A Framework for Flexibly Guiding Learning Agents

Mahmoud Elbarbari[1,2,3*], Florent Delgrange[1], Ivo Vervlimmeren[1], Kyriakos Efthymiadis[1,4], Bram Vanderborght[1,2,3] and Ann Nowé[1]

[1]Department of Computer Science, Vrije Universiteit Brussel, Pleinlaan, Brussels, 1050, Belgium.
[2]Department of Mechanical Engineering, Vrije Universiteit Brussel, Pleinlaan, Brussels, 1050, Belgium.
[3]Imec, Kapeldreef, Leuven, 3001, Belgium.
[4]Department of Pharmaceutical and Pharmacological Sciences, University of Leuven (KU Leuven), Herestraat, Leuven, 3000, Belgium.

*Corresponding author(s). E-mail(s): mahmoud.elbarbari@vub.be;
Contributing authors: florent.delgrange@vub.be; ivo.vervlimmeren@vub.be; kyriakos.efthymiadis@vub.be; bram.vanderborght@vub.be; ann.nowe@vub.be;

Abstract

Reinforcement Learning (RL) enables artificial agents to learn through direct interaction with the environment without the need for perfect models. However, RL usually does not scale up well to large problems as learning from interaction comes at a high sampling cost. Several approaches have been proposed to incorporate domain knowledge in RL to allow for more efficient learning. Reward Shaping is one well-established approach to incorporate domain knowledge in RL by providing the learning agent with supplementary rewards. In this work, we propose a novel methodology that automatically generates reward shaping functions from user-provided Linear Temporal Logic on finite traces (LTL_f) formulas. LTL_f in our work serves as a rich language that allows the user to communicate domain knowledge to the learning agent. In both single and multi-agent settings, we demonstrate that our approach performs at least as well as the baseline approach while providing essential advantages in terms of flexibility and ease of use. We elaborate on some of these advantages empirically by demonstrating that our approach can handle domain knowledge functions with different levels of accuracy and provides the user with the flexibility to express aspects of uncertainty in the provided advice.

Keywords: Linear Temporal Logic on finite traces, Reward Shaping, Reinforcement Learning, Multi-Agent Systems

1 Introduction

Reinforcement Learning (RL, [26]) enables artificial agents to learn through direct interaction with the environment without the need for perfect models. This makes RL relevant for controlling agents in numerous applications e.g. when it is difficult to obtain a model that allows for computing an optimal policy. However, vanilla RL starts from a blank slate and improves only through trial and error. Such a learning approach might take a huge amount of trials until the learning agent can reach a satisfying policy which is prohibitively expensive in many scenarios. One step in the right direction is to incorporate available domain knowledge in RL.

The use of prior knowledge in the form of advice, feedback, and demonstrations has been proven to significantly increase the sample efficiency of the RL algorithms in single-agent settings e.g. [27, 17, 16, 25, 21, 6]. But little has been done in the area in multi-agent settings e.g. [5].

Reward Shaping (RS) is a well-established method to incorporate domain knowledge in RL by providing the learning agent with supplementary rewards. When used properly, RS becomes a powerful tool that enables agents to learn more efficiently and thus reach a satisfying policy faster. But when used improperly, RS can change the optimal policy of the original task e.g [23]. That said, RS is however guaranteed to preserve the optimal policy in single-agent settings [20] and the Nash Equilibria in multi-agent settings [7] when expressed as potential-based reward shaping functions (PBRS). The guarantee also holds, given additional conditions, in the case of dynamic potential functions [9] and in episodic RL [11].

In this work, we introduce a novel methodology that incorporates domain knowledge in the RL agents through Linear Temporal Logic on finite traces (LTL_f) formulas. The LTL_f formulas are used to generate PBRS functions to provide the agent with supplementary rewards which in turn allow more efficient learning. This work also investigates several ways to extend the methodology to multi-agent systems (MAS). In MAS, we use our method to guide the coordination behaviors between agents in cooperative teams. In both single and multi-agent settings, we elaborate on the flexibility LTL_f provides in guiding the learning agents especially in the case of uncertainty.

Grzes et al. [12] propose a method to generate PBRS functions from STRIPS plans. The authors demonstrate their method on the challenging flag collection domain where a vanilla RL agent can easily get stuck into a sub-optimal policy. In this work, we compare our approach to the plan-based approach using different variants of the flag collection domain.

Camacho et al. [2] introduce a method to transform formal languages, including LTL_f , to Reward Machines (RMs) [14], a type of finite-state automaton used to specify Markovian and *a subset* of non-Markovian rewards. The authors argue that the artificial agents cannot inherently perceive the reward from the environment (although e.g. [18] is a counterexample), and thus they expose the reward function (in this case a set of RMs) to the agent so that it can exploit its underlying structure. As a consequence, the authors opt to develop *tailor-made* learning algorithms to take advantage of the exposed reward function. Although the authors' method is indeed interesting when a designer explicitly writes the reward function, this assumption is not aligned with the general goal of designing autonomous learning agents. In this work, we instead assume a Markovian reward function that is unknown to the agent, and we use LTL_f to provide supplementary rewards to guide the RL agent to reach a satisfying policy faster.

To the best of our knowledge, the work of Icarte et al. [15] is the only work that uses LTL_f to guide the RL agent's learning. The authors use heuristics generated from LTL_f formulas to bias the agent's exploration in a *tailor-made* variant of Rmax. Our work is highly motivated by their findings but instead, we propose to use LTL_f to automatically generate PBRS functions which makes our method more advantageous in two main ways: 1. Incorporating the heuristics in the RL agent as a supplementary reward via PBRS makes our method applicable to many RL algorithms while preserving the optimal policy (or Nash Equilibria in MAS) of the original problem. 2. Our method tracks the stage of satisfaction of the LTL_f formulas to guide the agent to achieve temporal subgoals and thus increasing the learning speed. Moreover, in our work, we focus on elaborating on the flexibility LTL_f provides in guiding learning agents in both single and multi-agent settings, including settings where the user provides LTL_f advice with aspects of uncertainty.

2 Preliminaries

For a *finite* set $\mathcal{X}$, let $\mathcal{D}(\mathcal{X}) = \left\{P\colon \mathcal{X} \to [0,1] \mid \sum_{x\in\mathcal{X}} P(x) = 1\right\}$ be the set of *probability distributions* over $\mathcal{X}$. We write $\mathbb{N}_0 = \mathbb{N}\setminus\{0\}$, $[T] = \{n \in \mathbb{N} \mid n \leq T\}$, and $[T]_0 = [T] \setminus \{0\}$. For $\boldsymbol{x} = \langle x_0, \ldots, x_n\rangle \in \mathbb{R}^n$, let $\boldsymbol{x}[\![i]\!]$ be the i^{th} entry of $\boldsymbol{x}$, i.e., x_i.

2.1 Reinforcement Learning and Markov Decision Processes

We start this section by introducing Markov Decision Processes (MDPs) as the framework for formalizing the problem of RL. MDPs [1] are used to formalize the problem of sequential decision-making. An MDP can be represented as a tuple $\langle S, A, \delta, R, \gamma \rangle$: S is the set of states, A is the set of actions, $\delta\colon S \times A \rightarrow \mathcal{D}(S)$ is the transition function, $R\colon S \times A \times S \rightarrow \mathbb{R}$ 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 the next state s' according to $\delta\ (s' \mid s, a)$ and receives a scalar reward $r \in \mathbb{R}$. Therefore, a *finite trajectory* of the MDP is a sequence of states and actions $\langle s_{0:T}, a_{0:T-1} \rangle$ such that $\delta(s_{t+1} \mid s_t, a_t) > 0$ for all $t \in [T]$, $T \in \mathbb{N}_0$. The RL agent should learn a policy $\pi\colon S \rightarrow \mathcal{D}(A)$ to maximize the cumulative discounted reward, where the policy is a probabilistic function that defines the probability of selecting an action $a \in A$ at a given state $s \in S$. The cumulative discounted reward under the agent's current policy π is the expectation of the sum of the discounted rewards the agent receives $\mathbb{E}_\pi[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots]$.

2.2 Potential-based Reward Shaping

In this work, we use PBRS to incorporate prior knowledge in RL. RS is a well-established method that allows the RL agent to learn more efficiently. RS works by providing the RL agent with a supplementary reward $F(s_t, s_{t+1})$. So the augmented reward function of the MDP becomes

$$R'(s_t, a_t, s_{t+1}) = F(s_t, s_{t+1}) + R(s_t, a_t, s_{t+1}) \tag{2.1}$$

PBRS represents the RS function as the difference of values of a potential function, formally defined as the following:

$$F(s_t, s_{t+1}) = \gamma \Phi(s_{t+1}) - \Phi(s_t) \tag{2.2}$$

The reward function in this form is both sufficient and necessary to guarantee policy invariance [20], meaning that even if the domain knowledge is wrong, the agent will eventually still learn the optimal policy. In the episodic case, in the case of multiple terminal states, the potential of the terminal states must also be set to zero [11].

The PBRS functions can also be dynamic i.e., change during the learning process [9]. Policy invariance is preserved if the RS function is given in the following form

$$F(s_t, s_{t+1}) = \gamma \Phi_{t+1}(s_{t+1}) - \Phi_t(s_t) \tag{2.3}$$

2.3 Multi-agent Reinforcement Learning

In this section, we introduce Multi-agent Reinforcement Learning (MARL) as our method can also be applied to MAS. MDPs can be extended to the multi-agent case by formalizing the environment as a *Markov Game* (MG), which is defined as a tuple $\langle N, \boldsymbol{S}, \boldsymbol{A}, \delta, R, \gamma \rangle$ where N is the number of agents, $\boldsymbol{S} \subseteq S^N$ is the joint state space of the MG with individual state space S, $\boldsymbol{A} \subseteq A^N$ is the joint action space with individual action space A, $R \colon \boldsymbol{S} \times \boldsymbol{A} \times \boldsymbol{S} \to \mathbb{R}$ is the reward function, and $\delta \colon \boldsymbol{S} \times \boldsymbol{A} \to \mathcal{D}(\boldsymbol{S})$ is the probabilistic transition function. Intuitively, $\delta(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a})$ gives the probability of going from joint state $\boldsymbol{s} = \langle s_1, \ldots, s_N \rangle$ to the next one $\boldsymbol{s}' = \langle s'_1, \ldots, s'_N \rangle$ when each agent has chosen its action $a_i \in A$, forming this way the joint action $\boldsymbol{a} = \langle a_1, \ldots, a_N \rangle$, where $s_i \in S$ represents the current state of agent $i \in [N]_0$. Finite trajectories of MGs are defined in a similar way to those of MDPs.

