Reward Machines: Effects of Noisy Labelling Functions in Complex Grid Environments in Reinforcement Learning

Bachelor's Project Thesis

Andro Erdelez, s4297237, a.erdelez@student.rug.nl,
Supervisor: prof D. Grossi

Abstract: Reinforcement learning (RL) commonly treats reward functions as black boxes, requiring extensive interaction with the environment for discovering rewards. Recently, a novel approach called reward machines (RM) has been introduced. These are finite state machines that make reward functions explicit, exploiting their internal structure. Innovative algorithms that enhance RL efficiency using reward machines have emerged, demonstrating better performance than standard Q-learning. These RM algorithms rely heavily on a labeling function to determine the next state in a reward machine. However, it is assumed that the event detectors, defining the labeling function, operate perfectly, which is rarely the case in real-world scenarios. Moreover, these algorithms are tested with reward machines that do not provide much free choice to the RL agent. This thesis addresses two main research questions: (1) How does noise at different levels affect the performance of the RM algorithms when introduced to the labeling function? and (2) How do these RM algorithms perform in a complex grid environment with significant free choice for the RL agent? To answer these questions, different levels of noise are introduced into the labeling function and the RM algorithms are evaluated in a new complex grid environment. Our results show the extent to which noise affects the RM algorithms' performance, robustness and adaptability under more realistic conditions. These contributions enhance the understanding of using reward machines in RL and identify potential problems that can be encountered in practical applications.

1 Introduction

The usual RL problem involves an agent interacting with its environment (Sutton & Barto, 2018). For each action taken by the agent, the environment sends feedback to the agent as a reward which is generated by a reward function. In this manner, the agent accumulates experience for learning the most optimal policy. Typically, a common assumption is that the reward function is obscure to the agent and, hence, treated as a black box. However, considering that the reward function is always designed by the programmer due to the fact that usually there are no real-world applications where the environment naturally provides rewards, the internal structure of the reward function could be exposed to possibly facilitate more efficient learning by the agent.

Previous research has focused on utilizing reward specifications through task definitions. Singh (1992a, 1992b) defined tasks based on sub-goal sequences whilst other research has given more emphasis on defining tasks using linear temporal logic (Li et al., 2017; Littman et al., 2017; Toro Icarte et al., 2018a; Hasanbeig et al., 2018; Camacho et al., 2019; De Giacomo et al., 2019; Shah et al., 2020). However, these previous works focused on generating the reward function rather than exposing it. In order to expose the reward function, one needs to use reward machines.

Reward machines are finite state machines which expose the internal structure of a reward function. They were first introduced by Icarte et al. (2018b) who have proposed an extension of Q-learning algorithm, Q-learning for reward machines (QRM), as the first algorithm that can exploit the reward machine structure. The results showed that algorithms that use reward machines (i.e. QRM) learn much faster optimal policies compared to the standard RL algorithms such as Q-learning. Further research on reward machines (Icarte et al., 2022) found new ways to utilize reward machines by combining established RL techniques and algorithms with the capabilities offered by the reward machines.

Given that the reward machines are based on the finite state machines, they rely on propositional symbols for enabling state transitions. These propositional symbols are formal description of events that can occur in agent's environment. Event detectors identify these events which are converted to their formal description by a labelling function. The labelling function in the aforementioned reward machine research is an optimal function whose event detectors have 100% accuracy in detecting correct events in the agent's environment. However, in the real-world applications such function is not always flawless and can encounter noise to some extent. Hence, the first research question of this thesis is: how does noise at different levels affect the performance of the RM algorithms when introduced to the labeling function?

Furthermore, the reward machines in Icarte's most recent research (Icarte et al., 2022) do not offer significant free choice to the agent in discrete domains. That means that the RM algorithms have not been tested thoroughly in highly-complex environments. Therefore, the second research question of this thesis is: How do these RM algorithms perform in a complex grid environment with significant free choice for the RL agent?

Now knowing what our research is, before delving into our methodology, we will cover the necessary background for understanding our research in the next section.

2 Preliminaries

In order to understand the idea behind the reward machines and its algorithms, we need to define the RL problem that reward machines tackle. In the standard RL problem, there is an RL agent which interacts with the environment over a series of discrete time steps. A classical formalization of agent's sequential decision making, known as Markov decision process, is usually used to model the environment. A Markov decision process is a tuple $\langle S, A, r, p, \gamma \rangle$ where S denotes a set of states, A denotes a set of actions, r is the reward function with signature $S \times A \times S \rightarrow \mathbb{R}$, $p(s_{t+1}|s_t, a_t)$ is the probability distribution of possible transitions and $\gamma \in (0, 1]$ is the discount factor.

The agent follows a certain structure in order to interact with the environment. The agent is situated in state s at time step t where it chooses a next action a in order to move to a new state s' at time step $t+1$. It selects its next action by following a policy $\pi(\cdot|s)$, a probability distribution of actions $a \in A$ from state s. Once action a has been chosen, the agent executes it and ends up in a new state s' according to a transition probability $p(\cdot|s, a)$ and receives a reward $r(s, a, s')$ from the environment as an evaluation of its most recent choice. The described process continues until the agent reaches a terminal state which marks the end of an episode.

To keep track of agent's progress in the environment, a Q-value $q^{\pi}(s, a)$ is defined as the expected utility of taking action a in state s following a policy π. The objective of the agent is to discover an optimal policy π^* that maximizes the expected discounted cumulative reward $G_t = E_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | S_t = s]$ from any state $s \in S$ at any time step t. Once the optimal policy has been discovered, it will satisfy the Bellman optimality equations for every state $s \in S$ and action $a \in A$:

$$\begin{aligned} q^{\pi^*}(s, a) = \sum_{s' \in S} p(s'|s, a)(r(s, a, s') + \\ \gamma \max_{a' \in A} q^{\pi^*}(s', a')) \end{aligned} \tag{2.1}$$

This will ensure that the best possible action will be chosen in any state $s \in S$.

2.1 Tabular Q-learning

Considering that this research focuses only on the grid environments which are discrete domains, an RL approach known as tabular Q-learning (Watkins & Dayan, 1992) will be used. Tabular Q-learning is an off-policy* learning method that evaluates each state-action pair using Q-values estimations $\tilde{q}(s, a)$, generated by the agent's experience that is utilized to find the most optimal policy.

*I.e. it can learn from experience generated by any policy.

When the agent is in a state s, it takes an action a based on some exploratory policy. The exploratory policy that will be used in our research is ϵ-greedy policy where the agent takes the most optimal action a following the highest Q-value $\tilde{q}(s,a)$ with the probability $1-\epsilon$; otherwise it takes a random action (probability ϵ). Once the agent receives the resulting state s' and its reward $r(s,a,s')$, the Q-value estimations $\tilde{q}(s,a)$ are updated as follows:

$$\begin{aligned}\tilde{q}(s,a) \leftarrow \tilde{q}(s,a) + \alpha \cdot (r(s,a,s') + \\ \gamma \max_{a'} \tilde{q}(s',a') - \tilde{q}(s,a))\end{aligned} \tag{2.2}$$

where α is a hyperparameter called learning rate which regulates how much new experiences change $\tilde{q}(s,a)$.

Tabular Q-learning is assured to converge to the most optimal policy, ensuring that with infinite visits to all state-action pairs, the agent will always discover the best solution.

2.2 Reward machines

Reward machines (Icarte et al., 2022) are finite state machines where states represent a specific atomic step of a given task in an environment and transitions between the states are triggered by events. As a result, the reward machines take the events as an input, together with a current state-action pair, outputting an appropriate reward function.

Reward machines are defined as a tuple $\langle U, u_0, F, \delta_u, \delta_r\rangle$ where, given a set of states S, a set of actions A and a set of propositional symbols $\mathcal{P}$, U denotes a set of RM states, u_0 denotes an RM initial state which is an element of the set U, F denotes a set of terminal states such that $U \cap F = \emptyset$, $\delta_u : U \times 2^{\mathcal{P}} \to U \cup F$ denotes the function describing state transitions, and $\delta_r : U \to [S \times A \times S \to \mathbb{R}]$ denotes the function outputting an appropriate reward function. Additionally, a labelling function L is defined as follows: $L : S \times A \times S \to 2^{\mathcal{P}}$, which is responsible for assigning truth values to propositional symbols in $\mathcal{P}$ based on the agent's experience (s,a,s').

