Exploiting Reward Machines with Deep Reinforcement Learning in Continuous Action Domains*

Haolin Sun[0009−0001−7185−0921] and Yves Lespérance[0000−0003−1625−0226]

York University, Toronto, Canada sun0907@yorku.ca,lesperan@yorku.ca

Abstract. In this paper, we address the challenges of non-Markovian rewards and learning efficiency in deep reinforcement learning (DRL) in continuous action domains by exploiting reward machines (RMs) and counterfactual experiences for reward machines (CRM). RM and CRM were proposed by Toro Icarte *et al.*. A reward machine can decompose a task, convey its high-level structure to an agent, and support certain non-Markovian task specifications. In this paper, we integrate state-of-the-art DRL algorithms with RMs to enhance learning efficiency. Our experimental results demonstrate that Soft Actor-Critic with counterfactual experiences for RMs (SAC-CRM) facilitates faster learning of better policies, while Deep Deterministic Policy Gradient with counterfactual experiences for RMs (DDPG-CRM) is slower, achieves lower rewards, but is more stable. Option-based Hierarchical Reinforcement Learning for reward machines (HRM) and Twin Delayed Deep Deterministic (TD3) with CRM generally underperform compared to SAC-CRM and DDPG-CRM. This work contributes to the ongoing development of more efficient and robust DRL approaches by leveraging the potential of RMs in practical problem-solving scenarios.

Keywords: Deep Reinforcement Learning · Reward Machines

1 Introduction

In *reinforcement learning (RL),* an agent interacts with the environment by performing actions in each state, receiving a reward signal in return and the agent's goal is to learn a policy (mapping observations to actions) that maximizes the expected cumulative reward and improves its policy from past experiences.

In simple discrete action domains, like turn-based games with finite states and actions, basic RL algorithms such as Q-learning [23] suffice to quickly find

* This version of the contribution has been accepted for publication, after peer review but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: https://doi.org/10.1007/978-3-031-43264-4\protect_6. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms.

the optimal policy. However, in more complex *continuous action domains* like autonomous driving, where variables like acceleration and steering angle have infinite domains, the agent cannot try all possible actions. Consequently, Q-learning fails to identify actions with the highest expected rewards and determine the optimal policy, and struggles to explore the state space effectively. *Deep reinforcement learning (DRL)* was developed to address hard RL problems such as those in continuous action domains. DRL combines neural networks' understanding capabilities with RL's decision-making, allowing agents to tackle more complex problems in such domains [7].

Reward functions in RL algorithms are typically "black boxes". As a result, learning requires extensive interaction with the environment, consuming significant time and computational resources. However, if the agent can access the reward function's internal structure and understand the task's high-level idea, it can leverage this information to expedite optimal policy learning.

To provide agents access to the reward function, Toro Icarte *et al.* proposed using finite state machines called *reward machines (RMs)* [21,22,20], which define a novel form for reward functions that support certain non-Markovian task specifications. The reward is non-Markovian when it doesn't just depend on the current world state but on the whole history. A reward machine can define multiple forms of reward functions, including concatenation, loops, and conditional rules. It can also decompose a complex task into subtasks, revealing each subtask's reward function to the agent. The RM is assumed to be fully known to the agent; as the agent transitions between RM states, the specific subtask's reward is returned, enabling state-by-state learning and thus allowing the agent to conduct less exploration and speed up the learning. Reward machines offer flexible expression, allowing tasks to be represented using Linear Temporal Logic over infinite or finite traces (LTL/LTL$_f$) [15,4] or other formal languages before translation into a reward machine. A related approach is that of "restraining bolts" [3], where LTL$_f$ restraining specifications are compiled into automata and used in RL to ensure that the learned behavior conforms to them [1]. Another related approach is called "logically constrained RL" [9], where one specifies rules about the finite set of actions that are allowed in a given state, avoiding an exhaustive update over the whole state space, thus guiding the agent to learn more efficiently and conform to desired behaviors.

To utilize an RM's structure, Toro Icarte *et al.* proposed a novel approach called *counterfactual experiences for reward machines (CRM)* [22,20]. CRM leverages reward function information from RMs during agent-environment interactions to generate synthetic experiences, helping the agent make more explicit judgments about RM states thus accelerating learning speed.

Reward machines can be applied in both discrete and continuous action domains. In discrete action domains, Toro Icarte *et al.* enhanced the learning efficiency of existing RL and DRL algorithms by combining reward machines with Q-learning [23] and Double DQN [10], where RM-based Q-learning can converge to the optimal policy. However, in continuous action domains, only DDPG [12] and option-based Hierarchical Reinforcement Learning (HRL) [19] have been

combined with reward machines. As new deep RL algorithms emerged, the performance of DDPG and option-based HRL has become less prominent, with some newly proposed algorithms surpassing their performance. To address this issue and further improve the learning efficiency of RM-based algorithms in continuous action domains, we focused on two aspects in our work.

First, we combined CRM with two widely used and well-performing deep RL algorithms, Soft Actor-Critic (SAC) [8] and Twin-Delayed Deep Deterministic Policy Gradient (TD3) [6]. We call the resulting algorithms Soft Actor-Critic with CRM (SAC-CRM) and Twin-Delayed Deep Deterministic Policy Gradient with CRM (TD3-CRM).

Next, we expanded the range of tasks tested compared to prior experiments, e.g., [22]. Based on the RM model, we defined six new tasks in two different continuous action domains. We ran experiments and compared the performance of existing and new RM-based deep RL algorithms and analyzed reasons for performance differences. Through these experiments, we found that SAC-CRM was generally the best-performing algorithm among those studied. The learning speed and reward values it achieved within the specified learning steps were generally the best amongst all the algorithms.

