LTL$_f$-based Reward Shaping for Reinforcement Learning

Mahmoud Elbarbari
Brubotics Lab and Artificial Intelligence Lab, Vrije Universiteit Brussel
Brussels, Belgium
Imec
Leuven, Belgium
mahmoud.elbarbari@vub.be

Kyriakos Efthymiadis
Artificial Intelligence Lab, Vrije Universiteit Brussel
Brussels, Belgium
kyriakos.efthymiadis@vub.be

Bram Vanderborght
Brubotics, Vrije Universiteit Brussel
Brussels, Belgium
Imec
Leuven, Belgium
bram.vanderborght@vub.be

Ann Nowé
Artificial Intelligence Lab, Vrije Universiteit Brussel
Brussels, Belgium
ann.nowe@vub.be

ABSTRACT

Reinforcement Learning usually does not scale up well to large problems. It typically takes a Reinforcement Learning agent many trials until it can reach a satisfying policy. A main contributing factor to this problem is the fact that Reinforcement Learning is often used for learning exclusively by means of trial and error. There has been much work that addresses incorporating domain knowledge in Reinforcement Learning to allow more efficient learning. Reward shaping is a well-established method to incorporate domain knowledge in Reinforcement Learning by providing the learning agent with a supplementary reward. In this work we propose a novel methodology that automatically generates reward shaping functions from user-provided Linear Temporal Logic formulas. Linear Temporal Logic in our work serves as a rich, yet compact, language that allows the user to express the domain knowledge with minimum effort. Linear Temporal Logic is also rather easy to be expressed in natural language which makes it easier for non-expert users. We use the flag collection domain to demonstrate empirically the increase in both the convergence speed and the quality of the learned policy despite the minimum domain knowledge provided.

KEYWORDS

Reinforcement Learning, Reward Shaping, Linear Temporal Logic on finite traces

1 INTRODUCTION

Most of the current research in Reinforcement Learning (RL) assumes that the learning starts from a blank slate and improves only by means of trial and error. This learning approach takes a huge amount of trials until the learning agent can reach a satisfying policy. This amount of trials is prohibitively expensive in many scenarios.

Reward shaping is a well-established method to incorporate domain knowledge in the RL agents. In reward shaping, the domain knowledge is represented as a supplementary reward that allows the RL agent to learn more efficiently. Representing the reward shaping functions as a difference between potential functions, as described in [10], provides the guarantee that the optimal policy does not change. The policy invariance guarantee holds in both cases of static [10] and dynamic potentials [3].

In this work, the domain knowledge is incorporated in the RL agent through Linear Temporal Logic on finite traces (LTL$_f$) formulas. The LTL$_f$ formulas are then used to generate potential-based reward shaping (PBRS) functions to provide the agent with an additional reward which in turn allows more efficient learning. Incorporating the domain knowledge in the RL agent as reward shaping functions makes our method applicable to a broad range of both model-based and model-free RL algorithms. In addition, PBRS provides the policy invariance guarantee which means that our method is guaranteed to preserve the optimal policy [3, 10].

This paper presents a novel methodology that automatically generates PBRS functions from user-provided LTL$_f$ formulas which provide a flexible and user-friendly way to incorporate domain knowledge in the RL agents.

2 RELATED WORK

LTL has been widely used as a task specification language in a variety of fields [7–9]. The agent in this case has to satisfy constraints provided to it through LTL formulas. This results in a reduction in the policy space which may lead to discarding optimal policies in case of inaccurate constraints. On the contrary, in our work, we use LTL$_f$ to generate PBRS functions that are guaranteed not to change the optimal policy.

Reward shaping, when used properly, is a powerful tool to help an RL agent to reach a satisfying policy faster. It has been shown that if the reward shaping is used incorrectly, even with good intentions, it can change the optimal policy of the task e.g [12]. In [10], the authors investigate the conditions under which the optimal policy is preserved with reward shaping. The authors conclude that it is both sufficient and necessary to have the reward shaping function as the difference between values of a static potential function.