However, Icarte et al. (2022) have proven that a particular case of reward machine, namely simple reward machine, showcases the same capabilities as classic reward machines, and have used it in their own research. Therefore, we will be using only simple reward machines in our research as well and we will refer to them as reward machines. Simple reward machine is a tuple $\langle U, u_0, F, \delta_u, \delta_r\rangle$ where, given a set of propositional symbols $\mathcal{P}$, everything is defined as in classic reward machines, except δ_r whose signature is different: $\delta_r : U \times 2^{\mathcal{P}} \to \mathbb{R}$. Note, how now $delta_r$ also depends on $2^{\mathcal{P}}$ and, instead of returning a function, returns a number. Moreover, as input, the reward machine only needs a propositional symbol(s) describing an event(s). An example of a reward machine is shown in Figure 3.3. An example of how a reward machine works in a specific environment will be given in Section 3.5.

2.3 RM algorithms

In order to utilize reward machines, Icarte et al. (2022) constructed the following algorithms: cross-product baseline (QL), counterfactual experiences for reward machines (CRM) and hierarchical RL for reward machines (HRM). Furthermore, a technique known as automated reward shaping (RS) has been used as a complement to the aforementioned algorithms[†]. These algorithms and techniques were used in our research and are discussed in more detail in the following subsections.

2.3.1 Cross-product baseline

Cross-product baseline is an RM algorithm that can be identified as standard Q-learning which utilizes reward machines. To make use of reward machines, cross-product baseline extends Q-value estimations with one additional entry, RM state u, leading from a 2D array storing each $\tilde{q}(s,a)$ to 3D array storing each $\tilde{q}(s,u,a)$. Note how the algorithm uses reward machines only to extract its reward, not exploiting the exposed structure of the reward specification. The algorithm's pseudocode can be found in Figure B.1 in Appendix B.

2.3.2 CRM

CRM is an RM algorithm that is an extension of cross-product baseline. Unlike cross-product baseline where the agent only receives experience for the current RM state, in CRM the agent generates

[†]If an algorithm A uses automated reward shaping, its abbreviation is A-RS

synthetic experiences for each RM state. These experiences are used to update Q-value estimation for each RM state, enabling more efficient learning to the agent by completely utilizing reward machines' structure. The algorithm's pseudocode can be found in Figure B.2 in Appendix B.

2.3.3 HRM

HRM is an RM algorithm that is based on the idea of hierarchical RL, namely the options framework (Sutton et al., 1999). The underlying concept of HRM is that a complex problem can be divided into simple subproblems which are easier to solve and less time-consuming for the agent. This introduces a hierarchy of policies where a high-level policy is a policy that chooses which subproblem is going to be solved while a low-level policy is a policy that is used to solve the chosen subproblem. In HRM (based on the options framework), these subproblems, called options, are formally described as a tuple $\langle \mathcal{I}, \pi, \beta \rangle$ where $\mathcal{I}$ denotes a subset of the state space where the option can be initiated, π is the low-level policy that selects actions while the option is executed, and β is the likelihood of the option ending in each state. β is defined as follows in HRM:

$$\beta_{\langle u,u_t \rangle}(s', u') = \begin{cases} 1 & \text{if } u' \neq u \text{ or } s' \text{ is terminal} \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

where $\langle u, u_t \rangle$ is an option being followed. Additionally, the low-level policy is trained using the following reward function:

$$r_{u,u_t}(s, a, s') = \begin{cases} \delta_r(u)(s, a, s') + r^+ & (1) \\ \delta_r(u)(s, a, s') + r^- & (2) \\ \delta_r(u)(s, a, s') & (3) \end{cases} \tag{2.4}$$

where r^+ and r^- are hyperparameters, (1) is *if* $u_t \neq u$ *and* $u_t = \delta(u, L(s, a, s'))$, (2) is *if* $u_t \neq u$ *and* $u_t \neq \delta(u, L(s, a, s'))$, and (3) is *otherwise*. The algorithm's pseudocode can be found in Figure B.3 in Appendix B.

2.3.4 Automated reward shaping

Automated reward shaping is a technique that focuses on providing intermediary rewards to the agent to facilitate easier and more efficient learning. The concept of reward shaping (Ng et al., 1999) is based on the idea that certain reward functions make it easier for an agent to learn effective policies, even if these reward functions ultimately lead to the same optimal policy as the original, unshaped reward function. Hence, the reward function is defined as the following:

$$r'(s, a, s') = r(s, a, s') + \gamma\Phi(s') - \Phi(s) \tag{2.5}$$

where Φ denotes a potential function with signature $S \to \mathbb{R}$ that promotes learning optimal policies faster. Icarte et al. (2022) used value iteration over RM states as a potential function and, therefore, we will as well in order to have the same experimental setup as them. The pseudocode for value iteration can be found in Figure B.4 in Appendix B.

3 Methods

In order to answer the first research question, noise needed to be infused in the labelling function of the reward machine. To answer the second question, a complex environment was created where more decision-making was required from the agent. Moreover, once the agent was tested on this environment with the optimal labelling function (no noise), it was tested with different levels of noise, following the same experimental structure of the first research question. This section discusses these methodologies in more detail.

3.1 Modernizing the system

All of the aforementioned preliminaries have been already programmed by Icarte et al. (2022). Therefore, the system that was used for this research is an extension of the system created by Icarte et al. (2022) and can be found at `https://github.com/aerdelez/reward_machines`. However, the original system used an outdated version of Python, namely *Python 3.6*, and, thus, it had to be modernized to meet the standards of modern *Python 3.11* and have the most up-to-date libraries for the experiment.

This proved to be a challenge due to the fact that the `baselines` library, whose `Logger` class is utilized to keep track of agent's progress, has not been updated since 2020 and, hence, the most recent version of Python accepted by it is *Python 3.7*. To solve this problem, `baselines` library was replaced with the `stable-baselines3` library, improved version of the `baselines` library that uses up-to-date Python. Unfortunately, this library is not complete meaning that its usage led to unexpected behavior of logging every single detail of the agent's progress as in the original experiment (Icarte et al., 2022). Nonetheless, it did capture agent's most important milestones (progress on every 10000th step) which were used to calculate agent's performance. This led to our replicated results being marginally different than the original results. These differences are thoroughly discussed in Section 4.

Next to this, we encountered problems with the `gym`[‡] library's functionality within the system where old functions had to be replaced with updated functions. All outdated functions were replaced, but there were some discrepancies in the `step`[§] function within the `gym` library itself due to the system having environments which were many-layered extensions of the original `gym`'s environment class. The reason behind these discrepancies was the usage of the old `step` API[¶] with the combination of the new `step` API. The new `step` API has been used only for QL which required no additional environment wrappers, whereas the old `step` API has been used for the other RM algorithms. However, this proved to have no effect on the final results because the new `step` API has an additional return argument which is not utilized by the original experiment, thus not being utilized by this paper as well.

3.2 Implementing noise

In our methodology we want to introduce different levels of noise to the labelling function in order to see how they affect the RM algorithms and whether RM algorithms that utilize more reward machines are more robust than the basic QL. In our experiment, there are 10 different levels of noise, ranging from 0 to 90 in increments of 10. These levels represent the probability of noise affecting the labelling function, e.g. 30 means that there is 30% chance of having noisy detection of the events.

The method of introducing noise into the labeling function does not change the labelling function itself; in fact, the labelling function always produces the correct output. Based on the given noise probability, once the labelling function outputs a propositional symbol, which describes the event in the agent's current state, it might change that propositional symbol. The probability of changing the output is generated using the Mersenne Twister pseudorandom number generator algorithm (Matsumoto & Nishimura, 1998) which generates integers ranging from 0 to 99. For example, when the noise probability is 30%, if the generated integer is lower than 30, then a random propositional symbol will be generated. The random propositional symbol is generated from the pool containing all propositional symbols of the given environment using the same pseudorandom number generator algorithm. When generating the propositional symbol, each propositional symbol is marked as an integer, ranging from 0 to $n - 1$, where n denotes the total number of distinct propositional symbols in the given environment. If no events have been detected originally, that state will not trigger the random event detection.

