Translating Bolts: LTL_f and STRIPS Based Restraining Bolts on the Taxi-v3 Environment

Claudia Melis Tonti [1888489], Lorenzo Papa [1699806],
Nicolas Zaccaria [1904366], Roberto Gallotta [1890251]

La Sapienza University of Rome

Abstract. We present here our work investigating the effects of Reward Shaping on a Hierarchical Reinforcement Learning (HRL) environment while specifying increasingly complex temporal goals defined as LTL_f formulas. Thanks to the flexibility of the LTL_f language, it is trivial to define such goals and, thanks to existing Python libraries, convert them to DFA automatons that work best with RL tasks.
We also investigate the possibility to discard the LTL_f goals and instead redefine the environment using STRIPS, which could later be extended to accept a multitude of temporal goals implicitly.

Keywords: Reinforcement Learning, Temporal Logic, LTL_f, DFA, STRIPS

Introduction

In this paper we report our findings when developing the experiment for the Reasoning Agents course. We first introduce the necessary background knowledge and current literature on the topics. We then introduce what our experiment consisted in and finally provide the results. Future work may consist in applying different learning methods, hyperparameter tuning or extensions on the STRIPS-based category of problems.

1 Reinforcement Learning

Reinforcement Learning (RL) is a branch of Machine Learning (ML), one of the most prominent and established field of study in the Artificial Intelligence (AI) sector. Despite an unpromising birth in the 1950s, thanks to the increment of computational power, achieved with the most recent Graphical Processor Units (GPUs), to the progress of the novel algorithms and research and, also, to the growth of the available amount of data (Big Data), ML has become a powerful tool in a wide set of tasks.

AI algorithms that belong to this sub-class are designed to improve their capabilities of performing some task with experience: *learning*. In general, RL aims at learning an optimal behavior function, called *policy*, $\pi : S \to A$, given

a set of states, actions and rewards, $D = \{(\langle s_0, a_1, r_1, s_1, ..., a_n, r_n, s_n \rangle_i)_{i=1}^n\}$, in order to maximize the final cumulative reward provided by a reward function.

More in detail, a RL agent interacts with an *environment*, represented as a defined *state* in a time-discrete system, by performing some *actions* from a set of possible ones. The dynamic evolution of both the environment and the agent is described as a *transition* (s_i, a_i, s_{i+1}): given the current state s_i and reward r_i, at every time-step i, the agent picks the action a_i to perform causing the evolution of the state, from s_i to s_{i+1}, and earning a reward r_{i+1}. Then, by maximizing the cumulative achieved reward, the agent learns an optimal policy function that defines which is the best action to perform considering the current state, $\pi(a, s) = Pr(a_i = a | s_i = s)$.

As evidenced above, to ensure the feasibility of this mechanism, a reinforcement learning problem may be modeled as a Markov Decision Process (MDP), an evolution of the Markov chain, consisting in a discrete-time stochastic control process based on the Markov property:

- The evolution of a dynamic system does not depend on the previous states, actions and observations once the current state is known.
- All the information needed to predict the future are contained in the current state.
- Given the current state, the future ones are conditionally independent of past states and observations.
- Past, present and future observations are statistically independent given the knowledge of the current state.

Then, a MDP model can be defined as:

$$MDP = \langle S, A, \delta, r \rangle$$

with:

- S is a finite set of states, they may be fully or partially observable
- A is a finite set of actions
- δ is a transition function, it may be deterministic, non-deterministic or stochastic
- $r : S \times A \times S \to \mathbb{R}$ is a reward function

The model's optimality is defined wrt the maximization of the expected cumulative discounted reward:

$$V^\pi(s_1) = E[\overline{r}_1 + \gamma \overline{r}_2 + \gamma^2 \overline{r}_3 + ...]$$

where $\overline{r}_i = r(s_i, a_i, s_{i+1}), a_i = \pi(s_i), \gamma \in [0, 1]$ is the discount factor and the optimal policy π^* is defined as:

$$\pi^* \equiv argmax_\pi V^\pi(s), \forall s \in S$$

Finally, these formulas underline one characterizing feature of the RL problems consisting is the notion of *regret* that pushes the agent to perform actions in a long-term reasoning way, even involving negative short-term rewards.

1.1 Taxi-v3

Taxi-v3 is one among all the proposed tasks in the set of environments provided by *OpenAI* [1], a prominent AI research and deployment company that offers a powerful toolkit, named *Gym* [2], for developing and testing reinforcement learning algorithms. This library consists in a collection of different challenges, called *environments*, which covers all the main known problems and weaknesses that afflicted, or still afflict, the RL field of study.

Introduced for the first time in 2000 by [3], Taxi-v3 has been used as a practical example that well shows some typical issues of Hierarchical Reinforcement Learning (HRL) [4]. Based on the decomposition of a goal into sub-goals, building a hierarchy where the parent tasks invokes the children ones as primitive actions, HRL is a reinforcement learning technique mainly designed to reduce the computational complexity.

The task, inspired by the taxi driver job, is relatively simple and takes place in a 5×5 grid with 4 marked tiles called *R, G, Y, B*. At each round, called *episode*, two of these positions are randomly chosen as passenger and destination locations, while another random one is picked along the whole grid as taxi's starting location. The goal of the agent is to bring the passenger to the destination, trying to maximize the final total reward. In case of coincidence of passenger and destination tiles, the agent still has to pick up the passenger and drop him off in place. To accomplish this objective, the driver can freely move inside the grid in any orthogonal direction with the exception of trespassing the outer edges or some fixed walls between tiles.