We use in this work potential functions that generally change over time (dynamic potential functions). The authors of [3] extend the proofs of the policy invariance to the case of dynamic potential functions. They proved that the potential functions can change during the learning while preserving the optimal policy, given the

condition that the potential of the states is evaluated at the time they were visited.

The domain knowledge in our work is generated manually but the conversion to reward shaping functions is automatic (called semi-automatic method). In [4], the authors propose a semi-automatic reward shaping method that generates PBRS functions from STRIPS plans. The authors demonstrate their method on the challenging flag collection domain where a vanilla RL agent (without reward shaping) can easily get stuck in a sub-optimal policy.

To the best of our knowledge, [6] is the only work that uses LTL_f to bias the RL agent's exploration. The authors use heuristics generated from LTL_f formulas to guide the agent's exploration in a variant of Rmax. Our work is highly motivated by their findings. We propose the use of LTL_f to automatically generate PBRS functions which makes our method more advantageous in two main ways:

- Using reward shaping makes our method applicable to many model-free and model-based RL algorithms.
- Our method utilizes the temporal aspect of LTL_f by using different reward shaping functions at different stages of satisfaction of the LTL_f formulas.

3 BACKGROUND

3.1 Reinforcement Learning and Markov Decision Processes

Markov Decision Processes (MDPs) [1] are used to formalize the problem of sequential decision making. An MDP can be represented as a tuple $(S, A, \delta, R, \gamma)$: S is the set of states , A is the set of actions, δ $(S, A|S')$ is the transition function, R is the reward function and γ is the discount factor. In MDPs, the agent interacts with the environment by executing an action a in a state s then makes the transition to next state s' according to the transition function δ and receives a scalar reward $r \in \mathbb{R}$. MDPs are used in this work to formalize the problem of RL. The RL agent should learn a policy $\pi(a|s)$ to maximize the cumulative discounted reward, where the policy is a probabilistic function that defines the probability of selecting an action a at a given state s. The cumulative discounted reward is the expectation of the sum of the discounted rewards the agent receives $\mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \dots]$.

3.2 Potential-based Reward Shaping

Reward shaping is a well-established method that allows the RL agent to learn more efficiently. Reward shaping works by providing the RL agent with a supplementary reward $F(s, a, s')$. So the augmented reward function of the MDP becomes

$$R'(s, a, s') = F(s, a, s') + R(s, a, s') \tag{3.1}$$

However, reward shaping can sometimes change the optimal policy. PBRS represents the reward shaping function as the difference of values of a potential function, formally as the following

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s) \tag{3.2}$$

The reward function in this form is both sufficient and necessary to guarantee policy invariance [10].

The PBRS functions can also be dynamic i.e., change during the learning process. Policy invariance is preserved if the reward shaping function is given in the following form

$$F(s, t, s', t') = \gamma\Phi(s', t') - \Phi(s, t) \tag{3.3}$$

where t is when the agent visited state s, and t' is when the agent visited state s' [3].

3.3 Linear Temporal Logic

Linear Temporal Logic (LTL) [11] is a type of temporal logic that allows expressing and reasoning over propositions qualified in terms of time. LTL is a modal temporal logic with an expressive and reasoning power limited to timelines. Since we assume finite horizon RL in this work, we utilize LTL formulas interpreted over *finite* traces (LTL_f). An LTL formula is composed of a finite set of propositional symbols $\mathcal{P}$, logical connectives $\neg$, $\wedge$, $\vee$ and $\rightarrow$ and modal operators X next and U until, from which other useful modal operators can be derived e.g. G globally and F finally. The modal operators can have the following meaning: G globally: $True$ now and at any time later, F finally: will become $True$ at some time, X next: will become $True$ in the next state and U until: $True$ until something else becomes $True$.

Any LTL formula f can be transformed to a Deterministic Finite Automaton (DFA) that accepts a trace T, a sequence of propositional symbols, only if T satisfies formula f [2]. The DFAs are used in this work to track the stage of satisfaction of the LTL formulas.