3.3 Environments

We tested our RM algorithms on two environments, namely the Office world and the Mordor world. These are 12×9 grid environments where each square is a possible state. Bold lines mark the walls while symbols in some of the squares are propositional symbols that represent the events. The agent can move in any cardinal direction (diagonal movements are forbidden) as long as it does not hit the wall. In both worlds, the agent's starting point is marked with X. In the following subsections, we will discuss these environments in detail.

3.3.1 Office world

The Office world (Icarte et al., 2022) mimics an environment where the agent is a laborer who works in an office and has daily tasks that it needs to perform. It consists of 12 3×3 rooms in which is possible to encounter a specific event in specific

[‡]A library that provides environments for RL algorithms.

[§]A function that enables agent's interaction with an environment.

[¶]An API that offers the functionality of the `step` function.

squares. There are eight events which the agent can encounter: it can pick up coffee (marked with c), it can pick up mail (marked with m), it can break decorations (marked with d), it can visit its office (marked with o), and it can visit environment landmarks (marked with A, B, C and D). For example, a task that can be given to the agent is to pick up a coffee and bring it to its office whilst avoiding breaking decorations. Figure 3.1 depicts such task in the Office world. In the figure, agent has two possible paths to take in order to achieve its goal due to the fact that coffee can be found in two different rooms. It is agent's goal to find the most optimal path, in this case the green path, which maximizes the cumulative reward that is gained from the reward machines.

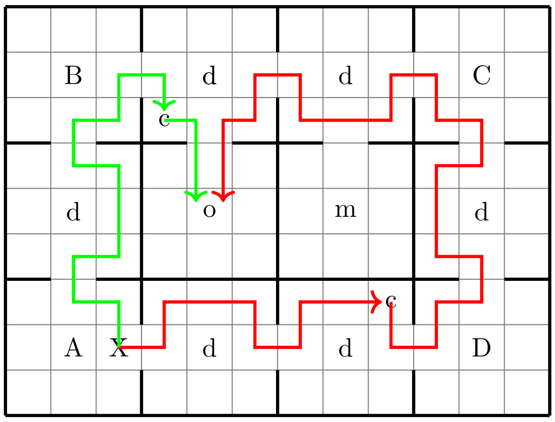

Figure 3.1: Office gridworld.

3.3.2 Mordor world

The Mordor world is an environment inspired by the *Lord of the Rings* series where the agent is a protagonist who needs to complete certain tasks before it leaves the world of Mordor. It consists of rooms of random shape which contain walls in random locations, and in which is possible to encounter a specific event in specific squares. There are six events which the agent can encounter: it can leave the world of Mordor (marked with E), it can get killed by an orc[‖] (marked with O), it can collect a prison key (marked with K), it can save its companion Sam from the prison (marked with S), it can collect a ring (marked with R), and it can visit a volcano (marked with V). For example, a task that can be given to the agent is to pick up the ring and destroy it in the volcano whilst avoiding orcs. Figure 3.2 depicts such task with red arrow in the Mordor world. Next to that task, the agent can also have a task where it needs to pick up the prison key and save its companion Sam (marked with red arrow in Figure 3.2). It is the agent's goal to learn which order of tasks yields the highest cumulative reward.

‖A fantasy creature.

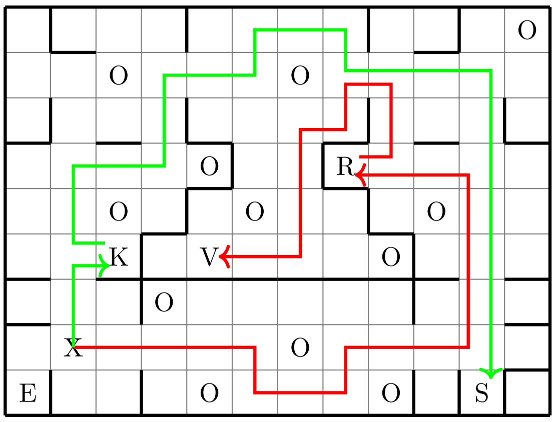

Figure 3.2: Mordor gridworld.

3.4 Reward machines

Unlike in usual RL tasks where the agent relies on opaque reward function, in the aforementioned environments the agent receives reward through reward machines. Reward machines depict important events during the agent's episode, i.e. necessary steps that the agent needs to take in order to reach its goal and steps that need to be avoided. This also means that atomic steps in the tasks can be viewed as states in the reward machines. The information that the reward machines convey is exploited by each RM algorithm in this experiment except QL. Hence, their careful construction is crucial.

The reward machines for the Office world were already constructed (Icarte et al., 2022). There are four reward machines, each depicting four possible tasks that can be assigned to the agent. These tasks can be found in Table 3.1, assigned to its respective reward machine. For example, for the task where the agent needs to collect the mail and go to the office, the reward machine in Figure A.4 feeds the agent with the necessary experience for succeeding its goal. The most complex task (the third task from Table 3.1) is covered by the reward machine in Figure 3.3. In that task, the agent has a choice between collecting either the mail or the coffee first. Throughout episodes, the generated experience from the reward machine will teach the agent that collecting the coffee first is more optimal due to the fact that the coffee is on the way to the room where the mail is located.

RM	Office world tasks
#1	Collect coffee and go to the office.
#2	Collect the mail and go to the office.
#3	Collect coffee, collect the mail and go to the office.
#4	Visit landmark A, visit landmark B, visit landmark C and visit landmark D.

Table 3.1: Possible tasks in the Office world with their respective reward machine (RM).

For the Mordor world, there are three reward machines, each depicting three possible tasks that can be assigned to the agent. These tasks can be found in Table 3.2, assigned to its respective reward machine. For example, for the task where the agent needs to collect the ring, destroy it in the volcano and leave Mordor, the reward machine in Figure A.2 feeds the agent with the necessary experience for succeeding its goal. Each atomic step in a task is ordered for every task except the third task from Table 3.2. The aforementioned task is the most complex task and is covered by the reward machine in Figure 3.3. In that task, due to the fact that it is a mixture of tasks covered by the reward machines #1 and #2 and the tasks are not ordered, the agent has significant free choice. Furthermore, because tasks consist of more atomic steps than tasks from the Office world, the complexity of the agent's choices is much higher, i.e. it leads to more possibilities. Note that the agent cannot destroy the ring in the volcano if it has not collected the ring, nor it can save Sam from the prison if it has not collected the prison key. Therefore, for some atomic steps in this task, the order still matters.

RM	Mordor world tasks
#1	Collect the ring, destroy it in the volcano and leave Mordor.
#2	Collect the prison key, save Sam and leave Mordor.
#3	Collect the ring, collect the prison key, destroy the ring in the volcano, save Sam and leave Mordor.

Table 3.2: Possible tasks in the Mordor world with their respective reward machine (RM).

3.5 Design

To grasp the complete idea of the design of our methodology, it will be explained using an example. The agent is situated in the Mordor world (Figure 3.2) with a staring position in the square marked with X. Its goal is to accomplish the third task from Table 3.2 with the highest cumulative reward using the information generated from the reward machine in Figure 3.3. The RM algorithm selected for the agent's training is CRM, meaning that at each step the reward machine will feed the agent with one experience per RM state. The noise probability of 10% will be incorporated into the labelling function, i.e. the labelling function will have a probability of 90% of the correct event detection. In one test run, the agent has a limit of 100000 steps, the same limit as in Icarte's paper (Icarte et al., 2022) because we wanted to have the same conditions for the both environments.

According to the reward machine, the agent is in state q_0 and it needs to either find the prison key or the ring in order to achieve progress (move to the next state) while avoiding the orcs. To get to the key, it would follow the green path from Figure 3.2. However, once the agent gets to the K square, in this episode the labelling function gets affected by noise, detecting an orc instead of the key. Therefore, the first episode terminates with zero reward (failure) due to the reward machine having no possible transitions from state q_0 (each transition has negated propositional symbol O).

