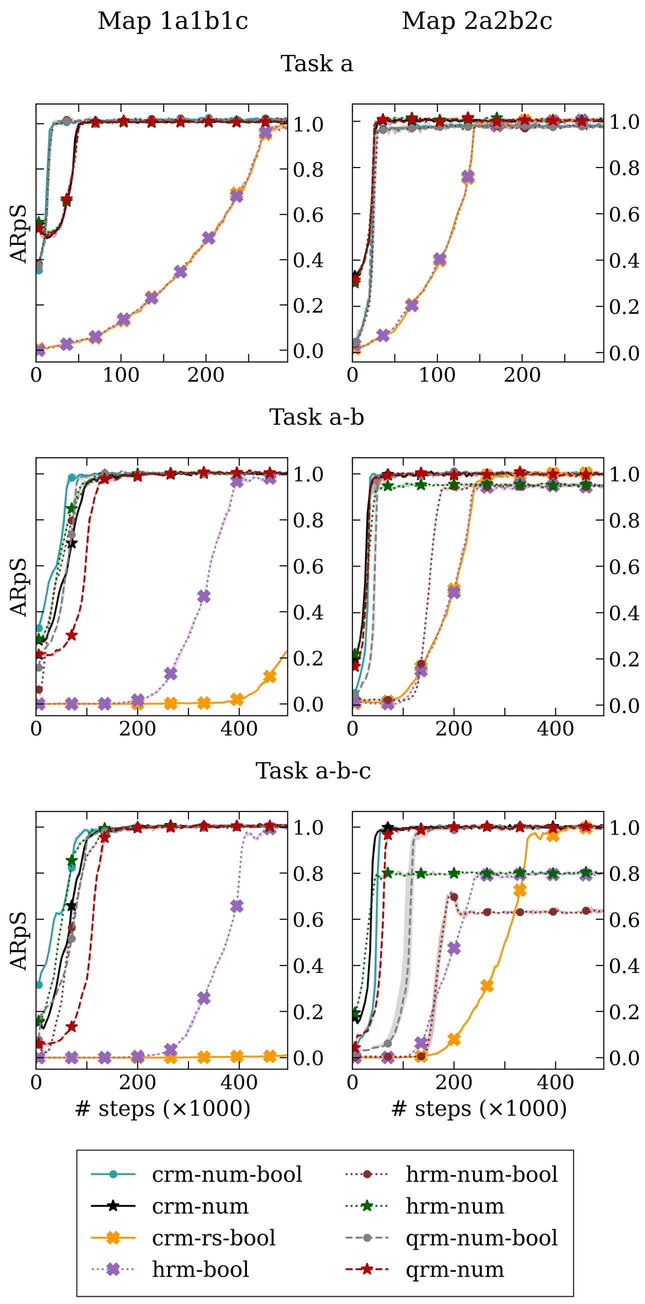


Figure 6: Median performance for map `1a1b1c` (left column) and map `2a2b2c` (right column). The 25th and 75th percentiles are shown in the shadowed regions. ARpS stands for average reward per step.

5 Results and Discussion

Using *Boolean* RMs, the agent receives a reward of 1 after transitioning to the terminal state and a reward of 0 for all other RM transitions. Regardless of the number of targets, the agent solves all evaluated tasks with the shortest path.

In *numeric–Boolean* RMs, we choose $r = 0.1$ and $R = 1000$ to guarantee that the agent finds shortest paths (see Sect. 3.5). Indeed, the agent finds shortest paths in all tasks.

With *numeric* RMs, the agent finds a shortest path only for task `a`, for which we proved that the optimal policy corresponds to the shortest-path solution in Sect. 3.5. However, for tasks `a-b` and `a-b-c`, the agent converges to non-optimal solutions. It is only able to find a shortest path once we change the terminal rewards from 0 to values of 10 000 and 100 000, respectively. We will investigate this outcome in future work.

We visualise the results in Fig. 6. The shadowed regions depicting the 25th and 75th percentiles are narrow. This is because randomisation in our experiments comes only from tie-breaking in the case of equal Q-values and from using ϵ-greedy for exploration. The methods using *Boolean* RMs converge slower than other methods.

In tasks `a-b` and `a-b-c`, all HRM-based methods converge to sub-optimal policies on map `2a2b2c`. This can be explained as follows. As shown in Fig. A.1, the Manhattan distances to the top-positioned `a` and bottom-positioned `a` from the initial position of agent `A` are 20 and 21, respectively. The HRM-based agent chooses the top-positioned `a` because it is always greedy towards the current transition in the RM. The total distances to the top-positioned `a-b` and bottom-positioned `a-b` are 27 and 26, respectively. Consequently, after choosing the top route, the HRM-based algorithms converge to a sub-optimal policy. In task `a-b-c`, the performance of the HRM-based algorithms deteriorates further. This is explained by the total distances to the top-positioned `a-b-c` and bottom-positioned `a-b-c` being 40 and 32, respectively.