3.4 Deterministic Finite Automaton

A Deterministic Finite Automaton (DFA) [5] is a mathematical model that deterministically maps an input sequence to an output so that the computation is unique. A DFA can be exactly in one state at a given time and it makes the transition from one state to another state given some input and according to the transition function. A DFA accepts an input sequence if the sequence drives the DFA to one of its accepting states. A DFA is defined formally as a tuple $(Q, q_0, \Delta, \Sigma, F)$ where q_0 is the initial state, Q is the set of states, Δ is the transition function (maps a state and an input to another state), Σ is the set of possible inputs and F is the set of accepting states.

4 METHODOLOGY

In this work, we utilize LTL_f as a means for incorporating domain knowledge in Reinforcement Learning agents. We use different potential functions at different satisfaction stages of the LTL_f formulas (at different states of the corresponding DFAs).

Three main advantages motivate our use of Linear Temporal Logic (LTL_f):

- LTL_f provides an intuitive and flexible way to describe many common RL tasks.
- LTL_f is a rich language that can validate whole sequences over time. This makes LTL_f a compact language which means less user effort is needed.
- LTL_f is rather easy to be expressed in natural language. This makes it easier for non-expert users.

We refrain from using more complex forms of Temporal Logics in our work e.g. time-branching logic and leave this for future work directions since our main goal in this paper is to demonstrate the

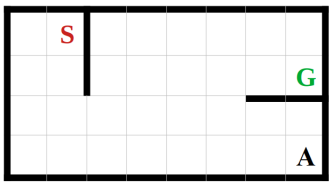

Figure 1: an example of flag collection problem; the start position (S in red), the flag A (A in black) and the goal position (G in green).

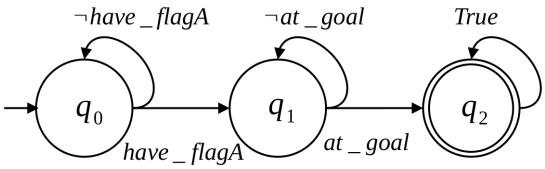

Figure 2: the corresponding DFA for the LTL$_f$ formula $F(have\_flagA \wedge XFat\_goal)$.

potential of using Temporal Logics as a means of guidance for the RL agent.

4.1 Example Task

We consider a simple flag collection problem (Figure 1) to serve as a running example to clarify our method. In the flag collection domain, the agent should learn how to collect all the flags and then go to the goal position in a minimum number of steps.

4.2 Constructing the LTL$_f$ Formulas

The user provides m-LTL$_f$ formulas either online (during learning) or offline (before the learning starts), where m is the number of formulas. To construct the LTL$_f$ formulas we first define a finite set of propositional symbols $\mathcal{P}$. In the example task, we define $\mathcal{P} = \{have\_flagA, at\_goal\}$ where $have\_flagA$ is $True$ *iff* the agent has collected flag A and at_goal is $True$ *iff* the agent is at the goal positon. And then we use $\mathcal{P}$ together with the propositional and modal operators discussed in 3.3 to construct the LTL$_f$ formulas.

In the example task, we would like to guide the agent to collect flag A and then go to the goal position. To this end we provide the agent with the following LTL$_f$ formula $F(have\_flagA \wedge XFat\_goal)$. This formula can be expressed in natural language as "*Finally collect flag A and then finally be at the goal position*".

4.3 Transforming LTL$_f$ Formulas to Deterministic Finite Automata

An LTL$_f$ formula can be translated to a corresponding DFA such that the DFA accepts a sequence of states *iff* the sequence satisfies the LTL$_f$formula [2]. We therefore transform the user-provided m-LTL$_f$ formulas to m-DFAs in order to track the satisfaction stage of the corresponding LTL$_f$ formulas. The goal is then to push the DFAs through their edges to their accepting states and thus satisfying the corresponding LTL$_f$ formulas. The LTL$_f$ formula we used in the example task has the corresponding DFA shown in Figure 2.