The second episode starts with the agent attempting to get the ring instead due to the bad experience it received in the previous episode. Hence, it follows the red path from Figure 3.2 until it reaches the ring. When it is in the square R, this

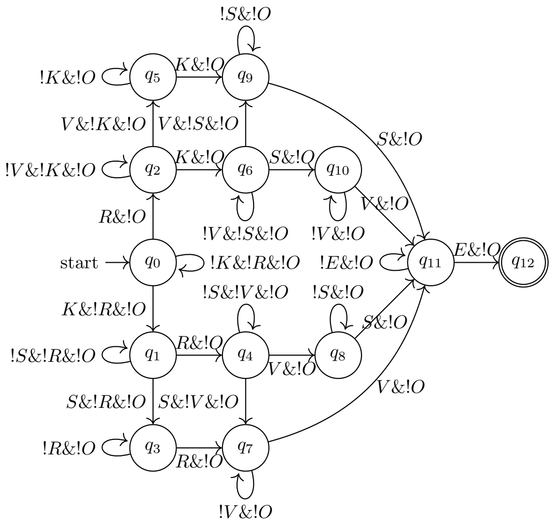

Figure 3.3: RM #3 in the Mordor world. Its (terminal) states are defined as follows: $U = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}\}$, $F = \{q_{12}\}$. Each edge represents a possible transition and their labels follow a specific notation that describes conditions that need to be satisfied in order for the transition to execute. Each transition returns a reward of 0; the only exception is the transition $E\&!O$ which returns a reward of 1. To explain the specific notation, here is an example: the transition $K\&!R\&!O$ that returns a reward of 0 would be translated to a tuple $\langle K \wedge \neg R \wedge \neg O, 0\rangle$ where the first tuple element represents the condition, and the second tuple element represents the reward.

time the labelling function gives the correct output, the propositional symbol R. That makes the agent transition from state q_0 to state q_2, getting new options to choose. This continues until the agent either has no possible transitions in the reward machine (failure) or reaches the terminal state q_{12} where it receives the reward of value one. Note that self-loops in each RM state make sure that the agent finds the appropriate event in that moment while avoiding the orcs.

Once the agent has reached its step limit, its RM algorithm, in this case CRM, is evaluated. The evaluation, together with the hyperparameters' values, is discussed in the next section, namely Section 4.

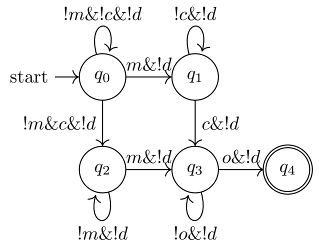

Figure 3.4: RM #3 in the Office world. Refer to Figure 3.3 for the notation explanation. Note that each transition returns a reward of 0; the only exception is the transition $o\&!d$ which returns a reward of 1.

4 Results

In this section, the evaluation of our methods and their initial settings will be discussed, together with the results they produce and consequences of the modernization of the system with no noise induced into the labelling function. Once that has been dealt with, the methods will be evaluated on the different levels of noise and their difference will be interpreted through performance plots.

Our research questions were tested on two grid-worlds, namely the Office world and the Mordor world. In order to evaluate the questions, 60 independent runs were conducted, each requiring the agent to solve one or more tasks: single-task environments are an environment variation that required solving only one task, while multi-task environments are an environment variation that required solving all available tasks in that environment. All environments were discrete domains, meaning that tabular Q-learning served as the fundamental off-policy learning method for all the RM algorithms. The RM algorithms that were used in the runs are the following: the cross-product baseline (QL), the cross-product baseline with automated reward shaping (QL-RS), Q-learning with counterfactual experiences (CRM), Q-learning with counterfactual experiences and automated reward shaping (CRM-RS), hierarchical RL with reward machines (HRM) and hierarchical RL with reward machines and automated reward shaping (HRM-RS). As for the hyperparameters, $\alpha = 0.5$ and $\gamma = 0.9$ were used for adjustment of learning speed, and $\epsilon = 0.1$ was used for exploration. In the case of HRM and HRM-RS, $r^+ = 1$ and $r^- = 0$ were used. Additionally, every initial Q-value was set to

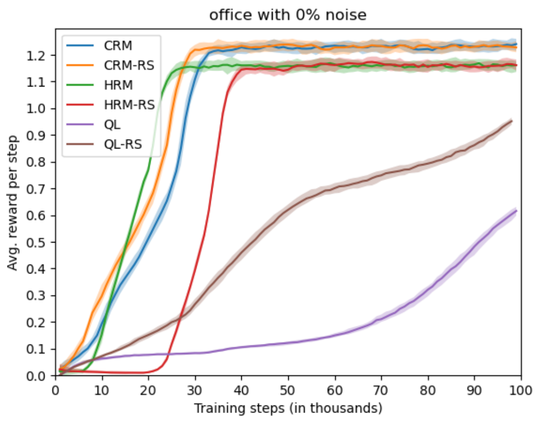

Figure 4.1: Replicated results for the Office world (multi-task variation) with no noise. The line represents medians over the 60 independent runs per step. The shaded area represents the area between their 25th and 75th percentiles.

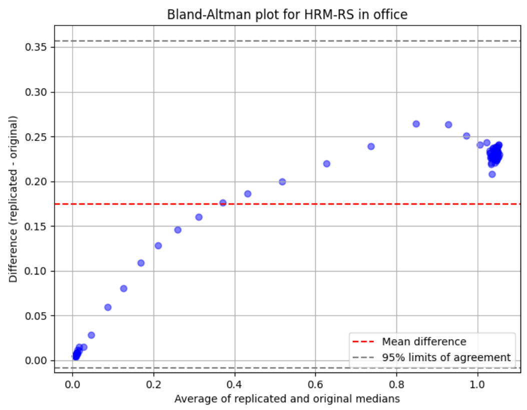

Figure 4.2: Bland-Altman plot for HRM-RS in the Office world (multi-task variation).

an optimistic value of 2. In order to get the performance plots for the algorithms with a certain level of noise, the average reward per step was normalized to be 1 for an optimal policy, pre-calculated using value iteration, and the median performance over the 60 runs was shown. The 25th and 75th percentiles were also shown in the shadowed area to understand the variability and range of rewards per step.

4.1 Office world results

Considering that the Office world is from the original experiment (Icarte et al., 2022), once we had modernized the system, we replicated the results from their experiment to see whether the system will produce the same results. The replicated results were very similar to the original results for the single-task variation, however, the results were different for the multi-task variation of the Office world. Particularly in that variation, the algorithms performance would converge to average reward per step higher by ≈0.2 than in the original results, exceeding the normalized value of 1 for each algorithm except QL and QL-RS. This is shown in Figure 4.1.

Still, when ignoring the y-values, the replicated performance plot looks the same as the one in the original experiment and the difference in the replicated results for each algorithm appears to be consistent. Therefore, we have decided to create a Bland-Altman plot for each algorithm in the multi-task variation of the Office world to verify whether that consistency is indeed consistent, meaning that the replicated results are credible.

Considering that Bland-Altman plots are fairly similar, we will concentrate on the Bland-Altman plot for HRM-RS, shown in Figure 4.2 (the rest can be found in Appendix A). The Bland-Altman plot showed a mean difference of ≈0.17 between the original and replicated results, indicating a small but insignificant systematic bias. The differences formed a logarithmic-function-like pattern within the lines of agreement, suggesting a proportional bias. This can be interpreted in a following way: as the median value rises, the bigger is the difference between the results. However, as all the differences fell within the acceptable range, the discrepancy between the results can be considered consistent, not affecting any further research.

One might notice in Appendix A that the lowest median values for some algorithms do not fall within the lines of agreement. Nonetheless, the differences almost do not exist for such values, thus not affecting the final interpretation of the Bland-Altman plot.