The environment is represented at each time step by its state: considering all the possibilities made by 25 taxi positions, 5 passenger positions, *on taxi* included, and 4 destinations, there are in total 500 states that are indexed with an encoding of discrete values in the interval $[0, 499]$.

The state is updated only by the agent's actions. There are 6 possible actions the agent can perform at any step of the episode, which are:

- *Move South*,
- *Move North*,
- *Move East*,
- *Move West*,
- *Pick up* the passenger and
- *Drop off* the passenger.

In order to train the agent, a -1 per-step reward is applied, except for the *Pick up* and *Drop off* actions which, if done illegally, costs -10. The episode ends if the agent performs the maximum number of fixed steps or if it manages to drop off the passenger at the destination obtaining a $+20$ reward.

1.2 Temporal Goals

We now explore Goal-Directed Exploration with a Reinforcement Learning Algorithm. This means that an agent has to learn a policy to reach a goal state in an

initially unknown state space. Once the reinforcement learning problem reaches the goal state the agent has to find some path to a goal state in an initially unknown (or partially known) state space with no need to find the shortest path. As mentioned in [5], the goal-directed exploration with reinforcement learning methods can be too complex or impossible to make it converge. The algorithm may degenerate into a random walk until it reaches the goal. The idea is using a set of temporal goals in order to reach the final goal. It is possible to consider a set of temporal goals expressed in LTL_f notation.

Reward Shaping Reward shaping is a method for engineering a reward function in order to provide more frequent feedback on appropriate behaviors. It is used in multiple field of applications as described in [6]. The most well-known work in the reward shaping domain is the Potential-Based Reward Shaping (PBRS) method [7], which is the first to show that policy invariance can be guaranteed if the shaping reward function is in the form of the difference of potential values.

Reward shaping is a well-established method to incorporate domain knowledge in the reinforcement learning (RL) task. The reward shaping in RL consist of supplying additional rewards to a learning agent to guide its learning process more efficiently, helping it in achieving the optimal policy faster. The reward shaping function can be integrated in the MDP, $M = (S, A, T, \gamma, R)$, modifying its reward function, R, as $R' = R + F$ where $F : S \times A \times S \Rightarrow \mathbb{R}$ is a bounded real-valued function. Then if the original MDP would have received a reward $R(s, a, s')$ for transitioning from s to s' on an action a then in the new MDP it would receive the following reward:

$$R'(s, a, s') = R(s, a, s') + F(s, a, s')$$

This process will encourage the learner to take the action a in some set of states S_0 leading faster to the optimal solution.

1.3 Restraining Bolts

A restraining bolt is a device that restricts an agent's actions when connected to its system. The restraining bolt has the aim of limiting actions to a set of desired behaviors. The restraining bolt gives an additional representation of the world with respect to the one that the agent already has. The agent has to conform as much as possible to the restraining specifications. The link between the two representation is given by studying this problem in the Reinforcement Learning scope. The restraining bolt is a high-level logical specification of desirable features. These features are specified as formulae in LTL_f formalism. To make the agent world representation conform to the specifications of the restraining bolt an additional reward is given by the restraining bolt.

In a RL task with restraining bolt:

- The learning agent is modeled by the MDP: $M_{ag} = \langle S, A, T_{r_{ag}}, R_{ag} \rangle$

– The restraining bolt is modeled as $RB = \langle L,\ \{(\phi_i,\ r_i)\}_{i=1}^{m} \rangle$, where L is the set of possible fluents and the set is the restraining specifications represented in LTL_f formulas.

The general problem definition given by the reference [8] is an RL problem with LTL_f /LDL_f restraining specifications is a pair $M_{ag}^{rb} = \langle M_{ag},\ RB \rangle$, where: $M_{ag} = \langle S,\ A,\ T_{r_{ag}},\ R_{ag} \rangle$ is a learning agent with transitions $T_{r_{ag}}$ and rewards R_{ag} hidden, and $RB = \langle L,\ (\phi_i,\ r_i)_{i=1}^{m} \rangle$ is a restraining bolt formed by a set of LTL_f /LDL_f formulas ϕ_i over L with associated rewards r_i. A solution to the problem is a policy $\bar{\rho} :\ (Q_1 \times \cdots \times Q_m \times S)^* \to A$ that maximizes the expected cumulative reward, that could be non-Markovian.

A solution to this problem may be:

1. For every Φ_i , compute the equivalent DFA A_{Φ_i}
2. Do RL on the MDP $M_{ag}^{q} = \langle Q_1 \times \cdots \times Q_m \times S,\ A,\ Tr'_{ag},\ R'_{ag} \rangle$ where:
   – The transition distribution Tr'_{ag} is unknown
   – The reward R'_{ag}, unknown to the agent, is defined as:

$$R'_{ag}(q_1, \ldots,\ q_m,\ s,\ a,\ q'_1, \ldots,\ q'_m,\ s') = \sum_{i : q'_i \in F_i} r_i + R_{ag}(s,\ a,\ s')$$

   – The states q_i of the DFAs A_{Φ_i} are progressed correctly by the environment.