4.4 Tracking the States of the DFAs

Our method generates potential functions depending on the *current* states of the DFAs (which represent the stage of satisfaction of the LTL$_f$ formulas) to guide the agent to push the DFAs to their accepting states (which means satisfying the corresponding LTL$_f$ formulas). To this end, we define a labeling function $L(S, \mathcal{P})$ that associates the truth assignment of each propositional symbol $A \in \mathcal{P}$ to each MDP state $s \in S$. By using the labeling function L we can now track the current states of the DFAs given the sequence of truth assignments associated with the MDP states the agent has visited. Iteratively at each time step t we use the truth assignments associated with the current MDP state $L(s_t, A)$ for each $A \in \mathcal{P}$ to update the states of the DFAs according to their transition functions.

In the example task, the DFA (in Figure 2) is at its initial state q_0 at the beginning of the episode. The DFA will not make a transition to another state until the agent collects flag A since $L(s_t, have\_flagA)$ will then be $True$ and the DFA will make the transition to q_1. Then again the DFA will not make a transition to another state until the agent reaches the goal position since $L(s_t, at\_goal)$ will be $True$ and the DFA will reach its only accepting state q_2 and thus satisfy the provided LTL$_f$ formula.

4.5 Defining Useful Edges

In order to guide the agent to push the DFAs to their accepting states, we first define what is called *useful edges*. We define useful edges for a DFA as the edges of the *current* DFA state that lead to progress towards the DFA accepting states. An edge $e(q, \phi, q')$ of the current DFA state q is a *useful* edge if the edge leads to a next DFA state q' which has a path to an accepting state and q is not part of this path. We denote the set of useful edges of the i^{th} DFA at time t as U_i^t.

In the example task, at any time t, if the DFA state is q_0, then $U_0^t = \{(q_0, have\_flagA, q_1)\}$, and if the DFA state is q_1, then $U_0^t = \{(q_1, at\_goal, q_2)\}$. And lastly, if the DFA current state is the accepting state q_2, then U_0^t is an empty set.

4.6 Domain Knowledge Functions

We will use potential functions that guide the agent to satisfy the formulas associated with the useful edges. For this we assume that the agent has a domain knowledge function $h_A(S)$ for each $A \in \mathcal{P}^+$, where $\mathcal{P}^+ = \mathcal{P} \cup \{\neg p : p \in \mathcal{P}\}$, that provides an estimate of the number of primitive actions needed for A to be $True$ from any MDP state $s \in S$. The estimates which the domain knowledge functions provide can vary widely depending on the domain knowledge available for the problem at hand.

Using the domain knowledge functions we can define a more general function $h(S, \phi)$ that provides an estimate of the number of primitive actions needed to satisfy any formula ϕ, given that ϕ is put in the Disjunctive Normal Form (DNF), which means that ϕ is a disjunction of conjunctions. We define the heuristics function h recursively as the following

$$\begin{aligned} h(S, A) &= h_A(S), \text{ for any } A \in \mathcal{P}^+ \\ h(S, \psi \wedge \chi) &= sum(h(S, \psi), h(S, \chi)), \text{ for any formulas } \psi \text{ and } \chi \\ h(S, \psi \vee \chi) &= \min(h(S, \psi), h(S, \chi)), \text{ for any formulas } \psi \text{ and } \chi \end{aligned} \tag{4.1}$$

In the example task, we can define a domain knowledge function for each propositional symbol $A \in \{have\_flagA, at\_goal, \neg have\_flagA, \neg at\_goal\}$ as the following

$$h_{have\_flagA}(s) = \begin{cases} distance_{flagA}(pos(s)) & \text{if } L(s, have\_flagA) \\ & \text{is } False \\ 0 & ; otherwise \end{cases} \tag{4.2}$$

$$h_{\neg have\_flagA}(s) = \begin{cases} 0 & \text{if } h_{have\_flagA}(s) \neq 0 \\ undefined & ; otherwise \end{cases} \tag{4.3}$$

$$h_{at\_goal}(s) = distance_{at\_goal}(pos(s)) \tag{4.4}$$

$$h_{\neg at\_goal}(s) = \begin{cases} 0 & \text{if } h_{at\_goal}(s) \neq 0 \\ 1 & ; otherwise \end{cases} \tag{4.5}$$

for each $s \in S$ and where $pos(s)$ returns the position of the agent at state s and $distance_{have\_flagA}$ and $distance_{at\_goal}$ are heuristics functions that return estimate distances e.g. the direct distances (disregarding the walls) to flag A and the goal position respectively.