2.4 Linear Temporal Logic

Linear Temporal Logic (LTL, [22]) 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*, i.e., LTL$_f$ [4]. Formulas of LTL$_f$ are constructed from a set of *propositional symbols* $\mathcal{P}$ and closed under boolean connectives and temporal operators X (next) and U (until). The LTL$_f$ syntax is recursively defined as

$$\varphi ::= \top \mid \alpha \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U} \varphi_2$$

where $\alpha \in \mathcal{P}$ and $\top$ is read as "True". Intuitively, $\mathsf{X}\varphi$ means that φ holds at the next time step, while $\varphi_1 \mathsf{U} \varphi_2$ means that φ_1 holds in the current step and *will* hold in the next steps until φ_2 holds. LTL$_f$ allows for standard logical connective abbreviations via $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, and $\varphi_1 \to \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$. Temporal operators can further be derived: $\mathsf{F}\varphi \equiv \top \mathsf{U} \varphi$ (finally) means that φ will eventually hold, and $\mathsf{G}\varphi \equiv \neg\mathsf{F}\neg\varphi$ (globally) means that φ always hold from the current time step. Note that, compared to classic LTL, $\neg\mathsf{X}\top$ always eventually holds, since executions are assumed finite.

To enable reasoning about finite executions, we attach a *labeling function* $\ell \colon S \to 2^{\mathcal{P}}$ to the MDP, mapping each state to its labels formed of literals. A *trace* τ of this MDP records the labels encountered through a trajectory: this is a sequence $\tau = l_{0:T}$ such that $l_t = \ell(s_t)$ for some trajectory $\langle s_{0:T}, a_{0:T-1} \rangle$ and all $t \in [T]$. Let τ_t define the *segment* of τ starting from $t \in [T]$, i.e., $\tau_t = l_{t:T}$, then the semantics of LTL$_f$ formulas are defined recursively as follows:

- $\tau_t \models \top$,
- $\tau_t \models \alpha$ iff $\alpha \in l_t$ for $\alpha \in \mathcal{P}$,
- $\tau_t \models \neg\varphi$ iff $\tau_t \not\models \varphi$,
- $\tau_t \models \varphi_1 \wedge \varphi_2$ iff $\tau_t \models \varphi_1$ and $\tau_t \models \varphi_2$,
- $\tau_t \models \mathsf{X}\varphi$ iff $t < T$ and $\tau_{t+1} \models \varphi$, and

- $\tau_t \models \varphi_1 \mathsf{U} \varphi_2$ iff $\exists i \in \mathbb{N}$ with $t \leq i \leq T$ such that $\tau_i \models \varphi_2$ and $\forall j \in \mathbb{N}$ such that $t \leq j < i$, $\tau_j \models \varphi_1$.

We enable LTL$_f$ for MGs by considering formulas constructed from $\mathcal{P}' = (\mathcal{P} \cup \{\star\})^N \setminus \{\boldsymbol{\alpha}^\star\}$, where $\star$ is a special symbol read as "any" and $\boldsymbol{\alpha}^\star[\![i]\!] = \star$ for all $i \in [N]_0$. Consider a Markov game with state space $\boldsymbol{S} \subseteq S^N$ and let ℓ be the labeling function defined over the individual state space S, a trace of this MG is a multi-labeled sequence $\tau = \boldsymbol{l}_{0:T}$ recording the labels encountered by each agent through some trajectory $\langle \boldsymbol{s}_{0:T}, \boldsymbol{a}_{0:T-1} \rangle$: $\boldsymbol{l}_t[\![i]\!] = \{\langle \alpha_1, \ldots, \alpha_N \rangle \mid \alpha_i \in \ell(\boldsymbol{s}_t[\![i]\!]) \cup \{\star\}, \text{ for all } i \in [N]_0\}$ with $t \in [T]$. Intuitively, an LTL$_f$ formula over $\mathcal{P}'$ features multi-dimensional literals $\boldsymbol{\alpha} = \langle \alpha_1, \ldots, \alpha_N \rangle \in \mathcal{P}'$ (one literal per agent), where α_i represents either a literal that could be encountered by the i^{th} agent through a trajectory, or *any* literal, encoded via the special symbol $\star$. That way, according to the LTL$_f$ semantics and by definition of MG traces, $\boldsymbol{\alpha}$ does not hold in τ_t (written $\tau_t \models \neg\boldsymbol{\alpha}$) iff there is an agent i which has not encountered the i^{th} (specific) literal from $\boldsymbol{\alpha}$, i.e., $\boldsymbol{\alpha}[\![i]\!] \notin \ell(\boldsymbol{s}_t[\![i]\!])$ and $\boldsymbol{\alpha}[\![i]\!] \neq \star$. Notice that $\boldsymbol{\alpha}$ records at least one specific literal: by definition of $\mathcal{P}'$, all entries of $\boldsymbol{\alpha}$ cannot be $\star$ simultaneously.

Any LTL$_f$ formula φ can be translated to a Deterministic Finite Automaton (DFA) that accepts a trace only if there exists an associated trajectory τ that satisfies formula φ, i.e., $\tau_0 \models \varphi$ [10]. DFAs are used in this work to track the stage of satisfaction of the LTL$_f$ formulas.

2.5 Deterministic Finite Automaton

A Deterministic Finite Automaton (DFA, e.g., [13]) is a tuple $\langle Q, q_I, \Delta, \Sigma, F \rangle$ where Q is the set of states, $q_I \in Q$ is the initial state, Σ is the set of possible inputs, $\Delta \colon Q \times \Sigma \to Q$ is the transition function (maps a state and an input to another state) and $F \subseteq Q$ is the set of accepting states. DFAs are mathematical models that deterministically map an input sequence to an output so that the computation is unique. Therefore, it can be exactly in one state at a given time and it makes the transition from one state $q \in Q$ given some input $\sigma \in \Sigma$ to another state $q' \in Q$ according to the transition function Δ, i.e., $\Delta(q, \sigma) = q'$. A finite input sequence $\sigma_{0:T}$ is accepted by the DFA if the sequence drives it to one of its accepting states, i.e., if there is a sequence $q_{0:T}$ where $q_0 = q_I$, $q_T \in F$, and $q_{t+1} = \Delta(q_t, \sigma_t)$ for all $t \leq T$ and $T \in \mathbb{N}$.

DFAs translated from LTL$_f$ take state labels as input, i.e., $\Sigma = 2^{\mathcal{P}}$ for MDPs and $\Sigma = 2^{\mathcal{P}'}$ for MGs. We define the *edges* of such DFAs via the *guarded* transition relation $\xrightarrow{\phi} = \{\langle q, l, q' \rangle \in Q \times \Sigma \times Q \mid \Delta(q, l) = q' \text{ and } l \models \phi\}$, where ϕ is an LTL$_f$ formula whose syntax is restricted to the following fragment without other logical connectives abbreviations permitted:

$$\phi ::= \top \mid \alpha \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2.$$

Whenever a state q goes to q' through a transition guarded by ϕ, written $q \xrightarrow{\phi} q'$, we have for all labels $l \in \Sigma$ that $\Delta(q, l) = q'$ iff $l \models \phi$, where l is

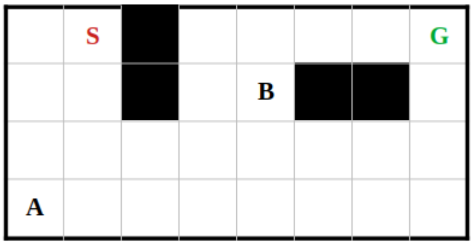

Fig. 1: An example of flag collection problem; the start position (S in red), the flag A and B (A and B in black) and the goal position (G in green). Starting from the start position the agent should collect flags A and B and then go to the goal, in the minimum number of steps.

here considered as a trace τ of unit size (i.e., $\tau = l$). We call this fragment $\text{LTL}_f^{\rightarrow}$ and we use it henceforth to formulate the guards of the transitions of DFAs translated from LTL_f formulas.

3 Methodology

In this section, we introduce our method that utilizes LTL_f to incorporate domain knowledge in RL agents. Our usage of LTL_f as a guidance formalism for RL agents is motivated by the following aspects: 1. LTL_f provides an intuitive way to describe many common RL tasks (e.g. [15, 2]). 2. LTL_f is a rich language that can validate whole sequences over time which makes LTL_f a compact language for providing guidance. 3. LTL_f is rather easy to be expressed in natural language. Thus, it has the potential to be easier for non-expert users, especially when syntactic sugar is added. 4. LTL_f can allow the user to express uncertainty about his/her advice as we will demonstrate in this work.

Example 3.1 (Flag collection domain [12])**.** We consider a simple flag collection problem (Figure 1) as a running example to introduce our approach. In the flag collection domain, the agent should learn how to collect all the flags and then go to the goal position, in the minimum number of steps. The state is defined by the agent's 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 the case of an obstacle in the intended direction of the executed action). The agent collects a flag automatically when it is at the position of the flag. 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. This sparsity in the reward makes the problem challenging to a vanilla RL agent.

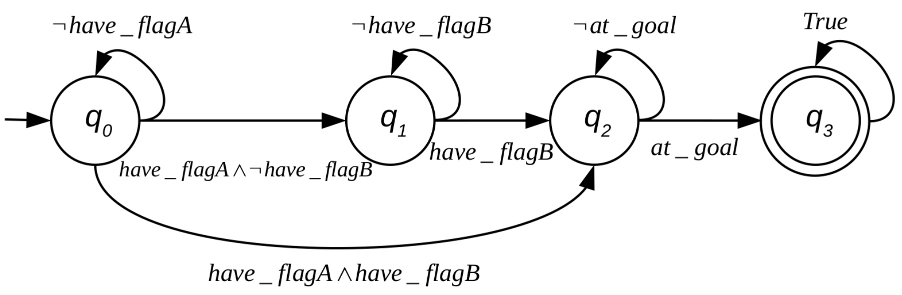

Fig. 2: The corresponding DFA for the LTL$_f$ formula in Equation 3.1. This DFA still accepts a trajectory even if flag B is collected first. For the sake of readability, we draw the transitions with their corresponding edges guarded by LTL$_f^{\rightarrow}$ formulas.

3.1 Constructing the LTL$_f$ Formula

Typically in RL tasks, the available domain knowledge is in the form of abstract (high-level) heuristics and it is up to the RL agents to learn the optimal policy i.e. which action should be taken in each of the states the agent encounters. Our method allows the user to communicate this abstract domain knowledge using an LTL$_f$ formula.

The LTL$_f$ formula is constructed from a finite set of propositional symbols $\mathcal{P}$. Intuitively, $\mathcal{P}$ relates to the task at hand and the guidance the user wants to provide.

Example 3.1. (*continued*) Related to the running example, the set of propositional symbols $\mathcal{P} = \{have\_flagA, have\_flagB, at\_goal\}$ where $have\_flagA$ and $have\_flagB$ hold in states where the agent has collected flag A and flag B respectively and at_goal holds in states where the agent is at the goal position. In this problem, the user may provide the following LTL$_f$ formula:

$$\mathsf{F}(have\_flagA \wedge \mathsf{F}(have\_flagB \wedge (at\_goal \vee \mathsf{XF}at\_goal))) \qquad (3.1)$$

this formula is still satisfied even if flag A is not collected before flag B (this is more clear when looking at the corresponding DFA in Figure 2). This kind of LTL$_f$ formula is relevant when the user is uncertain about the correct order. We will explain down the road how our method would utilize this kind of formulas to encourage the agent to follow the user's recommended ordering but still give the supplementary rewards even if the ordering is not respected. This formula is best expressed in natural language as "*Collect flag A then flag B, probably best in this ordering, and then go to the goal position*".