2 Preliminaries

2.1 Reward Machines in RL

Reward Machines To support non-Markovian rewards, Toro Icarte *et al.* [21,22,20] introduced a novel reward function form called the *reward machine (RM)*. Formally, given a set of propositional symbols $\mathcal{P}$, a set of (environment) states S, and a set of actions A, a reward machine (RM) is a tuple $\mathcal{R}_{\mathcal{P}SA} = \langle U, u_0, F, \delta_u, \delta_r \rangle$ where U is a finite set of states, $u_0 \in U$ is the initial state, F is a finite set of terminal states (where $U \cap F = \emptyset$), δ_u is the state-transition function, $\delta_u : U \times 2^{\mathcal{P}} \to U \cup F$, and δ_r is the state-reward function, $\delta_r : U \to S \times A \times S \to R$.

Consider a simple example where our agent (see Fig. 1) is a cheetah-like robot, as in the OpenAI Gym Half-Cheetah domain [2], and the task is to start from an arbitrary point between A and B, first go to point A, then to B and then C, then back to B, then back to C again, and then finally to point D to receive a reward of 1000 (which is Task 3 of the Half-Cheetah domain in Section 4). The agent can move in this 2D environment by choosing the moving angle and force to apply at each joint. Notice that this task involves non-Markovian rewards.

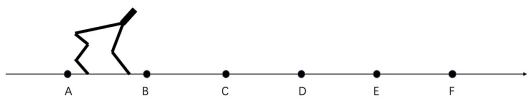

Fig. 1: An example RM environment (Half-Cheetah)

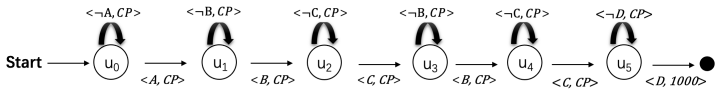

Fig. 2: The automaton for the task

Also, since the agent starts far from point D, and the task contains multiple back-and-forth operations (e.g., to do pick ups and deliveries), if the task description only specifies the final goal of reaching point D, the agent must spend significant time exploring. However, using a reward machine (RM) allows the task to be decomposed into subtasks by introducing multiple RM states to represent each intermediate reward function. With this, the agent can learn to reach each point sequentially, thus getting closer to the target with each subtask. This approach reduces exploration time and improves learning efficiency. The automaton for this task is shown in Fig. 2. In this automaton, the reward value is a small control penalty CP for transitions among the non-terminal RM states u_0 to u_5, and when the agent reaches point D while in u_5, it arrives at the terminal RM state, and it will receive a reward value of 1000. In this environment, the set of propositional symbols $\mathcal{P}$ can be defined as $\mathcal{P} = \{A, B, C, D\}$, where event $e \in \mathcal{P}$ occurs when the agent is at location e. To assign truth values to symbols in $\mathcal{P}$, a labelling function $L : S \times A \times S \to 2^{\mathcal{P}}$ will be needed. L can assign truth values to symbols in $\mathcal{P}$ given an environment experience (s, a, s'), where s' is the resulting state after executing action a from the environmental state s. In the example, U is the set of all the non-terminal RM states, including $\{u_0, u_1, u_2, u_3, u_4, u_5\}$; F is the set of the terminal RM state, which is the state after u_5. When the agent reaches point A, the state-transition function δ_u will transfer the agent's current RM state from u_0 to u_1 (otherwise it remains in u_0), and it will transfer the RM state from u_1 to u_2 when the agent reaches point B, and so forth. When the agent reaches point D, a terminal state, the state-reward function δ_r will give the agent a reward of 1000.

MDPRM In traditional reinforcement learning, the underlying environment model of the agent is assumed to be a *Markov Decision Process* (MDP) [5]. An MDP is a tuple $\mathcal{M} = \langle S, A, r, p, \gamma, \mu \rangle$, where S is a finite set of states, A is a finite set of actions, $r : S \times A \times S \to R$ is the reward function, $p(s_{t+1} \mid s_t, a_t)$ is the transition probability distribution, $\gamma \in (0, 1]$ is the discount factor, and μ is the initial state distribution where $\mu(s_0)$ is the probability that the agent starts in state $s_0 \in S$. By using reward machines, the agent learns in the environment considering not only the environmental state s_t at time t, but also the RM state u_t at time t. The extra consideration of the RM state u_t changes the learning environment from a traditional MDP to a *Markov Decision Process with a Reward Machine (MDPRM)* [21,22,20]. A *Markov Decision Process with a Reward Machine (MDPRM)* is a tuple $\mathcal{T} = \langle S, A, p, \gamma, \mu, \mathcal{P}, L, U, u_0, F, \delta_u, \delta_r \rangle$, where S, A, p, γ and μ are defined as in an MDP, $\mathcal{P}$ is a set of propositional

symbols, L is a labelling function $L : S \times A \times S \to 2^{\mathcal{P}}$, and U, u_0, F, δ_u, and δ_r are defined as in a reward machine. In an MDPRM, the policy learned by the agent then changes from $\pi(a \mid s)$ to $\pi(a \mid s, u)$, and the experience changes from $\langle s, a, r, s' \rangle$ to $\langle s, u, a, r, s', u' \rangle$. It can be seen that MDPRMs are regular MDPs when considering the cross-product between the environmental states S and the RM states U. As such, standard RL algorithms can learn in MDPRMs by using the *cross-product* of environment and RM states [1,11].