Furthermore, the results show that the CRM-based methods always converge at least as fast as pure-Q-learning-based methods. Indeed, the knowledge of the RM structure speeds up learning.

Another observation is that all CRM- and pure-Q-learning-based methods converge faster on map type `2a2b2c` than on map type `1a1b1c`. This is because placing more objects of the same kind on the map makes it more likely that the total distance to the final object is shorter from the initial position of the agent.

On map type `1a1b1c`, the *numeric–Boolean*-RM-based methods outperform the corresponding *numeric*-RM-based methods. In contrast, on map type `2a2b2c`, the *numeric*-RM-based methods perform at least as well as the corresponding *numeric–Boolean*-RM-based methods. We believe that this result is related to the numeric reward scheme. Furthermore, the *numeric–Boolean*-RM-based methods converge to a value slightly below 1.0 in task `a` on map type `2a2b2c`. We will investigate both of these phenomena in the future.

The performance difference between the methods increases with difficulty from task `a` to task `a-b-c`. In task `a`, all methods from the same feature category perform similarly. That is, crm-rs-bool and hrm-bool perform similar; qrm-num-bool, crm-num-bool and hrm-num-bool perform similar; and finally, qrm-num, qrm-num-bool, and hrm-num perform similar. The explanation is the task simplicity. Only once the task becomes challenging enough, performance differences start to appear.

Overall, the results show that *numeric–Boolean* and *numeric* RMs speed up learning in comparison with *Boolean* RMs with automatic reward shaping.

6 Conclusions

In this study, we extend RMs with numeric features by introducing *numeric–Boolean* and *numeric* RMs. We compare them with the original *Boolean* RMs introduced by Icarte et al. (2022).

For our experimental evaluation, we use the Craft domain. This is a simple grid world without obstacles, allowing the use of tabular RL methods. We test the developed methods in two directions. First, the results confirm that the developed reward acquisition schemes in *numeric–Boolean* and *numeric* RMs outperform automatic reward shaping used in *Boolean* RMs. Second, the results confirm that the exposure of the *numeric–Boolean*- and *numeric*-RM structures to the agent speeds up learning in CRM and HRM. The advantage of *numeric* RMs over *numeric–Boolean* RMs is not yet clear. Additional experiments with more complicated tasks should be performed to make meaningful conclusions.

Furthermore, we prove the shortest-path guarantees for the task with one target type. We show that *Boolean* RMs guarantee finding shortest paths regardless of the reward values in the RM. In contrast, in *numeric–Boolean* RMs, the terminal reward R should be higher than the intermediate reward r, and the difference between R and r will depend on the task. For *numeric* RMs, we prove that shortest paths are guaranteed for tasks with one target type. However, for sequential tasks with several targets, the experimental results show that this is not the case.

In future work, it will be interesting to test the *numeric–Boolean* and *numeric* RMs in tabular domains with obstacles, continuous-space domains, and continuous-control tasks. In addition, we will perform experiments for unordered tasks, where the agent can visit the assigned targets in any order. Furthermore, we will extend the proofs of the shortest-path guarantees for sequential tasks with several targets. Ideally, the problem of the solution dependence on the reward values in the RM should be resolved.

7 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 *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.

Gao, Y.; Matsunami, Y.; Miyata, S.; and Akashi, Y. 2022. Operational Optimization for Off-Grid Renewable Building Energy System using Deep Reinforcement Learning. *Applied Energy*, 325: 119783.

Icarte, R. T. 2021. Reward Machines Code. https://github.com/RodrigoToroIcarte/reward_machines. Accessed: March 19, 2024.

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.

Kazemi, M.; Perez, M.; Somenzi, F.; Soudjani, S.; Trivedi, A.; and Velasquez, A. 2022. Translating Omega-regular Specifications to Average Objectives for Model-free Reinforcement Learning. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022),*.

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.

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.

Skalse, J.; and Abate, A. 2023. On the Limitations of Markovian Rewards to Express Multi-objective, Risk-sensitive, and Modal Tasks. In *Uncertainty in Artificial Intelligence*, 1974–1984. PMLR.

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*.

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

Zorich, V. A.; and Paniagua, O. 2016. *Mathematical Analysis II*, volume 220. Springer.