Considering that our Office world results with no noise (Figures 4.1 and A.11) are almost the same as in Icarte's paper (Icarte et al., 2022), we refer to that paper for a detailed interpretation of the results. Now we will focus on the interpretation of the noise results.

The performance thorough different levels of noise, ranging from 0% noise to 90% noise in the labelling function, is shown in Figure 4.3 and Figure 4.4. Figure 4.3 shows how the performance of each algorithm with higher noise decreases for the multi-task variation (each task must be completed from Table 3.1 in the shown order). Each algorithm is experiencing a gradual decline due to the noise increasingly affecting the labelling function. However, unlike the other algorithms, QL-RS and HRM-RS have a steeper decline when introducing only 10% noise to the labelling function. A cause of this could be the way reward shaping is feeding the agent with intermediate rewards, a topic that will be tackled in Section 5.

In the single-task variation the agent was tasked with the most complex task in the Office world, the third task from Table 3.1. The effects of noise on each algorithm running in the aforementioned variation of the Office world is shown in Figure 4.4. In the figure, we notice more differences than in the multi-task variation. Again, all algorithms are roughly having a gradual decline. Particularly, QL has the steepest decline, showcasing how other algorithms are much more robust than QL for complex tasks. Furthermore, CRM-RS, as noise increases, slightly outperforms CRM in certain situations, especially with 90% noise. QL-RS, which had slightly better performance than CRM and CRM-RS when there is no noise, has been outperformed by CRM and CRM-RS for every other noise level. HRM-RS still performs the worst for all noise levels (except for 70% where it is better than QL) while HRM manages to outperform QL for the noise range 50-90%.

4.2 Mordor world results

In this section, considering that the Mordor world is a completely new environment, we will first discuss performance of each algorithm when no noise is induced in the labelling function. We will start with the multi-task variation of the Mordor world.

The multi-task variation of the Mordor world required the agent to solve all tasks from Table 3.2 in the shown order. Due to the modernization of the system, as for the multi-task variation of the Office world, the most optimal algorithms

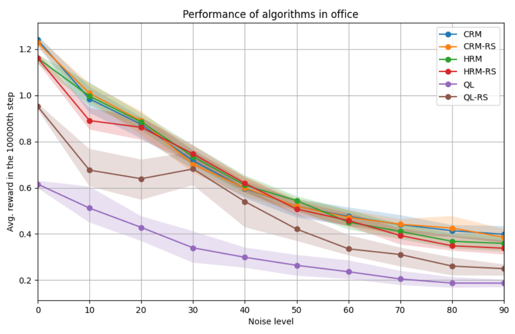

Figure 4.3: Performance results of the algorithms across different noise levels in the Office world (multi-task variation). The line represents the median at the 100000th step per noise level (there are only 10 noise levels and they are labelled on the x-axis). The shaded area represents the area between their 25th and 75th percentiles.

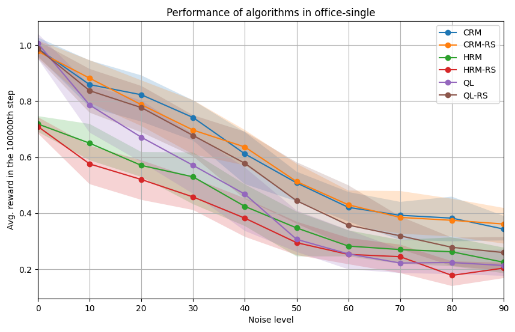

Figure 4.4: Performance results of the algorithms across different noise levels in the Office world (single-task variation). The line represents the median at the 100000th step per noise level (there are only 10 noise levels and they are labelled on the x-axis). The shaded area represents the area between their 25th and 75th percentiles.

converge at value 1.2 as well. Hence, the performance evaluation is scaled up from 0-1 to 0-1.2. For the multi-task variation, HRM-RS learns a policy extremely rapidly. Still, this policy is marginally worse than the most optimal policy which is learned at a slightly slower pace by CRM and CRM-RS. HRM-RS converges to the same policy as HRM but at a slower rate, showing that reward shaping might not contribute to faster learning in every situation. QL-RS and QL converge to suboptimal policies where QL-RS' policy has better performance and is learned faster when compared to QL.

For the single-task variation, the third task from Table 3.2, which is the most complex task in this thesis, was required to be solved by the agent. Note that the single-task variation has a scale 0-1, not 0-1.2, due to the fact that the modernization differences are not visible for single tasks. Once more, HRM converges extremely rapidly to a suboptimal policy, but, unlike in the multi-task variation, its performance is slightly worse. CRM and CRM-RS converge again to the most optimal policy whilst HRM-RS converges to the same policy as HRM but at a much slower learning rate. However, in the single-task variation, QL and QL-RS perform poorly, almost learning nothing new in 100000 steps. This is due to the task complexity which will be thoroughly discussed in Section 5. We can also notice that the task complexity affects the range of the 25th and 75th percentiles, showing that for complex tasks, the performance of each algorithm (except QL and QL-RS) is much more inconsistent.

Now that we have interpreted the results where no noise affects the labelling function, we can delve into understanding the performance of the algorithms in the Mordor world across different levels of noise induced in the labelling function. Each algorithm in the multi-task variation experiences a gradual decline due to the increase of noise. The performance of each algorithm when compared with one another stays roughly consistent. CRM-RS performs the best with 90% noise whilst QL-RS performs the same as QL on that same noise level. Interestingly, HRM outperforms HRM-RS, CRM and CRM-RS (QL and QL-RS do not perform well, thus not taken into account) for the noise range 10-30%, showing more robustness to such noise levels than the others.

As for the single-task variation, the differences

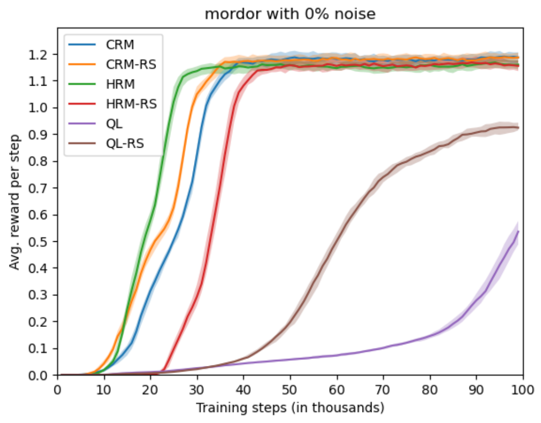

Figure 4.5: Performance results for the Mordor world (multi-task variation) with no noise. The line represents medians over the 60 independent runs per step. The shaded area represents the area between their 25th and 75th percentiles.

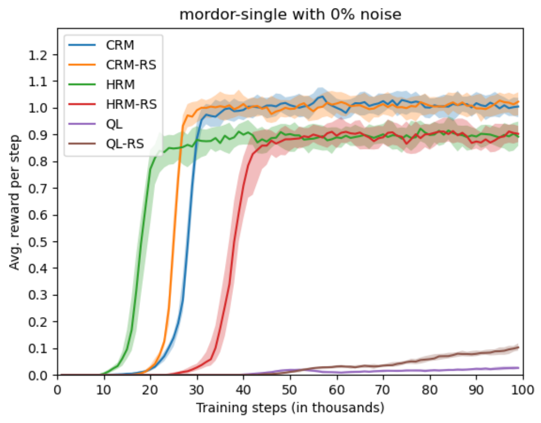

Figure 4.6: Performance results for the Mordor world (single-task variation) with no noise. The line represents medians over the 60 independent runs per step. The shaded area represents the area between their 25th and 75th percentiles.

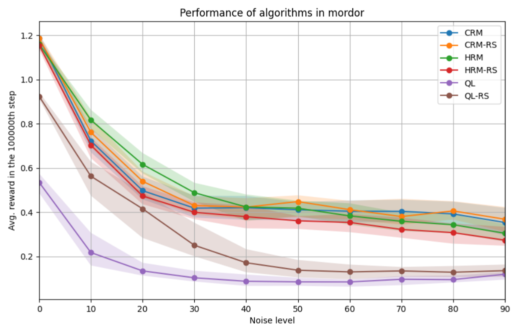

Figure 4.7: Performance results of the algorithms across different noise levels in the Mordor world (multi-task variation). The line represents the median at the 100000th step per noise level (there are only 10 noise levels and they are labelled on the x-axis). The shaded area represents the area between their 25th and 75th percentiles.