3.2 Transforming LTL$_f$ Formula to DFA

We transform the user-provided LTL$_f$ formula to a DFA in order to track the satisfaction stage of the corresponding LTL$_f$ formula[1]. In this case, the DFA would provide abstract subgoals that we use to guide the agent to satisfy. By doing so, we guide the agent towards the accepting states of the DFA and thus satisfying the corresponding LTL$_f$ formula.

Example 3.1. (*continued*) The DFA $\langle Q, q_I, \Delta, \Sigma, F\rangle$ translated from the LTL$_f$ formula of Equation 3.1 is depicted in Figure 2. Here, the set of states is $Q = \{q_0, q_1, q_2, q_3\}$, the initial state is $q_I = q_0$, the set of accepting states is $F = \{q_3\}$, and the alphabet is $\Sigma = 2^{\mathcal{P}} = 2^{\{have\_flagA, have\_flagB, at\_goal\}}$. We draw transitions with their corresponding edges, guarded by LTL$_f^{\rightarrow}$ formulas.

3.3 Tracking the Current State of the DFA

As stated earlier, the DFA provides subgoals towards satisfying the LTL$_f$ formula. Therefore, our method generates potential functions that depend on the *current* state of the DFA. Using the labeling function ℓ, one can track the current state of the DFA given the trace associated with the current trajectory resulting from the interaction of the agent with the environment: let s_t be the current state of the environment at time step $t \in \mathbb{N}$, the DFA updates its current state q_t to q_{t+1} according to the label observed in s_t, i.e., $q_{t+1} = \Delta(q_t, l)$, where Δ is the DFA transition function and l is the label observed.

Example 3.1. (*continued*) Let $\langle s_{0:T}, a_{0:T-1}\rangle$ be a finite trajectory resulting from the interaction of the agent with the environment. The initial state of the DFA of Figure 2 is q_0. The DFA transitions out of q_0 when the literal $have\_flagA$ is encountered for the first time, i.e., if $\exists t_1 \in [T]$ such that $have\_flagA \in \ell(s_{t_1})$ and $have\_flagA \notin \ell(s_t)$ for all $t < t_1$. If $have\_flagB \notin \ell(s_{t_1})$, then q_1 is visited. Otherwise, the DFA transitions directly to q_2.

The DFA transitions to q_2 when $have\_flagB$ is visited for the first time and $flag\_A$ has been visited beforehand, i.e., if $\exists t_2 \in [T]$, such that $t_2 \geq t_1$, $have\_flagB \in \ell(s_{t_2})$, and $have\_flagB \notin \ell(s_t)$ for $t_1 < t < t_2$. From q_2, the DFA transitions to the accepting state q_3 whenever at_goal is finally visited, i.e., if $\exists t_3 \in [T]$ where $\exists t_3 > t_2$ such that $at\_goal \in \ell(s_{t_3})$. In this case, the DFA has reached its only (absorbing) accepting state, thus the provided LTL$_f$ formula is satisfied.

3.4 Estimating the Number of Transitions to an Accepting State

Estimating how close the agent is to reach an accepting state of the DFA provides heuristics that we use later to assign potentials to the MDP states.

[1]We use the LTLf2DFA tool to obtain minimal DFA: github.com/whitemech/LTLf2DFA

Therefore, we make use of *the longest acyclic path* to an accepting state as a heuristic to estimate the number of transitions required in the DFA to finally enter an accepting state. We define the set of acyclic paths from a given DFA state q to a final state through the mapping $acyclic\colon Q \to Q^*$,

$$\begin{aligned} q \mapsto \{q_{0:T} \mid & \forall t \leq T,\, \exists \phi \text{ s.t. } \phi \text{ is an } \mathrm{LTL}_f^{\rightarrow} \text{ formula and } q_t \xrightarrow{\phi} q_{t+1}, \\ & q_t = q' \implies \forall t' \in [T] \text{ with } t < t',\, q_{t'} \neq q', \text{ and} \\ & q_0 = q,\, q_T \in F\}. \end{aligned}$$

Then, we define the longest acyclic path via the distance function $Distance\colon Q \to \mathbb{N}$,

$$q \mapsto \begin{cases} \max\limits_{q_{0:T} \in acyclic(q)} T & \text{if } acyclic(q) \neq \emptyset \\ \max\limits_{q' \in Q,\, q'_{0:T} \in acyclic(q')} T + 1 & \text{otherwise.} \end{cases}$$

Intuitively, *Distance* assigns the length of the longest path starting from a given DFA state $q \in Q$. In case such a path does not exist (i.e., $acyclic(q) = \emptyset$), then it assigns the length of the longest acyclic path which can be reached in the DFA plus one.

The estimate we choose here fits the type of guidance we want to provide to the RL agent (guidance with uncertainty) e.g. the LTL_f formula in Equation 3.1. However, the choice of the estimate is flexible and can be left for the designer to adapt it to the problem and/or the type of guidance desired.

Example 3.1. (*continued*) The mapping *Distance* assigns the following values to the DFA states of Figure 2: $Distance(q_0) = 3$, $Distance(q_1) = 2$, $Distance(q_2) = 1$, and $Distance(q_3) = 0$.

3.5 Defining the Guiding Formula

We also utilize another heuristic to guide the RL agent to satisfy the LTL_f formula. This heuristic estimates how close the agent is from satisfying any *useful edge*, an edge that leads to progress in the DFA. An edge $q \xrightarrow{\phi} q'$ of the current DFA state q is a *useful* edge if $Distance(q') < Distance(q)$. We denote the set of useful edges at time step $t \in \mathbb{N}$ as U_t.

Now, we define *the guiding formula* as the disjunction of the formulas associated to the useful edges. This is formally defined at step $t \in \mathbb{N}$ as follows:

$$\phi_t = \bigvee_{q \xrightarrow{\phi} q' \in U_t} \phi \tag{3.2}$$

Example 3.1. (*continued*) In Figure 2, if the DFA state is q_0 at time step $t \in \mathbb{N}$, then $U_t = \{q_0 \xrightarrow{have\_flagA \wedge \neg have\_flagB} q_1, q_0 \xrightarrow{have\_flagA \wedge have\_flagB} q_2\}$

and $\phi_t = have\_flagA$, if it is in q_1, then $U_t = \{q_1 \xrightarrow{have\_flagB} q_2\}$ and $\phi_t = have\_flagB$, and if it is in q_2, then $U_t = \{q_2 \xrightarrow{at\_goal} q_3\}$ and $\phi_t = at\_goal$. Finally, if the DFA current state is the accepting state q_2, then $U_t = \emptyset$ and $\phi_t = \top$.

3.6 Estimating the Number of Primitive Actions to a DFA Transition

To estimate how close the RL agent is to make a transition through a useful edge, we borrow the definition of a domain knowledge function from the work of Icarte et al. [15]. This assumes that the agent has domain knowledge expressed as a function $h_\alpha\colon S \to \mathbb{R}$ for each $\alpha \in \mathcal{P}^+$, where $\mathcal{P}^+ = \mathcal{P} \cup \{\neg p : p \in \mathcal{P}\}$, that provides an estimate of the number of actions needed for α to be satisfied from any MDP state $s \in S$. Using domain knowledge functions, we define a more general function $\vec{h}\colon S \times \vec{\Phi}(\mathcal{P}^+)$ (where $\vec{\Phi}(\mathcal{P}^+)$ is the set of all LTL$_f^{\rightarrow}$ formulas over $\mathcal{P}^+$), providing an estimate of the number of actions needed to satisfy any LTL$_f^{\rightarrow}$ formula ϕ from a given state $s \in S$. Given that ϕ is put in the Disjunctive Normal Form (DNF), which means that ϕ is a disjunction of conjunctions, we define $\vec{h}$ recursively as follows:

$$\begin{aligned} \vec{h}(s,\alpha) &= h_\alpha(s) && \textit{for any } \alpha \in \mathcal{P}^+ \\ \vec{h}(s,\phi_1 \wedge \phi_2) &= \vec{h}(s,\phi_1) + \vec{h}(s,\phi_2) && \textit{for any LTL}_f^{\rightarrow} \textit{ formulas } \phi_1 \textit{ and } \phi_2 \\ \vec{h}(s,\phi_1 \vee \phi_2) &= \min\left\{\vec{h}(s,\phi_1), \vec{h}(s,\phi_2)\right\} && \textit{for any LTL}_f^{\rightarrow} \textit{ formulas } \phi_1 \textit{ and } \phi_2 \end{aligned} \tag{3.3}$$

Example 3.1. (*continued*) Consider the MDP modeling the flag collection problem of Figure 1 as well as the formula translated to the DFA of Figure 2. We define a domain knowledge function for propositional symbols from $\mathcal{P} = \{have\_flagA, have\_flagB, at\_goal\}$ as follows. Let $\alpha \in \{have\_flagA, have\_flagB\}$,

$$h_\alpha(s) = \begin{cases} \vec{d}_\alpha(s) & \text{if } \alpha \notin \ell(s) \\ 0 & \textit{otherwise} \end{cases} \tag{3.4}$$

$$h_{\neg\alpha}(s) = \begin{cases} 0 & \text{if } h_\alpha(s) \neq 0 \\ \mathsf{undefined} & \textit{otherwise} \end{cases} \tag{3.5}$$

$$h_{at\_goal}(s) = \vec{d}_{at\_goal}(s) \tag{3.6}$$

$$h_{\neg at\_goal}(s) = \begin{cases} 0 & \text{if } h_{at\_goal}(s) \neq 0 \\ 1 & \textit{otherwise} \end{cases} \tag{3.7}$$

for each $s \in S$ and where $\vec{d}_{have\_flagA}(s)$, $\vec{d}_{have\_flagB}(s)$, and $\vec{d}_{at\_goal}(s)$ are heuristics functions that return estimated distances (e.g., ℓ_1 or ℓ_2 distance disregarding the obstacles) of state s to flag A, flag B and the goal position respectively. The domain knowledge functions for satisfying $\neg have\_flagA$ and $\neg have\_flagB$ are not defined when the agent already has flag A (or flag B) since the agent cannot drop a flag after it is collected in the flag collection domain.

3.7 Building the Potential Function

Finally, we define the potential of state $s \in S$ at step $t \in \mathbb{N}$ as follows

$$\Phi_t(s_t) = -\omega \times \left[Distance(q_t) + \frac{1}{n} \cdot \vec{h}(s_t, \phi_t) \right] \tag{3.8}$$

where ω is a scaling factor, q_t is the current state of the DFA and $1/n$ is a normalization factor where $n = \max_{s \in S} \vec{h}(s, \phi_t)$.

The intuition behind Equation 3.8 is that a state s_t gets a higher potential if the DFA is closer to their accepting states at time t. This "closeness" is proportional to the evaluation of the *Distance* function given the current DFA state q_t (3.4, an estimate of the number of DFA transitions to an accepting state) and the evaluation of the heuristics function $\vec{h}$ given the current guiding formula ϕ_t (3.6, an estimate of the number of primitive actions needed to make a "useful" DFA transition).