1.4 Automata as Reward Shaping

A Reward Machine indicates what reward function should be used to obtain a reward signal given the states label that the agent met up until now. The formal definition of a reward machine is given in the reference [9]: a reward machine (RM) for the setting $\langle S,\ A,\ P,\ L \rangle$ is a Mealy machine $\langle Q,\ q_0,\ \Sigma,\ R,\ \delta,\ \rho \rangle$ where the input alphabet is $\Sigma = 2^P$, and R is a finite set where each $R \in \mathbb{R}$ is a reward function from $S \times A \times S$ to R.

A Mealy machine is a tuple which takes a finite set of states, an initial state, a finite input alphabet, the resulting output alphabet, the transition function from the set of states and input alphabet to the resulting state and finally, the output function. Moreover, the A of the RM is the set of actions, P is a set of propositional symbols and L is a labeling function $L : S \times A \times S \to 2^P$.

In some cases, in order to have some feedback through the RL algorithm iterations, the potential function is extracted by a Reward Machine. The idea is to approximate the cumulative discounted reward in any RM state by treating the RM itself as an MDP. The value iteration would be the maximum value of the optimal policy. In the algorithm the resulting potential function will be extracted by the value iteration. Applying the value iteration approach with $\gamma = 0.9$ will have the RM states with the potential values on each node.

The reward shaping is computed in the following way:

$$\alpha(r(s,\ a,\ s') + \gamma \cdot \max \tilde{q}(s',\ a'))$$

The algorithm should then be able to give a positive reward in the case of an advancement towards the TG path.

2 Temporal Logic

Temporal Logic (TL) is a logic language of the *Modal Logic* set that offers some time-related rules and operators to deal with propositions in which truth varies with time.

Low expressive capabilities of well known and more simpler logic formalisms, such as First Order logic (FOL) or Propositional Logic (PL), urged the necessity of more powerful logic languages that are grouped in a macro set under the name of *Modal Logic*. This collection of formal systems offers a wider freedom of expression, allowing to model cases in which a set of facts can be either True or False in more than one situation called *possible worlds*. This result has been achieved with a more complex syntax that, among all, involves the introduction, along with the classical connectives $\neg, \wedge, \vee$, of two shared operators:

- The *Necessarily* operator, $\Box\varphi$: states that φ is true in all the possible worlds,
- The *Possibly* operator, $\diamond\varphi$: states that φ is true or will be true in all the possible worlds.

Similarly, a more complex semantic, able to unify all the Modal Logic applications, has to be designed and it is called *Kripke Model*:

$$\mathcal{KM} = \langle W, R, V \rangle$$

where:

- W is the set of possible worlds, named *universe*,
- R is the *accessibility relation*, a binary relation on W and
- V is the assignment function which states the truth value of each pair of an atomic formula and a world.

This semantic representation allows also simple graphical representations based on graphs.

Introduced by Arthur Prior in the 1950s, TL is one of the main application of Modal Logic that allows reasoning over a timeline. Since the set of possible worlds lies on that timeline, differently from others Modal Logic formalisms, its representative Kripke model graph contains only directional transitions from one state to another or loop from one state to itself.

2.1 LTL$_f$

In 1977, TL has been extended by Amir Pnueli in one of its variants called Linear Temporal Logic (LTL) [10] characterized by the formulation of a wider set of operators:

- The *Globally* operator, $G\varphi$: like $\Box\varphi$ states that y is true now and in all the future time points,
- The *Finally* operator, $F\varphi$: like $\diamond\varphi$ states that y is true now or it will at some time point in the future,

– The *Next* operator, $X\varphi$ or $\circ\varphi$: states that φ will be true in the next time point,
– The *Until* operator, $\varphi_1 \mathcal{U} \varphi_2$: states that φ_1 is true until φ_2 is true.

Finally, the complete description of evolution over time, consisting in a sequence of propositional interpretations, one for each time point, is called *trace* and a formula's truth is evaluated over that trace. For the standard LTL semantics, traces are infinite [10].

Though LTL formulas can still be used to express properties over finite traces, the expressive power of Linear Temporal Logic over finite traces (LTL_f) is reduced to the First Order Logic's one.

To solve this limit, a new logic called Linear Dynamic Logic over finite traces (LDL_f)[11] has been presented by De Giacomo and Vardi in 2013, aiming to increase LTL_f's expressiveness, maintaining the same original intuitiveness when applied over finite traces.

2.2 DFA

A Deterministic Finite Automaton (DFA) is an abstract representation shaped as a finite state machine that takes a string of symbols as input and, running through a sequence of states, may accept or reject it.

More in depth, a DFA is defined as:

$$\mathcal{DFA} = (\Sigma, S, s^0, \rho, F)$$

with:

1. Σ: a finite and non-empty *alphabet*, where a *word* is an element of Σ^* (i.e. a finite sequence of symbols $a_0...a_n, a_i \in \Sigma$),
2. S: a finite and non-empty set of *states*,
3. $s^0 \in S$: the *initial* state,
4. $F \subseteq S$: the set of *accepting states*,
5. $\rho : S \times \Sigma \to S$: a *transition function*.

So defined, an automaton can be represented as an edge-labeled directed graph by mapping the graph's *nodes* to the *states*, the *edges* to the *transitions* and labelling them with symbols $a_i \in \Sigma$, choosing s^0 as *initial* node and a certain set of nodes as F. In this way, a generic edge s', from s to s' and labelled with a, is denoted by $s' = \rho(s, a)$.