The domain knowledge function for satisfying $\neg have\_flagA$ is not defined when the agent already has flag A since the agent can not drop a flag after it is collected in the flag collection domain. We do not consider in this work misleading guidance, so this function will never be evaluated in its undefined range.

4.7 Building the Potential Functions

We now define what is called *the guiding formula* for the i^{th} DFA at time t as the following

$$\phi^t_{guiding} = \bigvee_{e(q,\phi,q') \in U^t_i} \phi \tag{4.6}$$

We also define a function $Distance(Q)$ which returns the length of the longest acyclic path to an accepting state from DFA state $q \in Q$. If an accepting state is not reachable from a DFA state q', then $Distance(q') = max(Distance(Q - \{q'\}) + 1$. Finally, we define the potential function at time t as the following

$$\Phi_t(s_t) = -\omega \times \left[Distance(q_t) + \frac{1}{n} \times h(s_t, \phi^t_{guiding}) \right] \tag{4.7}$$

where ω is a scaling factor and $1/n$ is a normalization factor where $n = max\big(h(S, \phi^t_{guiding})\big)$.

4.8 In Case of m-LTL$_f$ Formulas

In case of m-LTL$_f$ formulas, we define the guiding formula at time t as the following

$$\phi^t_{guiding} = \bigwedge_{i=0}^{m} \phi^i_t \tag{4.8}$$

where ϕ^i_t is the guiding formula of the i^{th} DFA at time t. And we redefine the potential function as the following

$$\Phi_t(s_t) = -\omega \times \left[\sum_{i=0}^{m} Distance(q^i_t) + \frac{1}{n} \times h(s_t, \phi^t_{guiding}) \right] \tag{4.9}$$

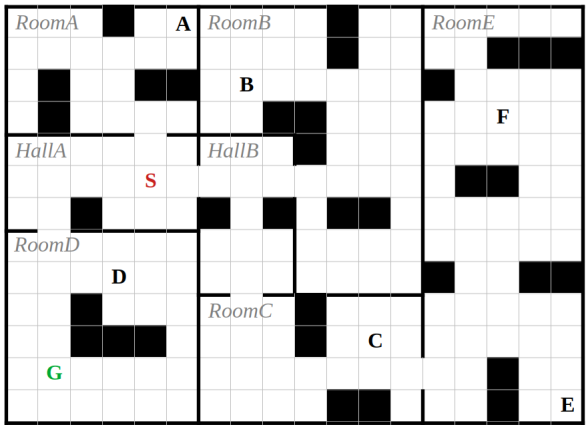

Figure 3: the classic flag collection problem (with added obstacles).

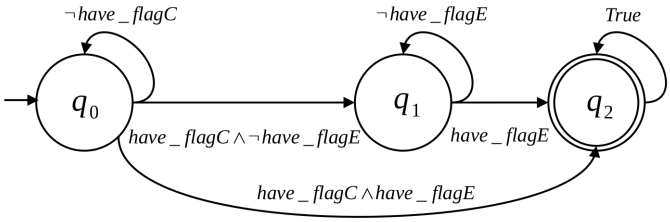

Figure 4: the corresponding DFA for the LTL$_f$ formula $F(have\_flagC \wedge (have\_flagE \vee XF\ have\_flagE))$.

5 EXPERIMENTS

We evaluate our method on two different variations of the flag collection domain. We compare our method performance to that of the plan-based reward shaping method in [4].

In the flag collection domain, the state is defined by the agent position and the flags collected by the agent. The agent has 8 actions that deterministically move it to the adjacent cells (the agent remains in place in case of a wall or an obstacle in the intended direction of the executed action). In our experiments, the agent is punished with a small negative reward of -0.1 at each time step and receives a positive reward of 100 multiplied by the number of flags collected only at the end of the episode. In all the experiments we use the SARSA algorithm with Q-table initialized with zero values. We use learning rate $\alpha = 0.1$ and discount factor $\gamma = 1$. We use a scaling factor $\omega = 10$ for our method and no scaling for the plan-based method (since it has no significant effect on the method performance). Each experiment consists of 10^4 episodes yet we plot the results for the first 1000 episodes only since they merely change afterward. We repeat each experiment 10 times and report the average results. For smoother learning curves we always take the average of every 10 consecutive episodes.