3.8 The Augmented Reward Function

We now build the PBRS function as the difference between different evaluations of the potential function Φ_t, formally as the following

$$F(s_t, s_{t+1}) = \gamma \Phi_{t+1}(s_{t+1}) - \Phi_t(s_t) \tag{3.9}$$

and then we combine the original reward function of the MDP and the shaping reward function as commonly done by taking the sum. Formally, the augmented reward function is

$$R'(s_t, a_t, s_{t+1}) = F(s_t, s_{t+1}) + R(s_t, a_t, s_{t+1}) \tag{3.10}$$

4 Experiments in Single-agent Settings

In this section, we evaluate our method in the single-agent settings in the flag collection domain [12]. First, we compare the performance of the LTL$_f$ method to that of the plan-based RS method [12] in the classic flag collection problem (Figure 3). Then, we demonstrate how the LTL$_f$ method is flexible regarding the accuracy of the domain knowledge used (from vague to accurate) reporting the performance of different domain knowledge functions with different accuracy levels. Finally, we demonstrate how our method can allow the user

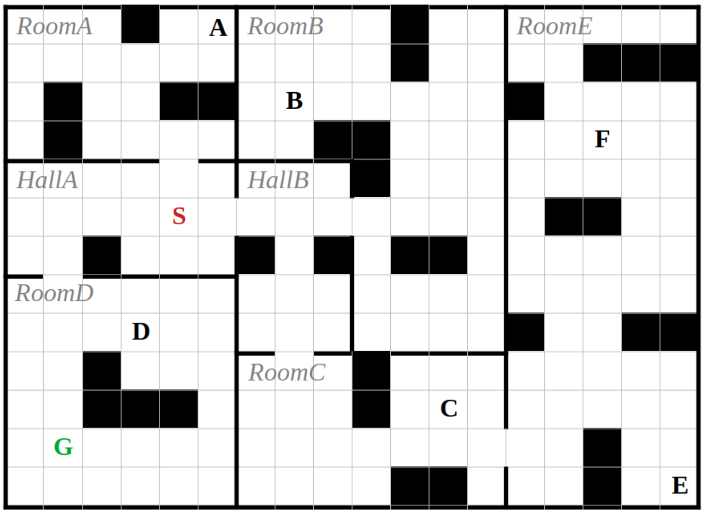

Fig. 3: The classic flag collection problem [12] (with added obstacles as black squares); the start position (S in red), the flags from A to F (in black) and the goal position (G in green). The agent should collect all flags (A to F) and then go to the goal, in the minimum number of steps.

to express partial knowledge (uncertainty) in the communicated advice using a variation of the flag collection domain (Figure 6).

Throughout this work, we use the following setup unless explicitly stated otherwise. We use vanilla SARSA [26] with the Q-table initialized to zero (we did not notice remarkable performance differences when using eligibility traces). We use a learning rate $\alpha = 0.1$ and a discount factor $\gamma = 1$ (this choice is motivated by the findings in the PhD thesis of Grzes regarding the effect of γ on the shaping rewards [2]). We use a scaling factor $\omega = 10$ for the shaping rewards. For ϵ-greedy and softmax strategies [26], we use a linearly decreasing ϵ from 0.3 to 0.01 and a fixed temperature $T = 0.25$ (where the numerical preference for choosing an action is: $H_t(a_t) = Q_t(s_t, a_t)/T$), respectively. Each experiment consists of 10^4 episodes and we report the average results over 10 experiments. The plots always show the mean (in solid line) and the standard deviation (shaded region) of the results data.

4.1 Comparing Performance to the Plan-based Approach

In this set of experiments, we compare the performance of the LTL$_f$ method and the plan-based method using the classic flag collection problem (Figure 3). This configuration is the original version used in [12] to which we add obstacles that the agent cannot go through. We add the obstacles in order to make the heuristics we use in our method realistic, more precisely by adding obstacles

[2]PhD thesis: Improving Exploration in RL through Domain Knowledge and Parameter Analysis. URL: https://etheses.whiterose.ac.uk/936/

the heuristics become less accurate yielding a more realistic setting . In this problem, there is 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 would only consider one optimal ordering in this case [3]. Our LTL$_f$ -based method instead provides the flexibility for the user to guide the agent to any optimal ordering. In our experiments, we provide the following LTL$_f$ formula

$$\mathsf{F}(have\_flagC \wedge \mathsf{F}(have\_flagE \wedge \ldots \wedge (have\_flagD \vee \mathsf{XF}\ have\_flagD)))))) \tag{4.1}$$

to help the agent to collect the flags in the following order (C, E, F, B, A then D). The LTL$_f$ formula in Equation 4.1 follows the same intuition as the LTL$_f$ formula considered in the running example (Equation 3.1). The formula guides the agent to collect the flags in a certain ordering but still provides supplementary reward even if the flags are not collected in this order.

We define the set of propositional symbols and the domain knowledge functions for all the flags in the same way as we did for flag A in the running example. Throughout this work, unless explicitly stated otherwise, we use a domain knowledge function $\vec{d}$ that returns ℓ_1 distance (disregarding the obstacles but not the walls). This heuristic knowledge is available in many applications in the real world where we would know the overall structure of the environment e.g. the room structure but not the obstacles e.g. the furniture.

In this set of experiments, we use the ϵ-greedy exploration strategy. In Figure 4, we report the results for the plan-based method and the LTL$_f$ -based method guiding the agent to the same optimal ordering, namely (C, E, F, B, A then D) (the corresponding STRIPS plan is reported in [12]). The LTL$_f$ -based method achieved slightly better performance compared to the plan-based method. Due to reward sparsity, the agent with no shaping easily gets stuck in the sub-optimal policy of picking flag D and then going directly to the goal (this behavior is also reported in [12]). The use of the LTL$_f$ method to guide the agent to other optimal orderings reported similar performance (though not reported in the plot for better visibility).

4.2 Evaluation with Different Levels of Heuristics Accuracy

In this section, we aim to demonstrate the flexibility that our method offers regarding the accuracy of the provided domain knowledge. Namely, we evaluate two less informative domain knowledge functions in the same problem (Figure 3). In the first function (we name it *to-doorstep*), the estimate distance from any state s is computed as before but to the door of the room where a literal lit becomes $\top$ (instead of its position). If the agent is in the same room in which lit becomes $\top$, the function returns 0. In the second function (we name

[3]The generated STRIPS plan in this problem is reported in Grzes et al. [12]

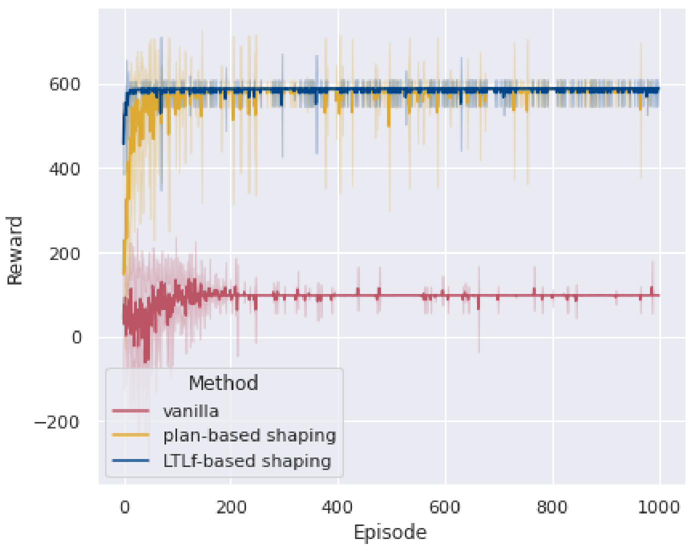

Fig. 4: Results in the classic flag collection domain show that the LTL$_f$ method at least performs as well as the plan-based method. While the vanilla SARSA agent converges to the sub-optimal solution of collecting one flag (flag D) and then go to straight to the goal. The first 1000 episodes are shown (no remarkable changes afterward).

it *no-of-doors* function), we estimate the distance between any state s to the position where a literal *lit* becomes $\top$ is the number of doors between the agent position and the position where a literal *lit* becomes $\top$ e.g. the position of a flag. We use the ϵ-greedy exploration strategy. And Figure 5 shows that the performance of the functions is comparable to ℓ_1-based function used in the previous section.

4.3 Demonstrating the Flexibility to Provide Guidance with Partial Knowledge

In this set of experiments, we aim at highlighting the flexibility LTL$_f$ can provide when the user has partial knowledge about the task. Therefore, 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 e.g. if the (Room C - Room E) door is open, the optimal ordering would be to collect flag B first then flag C. We assume the case that the user providing the guidance (in our case the LTL$_f$ formulas) does not know which door is open at the current experiment, therefore he/she does not know which ordering is the optimal one.

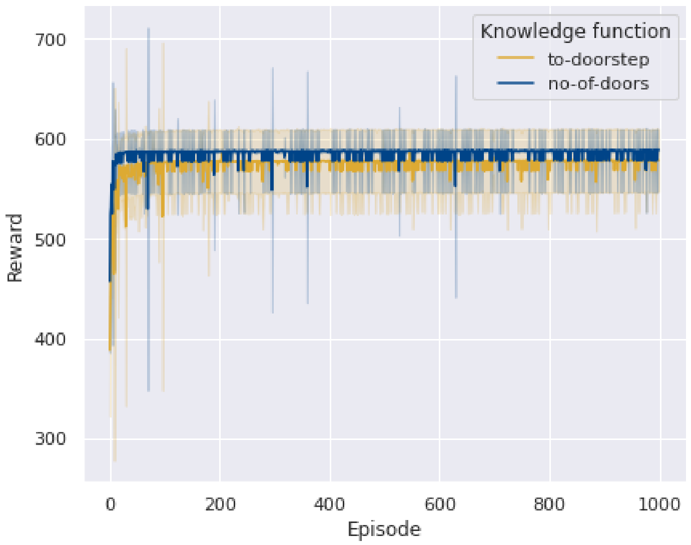

Fig. 5: The performance of domain knowledge functions with different accuracy levels in the class flag collection problem. Despite less knowledge domain , both functions perform comparably to the performance of the ℓ_1 distance function used in the previous section. The first 1000 episodes are shown (no remarkable changes afterward).

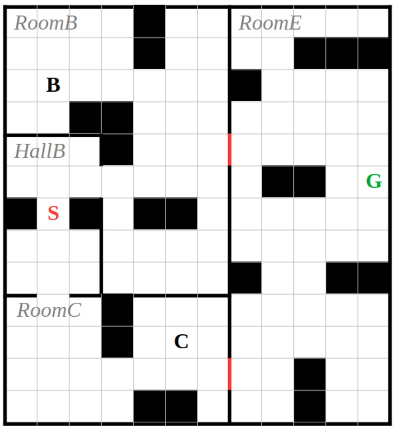

Fig. 6: A variation of the flag collection problem where the user does not know which door is open (in red). As before, S in red is the start position, G in green is the goal position and the flags B and C are in black.

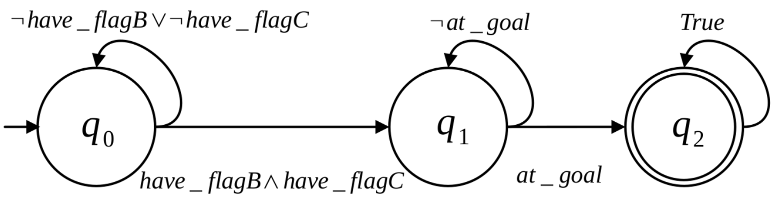
$\neg have\_flagB \vee \neg have\_flagC$
q_0
$have\_flagB \wedge have\_flagC$
$\neg at\_goal$
q_1
at_goal
True
q_2

Fig. 7: The corresponding DFA for the LTL$_f$ formula $F((have\_flagB \wedge have\_flagC) \wedge XFat\_goal))$. The formula guides the agent to collect flag B and flag C in any order and then go to the goal position.

