Lorenzi Flavio

June 2020 **Mat. 1662963**

Decision making with temporal goals : Reinforcement Learning For Restraining Bolts; De Giacomo, Iocchi, Patrizi

Report and analysis on this seminar topic; reproduction of experiments and related discussion

Abstract

In this work is discussed and performed a very atypical approach in which we can find a combination of Reinforcement Learning with Restraining Bolts, expressed through a particular formal system which derives from the classic First Order Logic: the **Linear Temporal Logic** (LTLf/LDLf). We will see a brief introduction of this topic, focusing on the mechanism that makes it possible, finally trying to repeat one of the original performed experiments, with a final discussion about it.

Theory focus and analysis

This work aims to show how one agent can learn while its targets changing with time, by adapting itself to the specifications expressed by the restraining bolts: the bolt provides an additional reward and features to the agent, that is able to act at different stages to get the rewards according to the temporal specifications.
So we'll have two different representations of the world, apparently unrelated: a **Reinforcement Learning Agent** modeled by a Markov Decision Process (MDP) and a logical specification of traces that are expressed in temporal logics over finite traces LTL /LDL, that will define the **Restraining Bolts**.

The latter was born in Science Novels as a device that restricts the capabilities of droids and thus allows a more effective control of these [2]. So in AI was taken up and developed thanks to Formal Systems Theory where in particular each specification for the agent (robot) can be formalized and expressed through fluents that define the LTL/LDL formulas φ; these can be expressed as following:

$$\varphi \quad ::= \quad \phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \circ\varphi \mid \varphi_1 \mathcal{U} \varphi_2$$

where φ is a propositional formula over P, ∘ is the *next* operator and $\mathcal{U}$ is the *until* operator.
So thanks to this we will be able to evaluate if and how much a generated trace is desiderable for the

agent. But here a problem arises: an LTL specification always generates a non-markovian reward that influences the learning agent. So a classical approach with MDP $M = \langle S,A,Tr,Ri \rangle$ is not possible and we must move to a non-markovian model (NMRDP) of this type: $M = \langle S, A, Tr, \{(\varphi_i , r_i)\}^{m}_{i=1} \rangle$ where instead it's possible apply the classic RL algorithms. So we have to learn an optimal policy for the NMRDP whose rewards are offered by the traces of the LTL and the transition function is hidden. The agent can see only the features from which Ri depends (RL part), but not the ones that influence ri (Restraining Bolts part). In Figure 1 we can see an abstract model of how this mechanism works:

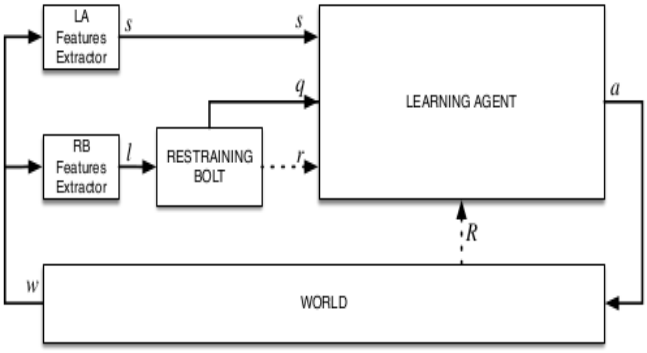

Figure 1

So a sequence of worl states w…wn defines a sequence of agent states s…sn and a sequence of fluent configuration l1…ln. The agent receives reward based on R_{ag} and the pairs (φ_i, r_i). We define it on the basis of which observations the agent can choose, before trying to satisfy a certain φ_i, since l1…ln are not seen directly [3].
The final solution is learning an excellent policy for NMRDP whose rewards are offered by the tracks of the LTL and the transition function is hidden: $\bar{\rho} : (Q_1 \times ... \times Q_m \times S)^* \rightarrow A$.