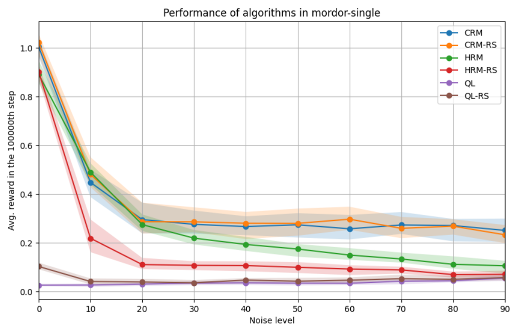

Figure 4.8: Performance results of the algorithms across different noise levels in the Mordor world (single-task variation). The line represents the median at the 100000th step per noise level (there are only 10 noise levels and they are labelled on the x-axis). The shaded area represents the area between their 25th and 75th percentiles.

between the algorithms are much more visible when the noise is increased. Each algorithm exhibits steep decline when the noise is increased to 10% (except QL and QL-RS whose performance without noise was already poor). HRM-RS has the steepest decline, having a significant loss of ≈0.7 average reward in the 100000th step. As for the range 20-90%, each algorithm converges to their own performance score which stays the same during the aforementioned noise levels. The only exception is HRM, which continues gradually dropping its performance score. Moreover, QL and QL-RS for the final noise levels show increase in their performance, but due to the fact that their score is very low from the start, this small increase is the consequence of random noise.

5 Discussion

The aim of this study was to evaluate the RM algorithms on different noise levels and compare them in order to see how noise affects them and to evaluate the performance of the RM algorithms in a complex grid environment which offers significant free choice to the agent with and without noise.

The results have shown that noise and environment complexity are important factors that need to be taken into account when building the reward machines and choosing the RM algorithms for the agent. In the both environments (all variations) without noise, CRM performs the best, finding extremely fast the most optimal policy. This is due to the facilitation of efficient learning by generating much more experience than the other RM algorithms. This also promotes enhanced exploration, having a higher probability to find the most optimal policy than the other algorithms. However, as shown by Icarte et al. (2022), the average runtime of the algorithm, especially in the Mordor world, is very high meaning that when having very complex reward machines CRM can be very time-inefficient, i.e. CRM has scalability issues. Furthermore, CRM depends the most on reward machines out of all the RM algorithms meaning that when the reward machines have errors, which is very plausible in the real-world applications, CRM's performance will be affected the most by it. Still, by performing the best on almost all noise levels as well, CRM has shown that it is very robust against the noise in the labelling function when compared to the other RM algorithms. Therefore, we can say that only a poor construction of the RM states and transitions can rapidly degrade the performance of CRM. Finally, in the Mordor world single-task variation, unlike any other RM algorithms, CRM converges to the same performance score for the noise level ranging 20-90% due to having additional experience. This shows that, when searching for both optimal performance and unaffected performance across higher noise levels, CRM would be the best option.

In the Office world single-task variation, HRM has shown that it performs the worst across almost all noise levels. It converges to a suboptimal policy because of the options framework being myopic, i.e. the learned option (low-level) policies will consistently aim to transition as quickly as possible, trapping themselves on a local optimum. Nonetheless, when disregarding the low starting performance, HRM shows similar robustness as CRM, managing to outperform QL with high-level noise. However, in the Mordor world single-task variation, due to the much higher task complexity, we see how really fast, compared to the other RM algorithms, can HRM learn a policy due to its structured learning. This means that if a complex real-world application requires a well-performing policy (not necessarily optimal) in a short amount of time, HRM would be the best option. When introducing noise to the labelling function, HRM exhibits similar robustness to CRM in the Office world, but, in the Mordor world, when complexity is much higher, it shows that dividing a problem into smaller subproblems might not be as robust as the CRM's additional experience.

As for the both Office and Mordor multi-task variation, HRM is slighlty worse than CRM and learns its policy the fastest. This is due to, next to the most complex task, having simple tasks as well that are robust to the myopic property. Note that in the Mordor world, HRM outperforms CRM for the lower noise levels. We could not find an explanation for this, it could be just pure coincidence in the way random noise is generated, or the structured learning is more robust to the noise for multiple tasks. In order to get certain answers, this should be further researched in a different complex environment with more data points.

As anticipated, QL has the worst performance (except for the Office world single-task variation due to the task simplicity). Considering how sparse reward is in our reward machines and how more complex environments have, it is of upmost importance that an algorithm completely utilizes the reward machines which is not the case for QL. This also means that it lacks a certain degree of robustness to the noise, offered by the reward machines, which is visible across all the noise levels in the both environments.

Automated reward shaping has accelerated learning for most of the RM algorithms with its intermediary rewards. However, in terms of the different noise levels, its contribution is usually not visible and would not mean much to any algorithm's performance if it was avoided (except for QL due to the sparsity of reward). However, there is an exception to this observation, namely HRM-RS. HRM-RS learns the same policy much slower than HRM. This could be due to the interference with HRM's hierarchical nature: reward shaping could encourage actions that do not effectively contribute to achieving the main task by choosing unnecessary options. Icarte et al. (2022) have already shown that experimenting with different hyperparameter values of HRM-RS does not do much. Therefore, instead of value iteration, some other potential function should be tested to see the HRM-RS' performance such as propositional logic-based functions that can utilize the events in the environments.

In general, higher complexity has shown how robustness provided by the reward machines rapidly drops when only 10% of noise is introduced to the labelling function. Complex environments usually involve many possibilities for the agent in order to complete its task. These possibilities can result in diverse reward patterns that are particularly vulnerable to the noise. As a result, the agent may struggle to distinguish meaningful rewards from noise-induced signals. Therefore, if noisy reward machines are unavoidable, either new RM algorithms or extensions of the current RM algorithms that are trained to distinguish between the noise and true information must be developed.

The main limitation of our research was exploring the effects of noise and environment complexity only in discrete domains. Icarte et al. (2018b, 2022) have done research in continuous domains and continuous control task domains, however, due to a time constraint, we were not able to extend on such experiments as well. Modernizing the system for only discrete domains already proved to be a challenge and led to some discrepancies in the results, meaning that modernizing neural networks in continuous domains could lead to even more differences, possibly making Icarte's research (Icarte et al., 2022) not extendable anymore in *Python 3.11.*

Furthermore, the simulated noise was generated by a pseudorandom number generator (Matsumoto & Nishimura, 1998) which does not completely capture the complexity of the real-world applications. Hence, for more plausible results, this kind of experimentation should be done in the applications that already have a known noise model.

To conclude this thesis, we will look into potential future work that can be derived from this. The current reward machine research uses only positive terminal states, having no negative rewards. Currently, when the agent detects a failing event, the ongoing episode only terminates. However, introducing negative rewards could facilitate faster and more efficient learning and, hence, it should be explored. One note, in that case, the way reward shaping is implemented should be reconsidered due to the fact that it is currently adapted only to the reward range of 0-1.

Additionally, the noise probability distribution depends on the amount of events that can be detected by the agent. For example, in the Mordor world there are 17 events while there are 14 events in the Office world. This means that there is a much higher chance to detect an orc in the Mordor world than to detect a decoration in the Office world. Moreover, the amount of distinct events and what their purpose is also affects the noise probability distribution and should be studied further.

Finally, reward machines are built manually, requiring a programmer to design it themselves. However, it might be possible to automate this process considering that the reward machines are constructed from the available tasks in the environment. As long as these tasks can be divided into atomic steps (in case of our research, these steps are easy to detect because they are divided by comas) and they are translated into regular language, states and transitions should be derivable from that information. However, this is not straightforward and, hence, should be looked into with more detail.

References

Camacho, A., Icarte, R. T., Klassen, T. Q., Valenzano, R. A., & McIlraith, S. A. (2019). Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *Ijcai* (Vol. 19, pp. 6065–6073).