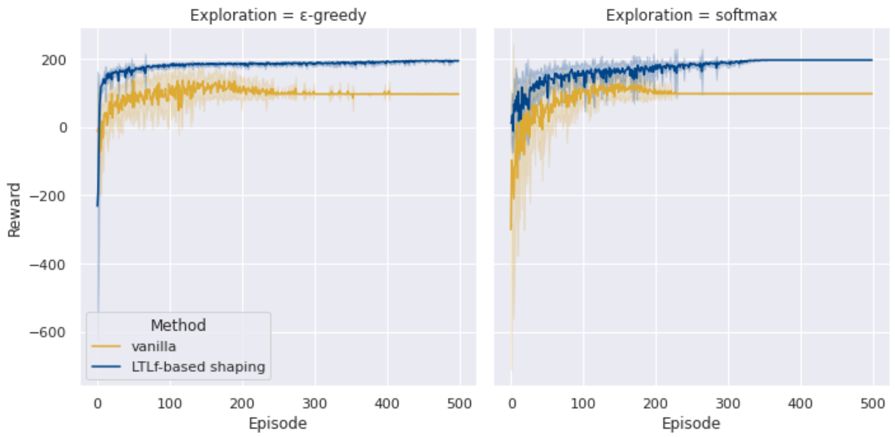

Fig. 8: Results in the flag collection domain with uncertainty (in the case of the door (Room C - Room E) is the open door) show that with no shaping the agent fails to collect both flags regardless of the exploration strategy used. While the agent with LTL$_f$-based shaping always collected both flags in the correct ordering when utilizing softmax (although this is not clear in the plots due to the relatively low difference in reward between the orderings).

In this case, we would like to have the flexibility to guide the agent to collect both flags without indicating an ordering and then to go to the goal. To the best of our knowledge, the plan-based RS method does not offer such flexibility. In LTL$_f$, this can be expressed elegantly by providing the following formula

$$\mathsf{F}((have\_flagB \wedge have\_flagC) \wedge \mathsf{XF}at\_goal)) \tag{4.2}$$

which can be expressed in natural language as "*Collect flag B and flag C in any order, and then go to the goal*". The corresponding DFA is depicted 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 RS 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.

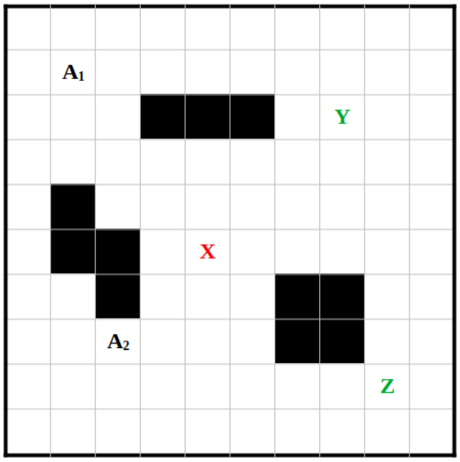

Fig. 9: An instance of the rendez-vous domain [19]. The agents $A1$ and $A2$ (both in black) need to meet at position X (in red) and then go to positions Y and Z (both in green), respectively, in the minimum number of steps.

In this set of experiments, we use both ϵ-greedy and softmax exploration strategies with LTL$_f$-based RS, and we also compare between the performance of LTL$_f$-based agent and vanilla SARSA.

In the set of experiments where (Room B - Room E) is the open door, the LTL$_f$-based agent managed to converge to the optimal ordering regardless of the exploration strategy used[4]. However, when (Room B - Room E) door is the open door, the LTL$_f$-based agent utilizing ϵ-greedy could not converge even once to the optimal ordering, while the softmax strategy always converged to the optimal ordering (Figure 8) (although this is not clear in the plots due to the relatively low difference in reward). 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. And finally, the agent with no shaping, as expected, always gets stuck at the sub-optimal policy of collecting one flag and then going directly to the goal.

5 Extension to Multi-agent Settings

In this section, we extend our method to MAS and then demonstrate in different domains the advantages our method can offer in MAS. Namely, we investigate two ways to utilize our method in MAS: *centralized guidance* and *decentralized guidance*. We use a variant of the rendez-vous domain [19] to

[4]We do not display the plots of this set of experiments because of the similarity to the plots in Figure 8

serve as a running example to introduce our method in MAS. In the rendez-vous domain, the agents should all meet at a rendez-vous position and then go to their (own) final positions, in the minimum number of steps.

Example 5.1 (Rendez-vous domain)**.** We consider a rendez-vous problem (Figure 9) with two agents. The agents should meet at position X and then should go to positions Y and Z, respectively. Each agent's state is defined by its position and whether the rendez-vous happened or not. The agents have 4 actions that deterministically move it to the adjacent cells in the conventional directions (the agent remains in place in the case of an obstacle). The agents share the same reward, namely the agents are punished with a small negative reward of -0.1 at each time step and receive a positive reward of 100 when they complete the task successfully. The positions Y and Z are absorbing states for the first and second agents, respectively. And the episode ends when both agents reach their final positions whether the task was completed successfully or not, i.e. whether the rendez-vous happened.

5.1 Decentralized Guidance

In the decentralized guidance approach, we consider the case where different users are guiding the different agents in a MAS without coordination e.g. due to lack of communication capabilities. In this case, the LTL_f method can be used to guide the individual behavior of each agent separately. Therefore, we treat each agent as a standalone system and provide the LTL_f guidance as we did in single-agent settings.

Example 5.1. (*continued*) For the rendez-vous problem in Figure 9, the LTL_f formulas will be constructed from the set of propositional symbols $\mathcal{P} = \{at\_x, at\_y, at\_z\}$. And we consider the following LTL_f formulas for the first and second agent respectively: $\mathsf{F}(at\_x \wedge \mathsf{XF}at\_y))$ and $\mathsf{F}(at\_x \wedge \mathsf{XF}at\_z))$. It is clear the agents in this case are not guided to coordinate with each other to meet at position X, but instead each agent is guided to reach position X and then reach their final positions, independently.

5.2 Centralized Guidance

In the centralized guidance approach, we consider the case where the settings allow that one user guides the whole MAS as one unit. In this case, the user provides the LTL_f formula to a central entity which in turn provides each individual agent with the appropriate shaping rewards. Therefore, we use our extension of LTL_f to MGs (in Section 2.4) to provide the MAS with *multi-dimensional* LTL_f formulas (i.e., LTL_f formulas over $\mathcal{P}'$) that guide the coordination between the agents.

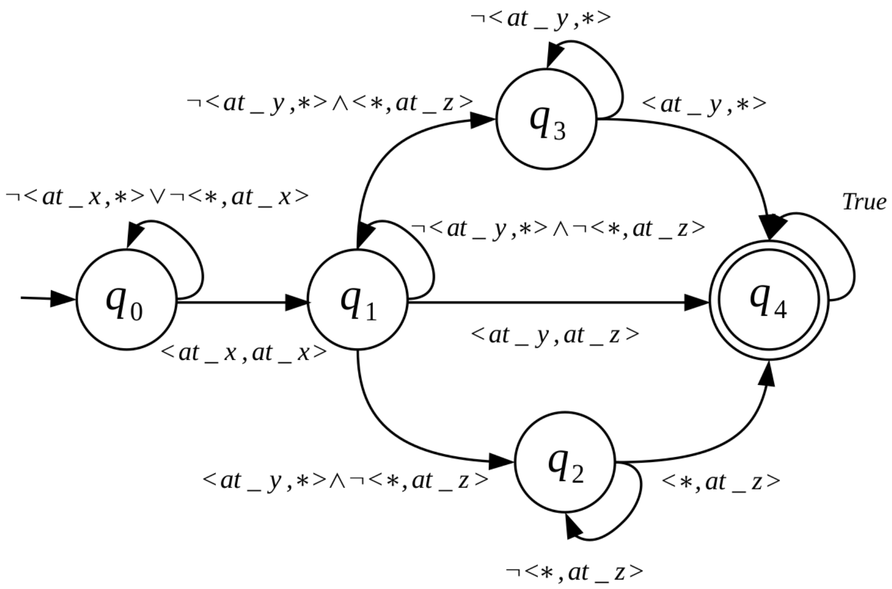

Fig. 10: The corresponding DFA for the multi-dimensional LTL$_f$ formula $\mathsf{F}(\langle at\_x, at\_x\rangle \wedge \mathsf{XF}\langle at\_y, \star\rangle \vee \langle\star, at\_z\rangle))$. The formula guides the agents to be at position X at the same time and then go to positions Y and Z respectively.

Example 5.1. (*continued*) For the rendez-vous problem in Figure 9, we define the set of propositional symbols $\mathcal{P} = \{at\_x, at\_y, at\_z\}$ and the multi-dimensional LTL$_f$ formula will be then constructed from $\mathcal{P}'$ (as defined in Section 2.4). We can now consider the following multi-dimensional LTL$_f$ formula: $\mathsf{F}(\langle at\_x, at\_x\rangle \wedge \mathsf{XF}\langle at\_y, \star\rangle \vee \langle\star, at\_z\rangle))$ (The corresponding DFA shown in Figure 10). This LTL$_f$ formula guides the agent to coordinate and be at position X at the same time and then go to their final positions.

In order to use multi-dimensional LTL$_f$ formulas to generate meaningful heuristics for each agent in the MAS centralized guidance setup, we need to redefine the following from our methodology in Section 3.