From *theorem 6* in paper "Foundation For Restraining Bolts" [2], we have the following problem definition:
Reinforcement Learning with LTL$_f$ / LDL$_f$ restraining specifications $M^{rb}_{ag} =< M_{ag,} RB >$ *with*
$M_{ag} = < S_{ag}, A_{ag}, Tr_{ag}, R_{ag} >$ *and* $RB = < \mathcal{L}, \{(\varphi_i, r_i)\}^{m}_{i=1} >$
can be reduced to RL over the MDP: $M^{q}_{ag} =< Q_1 \times ... \times Q_m \times S, A, T''_r, R''_{ag} >$
and optimal policies ρ^{new}_{ag} *for* M^{rb}_{ag} *can be learned by learning corresponding optimal policies for* M^{q}_{ag}.

This means that a minimal intervention is made to the learning agent: it is fed with reward ri at the appropriate times and this must be allowed to keep track of the satisfaction phase of the RB formulas, then feeding it with new features Qn. The policy $\bar{\rho}$ must always maximizes the cumulative reward.

Implementation, code and results

Four different experiments (Sapientino, Breakout, Minecraft, Cocktail Party) are performed [2], thanks to a simulate environment that is able to model all the relevant evolutions of the world; in general it was used the non-deterministic SARSA algorithm with $\gamma = 0.9$, Epsilon = 0.2,
learning rate (alfa) = 0.9 and with 100 n-episode steps.
Now let's focus in paricular on **Sapientino** experiment, an educational game where a small mobile robot has to be programmed to visit specific cells in a 5x7 grid. The state representation is given by the

agent pose (fx, fy, fθ) and the agent must learn just how to move through the grid. It can be done in a Omnidirectional (up, down, left, right actions) or a Differential way (forwards, backwards, turn left, turn right). Then we have two Restraining Bolt specifications, divided into this two type:

1. Visit at least two cells per color for each color
2. Visit all triplets of each color, given an order among colors (*)

I have tried to reproduce this experiment thanks to the *open source code* [4], here we have a Sapientino.py script in which in particular we have to focus on four important things:

1) Given an initial token (name, color, grid_x, grid_y) thanks to python dictionaries **Tokenbip** and **Colorbip**, that take respectively the first and the second element of this token, it's possible to make some check on the current status of the DFA (automa) and assign rewards.
2) There is a **class RewardAutoma** where each method is finalized to perform what is described by LTL formulas, according to the Restraining Bolts specifications. Here for example the method "*update*" checks for color and motion orders, updating the current state with rewards
3) There is a **class Sapientino** in which we can found all we need to perform the RL implementation: *Update positions, actions and colors + Reset functions + Rendering*
4) The **cumulative reward** will be the sum of goals and fails coming from both part (RL with goalSteps and RB with RAfail and RAgoal).

To perform a reproduction of the experiment I change the variable nVisitPerColor from 2 to 3, to see how it works with the second type of Restraining Specification (named above) (*); I have also switched True the boolean variable StopOnGoal, that can avoid overfitting, by stopping the training when a good/optimal policy is found. So training the experiment (command : *python experiment1.py*) I just immediately reached after a short time good results: the robot visits first the red cells, by keeping the order of colors specified in the code into *Tokens variable*, as we can see in Fig. 2 and 3.

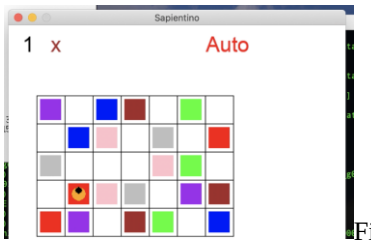

Figure 2

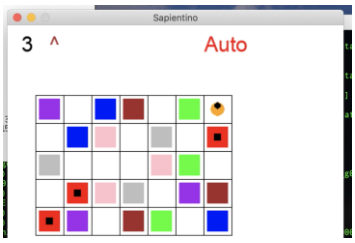

Figure 3

After 30 seconds the robot has learned the entire token and can perform it without any error: the final result is showed in Fig. 4 with the maximum performed score.