5.1 Evaluation in the Classic Flag Collection Domain

In this set of experiments, we consider the flag collection problem shown in Figure 3. This configuration is the original version used in [4] with only added obstacles that the agent cannot go through. The agent should collect all the flags (A-F) and go to the goal position in the minimum number of steps. In this problem there are more than one optimal ordering of collecting the flags e.g. (C, E, F, B, A then D) and (A, B, C, F, E then D) are examples of optimal orderings.

The plan-based method, as described in [4], in this case would only consider one optimal ordering. Our LTL_f-based method instead provides the flexibility for the user to guide the agent to any optimal ordering. In our experiments we give the agent different LTL_f formulas, each guiding the agent to a different optimal ordering and we compare the performance to that of the plan-based method. We provide the following LTL_f formula

$$F(have\_flagC \wedge F(have\_flagE \wedge \ldots \\ \wedge (have\_flagD \vee XF\ have\_flagD))))) \tag{5.1}$$

to help the agent to collect the flags in the following order (C, E, F, B, A then D), and similar formulas for other optimal orderings. A guidance formula expressed as in 5.1 would guide the agent to follow a certain order, but still provides supplementary reward even if the flags are not collected in this order. Therefore this formula is more relevant when the user is not completely sure about the optimal ordering. The corresponding DFA to formula 5.1 cannot be shown in its entirety due to insufficient space. Nevertheless, we show the corresponding DFA of a simpler formula similar to 5.1 to provide the intuition. We consider the formula that guides the agent to collect only the first 2 flags, namely C and E expressed in LTL_f as the following

$$F(have\_flagC \wedge (have\_flagE \vee XF\ have\_flagE)) \tag{5.2}$$

with the corresponding DFA in Figure 4.

In order to build the LTL_f formulas, as discussed in 4.2, we need to define a set of propositional symbols $\mathcal{P} = \{have\_flagA, \ldots, have\_flagF\}$. Then we define domain knowledge functions (4.6) for each propositional symbol $A \in \mathcal{P}^+$. We define the domain knowledge functions for all the flags in the same way as we did for flag A in 4.2 and 4.3. We redefine the heuristics function $distance_A$ for all $A \in \mathcal{P}$ to return the distance (disregarding the obstacles but not the walls) to the corresponding flag. This heuristic knowledge is available in many applications in the real world where we would know the main structure of the environment e.g. the room structure of a floor but not the obstacles e.g. the furniture.

In this set of experiments, we use ϵ-greedy exploration strategy with $\epsilon = 0.3$ at the start of the episode and linearly decreasing to 0.01. In Figure 5 we report the results for plan-based and LTL_f-based method guiding the agent to the same optimal ordering, namely (C, E, F, B, A then D). LTL_f-based method achieved slightly better performance in both terms of convergence speed and the final policy compared to the plan-based method. The LTL_f formulas that guide the agent to other optimal orderings report similar results (although not reported in Figure 5 for clarity since the learning curves usually override). The agent with no shaping easily converges to the sub-optimal policy of picking flag D and then go directly to the goal.

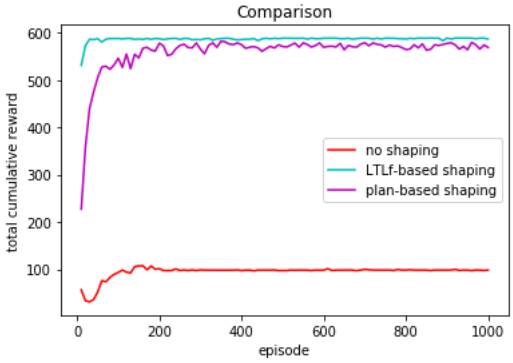

Figure 5: results in the classic flag collection domain.