De Giacomo, G., Iocchi, L., Favorito, M., & Patrizi, F. (2019). Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. In *Proceedings of the international conference on automated planning and scheduling* (Vol. 29, pp. 128–136).

Hasanbeig, M., Abate, A., & Kroening, D. (2018). Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*.

Icarte, R. T., Klassen, T., Valenzano, R., & McIlraith, S. (2018b). Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International conference on machine learning* (pp. 2107–2116).

Icarte, R. T., Klassen, T. Q., Valenzano, R., & McIlraith, S. A. (2022). Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, *73*, 173–208.

Li, X., Vasile, C.-I., & Belta, C. (2017). Reinforcement learning with temporal logic rewards. In *2017 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 3834–3839).

Littman, M. L., Topcu, U., Fu, J., Isbell, C., Wen, M., & MacGlashan, J. (2017). Environment-independent task specifications via gltl. *arXiv preprint arXiv:1704.04341*.

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, *8*(1), 3–30.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml* (Vol. 99, pp. 278–287).

Shah, A., Li, S., & Shah, J. (2020). Planning with uncertain specifications (puns). *IEEE Robotics and Automation Letters*, *5*(2), 3414–3421.

Singh, S. P. (1992a). Reinforcement learning with a hierarchy of abstract models. In *Proceedings of the national conference on artificial intelligence* (p. 202).

Singh, S. P. (1992b). Transfer of learning by composing solutions of elemental sequential tasks. *Machine learning*, *8*, 323–339.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, *112*(1-2), 181–211.

Toro Icarte, R., Klassen, T. Q., Valenzano, R., & McIlraith, S. A. (2018a). Teaching multiple tasks to an rl agent using ltl. In *Proceedings of the 17th international conference on autonomous agents and multiagent systems* (pp. 452–461).

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*, 279–292.

A Appendix A

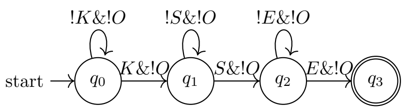

Figure A.1: RM #1 in the Mordor world. Refer to Figure 3.3 for the notation explanation. Note that each transition returns a reward of 0; the only exception is the transition $E\&!O$ which returns a reward of 1.

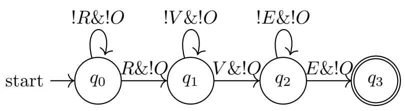

Figure A.2: RM #2 in the Mordor world. Refer to Figure 3.3 for the notation explanation. Note that each transition returns a reward of 0; the only exception is the transition $E\&!O$ which returns a reward of 1.

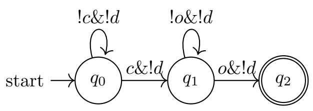

Figure A.3: RM #1 in the Office world. Refer to Figure 3.3 for the notation explanation. Note that each transition returns a reward of 0; the only exception is the transition $o\&!d$ which returns a reward of 1.

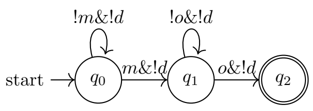

Figure A.4: RM #2 in the Office world. Refer to Figure 3.3 for the notation explanation. Note that each transition returns a reward of 0; the only exception is the transition $o\&!d$ which returns a reward of 1.

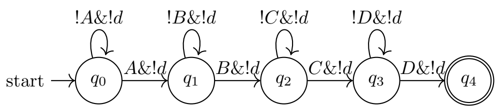

Figure A.5: RM #4 in the Office world. Refer to Figure 3.3 for the notation explanation. Note that each transition returns a reward of 0; the only exception is the transition $D\&!d$ which returns a reward of 1.

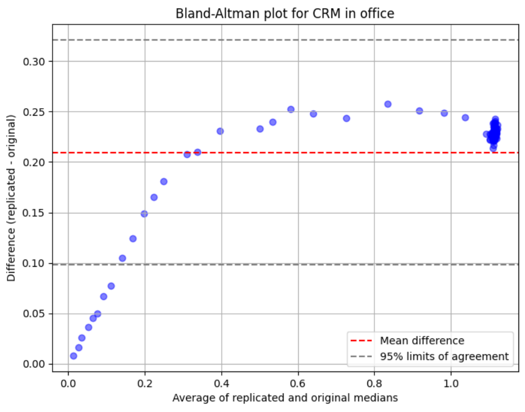

Figure A.6: Bland-Altman plot for CRM in the Office world (multi-task variation).

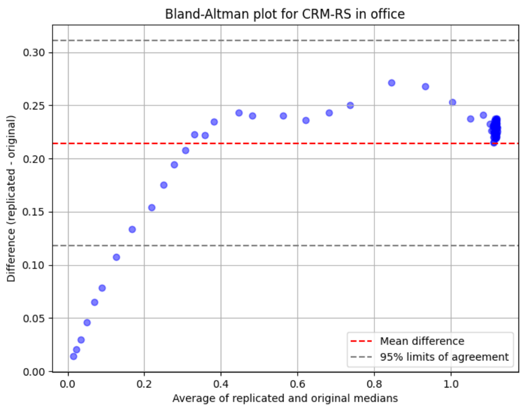

Figure A.7: Bland-Altman plot for CRM-RS in the Office world (multi-task variation).

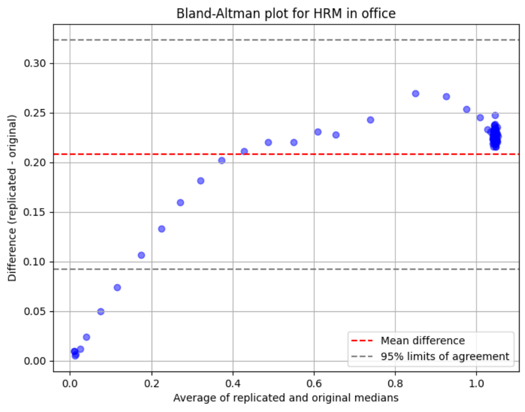

Figure A.8: Bland-Altman plot for HRM in the Office world (multi-task variation).

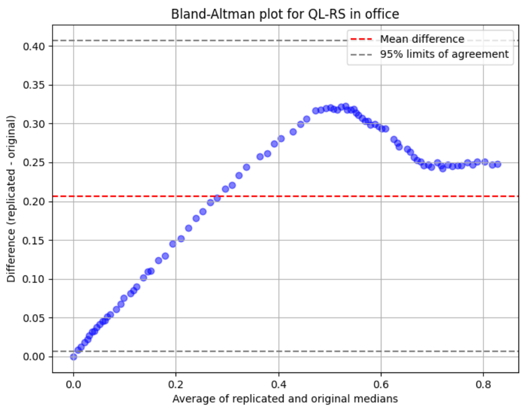

Figure A.10: Bland-Altman plot for QL-RS in the Office world (multi-task variation).

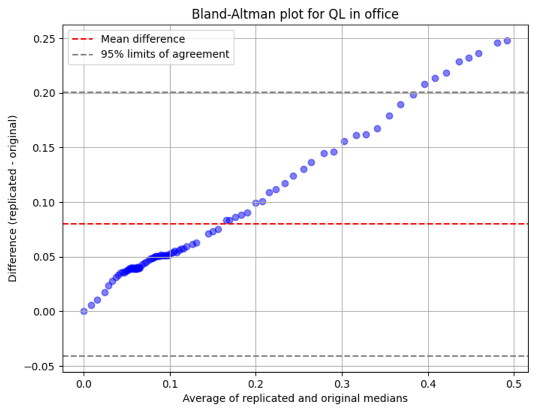

Figure A.9: Bland-Altman plot for QL in the Office world (multi-task variation).

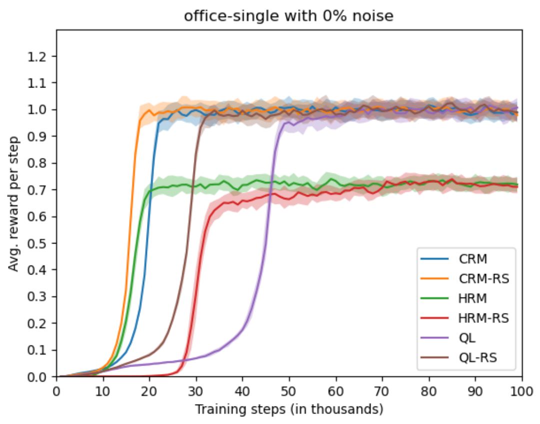

Figure A.11: Performance results for the Office world (single-task variation) with no noise. The line represents medians over the 60 independent runs per step. The shaded area represents the area between their 25th and 75th percentiles.