Moreover, the transition function ρ can be either a partial mapping $S \times \Sigma \to S$ or a partial mapping $S \times \Sigma^* \to S$ if extended as following:

1. $\rho(s, \epsilon) = s$,
2. $\rho(s, xw) = \rho(\rho(s, x), w)$, for $s \in S,\ x \in \Sigma,\ w \in \Sigma^*$.

Finally, a DFA's run r over a finite word is a sequence of $n+1$ states, starting from $s_0 = s^0$ and passing through $s_{i+1} = \rho(s_i, a_i)$, for $i \in [0, n]$. If $s_n \in F$, r is accepting and the DFA accepts the input word w.

Then, given an input w, the resulting run r of a DFA A, as suggested by the name itself, is deterministic, meaning that exists at most one run only for each input and $w \in \mathcal{L}(A) \iff \rho(s^0, w) \in F$, where $\mathcal{L}(A)$ is the *language of A* defined as the set of finite words accepted by A.

3 STRIPS

The STanford Research Institute Problem Solver (STRIPS) [12], formalism was introduced by Fikes and Nilsson, in 1972. It is a problem solver that aims to find the optimal composition of operators that transforms a given initial world model/state into one that satisfies some stated goal condition, if reachable. The operators are the basic elements from which a solution is built. The problem space for STRIPS is defined by a quadruple

$$\mathit{\Pi} = \langle P, O, I, G \rangle$$

where:

- P is the set of conditions,
- O is the finite set of available operators and their effects on world models,
- $I \subseteq P$ the initial world model,
- $G \subseteq P$ is the goal statement.

The optimal path π_{opt} that can be obtained is the path with the lowest possible cost. The cost is defined as $c(x_t, u_t)$, where x_t and u_t are the Markovian system state and action at the time step $t \in [1, T]$, respectively. This means that there exists neither path nor plan π' such that $c(\pi') < c(\pi_{opt})$.

It is possible to translate the STRIPS domain into an equivalent LTL_f formula. This is accomplished by the following steps:

1. Define the STRIPS domain as the aforementioned set using fluents.
2. Convert STRIPS domain into a LTL_f formula; this process is executed in three steps, as described in the original LTL_f paper [11].
   (a) Each action $A \in Act$ defines
      - a precondition φ,
      - the *add* effects $\bigwedge_{F \in Add(A)} F$ and
      - the *del* effects $\bigwedge_{F \in Del(A)} \neg F$.

      We can express in LTL_f each of these properties:
      - Satisfy precondition to execute the action:

$$\Box(\circ A \rightarrow \varphi)$$

      - Satisfy effects after the action is executed:

$$\Box(\circ A \rightarrow \circ(\bigwedge_{F \in Add(A)} F \wedge \bigwedge_{F \in Del(A)} \neg F)$$

– Ensure any other fluent that is not changed by the action remains unaltered:

$$\Box(\circ A \rightarrow \bigwedge_{F \notin Add(A) \cup Del(A)} (F \equiv \circ F)$$

Additionally, since we want that only one action can be executed per step, we also have

$$\Box((\bigvee_{A \in Act} A) \wedge (\bigwedge_{A_i, A_j \in Act, A_i \neq A_j} A_i \rightarrow \neg A_j))$$

(b) Encode the initial propositions $F \in Init$ that are initially *True*:

$$\bigwedge_{F \in Init} F \wedge \bigwedge_{F \notin init} \neg F$$

(c) State the goal function, defined as φ_g as $\diamond \varphi_g$, requiring that eventually its fluents hold.

Once we have expressed the STRIPS domain in a LTL_f formula, it is possible to convert such formula in DFA to obtain the corresponding automaton, as described in Sec. 2.2.

4 Our experiments

We decided to test two categories of experiments:

1. Translation of non-trivial LTLf formulae to a DFA Automaton to use as reward machine in a RL task
2. Translation of a complete STRIPS description of an environment to a DFA Automaton to use as reward machine in a RL taks

In the first category we tested the following formulae, ordered by increasing complexity:

– Base environment goal: a U (b U c). In this goal we check first that there is a taxi anywhere in the map (a), then that the passenger is picked up by the taxi (b) and finally that the destination is reached (c). Note that this is the equivalent goal of the environment, which we used as baseline to ensure the correctness of our solution.
The generated DFA for this task is reported in Fig. 1.
– Pass through center: a U (b U (c U d)). In this goal first that there is a taxi anywhere in the map (a), then that the passenger is picked up by the taxi (b), that the taxi passes through the center of the map (c) and finally that the destination is reached (d).
The generated DFA for this task is reported in Fig. 2.

– Pass through 1 checkpoint: a U (b U (c U d)). This goal is similar to the previous one, but instead of checking if the taxi passes through the center, we check that it passes through a different checkpoint before reaching the destination.
  The generated DFA for this task is reported in Fig. 3.
– Pass through 2 checkpoints: a U (b U (c U (d U e))). This goal is similar to the previous one, but we check that it passes through two different checkpoints before reaching the destination.
  The generated DFA for this task is reported in Fig. 4.
– Pass through 3 checkpoints: a U (b U (c U (d U (e U f)))). This goal is similar to the previous one, but we check that it passes through two different checkpoints before reaching the destination.
  The generated DFA for this task is reported in Fig. 5.