5.2.1 Estimate of Primitive Actions Needed for a DFA Transition in MAS

Let $\boldsymbol{S} = S^N$ be the state space of an MG and N be the number of agents involved in this MG, then we can define a general function $\vec{h}\colon \boldsymbol{S} \times \vec{\Phi}((\mathcal{P}^+)') \to \mathbb{R}^N$ (where $\vec{\Phi}((\mathcal{P}^+)')$ is the set of all LTL$_{\vec{f}}$ formulas over $(\mathcal{P}^+)'$), providing an estimate of the number of steps required to satisfy a given LTL$_{\vec{f}}$ formula ϕ from an MG state $\boldsymbol{s} \in \boldsymbol{S}$. To do so, we estimate how close are the RL agents from making a transition through a useful edge in the MAS by using a similar domain knowledge function to the one used in Section 3.6, namely $h_\alpha\colon S \to \mathbb{R}$

for each $\alpha \in \mathcal{P}^+$, which provides an estimate of the number of steps required for the agent to satisfy α from state $s \in S$. Given that α is in DNF, we define $\vec{h}$ recursively as follows:

$$\begin{aligned}
&\vec{h}[\![i]\!](\boldsymbol{s},\boldsymbol{\alpha}) = \begin{cases} h_{\boldsymbol{\alpha}[\![i]\!]}(\boldsymbol{s}[\![i]\!]) & \text{if } \boldsymbol{\alpha}[\![i]\!] \neq \star \text{, and} \\ 0 & \text{otherwise.} \end{cases} && \text{for } \boldsymbol{\alpha} \in \mathcal{P}', \\
&\vec{h}(\boldsymbol{s},\neg\boldsymbol{\alpha}) = \begin{cases} \min_{j\in[N]_0} h_{\neg\boldsymbol{\alpha}[\![j]\!]}(\boldsymbol{s}[\![j]\!]) & \text{if } \vec{h}(\boldsymbol{s},\boldsymbol{\alpha}) = \boldsymbol{0} \text{, and} \\ 0 & \text{otherwise} \end{cases} && i \in [N]_0 \\
&\vec{h}(\boldsymbol{s},\phi_1 \wedge \phi_2) = \vec{h}\,(\boldsymbol{s},\phi_1) + \vec{h}\,(\boldsymbol{s},\phi_2) \\
&\vec{h}(\boldsymbol{s},\bigvee_{k=1}^{M}\phi_k) = \vec{h}(\boldsymbol{s},\phi') \text{ s.t. } \phi' = \underset{k\in[M]_0}{\operatorname{argmin}} \max_{j\in[N]_0} \vec{h_j}(\boldsymbol{s},\phi_k) && \text{for } \phi_1,\phi_2,\phi_k \in \vec{\Phi}((\mathcal{P}^+)').
\end{aligned} \tag{5.1}$$

we highlight the following facts to clarify the intuition behind the equations: 1. $\boldsymbol{\alpha}$ holds iff for all $i \in [N]_0$, $\boldsymbol{\alpha}[\![i]\!]$ holds. 2. $\neg\boldsymbol{\alpha}$ holds iff $\exists i \in [N]_0$ where $\neg\boldsymbol{\alpha}[\![i]\!]$ holds. With the same intuition as in the single-agent settings, the heuristics of satisfying a disjunction between a set of formulas is equal to the heuristics of satisfying the formula which is "easier" to satisfy e.g. lower estimated distance to satisfaction. Given the first fact and the fact that the agents are independent, we choose the heuristics to satisfy a disjunction to be: the heuristics of the formula (a vector of size N) with the lower maximum element-wise heuristic value. And given the second fact, we choose the heuristics to satisfy a negation of $\boldsymbol{\alpha}$ to be: the minimum element-wise heuristic value of $\boldsymbol{\alpha}$.

Example 5.1. (*continued*) For the rendez-vous problem in Figure 9, we define a domain knowledge function for each propositional symbol $\alpha \in \{at\_x, at\_y, at\_z\}$ as follows

$$\begin{aligned}
h_\alpha(s) &= \vec{d_\alpha}(s) \\
h_{\neg\alpha}(s) &= \begin{cases} 0 & \text{if } h_\alpha(s) \neq 0 \\ 1 & \textit{otherwise} \end{cases}
\end{aligned} \tag{5.2}$$

for each $s \in S$ and where $\vec{d_\alpha}(s)$ are heuristics functions that return estimate distance from state s to the state where α holds.

5.2.2 Building the Potential Function in MAS

We then define the potential of $\boldsymbol{s}_t$ at time t for agent i where $i \in [N]_0$ as follows:

$$\Phi^i_t(\boldsymbol{s}_t) = -\omega \times \left[Distance(q_t) + \frac{1}{n} \cdot \vec{h}[\![i]\!](\boldsymbol{s}_t, \phi_t) \right] \tag{5.3}$$

where ω is a scaling factor, q_t is the current state of the DFA and $1/n$ is a normalization factor where $n = \max_{\boldsymbol{s}\in\boldsymbol{S}} \vec{h}[\![i]\!](\boldsymbol{s}, \phi_t)$. The potential formula has the same intuition as the one used in single-agent settings (Section 3.7). The augmented reward functions will then be built in a similar way as in single agent (Section 3.8).

6 Experiments in Multi-agent Settings

In this section, we evaluate the performance of LTL$_f$-based centralized guidance in the rendez-vous domain and buttons domain [19], two benchmark problems for coordination in MAS. Then, we move back to the flag collection domain where we compare to the plan-based method and demonstrate the flexibility of communicating guidance with partial knowledge (uncertainty) using LTL$_f$ in MAS.

In all the experiments, we use independent learners (ILs), namely, we use independent SARSA agents. Also, we only consider softmax exploration strategy (motivated by the findings of Claus et al. [3] on the convergence of exploitative-exploration strategies in the case of ILs).

6.1 Evaluating the Performance in the Rendez-vous Domain

In this section, we evaluate the performance of our method in the rendez-vous problem (in Figure 9). Since in the rendez-vous domain the agents need to meet at a rendez-vous position, we choose to use the centralized guidance approach since it is most relevant when the agents need to explicitly coordinate to perform the task successfully.

The results are depicted in Figures 11. The LTL$_f$-shaped agents always converge to the optimal policy while the vanilla agents get stuck in the suboptimal solution of going directly to the final positions (without the rendez-vous happening) in 2 out of 10 runs. This suboptimal solution has -1.3 payoff (compared to 98.7 of the optimal solution). This demonstrates that the LTL$_f$ method is capable of guiding the agents to coordinate even if the agents do not explicitly take each other into account (ILs). Also, the LTL$_f$-based shaped MAS has a remarkable initial performance compared to the vanilla MAS.

6.2 Evaluating the Performance in the Buttons domain

In this section, we again evaluate the LTL$_f$-based centralized guidance in a variant of the buttons domain [19]. Namely, we consider the buttons problem in Figure 12. The goal is for the agents to coordinate to enable the agent A_1 to reach the goal state G. The agents A_1, A_2 and A_3 are penalized with -5 reward when they step on the red, yellow and green colored regions, respectively, if the buttons with the corresponding colors are not pushed. Agent A_1 and A_3 can press the yellow and green buttons respectively while the red button requires both agents A_2 and A_3 to push it simultaneously. The agents share the same reward and get a positive reward of 100 reward when the agent A_1 reaches

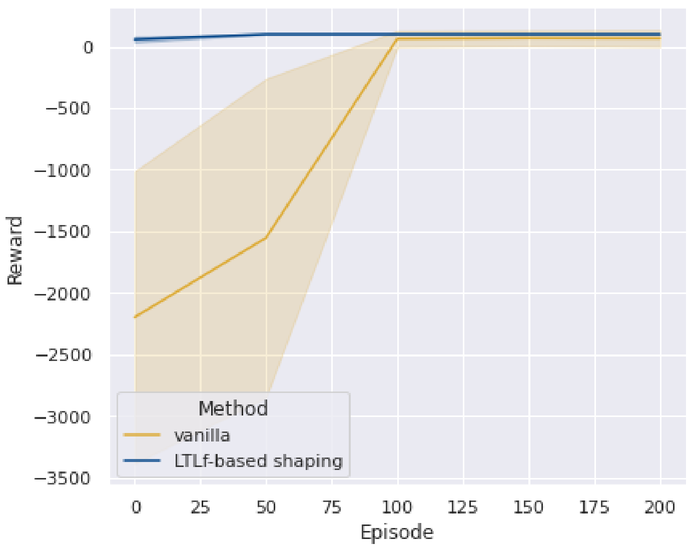

Fig. 11: Results in the rendez-vous problem 9 show smooth convergence to the optimal solution in the case of LTL$_f$-shaped agents while the vanilla agents take much more time to reach a near optimal solution. As explained in the text, vanilla agents get stuck in sub-optimal policies in 2 out of 10 runs. The first 200 episodes are shown (no remarkable changes afterward) and the curve smoothed over 50 consecutive episodes, for better visibility.

the goal position and is penalized at each time step with -0.1 reward. Each agent's state is defined by its position and whether the buttons (relevant to the agent) are pushed e.g. the yellow button is relevant for agents A_1 and A_2 only. The agents can move in the 4 conventional directions and can also stay still. The agents remain in place if there is an obstacle in the intended direction of the executed action.

We define $\mathcal{P} = \{rb, gb, yb, g\}$ corresponding to the 3 buttons and the goal position. To guide the agents to the optimal policy, we provide the following LTL$_f$ formula:

$$\mathsf{F}(\langle yb, \star, \star\rangle \wedge \mathsf{XF}(\langle \star, gb, \star\rangle \wedge \mathsf{XF}(\langle \star, rb, rb\rangle \wedge \mathsf{XF}\langle g, \star, \star\rangle))) \qquad (6.1)$$

with the corresponding DFA in Figure 13.

Results in Figure 14 shows that the LTL$_f$-shaped MAS converges to the optimal policy in which agent A_1 pushes the yellow button, agent A_3 pushes the green button, agents A_2 and A_3 push the red button, and then finally agent A_1 goes to the goal position. In 6 out of 10 runs, the vanilla MAS gets stuck in the sub-optimal policy in which the agent A_1 goes directly to the goal position stepping on the red region while the red button is not pushed by

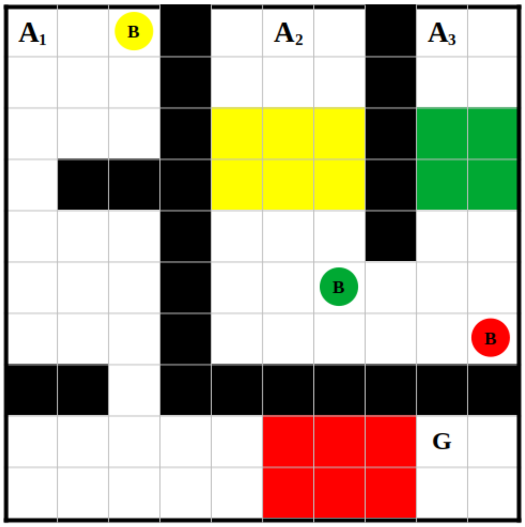

Fig. 12: An instance of the buttons domain [19]. The goal for the agents A_1, A_2, and A_3 is to coordinate to enable the agent A_1 to reach the goal state G. The optimal policy is: A_1 pushes the yellow button, agent A_3 pushes the green button, agents A_2 and A_3 push the red button, and then finally agent A_1 goes to the goal position.

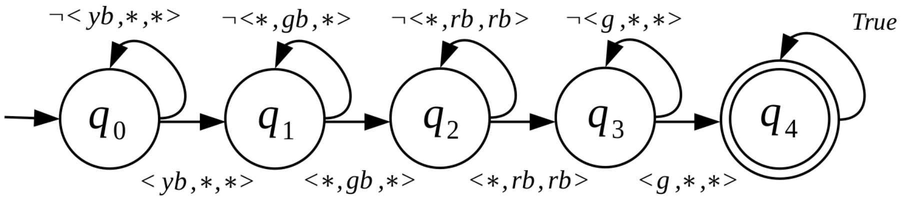

Fig. 13: The corresponding DFA for the multi-dimensional LTL$_f$ formula in Equation 6.1.

agents A_2 and A_3. This suboptimal solution has 93.4 payoff (compared to 98 payoff of the optimal policy).

6.3 Evaluating the LTL$_f$-based Method in the Flag Collection Domain

In this section, we evaluate our method in the classic flag collection problem (Figure 3) in the MAS. First, we compare our method to the plan-based RS method in MAS [8]. Then, we demonstrate the flexibility of LTL$_f$ in multi-agent settings using different LTL$_f$ formulas to express different advice with partial knowledge.