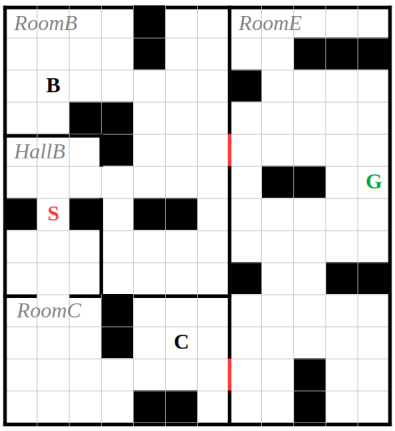

Figure 6: flag collection problem with uncertainty over which door (in red) is open.

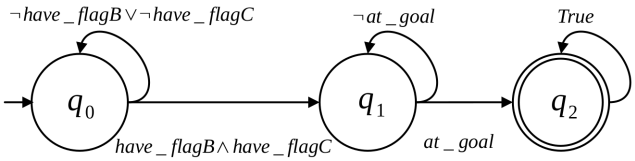

Figure 7: the corresponding DFA for the LTL_f formula $F((have\_flagB \wedge have\_flagC) \wedge XFat\_goal))$.

5.2 Evaluation in Flag Collection Domain with an Aspect of Uncertainty

In this set of experiments, we consider the flag collection problem shown in Figure 6. At each experiment one of the two doors in the color red can be closed and the other is open. The optimal ordering of collecting the flags depends on which door is open at the current experiment. If the (Room C - Room E) door is the one which is open, the optimal ordering would be to collect flag B first then flag C. And if (Room B - Room E) door is the one which is open, the optimal ordering would be to collect flag C first then flag B. We assume the case that the user who provides the guidance (in our case the

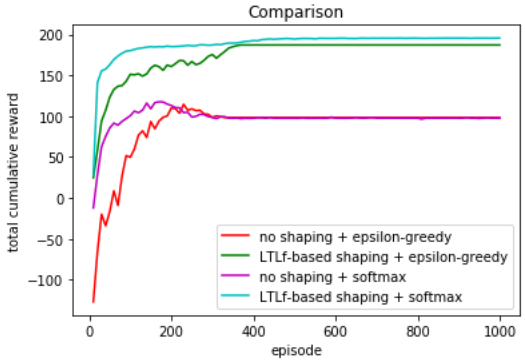

Figure 8: results in the flag collection domain with uncertainty (in case of door (Room B - Room E) is the open door).

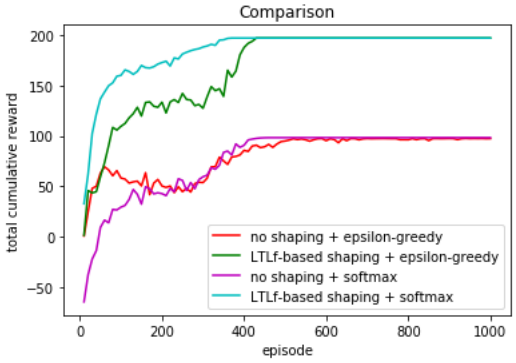

Figure 9: results in the flag collection domain with uncertainty (in case of door (Room C - Room E) is the open door).

LTL_f formulas) does not know which door is open at the current experiment, therefore the user does not know which ordering is the optimal one.

In this case, we would like to have the flexibility to guide the agent to collect flag B first then flag C *or* collect flag C first then flag B (which means guiding the agent to collect both flags without pushing for a certain ordering) then go to the goal. To the best of our knowledge, the plan-based reward shaping method does not offer such flexibility. In LTL_f this can be expressed elegantly by providing the following formula

$$F((have\_flagB \wedge have\_flagC) \wedge XFat\_goal)) \tag{5.3}$$

which can be expressed in natural language as "*Finally collect flag B and flag C, and then go to the goal*" (with the corresponding DFA in Figure 7). Although the previous formula guides the agent to collect both flags, it clearly does not recommend a certain order to collect them. The reward shaping functions that the previous LTL_f formula generates will not bias the agent in any way to collect one of the flags before the other one.