For all of these, the training process ran for 2500 steps on consumer level hardware.

For the second category, we defined the STRIPS-equivalent problem. This was then used to generate the LTL_f formula that would, in turn, generate the DFA automaton we needed as temporal goal. As a test, we implemented only the base environment in STRIPS, so the environment goal and the temporal goal coincided. It is however important to note that the STRIPS problem describes the *entirety* of the original environment, so all possible initial states and goals are considered. Using STRIPS formalism makes it easier to define even more complex temporal goals and it is able to encapsulate a much broader variety of goals. In our case, the definition of the base environment goal actually generated a DFA that represented *all* possible environment runs (since, as we explained earlier, the environment is randomly initialized).

We developed the experiments using Python 3.7. We employed the following libraries during our development:

– NumPy [13] for mathematical computations,
– Logaut [14] defines most of the temporal logic paradigm we used,
– Lydia [15] is used as a backend during the LTL_f to DFA translation and
– TempRL [16] offers a high-level API for temporal goal (TG) based reinforcement learning.

When exploring the effects of reward shaping on different TG tasks, we used TempRL to wrap the base Gym environment in a TG-oriented environment that contained the DFA automaton of the task expressed in LTL_f logic. With such an environment it was possible to compute both observations for the agent and rewards that reflected not only the base environment but the DFA as well. In particular, we have that

– The observations are tuple of $(state_{env},\ state_{DFA})$.
– The reward is either $reward_{env}+reward_{DFA}$ when reward shaping is applied or just $reward_{env}$ for all transitions that do not result in a goal state to be reached, otherwise $reward_{env} + reward_{DFA}$.

When testing the STRIPS formalism to define the temporal goals, we first defined the STRIPS problem. We then converted it into an LTL_f formula, making sure that

1. we defined a `start` fluent that is True in the initialization and is False when the goal is reached,
2. all negated fluents are computed for any possible initial condition, as we noticed that leaving them as *don't care* resulted in an explosion of possible DFA states, and
3. all traces end in a `sink` state after the goal is satisfied, otherwise there could be duplicated states that differ only in whether the goal has been reached or not.

The conversion from STRIPS to LTL_f follows the proposed method in [11] and is implemented in a single Python script[1].

Since the states in the DFA generated in this way had no loops over themselves, we introduced an object in the TempRL library that would control whether or not to apply a step on the DFA[2]. This object, called *Step Controller*, is a state-based controller that determines when a given set of fluents may be used to traverse along the DFA and thus checking the progress of the temporal goal. The user then needs only to provide a step rule (i.e.: a formula that accepts a set of fluents and, based on those returns either True or False) and, during the training and testing loops, use this object to determine when to apply a step on the temporal goal, collecting the update states and temporal rewards.

For both experiments, the agent itself was trained via tabular Q-Learning. Exploration was ensured via an ϵ-greedy strategy with a variable ϵ value of 0.5 for all experiments but passing through 2 and 3 checkpoints, where a higher ϵ value was used (0.8). Additionally, the DFA reward was set to 100 for all tasks except for passing through 2 and 3 checkpoints, where we set the DFA reward to 500 and 5000, respectively, as we noticed that a lower reward lead to worse performance (probably due to the nature of the learning algorithm itself).

We provide the code for our solution at `https://github.com/gallorob/ra_project` and the website at `https://gallorob.github.io/ra_project/`.

5 Results

We report here the results of our experiments. Each experiment consists in training the agent on the assigned LTL_f formula and checking if reward shaping has a significant impact on performances. For each experiment we tracked the episodic rewards during both training and testing. All images referenced here are presented in Sec. 5.

[1] `https://github.com/gallorob/ra_project/blob/main/code/strips/src/strips_to_ltlf.py`

[2] PR available at `https://github.com/whitemech/temprl/pull/44`

Base environment goal We found that the agent reliably clears the environment, obtaining the temporal goal reward consistently during training and during testing, where the episodic reward is higher than the temporal goal reward (set at 100) and varies only on the number of steps taken by the agent. We report the graphs for this experiment at Fig. 7.

When reward shaping is applied, we can see that the agent converges more rapidly to a stable high reward during training. During testing, however, the learned Q values are good enough that no episode failed. We report the graphs for this experiment at Fig. 8.

Pass through center goal This modified temporal goal is still easily solved by the agent during training and testing, as shown in Fig. 9. All of the tests succeed, as shown in Fig. 10. As the taxi almost always has to move through the center tile, it does not surprises us that this goal is achieved easily.

Pass through 1 checkpoint goal This modified temporal goal is similar to the previous one and as such is still easily solved by the agent during training, though in testing it is still possible to see some failed episodes, as shown in Fig. 11.

Similarly to the previous goal, the addition of the reward shaping effect makes no real difference: the training is still solved and some of the tests episode actually fail, as shown in Fig. 12. The episodic rewards curve plateaus during training so we do not believe additional training steps would improve the agent's performances.

Pass through 2 checkpoints goal This modified temporal goal is more complex than the previous one but the agent is still able to obtain high rewards during training and testing, as shown in Fig. 13. We note however that some of the tests fail or have a final rewards that consists only on the base environment's reward (ie: it does not solve the temporal goal). The higher ϵ value used during training is the cause for the much diverse training rewards, as many more episodes were solved before actually ensuring the temporal goal.