CRM To exploit the information provided by the RM, Toro Icarte *et al.* proposed a method called *Counterfactual experience for Reward Machines (CRM)* [22,20]. CRM also learns policies over the cross-product $\pi(a \mid s, u)$, but uses counterfactual reasoning to generate *synthetic* experiences. In CRM, the RM will go through every RM state $\bar{u} \in U$ after each action, and use the state transition function $\delta_u(\bar{u}, L(s, a, s'))$ to determine the next RM state $\bar{u}'$; the agent will also receive a reward of $\bar{r}$ using the reward transition function $\delta_r(\bar{u})(s, a, s')$. That is, instead of just providing the actual experience in an MDPRM, the RM can now provide one experience per RM state. In this manner, after taking just one action, the agent will get to know whether the action could cause a transition in any of the RM states and what the reward would be if that happened. In other words, the agent will be able to determine precisely whether its current action, made in the current environmental state, would have an impact on any subtask. This greatly improves the efficiency of the agent's exploration.

2.2 Deep RL Algorithms

Deep Deterministic Policy Gradient (DDPG) Deep Deterministic Policy Gradient (DDPG) [12] is an off-policy deep reinforcement learning algorithm that incorporates an actor-critic architecture to address complex, continuous control problems. DDPG utilizes two distinct neural networks, namely the actor network and the critic network. The actor network is responsible for learning the optimal policy, while the critic network approximates the optimal Q-function, which estimates the expected reward of taking a given action in a given state.

In DDPG, the actor network takes the current environment state as input and outputs a continuous-valued action derived from the current policy. The critic network estimates the value of state-action pairs based on the actor network's output. By adopting a deterministic policy gradient approach, DDPG is able to effectively handle continuous action spaces, while the incorporation of experience replay and target networks stabilizes the learning process.

Option-based Hierarchical Reinforcement Learning (HRL) Option-based Hierarchical Reinforcement Learning (HRL) [19] is a framework for efficiently learning and planning in complex environments with long-term goals and multiple abstraction levels. In HRL, agents learn a set of subgoals, or "options", which can be combined to create high-level plans. Options serve as reusable subroutines learned through experience. During training, agents learn intra-option policies to

achieve each subgoal and inter-option policies for transitioning between subgoals. This allows agents to navigate complex environments by decomposing problems into smaller, more manageable subtasks. Thus a key advantage of HRL is its ability to reduce the amount of training needed, particularly in tasks involving long action sequences.

Soft Actor-Critic (SAC) Soft Actor-Critic (SAC) [8] is an off-policy deep RL algorithm specifically designed for continuous control tasks. SAC aims to concurrently maximize the policy's entropy and its cumulative return, i.e., obtain an agent that succeeds at the task while acting as randomly as possible. To do this, it incorporates an entropy term into the Q-function:

$$Q^{\pi}_{soft}(s,a) = \mathop{\mathbb{E}}_{s_t,a_t\sim\rho_\pi}\left[\sum_{t=0}^{\infty}\gamma^t r\left(s_t,a_t\right) + \alpha\sum_{t=1}^{\infty}\gamma^t H\left(\pi\left(\cdot \mid s_t\right)\right) \mid s_0 = s, a_0 = a\right]$$

where entropy is defined as: $H(P) = \mathop{\mathrm{E}}_{x\sim P}[-\log P(x)]$.
Adding this entropy component enables deeper exploration of the state space, which is crucial in continuous control tasks characterized by high-dimensional state and action spaces.

The maximum entropy model offers several advantages, including making the fewest assumptions about the environment's unknown information while matching observed data. This approach ensures that the model remains robust and adaptable to various environments. Furthermore, by controlling the entropy value, the agent can maintain a high level of exploration capability. This prevents the agent from prematurely converging to a local optimum and allows for the discovery of more optimal solutions in complex problem domains.

Twin Delayed Deep Deterministic Policy Gradient (TD3) Twin Delayed Deep Deterministic Policy Gradient (TD3) [6] is an off-policy deep RL algorithm for continuous control tasks, improving upon the original Deep Deterministic Policy Gradient (DDPG) algorithm by addressing several limitations. A primary enhancement in TD3 is the use of two critic networks instead of one, estimating the value of state-action pairs and reducing overestimation bias. TD3 also employs delayed policy updates, updating the policy less frequently than the critic networks to decrease policy update variance and stabilize learning. Another notable feature of TD3 is target policy smoothing, which adds noise to actions selected by the actor network, regularizing the policy and increasing its robustness to environmental perturbations. This is especially beneficial in continuous control tasks where minor action changes significantly impact the agent's behavior.

For more technical details about these algorithms, see [18].

3 Adapting Deep RL Algorithms with Reward Machines

Toro Icarte *et al.* [22,20] proposed a variant of DDPG that incorporates the CRM approach, calling it DDPG-CRM. Concurrently, they introduced an options-

based Hierarchical Reinforcement Learning (HRL) algorithm that learns options to move between states of a RM, which they call HRM. The integration of CRM into DDPG is achieved by initially modifying the learning environment to suit the Markov Decision Process with a Reward Machine (MDPRM), followed by the inclusion of counterfactual experiences into the replay buffer. Instead, HRM applies DDPG to learn the option policies while employing Deep Q-Network (DQN) [13] to learn the high-level policy. In this work, we incorporate the CRM approach into two additional deep RL algorithms that are currently widely recognized for their strong performance, namely Soft Actor-Critic (SAC) [8] and Twin Delayed Deep Deterministic Policy Gradient (TD3) [6]. Note that we also experimented with combining CRM with PPO [17] but the performance/learning efficiency was very poor, see [18] for details. PPO is an on-policy RL method and it is not clear how counterfactual experiences can be incorporated effectively in such approaches.