It is worth mentioning that in our experiments we assume that the domain knowledge functions would report the correct estimates depending on which door is open. Although this assumption restricts the flexibility we would like to have, this flexibility is still useful in cases where the user does not know which door is open but the agent has this knowledge. We can remove this assumption if the agent is capable to identify the situation and switch between two different sets of domain knowledge functions (or even learn them) depending on the situation .

In this set of experiments we use both ϵ-greedy and softmax exploration strategies with LTL_f-based reward shaping, and we also compare to the performance of no shaping. We use a constant temperature $T = 0.3$ for softmax, and we use the same values for ϵ as in the first set of experiments for the ϵ-greedy.

In Figure 8 we report the results for the case in which the (Room B - Room E) door is the open door. The ϵ-greedy strategy could not converge even once to the optimal ordering (flag C then flag B), while the softmax strategy always converged to the optimal ordering. These results support our initial anticipation that the softmax strategy would do more efficient exploration for two (or more) *potentially* optimal solutions. We expect the differences in performance to be even more profound as we move to more complex settings. We also report the results for the case in which the (Room C - Room E) is the open door in Figure 9. In this case, both ϵ-greedy and softmax could converge to the optimal ordering (flag B then flag C) in all the experiments. The agent with no shaping, as expected, always converges to the sub-optimal policy of collecting one flag and then going directly to the goal.

6 CONCLUSION

In this paper, we introduced a novel methodology to incorporate domain knowledge in RL agents. We used LTL_f to automatically generate reward shaping functions to allow for more efficient learning. Our method is applicable to many RL algorithms and provides benefits over previous works in terms of specifying domain knowledge. LTL_f is an intuitive and very rich language that can express complex whole sequences over time. It is rather easy to be expressed using natural language and this makes it compact and easier to use by non-experts. LTL_f can provide a mechanism for domain knowledge to be incorporated with minimal user effort.

We demonstrated empirically the increase in the ability and the speed of the RL agent convergence in the classic flag collection domain and a variant with an aspect of uncertainty. Our agent manages to learn the correct order of picking up the flags even in the case of uncertainty in the environment, something that the STRIPS method fails to do.

We are motivated to pursue three main future directions to build on this work. We will investigate and further expand our experiments to scenarios where LTL_f can offer needed flexibility in guiding the learning agent e.g. in case of uncertainties. We will also investigate using LTL_f in multi-agent systems as we think that our method can efficiently guide coordination behaviors between agents. Lastly, we think that our method can make RL more feasible in real-world scenarios, therefore we are interested in expanding our experiments to real robotic systems.

ACKNOWLEDGMENTS

This research was supported by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme, and the Vrije Universiteit Brussel.

REFERENCES

[1] Richard Bellman. 1957. A Markovian decision process. *Journal of mathematics and mechanics* (1957), 679–684.

[2] Giuseppe De Giacomo and Moshe Y Vardi. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

[3] Sam Michael Devlin and Daniel Kudenko. 2012. Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS, 433–440.

[4] Marek Grzes and Daniel Kudenko. 2008. Plan-based reward shaping for reinforcement learning. In *2008 4th International IEEE Conference Intelligent Systems*, Vol. 2. IEEE, 10–22.

[5] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. 2001. Introduction to automata theory, languages, and computation. *Acm Sigact News* 32, 1 (2001), 60–65.

[6] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Anthony Valenzano, and Sheila A Mcllraith. 2018. Advice-based exploration in model-based reinforcement learning. In *Canadian Conference on Artificial Intelligence*. Springer, 72–83.

[7] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. 2009. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics* 25, 6 (2009), 1370–1381.

[8] Bruno Lacerda and Pedro U Lima. 2011. Designing Petri net supervisors from LTL specifications. *Proc. of Robotics: Science and Systems VII* (2011).

[9] Bruno Lacerda, David Parker, and Nick Hawes. 2014. Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications. In *2 014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 1511–1516.

[10] Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, Vol. 99. 278–287.

[11] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. IEEE, 46–57.

[12] Jette Randløv and Preben Alstrøm. 1998. Learning to Drive a Bicycle Using Reinforcement Learning and Shaping.. In *ICML*, Vol. 98. 463–471.