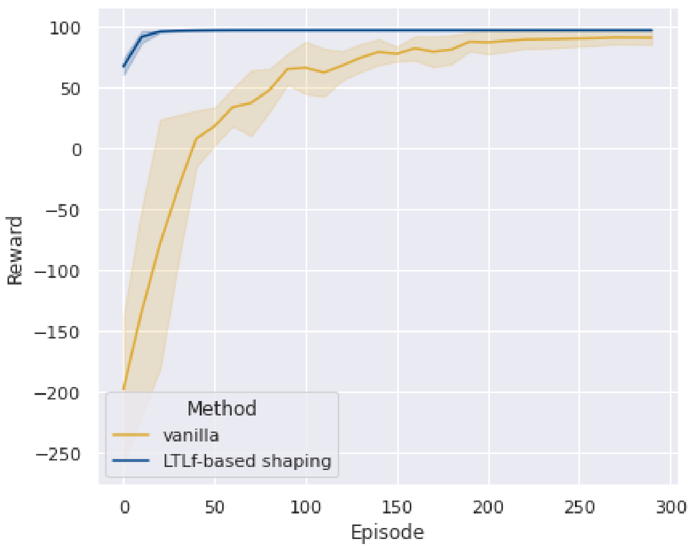

Fig. 14: Results in the buttons problem in Figure 12. The LTL$_f$ -shaped MAS always converges to the optimal policy while the vanilla MAS not only takes much longer to improve their behavior but also gets stuck in a sub-optimal policy in 6 out of 10 runs. The first 300 episodes are shown (no remarkable changes afterward) and the curve smoothed over 10 consecutive episodes, for better visibility.

In all experiments in this section, we run each experiment for 5×10^4 episodes with maximum of 10^5 steps per episode. In all the plots, we smooth the curves by taking the average of each 10 consecutive episodes, for better visibility. We calculate the temperature parameter T [24] as follows:

$$T = \frac{c \times \max_{a \in A}(\max_{b \in A} Q(s,b) - Q(s,a))}{\ln n_t(s)} \tag{6.2}$$

where $n_t(s)$ is the amount of times the agent visited state s in the current experiment, and c is the scaling parameter which we empirically found to provide good results when set to $c = 0.5$.

6.3.1 Comparing to the Plan-based Method

In this section, we compare the LTL$_f$ -based method to the plan-based method in MAS [8]. Since Devlin et al. [8] do not assume any type of information sharing between agents, we use the *decentralized guidance* of the LTL$_f$ method to compare to their approach.

The agents start at positions $(5, 4)$ and $(5, 14)$, respectively. Both plan-based and LTL$_f$ -based method guide the agents according to the optimal

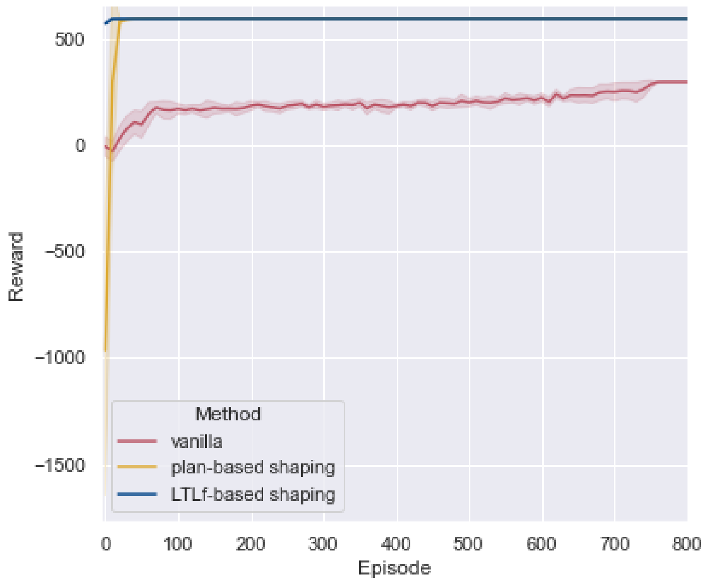

Fig. 15: Comparing the plan-based and the LTL$_f$-based methods in the flag collection problem (Figure 3) with a second agent starting at $(5, 14)$. The results remain stable after episode 800 (not shown for better visibility).

solution which has the first agent collecting flag A then flag D and the second agent collecting flag F, E, C, then B. For the plan-based method we provide the agents with plans 16a and 16b respectively. For the LTL$_f$-based method, we provide the agents with the formulas 6.5 and 6.6, respectively. As in the single-agent experiments 4.1, these formulas still reward the agents if they collect their assigned flags in a different order.

$$\mathsf{F}(have\_flagA \wedge \mathsf{F}(have\_flagD \wedge (at\_goal \vee \mathsf{XF}(at\_goal)))) \tag{6.3}$$

$$\begin{aligned}\mathsf{F}(have\_flagF \wedge \mathsf{F}(have\_flagE \wedge \mathsf{F}(have\_flagC \wedge \ldots \\ \wedge (at\_goal \vee \mathsf{XF}(at\_goal))))))\end{aligned} \tag{6.4}$$

The results are depicted in Figure 15. The vanilla agents get stuck in the sub-optimal solution of collecting only 3 flags: C, E, and F are collected by agent 2, while the shaped agents successfully converge to the optimal solution with the LTL$_f$-based method achieving a better initial performance. Both shaping methods successfully guide the agents to converge to the optimal

```
1 MOVE( hallA ,  roomA)
2 TAKE( flagA )
3 MOVE( hallA )
4 MOVE(roomD)
5 TAKE(FlagD)
```

(a) Joint-plan part for agent 1

```
1 TAKE( flagF )
2 TAKE( flagE )
3 MOVE(roomE,  roomC)
4 TAKE( flagC )
5 MOVE(roomC,  hallB )
6 MOVE( hallB ,  roomB)
7 TAKE( flagB )
8 MOVE(roomB,  hallB )
9 MOVE( hallB ,  hallA )
10 MOVE( hallA ,  roomD)
```

(b) Joint-plan part for agent 2

Fig. 16: Joint plan for plan-based RS method abstractly representing the optimal solution for the flag collection domain in MAS. The plan actions have obvious effects as indicated by their names. For more details, please refer to the original work [8]

solution with each agent collecting their assigned flags and then going to the goal.

6.4 Demonstrating the Flexibility of the LTL$_f$ Method in MAS

In this section, we demonstrate different aspects of flexibility centralized LTL$_f$-based RS can provide in the MAS. This is something the STRIPS plan-based method can not express.

6.4.1 The user knows which agent should collect which set of flags but not the ordering

Figure 17 shows the results with decentralized guidance where the agents were given the formulas 6.5 and 6.6. The user does not know the correct order for the flags to be collected in but does know which agent should collect which set of flags. So the first agent is advised to collect flags A and D and then to go to the goal, and the second agent is advised to do the same for flags B, C, E and F. As shown in figure 17, the agents swiftly converge on the optimal solution, learning to collect the flags in the optimal order. For the first agent that is A and D, and for the second agent that is F, E, C, and B. The agents without shaping converge on a solution that has agent 1 collecting no flags and agent 2 collecting flags F, E, and C in that order.

$$\mathsf{F}((\mathit{have\_flagA} \wedge \mathit{have\_flagD}) \wedge \mathsf{XF}(\mathit{at\_goal})) \tag{6.5}$$

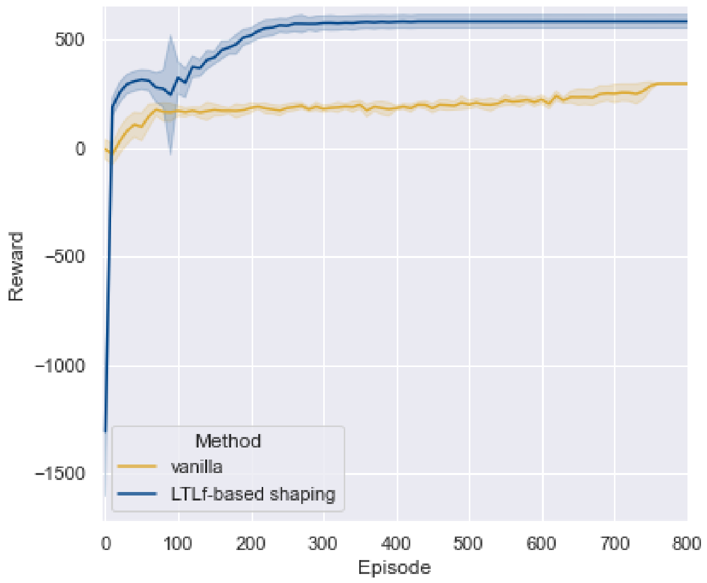

Fig. 17: LTL$_f$ -based decentralized RS with LTL$_f$ formulas 6.5 and 6.6 on the MAS flag-collection domain (Figure 3 with added agent starting at $(5, 14)$). The advice with partial knowledge causes the agents to need some more time after learning to collect the easily accessible flags, but it still allows the agents to converge upon the optimal solution. There are no remarkable changes in the curves after episode 800 (not shown for better visibility).

$$\mathsf{F}((\mathit{have\_flagF} \wedge \mathit{have\_flagE} \wedge \mathit{have\_flagC} \wedge \mathit{have\_flagB}) \wedge \mathsf{XF}(\mathit{at\_goal})) \tag{6.6}$$

6.4.2 The user does not know which agent should collect which set of flags

The results demonstrating the flexibility of LTL$_f$ are shown in Figure 18. The user providing the decentralized guidance knows the order in which each set of flags is best gathered. But the user does not know which agent should gather which set of flags. The LTL$_f$ formula is the same for both agents, formula 6.7 advises to take the flags in a specified order and then go to the goal, reflecting the uncertainty of the user. More precisely the agents are advised to either take flags A and D in order and then go to the goal, or take flags F, E, C, and B in that order and then go the goal. As can be seen in Figure 18, the agents converge rapidly to the optimal solution even with the uncertainty present in the advice. The agents without shaping converge on a solution that has agent 1 collecting no flags and agent 2 collecting flags F, E, and C in that order.