3.1 Soft Actor-Critic (SAC) with CRM

First, we use SAC as a base and propose a new algorithm, SAC-CRM, that takes advantage of the task structure that the RM has made visible. In SAC-CRM, the agent still uses the entropy value from the baseline SAC when updating the Q-function and continues the Energy-Based Policy model from the baseline SAC. In contrast to the baseline, SAC-CRM changes the type of the actual experience compared to the baseline SAC and also adds counterfactual experiences to the replay buffer. The pseudocode of SAC-CRM is shown in Algorithm 1.

In SAC-CRM, the learning environment becomes an MDPRM, so the RM experience will be added to the replay buffer. The actual experience learned by the agent will change from the original $\langle \mathrm{s}, \mathrm{a}, \mathrm{r}, \mathrm{s}' \rangle$ to $\langle \mathrm{s}, \bar{u}, \mathrm{a}, \bar{r}, \mathrm{s}', \bar{u}' \rangle$, where $\bar{u}$ and $\bar{u}'$ are the RM states before and after the action a, and $\bar{r}$ is the reward given by the reward machine. Also, CRM will generate one counterfactual experience for each RM state after the agent takes an action (see line 7 in Alg. 1). To generate the counterfactual experiences, the agent will traverse each RM state $\bar{u} \in U$ after making an action. If the agent's action in $\bar{u}$ causes the environmental state s change to the next environmental state s', then the next RM state will be calculated by the state-transition function, which is $\bar{u}' = \delta_u\left(\bar{u}, L\left(s, a, s'\right)\right)$, and the agent will receive a reward given by the state-reward function, which is $\bar{r} = \delta_r(\bar{u})\left(s, a, s'\right)$. CRM generates one counterfactual experience for each RM state. The expression of the counterfactual experience set is:

$$\left\{\left(s, \bar{u}, a, \delta_r(\bar{u})\left(s, a, s'\right), s', \delta_u\left(\bar{u}, L\left(s, a, s'\right)\right)\right) \mid \bar{u} \in U\right\}$$

Correspondingly, SAC-CRM will learn the information provided by CRM when updating the policy. Specifically, the agent will consider both the actual experience and counterfactual experiences. In terms of reward, the agent will now consider the RM reward provided by the state-reward function. At this point, since the agent learns in an MDPRM, SAC-CRM will not only consider the actual environmental state but the cross-product of the environmental state and

Algorithm 1 Soft Actor-Critic with counterfactual experiences for RMs (CRM).

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$, labelling function L, a finite set of states U, a finite set of terminal states F, state-transition function δ_u, state-reward function δ_r, initial RM state $u_0 \in U$

Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1, \phi_{\text{targ},2} \leftarrow \phi_2$
Initialize $u \leftarrow u_0$ and $s \leftarrow$ EnvInitialState()
repeat
  Observe state s and select action $a \sim \pi_\theta(\cdot \mid s, u)$
  Execute a in the environment and observe next state s'
  Compute the reward $r \leftarrow \delta_r(u)(s, a, s')$ and next RM state $u' \leftarrow \delta_u(u, L(s, a, s'))$, and done signal d to indicate whether s' is terminal
  Set experience $\leftarrow \{(s, \bar{u}, a, \delta_r(\bar{u})(s, a, s'), s', \delta_u(\bar{u}, L(s, a, s')), d) \mid \bar{u} \in U\}$
  Store experience in replay buffer $\mathcal{D}$
  If s' is terminal or $\bar{u} \in F$, reset environment state.
  if it's time to update **then**
    for j in range (however many updates) **do**
      Randomly sample a batch B of transitions from $\mathcal{D}$
      Compute targets for the Q functions:

$$y(\bar{r}, s', \bar{u}', d) = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \bar{u}', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}' \mid s', \bar{u}')\right),$$

$$\tilde{a}' \sim \pi_\theta(\cdot \mid s', \bar{u}')$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,\bar{u},a,\bar{r},s',\bar{u}',d) \in B} \left(Q_{\phi_i}(s, \bar{u}, a) - y(\bar{r}, s', \bar{u}', d)\right)^2 \quad \text{for } i = 1, 2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s,\bar{u} \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \bar{u}, \tilde{a}_\theta(s, \bar{u})) - \alpha \log \pi_\theta(\tilde{a}_\theta(s, \bar{u}) \mid s, \bar{u})\right)$$

      where $\tilde{a}_\theta(s, \bar{u})$ is a sample from $\pi_\theta(\cdot \mid s, \bar{u})$ which is differentiable wrt θ via the reparametrization trick.
      Update target network with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1-\rho)\phi_i \quad \text{for } i = 1, 2$$

    end for
  end if
until convergence or maximum training step reached

the RM state, as well as the counterfactual experiences provided by CRM (line 9 to line 15 in Alg. 1). Note that we follow [8] and maintain two independent Q functions and use the minimum of the two in the policy improvement step (line 15) to mitigate positive bias. The updated policy is selected by minimizing the distance between it and the energy-based policy (EBP) for the Q function, where the distance is measured via Kullback-Leibler divergence.

3.2 Twin Delayed Deep Deterministic Policy Gradient (TD3) with CRM

We chose Twin Delayed Deep Deterministic Policy Gradient (TD3) as another algorithm to integrate with CRM. The integration process is similar to SAC-CRM and involves two steps. First, we added reward machine information to the actual experience, which includes current and next RM states and the RM reward. Then, we added counterfactual experiences to the replay buffer.