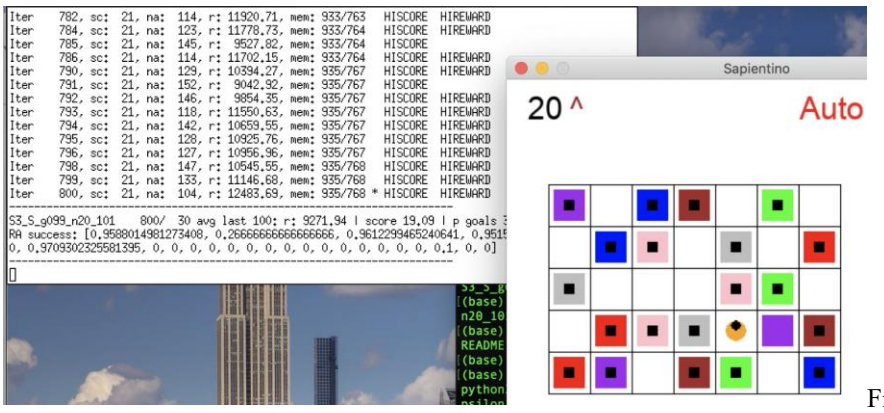

Figure 4

Final discussion and conclusion

By plotting the saved trained file *(python plotresults.py -datafiles data/S3_S_g099_n20_101)* we can see the internal average reward over time. By comparing the original experiment [1][2] plot (Fig. 5) with the one I have found (Fig.6), I can conclude that results are consistent and very good, because we can find an optimal policy in a very short training time.

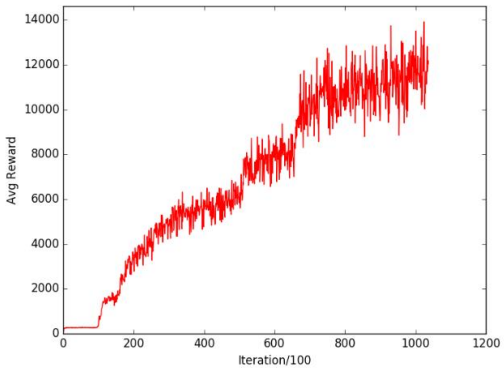

Figure 5

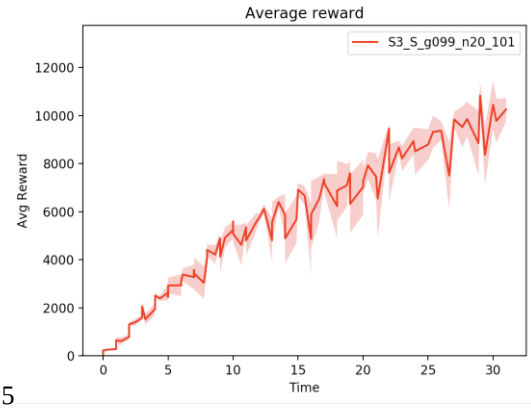

Figure 6

An important observation should be made about the fact that LTL / LDL formulas are not implemented in the code; in fact, the restraining bolts are generated through explicit specifications in the code, assigning positive or negative rewards (hard constraints) according to the state of the FDA.

In fact as it was said in Seminars [1] this is still a complex exsperimental topic; a full implementation with implicit creation of specifications from LDL formulas could be done as future work, maybe with the FLLOAT python LTL/LDL library (see 4c).

To conclude we have to say that despite in simulators exellent results are achieved, it is not robust in real life because we would need too much features of the environment, that is very difficult to realize. For example during the CocktailParty "Real" experiment [1] too many difficulties were found, which got stuck the experiment and made it still not complete.

References

- (1) Seminars in Artificial Intelligence and Robotics, Restraining Bolts (De Giacomo, Patrizi, Iocchi), May the 5th seminars, YouTube channel: https://www.youtube.com/watch?v=kbUYQHiryXs
- (2) Foundations for Restraining Bolts: Reinforcement Learning with LTLf/LDLf restraining specifications, Università La Sapienza, Roma Italy
- (3) Non-Markovian Rewards Expressed in LTL: Guiding Search Via Reward Shaping, Alberto Camacho, Oscar Chen, Scott Sanner, Sheila A. Mcllraith, University of Toronto, Canada
- (4) Restraining Bolts, available code for experiments:
  https://sites.google.com/diag.uniroma1.it/restraining-bolt
  https://github.com/iocchi/RLgames
  https://github.com/marcofavorito/master-thesis