The addition of the reward shaping leads to faster convergence during training and higher mean rewards but some of the tests still fail, as shown in Fig. 14.

Pass through 3 checkpoints goal This modified temporal goal is more complex than the previous one and the agent is not able to obtain high rewards during training and testing, as shown in Fig. 15. While sometimes the agent solves the temporal goal along with the base environment goal, it clearly does not solve the entire environment. This result may change with many, many more time steps provided to allow more exploration to take place, though we believe Q-Learning with a simple ϵ-greey strategy may not be ideal in this case.

The addition of the reward shaping does not improve the results, as shown in Fig. 16. In fact, during test we see only a few episodes where the base environment goal is solved, whereas no episodes solves the temporal goal.

STRIPS-based TG We report the generated graph of the STRIPS problem in Fig. 6. We can see that different initial states share, as expected, the same sub-goal, though the intermediate steps are obviously different. We ran the training for just 1500 steps as it rapidly converged to a solution, as shown in Fig. 17.

The rewards graph is similar to that of the base environment goal with reward shaping, reported in Fig. 8, but with less bad rewards during training. We think this is due to the smaller value of ϵ, set to 0.1. The agent learned a good policy that solved both the base environment goal and the temporal goal (though, as previously stated, these two coincide) and obtained a high-enough score in the test phase.

To qualitatively measure possible overhead during training with the STRIPS approach, we evaluated a simple Q-Learning approach with the same hyper-parameters but no temporal goals. We report the rewards plots in Fig. 18. It is clear that the agent in both environments learns roughly at the same rate, indicating the there is no apparent overhead between the two approaches.

Acknowledgements

This project was developed as final project for the "Reasoning Agents" course in the Master's course of "Artificial Intelligence and Robotics" at La Sapienza University, Rome. The project was supervised by PhD student Marco Favorito.

References

1. OpenAI: Ai research and deployment company
2. OpenAI: Gym, reinforcement learning toolkit
3. Dietterich, T.G.: Hierarchical reinforcement learning with the maxq value function decomposition. Journal of Artificial Intelligence Research **13**(1) (2000)
4. Hengst, B. In: Hierarchical Reinforcement Learning. Springer US, Boston, MA (2010) 495–502
5. Koenig, S., Simmons, R.G.: The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. Machine Learning **22**(1) (Mar 1996) 227–250
6. E., W. In: Hierarchical Reinforcement Learning. Springer US, Boston, MA (2011)
7. Andrew Y Ng, D.H., Russell, S.: Policy invariance under reward transformations: Theory and application to reward shaping (1999)
8. Giacomo, G.D., Iocchi, L., Favorito, M., Patrizi, F.: Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications (2019)
9. Camacho, A., Toro Icarte, R., Klassen, T.Q., Valenzano, R., McIlraith, S.A.: Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, International Joint Conferences on Artificial Intelligence Organization (7 2019) 6065–6073

10. Pnueli, A.: The temporal logic of programs. In: 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, Los Alamitos, CA, USA, IEEE Computer Society (oct 1977) 46–57
11. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence. IJCAI '13, AAAI Press (2013) 854–860
12. Fikes, R.E., Nilsson, N.J.: Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **2**(3) (1971) 189–208
13. Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with NumPy. Nature **585**(7825) (September 2020) 357–362
14. Favorito, M.: Logaut (v0.1.1) (2021)
15. Favorito, M., Fuggitti, F.: Lydia (v0.1.2) (2021)
16. Favorito, M.: Temprl (v0.3.0) (2021)