Because the learning environment becomes an MDPRM, we need to include reward machine information in the actual experience. Specifically, we added the cross-product of the environmental states and the RM states to the actual experience, as well as the reward provided by the reward machine, changing the agent's experience from $\langle s, a, r, s' \rangle$ to $\langle s, u, a, r, s', u' \rangle$. Then, we added counterfactual experiences by generating a corresponding counterfactual experience for each RM state after the agent executes each action. This experience contains the next RM state $\bar{u}'$ calculated using the state-transition function $\delta_u(\bar{u})$ and the RM reward $\bar{r}$ calculated by the state-reward function $\delta_r(\bar{u})$, which is the same form as the counterfactual experience in SAC-CRM.

For more details about all these algorithms, see [18]; the code is available at `https://github.com/haolinsun0907/Exploiting_RMs_with_DRL`).

4 Experimental Evaluation

In this section, we test the proposed algorithms (SAC-CRM and TD3-CRM) in two continuous action domains, comparing their performance with existing algorithms (DDPG-CRM and HRM). The test environments are the Half-Cheetah (2D) and Ant (3D) domains in OpenAI Gym [2]. All CRM-based algorithms have a batch size of $100n$, where $n = |U|$ represents the number of non-terminal RM states. For HRM, option policies are learned using DDPG, and the high-level policy is learned using DQN. The batch size of HRM is $100n$, where n represents the available options. The neural networks for all algorithms use two hidden layers with 256 units and *RELU* activation functions.

In both domains, the agent's task involves reaching multiple points in a specific order, making the rewards non-Markovian. This tests each algorithm's ability to control the agent's movement by coordinating its limbs, as well as CRM's impact on task completion. The efficiency of the baseline algorithms determines the agents' movement speed, affecting the steps required to reach target points. Additionally, since the environments are RM environments with complex tasks,

it is difficult for the agent to learn to complete the task using only the baseline algorithm running over the cross-product of the environment and reward machine states. To substantiate this assertion, we conducted a performance comparison between the baseline SAC and DDPG running over the cross-product states versus their counterparts, SAC-CRM and DDPG-CRM, that generate counterfactual experiences. The test results (on the Half-Cheetah Tasks 1 and 2 below) revealed a significant performance boost when counterfactual experiences were utilized, thus underscoring their pivotal role in enhancing the efficacy of these algorithms; see [18] for details. CRM provides more specific task information, improving the agent's learning efficiency in completing multi-target point tasks.

4.1 Results in the Half-Cheetah Domain

In the Half-Cheetah domain, our first experimental environment, the agent is a cheetah-like robot with six joints, see Fig. 1. The robot must learn to control these joints to stand, move forward, or backward. It chooses how much force to apply to each joint per step, resulting in an infinite action space. The continuous state space includes each joint's location (coordinates' values on the plane) and velocity. We will test the new and existing algorithms on four tasks, including one original task defined by Toro Icarte *et al.* [22,20] (Task 1). The tasks are:

- **Task 1**: Starting between points A and B, first go to point B, then repeatedly go back and forth between A and B.
- **Task 2**: Starting between points A and B, first go to point A, then to B, then to C, then back to B, and then A, and repeat indefinitely.
- **Task 3**: Starting between points A and B, first go to point A, then to B, then C, then back to B, then to C again, then reach point D and stop.
- **Task 4**: Starting between points A and B, either go to point A or to B, then go to point C, and finally reach point D and stop.

We will use these tasks to test and compare the performance of our new algorithms, SAC-CRM and TD3-CRM, against [22]'s RM-based algorithms, DDPG-CRM and HRM. For all tasks, the agent starts from an arbitrary position between points A and B. Following the original approach in [22], to prevent the agent from ceasing exploration, the agent receives a small negative reward value, called *Control Penalty (CP)*, after each RM state transition.

Fig. 3 displays the performance of the evaluated algorithms. The horizontal axis represents the total number of training steps (three million), while the vertical axis indicates the total reward received by the agent within an episode (of 1,000 training steps). Different coloured lines represent the mean episode reward among 10 trials for each algorithm, and the shaded area indicates the range between the highest and lowest episode rewards for each trial. Note that the results for the first task with DDPG-CRM and HRM, initially presented by Toro Icarte *et al.* [22], were reconfirmed in our study. After repeating the task ten times, we found our results consistent with theirs.

It can be seen that SAC-CRM outperforms all other algorithms in this domain, exhibiting faster learning speeds and higher reward values. In Task 1,