B Appendix B

Algorithm B.1 The cross-product baseline using tabular Q-learning. This pseudocode is the same as the one presented by Icarte et al. (2022).

```
 1: Input: $S, A, \gamma \in (0,1], \alpha \in (0,1], \epsilon \in (0,1], \mathcal{P}, L, U, u_0, F, \delta_u, \delta_r$.
 2: For all $s \in S, u \in U$ and $a \in A$, initialize $\tilde{q}(s,u,a)$ arbitrarily
 3: for $l \leftarrow 0$ to num_episodes do
 4:   Initialize $u \leftarrow u_0$ and $s \leftarrow$EnvInitialState()
 5:   while $s$ is not terminal and $u \notin F$ do
 6:     Choose action $a$ from $(s,u)$ using $\epsilon$-greedy
 7:     Take action $a$ and observe the next state $s'$
 8:     Compute the reward $r \leftarrow \delta_r(u)(s,a,s')$ and next RM state $u' \leftarrow \delta_u(u, L(s,a,s'))$
 9:     if $s'$ is terminal or $u' \in F$ then
10:       $\tilde{q}(s,u,a) \overset{\alpha}{\leftarrow} r$
11:     else
12:       $\tilde{q}(s,u,a) \overset{\alpha}{\leftarrow} r + \gamma \max_{a' \in A} \tilde{q}(s',u',a')$
13:     end if
14:     Update $s \leftarrow s'$ and $u \leftarrow u'$
15:   end while
16: end for
```

Algorithm B.2 Tabular Q-learning with counterfactual experiences for RMs (CRM). This pseudocode is the same as the one presented by Icarte et al. (2022).

```
 1: Input: $S, A, \gamma \in (0,1], \alpha \in (0,1], \epsilon \in (0,1], \mathcal{P}, L, U, u_0, F, \delta_u, \delta_r$.
 2: For all $s \in S, u \in U$ and $a \in A$, initialize $\tilde{q}(s,u,a)$ arbitrarily
 3: for $l \leftarrow 0$ to num_episodes do
 4:   Initialize $u \leftarrow u_0$ and $s \leftarrow$EnvInitialState()
 5:   while $s$ is not terminal and $u \notin F$ do
 6:     Choose action $a$ from $(s,u)$ using $\epsilon$-greedy
 7:     Take action $a$ and observe the next state $s'$
 8:     Compute the reward $r \leftarrow \delta_r(u)(s,a,s')$ and next RM state $u' \leftarrow \delta_u(u, L(s,a,s'))$
 9:     Set experience $\leftarrow \{\langle s, \bar{u}, a, \delta_r(\bar{u})(s,a,s'), s', \delta_u(\bar{u}, L(s,a,s'))\rangle | \forall \bar{u} \in U\}$
10:     for $\langle s, \bar{u}, a, \bar{r}, s', \bar{u}' \rangle \in$ experience do
11:       if $s'$ is terminal or $u' \in F$ then
12:         $\tilde{q}(s,\bar{u},a) \overset{\alpha}{\leftarrow} \bar{r}$
13:       else
14:         $\tilde{q}(s,\bar{u},a) \overset{\alpha}{\leftarrow} \bar{r} + \gamma \max_{a' \in A} \tilde{q}(s',\bar{u}',a')$
15:       end if
16:     end for
17:     Update $s \leftarrow s'$ and $u \leftarrow u'$
18:   end while
19: end for
```

Algorithm B.3 Tabular hierarchical RL for reward machines (HRM). This pseudocode is the same as the one presented by Icarte et al. (2022).

```
1: **Input:** $S, A, \gamma \in (0,1], \alpha \in (0,1], \epsilon \in (0,1], \mathcal{P}, L, U, u_0,\ F, \delta_u, \delta_r$.
2: $\mathcal{A}(u) \leftarrow \{u_t = \delta_u(u,\sigma)$ for some $u_t \in U \cup F, \sigma \in 2^{\mathcal{P}}\}$ for all $u \in U$
3: For all $s \in S, u \in U$ and $u_t \in \mathcal{A}(u)$, initialize the high-level $\tilde{q}(s,u,u_t)$ arbitrarily
4: For all $s \in S, u \in U, u_t \in \mathcal{A}(u)$ and $a \in A$, initialize option $\tilde{q}_{u,u_t}(s,a)$ arbitrarily
5: **for** $l \leftarrow 0$ **to** num_episodes **do**
6:   Initialize $u \leftarrow u_0$, $s \leftarrow$EnvInitialState() and $u_t \leftarrow \emptyset$
7:   **while** $s$ is not terminal **and** $u \notin F$ **do**
8:     **if** $u_t = \emptyset$ **then**
9:       Choose option $u_t \in \mathcal{A}(u)$ using $\epsilon$-greedy policy derived from $\tilde{q}$
10:      Set $r_t \leftarrow 0$ and $t \leftarrow 0$
11:    **end if**
12:    Choose action $a$ from $s$ using $\epsilon$-greedy policy derived from $\tilde{q}_{u,u_t}$
13:    Take action $a$ and observe the next state $s'$
14:    Compute the reward $r \leftarrow \delta_r(u)(s,a,s')$ and next RM state $u' \leftarrow \delta_u(u, L(s,a,s'))$
15:    **for** $\bar{u} \in U, \bar{u}_t \in \mathcal{A}(\bar{u})$ **do**
16:      **if** $\delta(\bar{u}, L(s,a,s')) \neq u$ **or** $s'$ is terminal **then**
17:        $\tilde{q}_{\bar{u},\bar{u}_t}(s,a) \xleftarrow{\alpha} r_{\bar{u},\bar{u}_t}(s,a,s')$
18:      **else**
19:        $\tilde{q}_{\bar{u},\bar{u}_t}(s,a) \xleftarrow{\alpha} r_{\bar{u},\bar{u}_t}(s,a,s') + \gamma \max_{a' \in A} \tilde{q}_{\bar{u},\bar{u}_t}(s',a')$
20:      **end if**
21:    **end for**
22:    **if** $s'$ is terminal **or** $u' \neq u$ **then**
23:      **if** $s'$ is terminal **or** $u' \in F$ **then**
24:        $\tilde{q}(s,u,u_t) \xleftarrow{\alpha} r_t + \gamma^t r$
25:      **else**
26:        $\tilde{q}(s,u,u_t) \xleftarrow{\alpha} r_t + \gamma^t r + \gamma^{t+1} \max_{u'_t \in \mathcal{A}(u')} \tilde{q}(s',u',u'_t)$
27:      **end if**
28:      Set $u_t \leftarrow \emptyset$
29:    **end if**
30:    Update $s \leftarrow s'$ and $u \leftarrow u'$
31:    Update $r_t \leftarrow r_t + \gamma^t r$
32:    Update $t \leftarrow t + 1$
33:  **end while**
34: **end for**
```

Algorithm B.4 Value iteration for automated reward shaping. This pseudocode is the same as the one presented by Icarte et al. (2022).

```
1: **Input:** $U, F, \mathcal{P}, \delta_u, \delta_r, \gamma$.
2: for $u \in U \cup F$ do
3:     $v(u) \leftarrow 0$
4: end for
5: $e \leftarrow 1$
6: while $e > 0$ do
7:     $e \leftarrow 0$
8:     for $u \in U$ do
9:         $v' \leftarrow \max\{\delta_r(u, \sigma) + \gamma v(\delta_u(u, \sigma)) | \forall \sigma \in 2^{\mathcal{P}}\}$
10:        $e = \max\{e, |v(u) - v'|\}$
11:        $v(u) \leftarrow v'$
12:    end for
13: end while
14: return $v$
```