List of figures

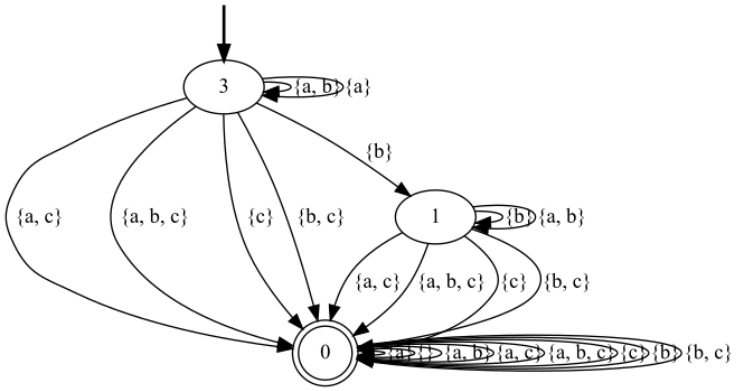


Fig. 1: The Base environment goal generated DFA

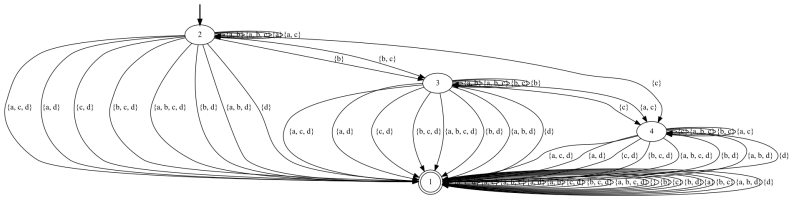


Fig. 2: The Through center environment goal generated DFA

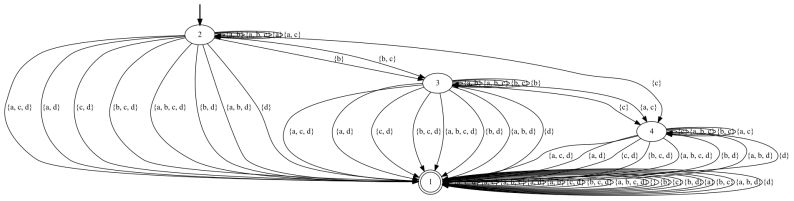


Fig. 3: The Through 1 checkpoint goal generated DFA

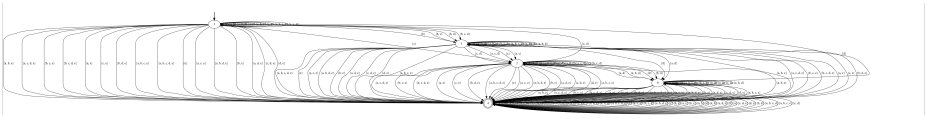


Fig. 4: The Through 2 checkpoints goal generated DFA

Fig. 5: The Through 3 checkpoints goal generated DFA

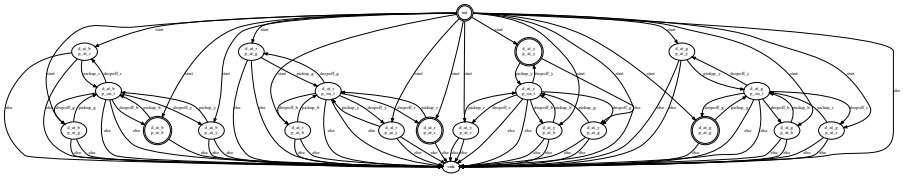
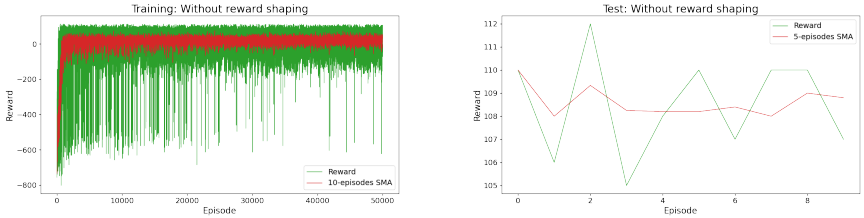
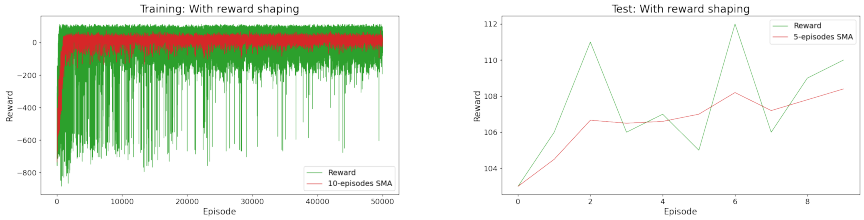


Fig. 6: The complete Taxi environment described as a STRIPS problem



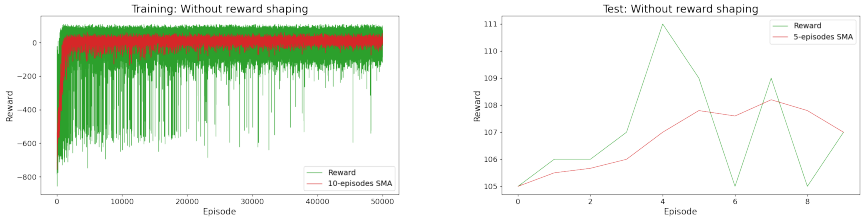
(a) Train rewards (b) Test rewards

Fig. 7: Train and test rewards for the base environment goal



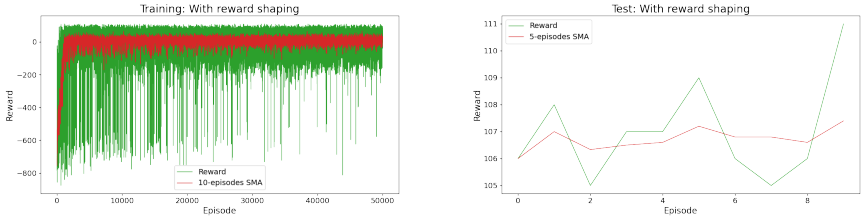
(a) Train rewards (b) Test rewards

Fig. 8: Train and test rewards for the base environment goal with reward shaping



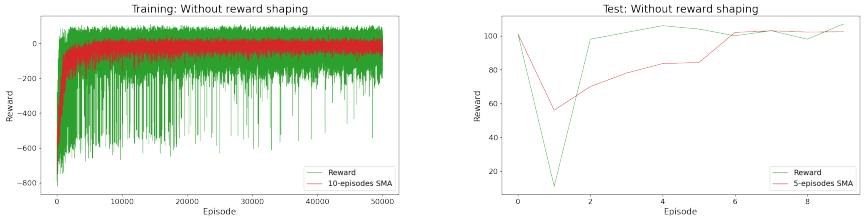
(a) Train rewards (b) Test rewards

Fig. 9: Train and test rewards for the passing through the center goal



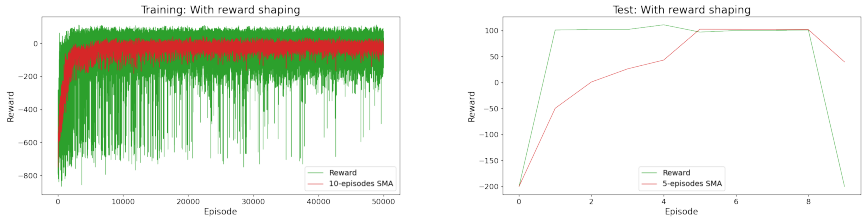
(a) Train rewards (b) Test rewards

Fig. 10: Train and test rewards for the passing through the center goal with reward shaping