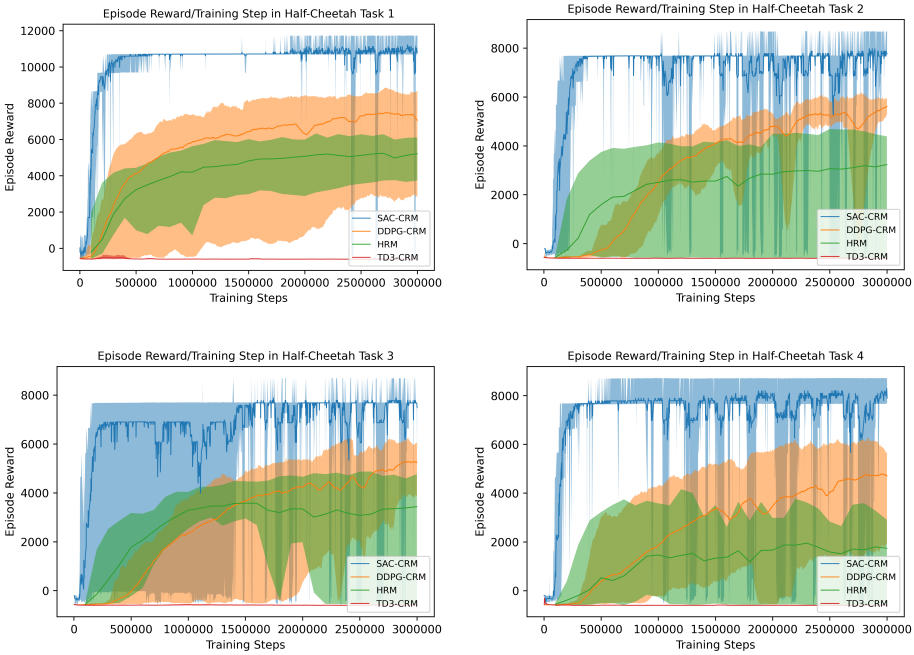

Fig. 3: Results in Half-Cheetah domain

SAC-CRM achieves the same performance level as the second-best performer, DDPG-CRM, in 150,000 training steps—up to 20 times faster. The mean reward value after two million training steps for SAC-CRM is about 30% higher than DDPG-CRM. With a highest episode reward of around 11,000, SAC-CRM can complete the task approximately 11 times in one episode, three more than DDPG-CRM. SAC-CRM also excels in the other tasks, demonstrating the fastest learning speed and highest episode reward.

Other algorithms do not perform as well as SAC-CRM. DDPG-CRM ranks second, with more stable learning curves than SAC-CRM. HRM performs reasonably well but has lower rewards than SAC-CRM and DDPG-CRM. TD3-CRM fails to complete the task within the training period (without counterfactual experiences, it works effectively in the easier tasks).

4.2 Results in the Ant Domain

We further tested the algorithms' performance in the Ant domain to increase the environment and task complexity. The Ant robot is a 3D robot with a torso and four legs, each with two links. The main goal is to coordinate the four legs by applying torques to the eight hinges, allowing movement in any direction on the plane. The state space (coordinates of the joints in 3D space) and action space (torque on the joints) are also continuous, similar to the Half-Cheetah domain.

The Ant domain was chosen for several reasons. First, it is a 3D environment, providing a larger moving space and more diverse states. Second, the Ant robot's higher number of joints requires more complex movement coordination, making learning more difficult. Consequently, the Ant domain is ideal for testing the performance of RM-based algorithms in a more complex environment.

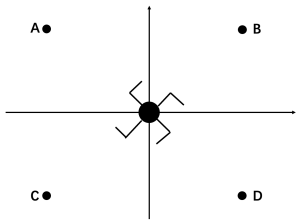

Fig. 4: The abstract representation of Ant

We will test three tasks in the Ant environment, which has designated points similar to the Half-Cheetah domain. Fig. 4 shows an abstract representation of the domain. In all three tasks, the ant robot starts at a random location near the origin:

- **Task 1**: Starting nearby the origin, go to point B, then repeatedly move between points A and B.
- **Task 2**: Starting nearby the origin, go to points A, B, and C sequentially, then back to B, A, and repeat indefinitely.
- **Task 3**: Starting nearby the origin, choose either point A or B, go to the chosen point, then to points C and D. From point D, return to the chosen point (A or B) and stop.

Fig. 5 displays the learning curves of all algorithms across tasks. Only SAC-CRM and DDPG-CRM achieve significant rewards within the specified learning steps, while the other algorithms do not.

It can be seen that SAC-CRM outperforms all other algorithms, demonstrating faster learning and higher reward values compared to DDPG-CRM. This is primarily due to SAC's greater exploration capability. In the Ant domain, the agent's movement expands to backward, forward, left, and right, increasing the movement space. SAC's high exploration tendency enables it to try new directions and explore joint coordination more effectively, improving movement speed faster than other algorithms. Conversely, DDPG-CRM's exploration rate diminishes as it learns, resulting in slower performance improvement.

Notably, HRM performs poorly in the Ant domain, with a significant performance gap compared to the Half-Cheetah domain. Although it quickly finds local optimal solutions, HRM's policies often get stuck in local optima. In the Half-Cheetah domain, lower environmental complexity allows the agent to rely

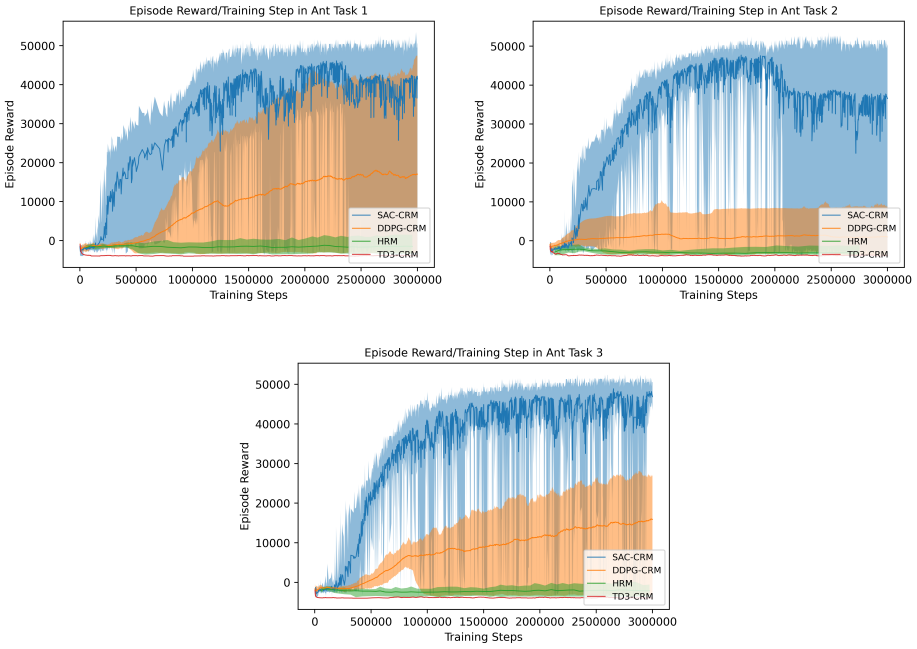

Fig. 5: Results in Ant domain

on local optimal policies for relatively high rewards. However, in more complex environments, the gap between local and global optimal policies widens, and local optimal policies become insufficient for obtaining high rewards, resulting in HRM's poor performance in the Ant domain.