Appendix

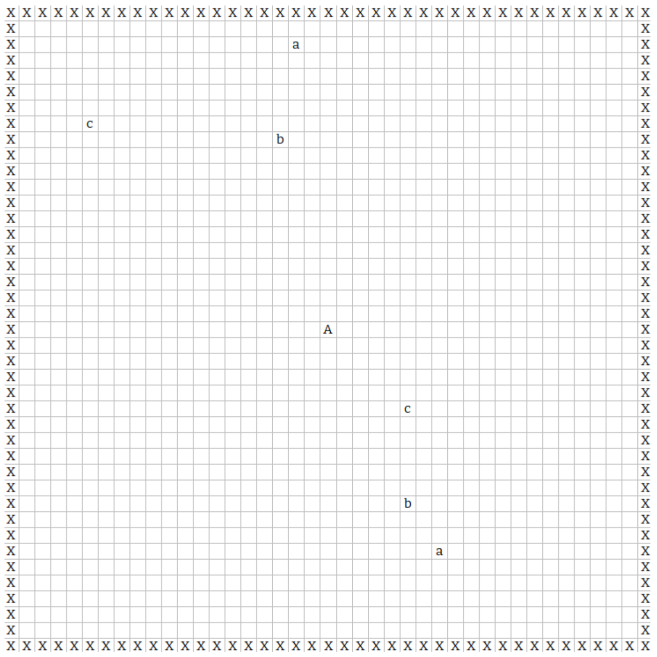


Figure A.1: Analysed map with setup `2a2b2c`, agent `A`, and walls `X`.

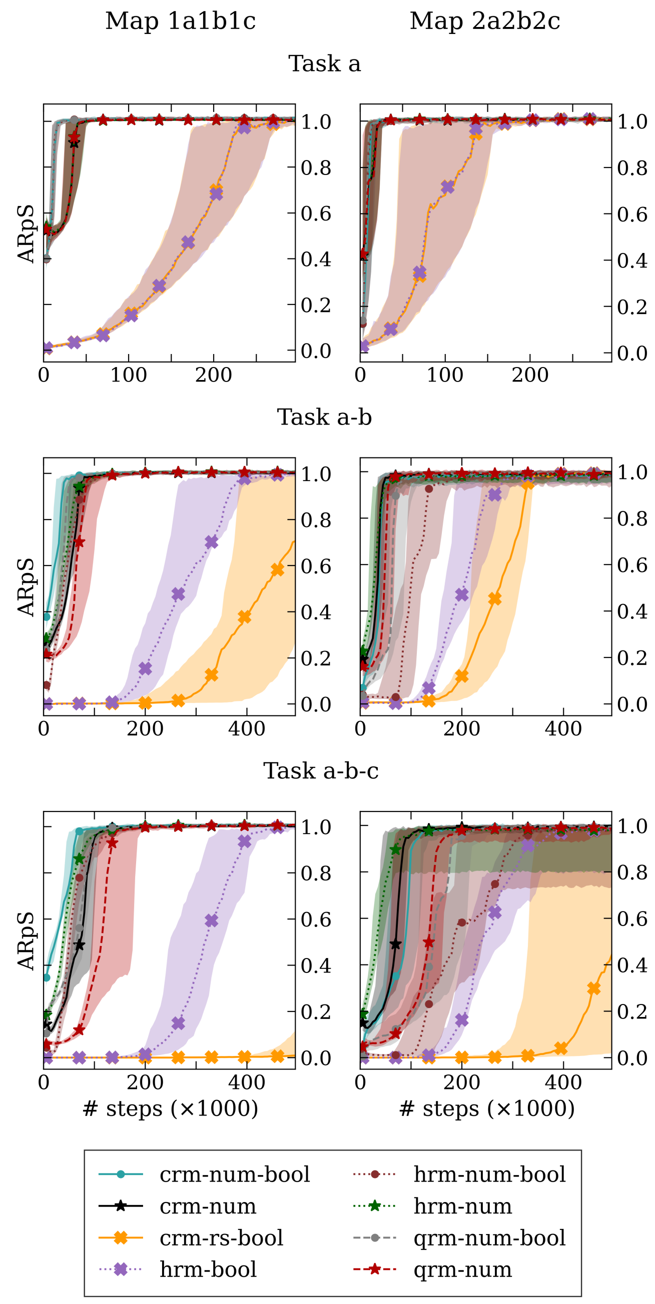


Figure A.2: Median performance for all 10 `1a1b1c` and `2a2b2c` maps (left and right columns, respectively). The 25th and 75th percentiles are shown in the shadowed regions. ARpS stands for average reward per step.