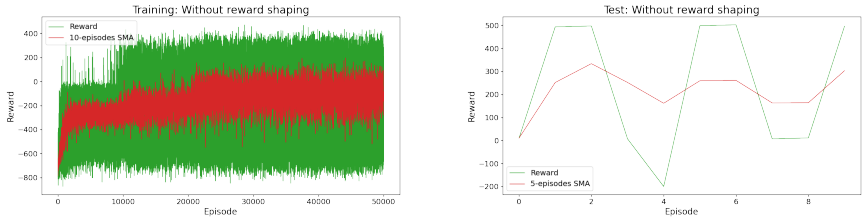
(a) Train rewards (b) Test rewards

Fig. 11: Train and test rewards for the passing through 1 checkpoint goal



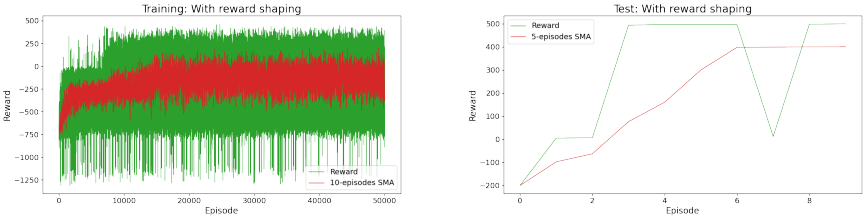
(a) Train rewards (b) Test rewards

Fig. 12: Train and test rewards for the passing through 1 checkpoint goal with reward shaping



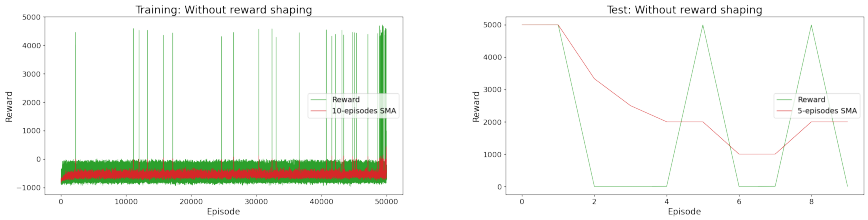
(a) Train rewards (b) Test rewards

Fig. 13: Train and test rewards for the passing through 2 checkpoints goal



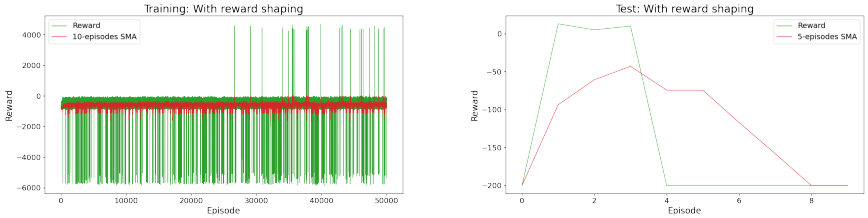
(a) Train rewards (b) Test rewards

Fig. 14: Train and test rewards for the passing through 2 checkpoints goal with reward shaping



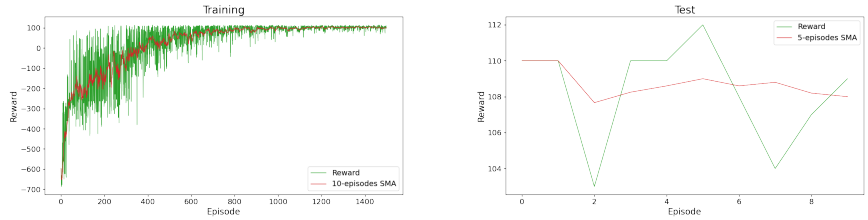
(a) Train rewards (b) Test rewards

Fig. 15: Train and test rewards for the passing through 3 checkpoints goal



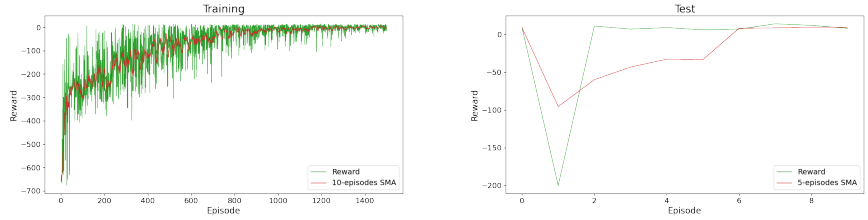
(a) Train rewards (b) Test rewards

Fig. 16: Train and test rewards for the passing through 3 checkpoints goal with reward shaping



(a) Train rewards (b) Test rewards

Fig. 17: Train and test rewards for the STRIPS-defined temporal goal.



(a) Train rewards (b) Test rewards

Fig. 18: Train and test rewards for the base environment with no temporal goal.