Lastly, TD3-CRM's performance remains weak, similar to its results in the Half-Cheetah domain.

5 Discussion

Performance of SAC-CRM The experimental results reveal that SAC-CRM consistently outperforms the other tested RM-based algorithms across all tasks, demonstrating superior learning speed and policy quality. Its advantage is particularly pronounced in the more complex Ant domain. Therefore, SAC-CRM is deemed the optimal choice for all tasks in both Half-Cheetah and Ant domains explored in this paper.

The standout performance of SAC-CRM can be attributed mainly to its unique entropy-based policy update mechanism. In our continuous experimental environments, numerous action combinations influence the agent's movement, some leading to failure, some to slow progress, and others to rapid advancement. The entropy-based mechanism encourages extensive exploration, enabling the

policy to avoid early local optima and maintain high exploration levels while maximizing reward value.

Furthermore, SAC-CRM's success is bolstered by its generation of stochastic policies. In the continuous action domain, multiple optimal actions often exist in a specific state. Unlike deterministic policies, which limit the discovery of better action combinations by outputting a unique action, SAC-CRM saves all available actions in a given state, allocating their probabilities based on their Q-values. This broader 'vision' in action selection allows more frequent testing of different action combinations, accelerating learning for complex tasks such as multi-limb robot control.

Performance of DDPG-CRM Despite not achieving the highest rewards in most tasks, DDPG-CRM displays consistent performance, outshining other RM-based algorithms in stability, especially compared to SAC-CRM.

However, DDPG-CRM's drawbacks include its slow learning speed and lower rewards compared to SAC-CRM. The experimental results show that in all tasks, DDPG-CRM lags behind SAC-CRM in both learning speed and reward achieved. This gap widens in the more complex Ant domain. Due to its deterministic policy learning, DDPG-CRM is less exploratory, limiting the agent's capacity to seek better action combinations. Moreover, its traditional Q-function-based learning may overestimate Q-values, causing premature convergence to a local optimum.

In summary, DDPG-CRM is robust and stable, making it a viable choice for simpler environments like Half-Cheetah when stability is more important than optimal performance. However, its reward output falls short compared to SAC-CRM, making it less suitable for complex environments like the Ant domain.

Performance of HRM The experimental results show that HRM performs reasonably well in the Half-cheetah domain. While not as efficient as SAC-CRM, it achieves comparable performance to DDPG-CRM in learning speed and reward, and it also often surpasses DDPG-CRM in early training stages.

However, HRM's performance declines sharply in the more complex Ant domain. This is due to HRM's predisposition to find local optima. HRM has no guarantee of convergence to a global optimum, which becomes problematic as the gap between local and global optima widens in complex environments. Thus, while HRM performs well in low-complexity settings, it becomes less effective in more complex environments. Note that HRM's performance depends on it using a good decomposition for the task; but we think that the RM-based task decompositions are reasonably good for our test tasks.

Performance of TD3-CRM TD3-CRM's performance in all tasks is far from ideal, earning the lowest rewards among all the evaluated algorithms. The reason appears to be a conflict between CRM and TD3's policy updating mechanism.

First, TD3-CRM assigns the lower Q-value to an action using two learned Q-functions. This works well in MDPs, where Q-values are often overestimated,

but not in MDPRMs. In an MDPRM, the RM information is critical for the agent to transfer from one RM state to another; specifically, the actions that can make the RM state change usually have high Q-values, which encourage the agent to keep using these actions to make transitions between the RM states. Nevertheless, TD3 always tries to "underestimate" the Q-values, which will avoid these beneficial actions.

Furthermore, TD3's target policy smoothing regularization, which adds noise to actions, restricts optimal action selection and steers the agent towards close alternatives instead. This contrasts with CRM's encouragement for the agent to execute optimal actions that trigger RM state changes. Consequently, this contradiction confuses the agent, causing infrequent correct actions and resulting in poor performance.

6 Conclusion

Training a practical deep RL agent for specific scenarios typically requires extensive training data and time. Furthermore, agents often face complex tasks with non-Markovian rewards, making learning high-quality policies from limited information a significant challenge. Therefore, observing more information and fully utilizing it is crucial for improving training efficiency.

Our research is inspired by previous work on reward machines and deep RL algorithms, particularly the work by Toro Icarte *et al.* [22,20]. Our contributions include extending two mainstream deep RL algorithms, SAC and TD3, to exploit reward machine models and counterfactual experiences, yielding two new reward machine-based algorithms, SAC-CRM and TD3-CRM. In order to simulate the tasks that an intelligent agent might encounter in the real world, we introduced seven different task types in two simulated continuous action domains. We evaluated experimentally the performance of all RM-based deep RL algorithms across these tasks. We found that the newly proposed SAC-CRM performed best in most tasks.

For future work, there are several key areas of focus. First, more extensive parameter tuning could potentially enhance algorithm performance, as the current uniform parameters may not allow for optimal performance. Second, expanding experimental evaluations to include a wider variety of tasks and domains would allow for more comprehensive robustness testing and a better understanding of the environments and tasks best suited for each algorithm. Third, finding ways to stabilize the policies of SAC-CRM, which currently fluctuate in learning curves across tasks, could make it a more robust algorithm. Fourth, incorporating Automated Reward Shaping [14] into CRM-based algorithms may further improve learning speed by providing intermediate rewards for subtask completion. Fifth, it's worth exploring ways to combine CRM and on-policy deep RL algorithms such as PPO [17] and TRPO [16]. This could further expand the use cases for RM and CRM. Lastly, applying these RL algorithms to real-world hybrid domains, which involve both discrete and continuous decision variables, could offer more

practical solutions to real-world problems, expanding their usability beyond the purely continuous control problems they currently address.