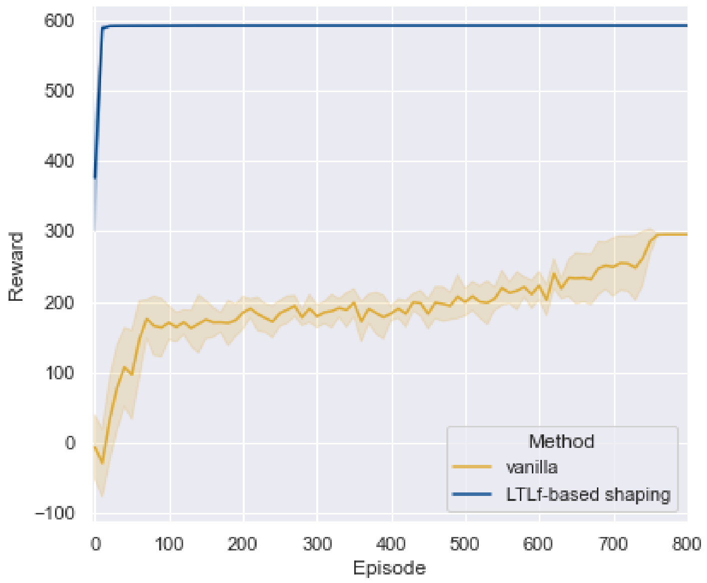

Fig. 18: LTL$_f$-based decentralized RS with LTL$_f$ formula 6.7 given to both agents, on the MAS flag-collection domain (Figure 3 with added agent starting at $(5, 14)$). The agents swiftly learn which set of flags is optimal for them and converge upon the optimal solution. There are no remarkable changes in the curves after episode 800 (not shown for better visibility).

$$\begin{aligned}&\mathsf{F}(\mathsf{F}(have\_flagA \wedge \mathsf{F}(have\_flagD \wedge (at\_goal \vee \mathsf{XF}(at\_goal))))\vee \\ &\qquad \mathsf{F}(have\_flagF \wedge \mathsf{F}(have\_flagE \wedge \ldots \wedge (at\_goal \vee \mathsf{XF}(at\_goal))))))\end{aligned} \tag{6.7}$$

6.4.3 The user knows which agent should collect which flag except for one flag

We consider the scenario where the user knows which agent should collect which set of flags in which order except that he/she does not know which agent should gather flag B. We consider the LTL$_f$ formula in Equation 6.12 which express the user's partial knowledge. We split the formula into sub-formulas for readability. The meaning of the formulas 6.8, 6.9, 6.10, 6.11 and 6.12 are relatively straightforward, Formula 6.8 guides the first agent to take flags A, B, then D.

$$\varphi_1 = \langle have\_flagA, \star\rangle \wedge \mathsf{XF}(\langle have\_flagB, \star\rangle \wedge \mathsf{XF}(\langle have\_flagD, \star\rangle)) \tag{6.8}$$

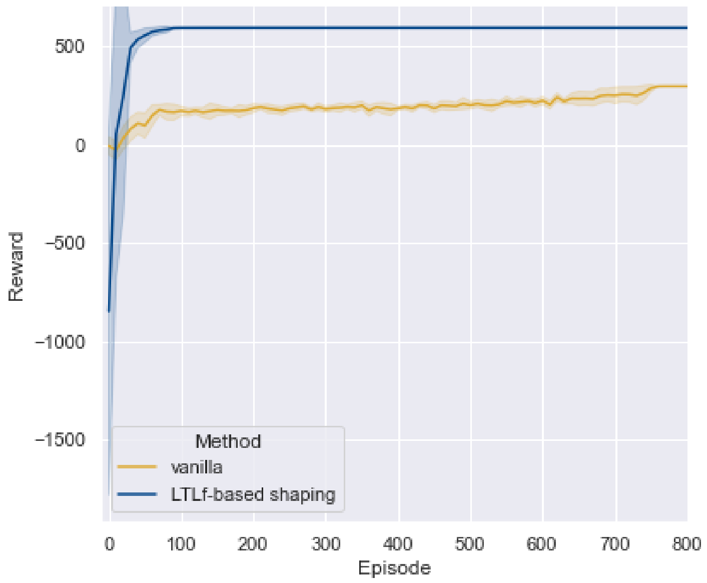

Fig. 19: LTL$_f$ -based centralized RS with LTL$_f$ formula 6.12 given to both agents, on the MAS flag-collection domain (Figure 3 with added agent starting at $(5, 14)$). There are no remarkable changes in the curves after episode 800 (not shown for better visibility).

Formula 6.9 guides the second agent to take flags F, E, then C.

$$\varphi_2 = \langle \star, have\_flagF \rangle \land \mathsf{XF}(\langle \star, have\_flagE \rangle \land \mathsf{XF}(\langle \star, have\_flagC \rangle)) \tag{6.9}$$

Formula 6.10 guides the first agent to take flags A then D.

$$\varphi_3 = \langle have\_flagA, \star \rangle \land \mathsf{XF}(\langle have\_flagD, \star \rangle) \tag{6.10}$$

Formula 6.11 guides the second agent to take flags F, E, C, then B.

$$\begin{split}\varphi_4 = \langle \star, have\_flagF \rangle \land \mathsf{XF}(\langle \star, have\_flagE \rangle \land \\ \mathsf{XF}(\langle \star, have\_flagC \rangle \land \mathsf{XF}(\langle \star, have\_flagB \rangle)))\end{split} \tag{6.11}$$

Finally formula 6.12 combines formulas 6.8, 6.9, 6.10 and 6.11 so that the first and second agents respectively either take flags (A, B, D) and (F, E, C) or flags (A, D) and(F, E, C, B).

$$\mathsf{F}(\varphi_1 \land \varphi_2) \lor \mathsf{F}(\varphi_3 \land \varphi_4) \tag{6.12}$$

The results demonstrating the flexibility of LTL$_f$ are shown in Figure 19. The agents using centralized guidance swiftly learn which set of flags is optimal for them and converge upon the optimal solution. Which has agent 1 collecting flags A and D in order and agent 2 collecting flags F, E, C, B in that order. The agents without shaping converge on a solution that has agent 1 collecting no flags and agent 2 collecting flags F, E, and C in that order.

7 Conclusion

In this paper, we introduced a framework that allows the flexible incorporation of domain knowledge in RL agents. Namely, we generate PBRS functions, which guaranteed to preserve the optimal policy, from user-provided LTL$_f$ formulas. We demonstrated in both single and multi-agent settings that LTL$_f$ provides the flexibility to communicate guidance in the case of partial knowledge (uncertain advice). Namely, we demonstrated empirically in the flag collection domain that our method performs at least as well as the STRIPS plan-based method while providing essential advantages in terms of flexibility and ease of use. Our method could guide the learning agents to the optimal policy despite the user's partial knowledge. We also introduced two approaches to extend our method to MAS: centralized and decentralized guidance. We demonstrated that the LTL$_f$ -based centralized guidance could guide the agents to coordinate, and thus learn the optimal policy, in two benchmark domains: the rendez-vous domain and the buttons domain. Lastly, we believe the flexibility we demonstrated in this work is an important added value for RL to be used in real-life applications, such as robotic systems e.g. a machine tending system.

Acknowledgments. This research was supported by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

References

[1] Bellman R (1957) A markovian decision process. Journal of mathematics and mechanics pp 679–684. URL https://doi.org/10.1512/iumj.1957.6.56038

[2] Camacho A, Icarte RT, Klassen TQ, et al (2019) Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In: IJCAI, pp 6065–6073, URL https://doi.org/10.24963/ijcai.2019/840

[3] Claus C, Boutilier C (1998) The dynamics of reinforcement learning in cooperative multiagent systems. AAAI/IAAI 1998(746-752):2

[4] De Giacomo G, Vardi MY (2013) Linear temporal logic and linear dynamic logic on finite traces. In: Twenty-Third International Joint Conference on Artificial Intelligence

[5] De Hauwere YM, Devlin S, Kudenko D, et al (2016) Context-sensitive reward shaping for sparse interaction multi-agent systems. The Knowledge Engineering Review 31(1):59–76. URL https://doi.org/10.1017/S0269888915000193

[6] De Winter J, De Beir A, El Makrini I, et al (2019) Accelerating interactive reinforcement learning by human advice for an assembly task by a cobot. Robotics 8(4):104. URL https://doi.org/10.3390/robotics8040104

[7] Devlin S, Kudenko D (2011) Theoretical considerations of potential-based reward shaping for multi-agent systems. In: The 10th International Conference on Autonomous Agents and Multiagent Systems, ACM, pp 225–232

[8] Devlin S, Kudenko D (2016) Plan-based reward shaping for multi-agent reinforcement learning. The Knowledge Engineering Review 31(1):44–58

[9] Devlin SM, Kudenko D (2012) Dynamic potential-based reward shaping. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, IFAAMAS, pp 433–440

[10] Giacomo GD, Vardi MY (2015) Synthesis for LTL and LDL on finite traces. In: Yang Q, Wooldridge MJ (eds) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. AAAI Press, pp 1558–1564

[11] Grzes M (2017) Reward shaping in episodic reinforcement learning. International Foundation for Autonomous Agents and Multiagent Systems, AAMAS '17, p 565–573

[12] Grzes M, Kudenko D (2008) Plan-based reward shaping for reinforcement learning. In: 2008 4th International IEEE Conference Intelligent Systems, IEEE, pp 10–22, URL https://doi.org/10.1109/IS.2008.4670492

[13] Hopcroft JE, Motwani R, Ullman JD (2001) Introduction to automata theory, languages, and computation. Acm Sigact News 32(1):60–65. URL https://doi.org/10.1145/568438.568455

[14] Icarte RT, Klassen T, Valenzano R, et al (2018) Using reward machines for high-level task specification and decomposition in reinforcement learning. In: International Conference on Machine Learning, PMLR, pp 2107–2116

[15] Icarte RT, Klassen TQ, Valenzano RA, et al (2018) Advice-based exploration in model-based reinforcement learning. In: Canadian Conference on Artificial Intelligence, Springer, pp 72–83

[16] Judah K, Roy S, Fern A, et al (2010) Reinforcement learning via practice and critique advice. In: Twenty-Fourth AAAI Conference on Artificial Intelligence

[17] Knox WB, Stone P (2009) Interactively shaping agents via human reinforcement: The tamer framework. In: Proceedings of the fifth international conference on Knowledge capture, pp 9–16, URL https://doi.org/10.1145/1597735.1597738

[18] Mnih V, Kavukcuoglu K, Silver D, et al (2015) Human-level control through deep reinforcement learning. nature 518(7540):529–533. URL https://doi.org/10.1038/nature14236

[19] Neary C, Xu Z, Wu B, et al (2021) Reward machines for cooperative multi-agent reinforcement learning. AAMAS '21, p 934–942, URL https://doi.org/10.48448/cawm-bw32

[20] Ng AY, Harada D, Russell S (1999) Policy invariance under reward transformations: Theory and application to reward shaping. In: ICML, pp 278–287

[21] Peng B, MacGlashan J, Loftin R, et al (2016) A need for speed: Adapting agent action speed to improve task learning from non-expert humans. In: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems

[22] Pnueli A (1977) The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), IEEE, pp 46–57, URL https://doi.org/10.1109/SFCS.1977.32

[23] Randløv J, Alstrøm P (1998) Learning to drive a bicycle using reinforcement learning and shaping. In: ICML, pp 463–471

[24] Singh S, Jaakkola T, Littman ML, et al (2000) Convergence results for single-step on-policy reinforcement-learning algorithms. Machine learning 38(3):287–308. URL https://doi.org/10.1023/A:1007678930559

[25] Suay HB, Chernova S (2011) Effect of human guidance and state space size on interactive reinforcement learning. In: 2011 Ro-Man, IEEE, pp 1–6, URL doi.org/10.1109/ROMAN.2011.6005223

[26] Sutton RS, Barto AG, et al (1998) Introduction to reinforcement learning, vol 135. MIT press Cambridge

[27] Thomaz AL, Breazeal C (2007) Robot learning via socially guided exploration. Development and Learning pp 82–87. URL doi.org/10.1109/DEVLRN.2007.4354078