Acknowledgements

Work supported by the National Science and Engineering Research Council of Canada and York University.

References

1. Brafman, R.I., Giacomo, G.D., Patrizi, F.: LTLf/LDLf Non-Markovian Rewards. In: McIlraith, S.A., Weinberger, K.Q. (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. pp. 1771–1778. AAAI Press (2018), https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17342
2. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym. CoRR **abs/1606.01540** (2016), http://arxiv.org/abs/1606.01540
3. De Giacomo, G., Iocchi, L., Favorito, M., Patrizi, F.: Restraining bolts for reinforcement learning agents. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020. pp. 13659–13662. AAAI Press (2020), https://ojs.aaai.org/index.php/AAAI/article/view/7114
4. De Giacomo, G., Vardi, M.Y.: Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In: Rossi, F. (ed.) IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013. pp. 854–860. IJCAI/AAAI (2013), http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997
5. Feinberg, A.: Markov Decision Processes: Discrete Stochastic Dynamic Programming (Martin L. Puterman). SIAM Rev. **38**(4), 689 (1996). https://doi.org/10.1137/1038137, https://doi.org/10.1137/1038137
6. Fujimoto, S., van Hoof, H., Meger, D.: Addressing Function Approximation Error in Actor-Critic Methods. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 1582–1591. PMLR (2018), http://proceedings.mlr.press/v80/fujimoto18a.html
7. Guillen-Perez, A., Cano, M.: Learning from oracle demonstrations - A new approach to develop autonomous intersection management control algorithms based on multiagent deep reinforcement learning. IEEE Access **10**, 53601–53613 (2022). https://doi.org/10.1109/ACCESS.2022.3175493, https://doi.org/10.1109/ACCESS.2022.3175493

8. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 1856–1865. PMLR (2018), http://proceedings.mlr.press/v80/haarnoja18b.html
9. Hasanbeig, M., Kroening, D., Abate, A.: LCRL: certified policy synthesis via logically-constrained reinforcement learning. In: Ábrahám, E., Paolieri, M. (eds.) Quantitative Evaluation of Systems - 19th International Conference, QEST 2022, Warsaw, Poland, September 12-16, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13479, pp. 217–231. Springer (2022). https://doi.org/10.1007/978-3-031-16336-4_11, https://doi.org/10.1007/978-3-031-16336-4_11
10. van Hasselt, H., Guez, A., Silver, D.: Deep Reinforcement Learning with Double Q-Learning. In: Schuurmans, D., Wellman, M.P. (eds.) Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. pp. 2094–2100. AAAI Press (2016), http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389
11. Lacerda, B., Parker, D., Hawes, N.: Optimal policy generation for partially satisfiable co-safe LTL specifications. In: Yang, Q., Wooldridge, M.J. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. pp. 1587–1593. AAAI Press (2015), http://ijcai.org/Abstract/15/227
12. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: Bengio, Y., LeCun, Y. (eds.) 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016), http://arxiv.org/abs/1509.02971
13. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M.A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nat. **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236, https://doi.org/10.1038/nature14236
14. Ng, A.Y., Harada, D., Russell, S.: Policy invariance under reward transformations: Theory and application to reward shaping. In: Bratko, I., Dzeroski, S. (eds.) Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999. pp. 278–287. Morgan Kaufmann (1999)
15. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977. pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32, https://doi.org/10.1109/SFCS.1977.32
16. Schulman, J., Levine, S., Abbeel, P., Jordan, M.I., Moritz, P.: Trust region policy optimization. In: Bach, F.R., Blei, D.M. (eds.) Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015. JMLR Workshop and Conference Proceedings, vol. 37, pp. 1889–1897. JMLR.org (2015), http://proceedings.mlr.press/v37/schulman15.html
17. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR **abs/1707.06347** (2017), http://arxiv.org/abs/1707.06347

18. Sun, H.: Exploiting Reward Machines with Deep Reinforcement Learning in Continuous Action Domains. Master's thesis, EECS Dept., York University, Toronto, Canada (2022)
19. Sutton, R.S., Precup, D., Singh, S.: Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. Artif. Intell. **112**(1-2), 181–211 (1999). https://doi.org/10.1016/S0004-3702(99)00052-1, https://doi.org/10.1016/S0004-3702(99)00052-1
20. Toro Icarte, R.: Reward Machines. Ph.D. thesis, University of Toronto, Canada (2022), http://hdl.handle.net/1807/110754
21. Toro Icarte, R., Klassen, T.Q., Valenzano, R.A., McIlraith, S.A.: Using reward machines for high-level task specification and decomposition in reinforcement learning. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 2112–2121. PMLR (2018), http://proceedings.mlr.press/v80/icarte18a.html
22. Toro Icarte, R., Klassen, T.Q., Valenzano, R.A., McIlraith, S.A.: Reward machines: Exploiting reward function structure in reinforcement learning. J. Artif. Intell. Res. **73**, 173–208 (2022). https://doi.org/10.1613/jair.1.12440, https://doi.org/10.1613/jair.1.12440
23. Watkins, C.J.C.H., Dayan, P.: Q-Learning. Mach. Learn. **8**, 279–292 (1992). https://doi.org/10.1007/BF00992698, https://doi.org/10.1007/BF00992698