Teaching Multiple Tasks to an RL Agent using LTL

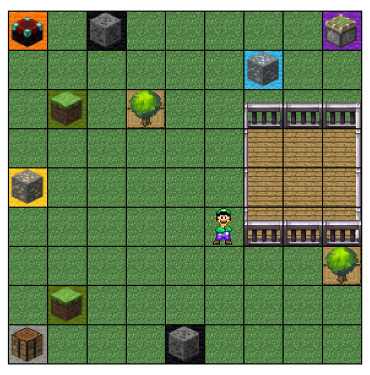
Rodrigo Toro Icarte[†‡] Toryn Q. Klassen[†] Richard Valenzano[§] Sheila A. McIlraith[†‡]

[†]Department of Computer Science, University of Toronto [‡]Vector Institute [§]Element AI

[†]{rntoro, toryn, sheila}@cs.toronto.edu, [§]rick.valenzano@elementai.com

Abstract. This paper examines the problem of how to teach multiple tasks to a *Reinforcement Learning (RL)* agent. To this end, we use Linear Temporal Logic (LTL) as a language for specifying multiple tasks in a manner that supports the composition of learned skills. We also propose a novel algorithm that exploits LTL progression and off-policy RL to speed up learning without compromising convergence guarantees, and show that our method outperforms the state-of-the-art approach on randomly generated Minecraft-like grids.

Running Example



Luigi can collect raw materials:

... and make new objects in:

E.g., make a bridge: get wood, get iron, and use the factory

Motivation

How do you describe a task to an RL agent?

Task specification ≠ Reward function

Language → Reward function

Why would we want such a language?

To define new task faster.

To transfer learning between tasks.

We use **Linear Temporal Logic (LTL)** to specify tasks and **LPOPL** to transfer learning between multiple tasks.

Related Work

Exemplar task	HER [2]	Sketches [1]	LPOPL
get wood	✓	✓	✓
get wood and then use the factory		✓	✓
get wood or iron			✓
get grass and iron			✓
do not leave the shelter at night			✓
Off-policy learning	✓		✓
Task decomposition		✓	✓

Previous works using variants of LTL in RL (e.g. [4, 5, 3]) **do not** exploit task decomposition or off-policy RL.

Specifying Tasks in LTL

Given a set of high-level events $\mathscr{P}$ that the RL agent can detect, such as

$$\mathscr{P} = \{\texttt{got\_wood}, \texttt{got\_iron}, \texttt{got\_grass}, \texttt{used\_workbench}, \texttt{used\_factory}, \texttt{is\_night}, \texttt{at\_shelter}, \ldots\},$$

we can use LTL to define tasks by composing occurrences of events in $\mathscr{P}$. LTL augments propositional logic with temporal operators $\bigcirc$ (*next*), $\Diamond$ (*eventually*), and U (*until*):

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \Diamond\varphi \mid \varphi_1 \mathsf{U} \varphi_2 \text{ with } p \in \mathscr{P}$$

Examples:

- **eventually** `got_wood`
- **eventually** (`got_grass` and **eventually** `used_factory`)
- **eventually** `got_wood` or **eventually** `got_iron`
- **eventually** `got_grass` and **eventually** `got_iron`
- (`is_night`→`at_shelter`) **until** `got_wood`

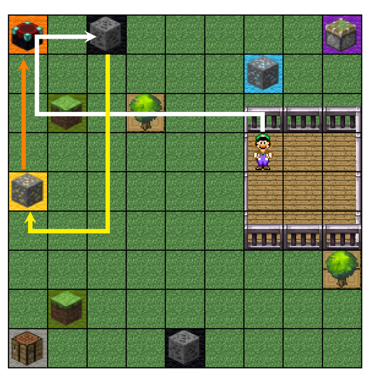
From LTL formulae to rewards

LTL formulas can be progressed as the agent accomplishes part of them. **We reward the agent when it finishes the task.**

Given an LTL formula φ and state s, we can *progress* φ using s:

- $\text{prog}(s,p) = \text{true}$ if $p \in L(s)$, where $p \in \mathscr{P}$
- $\text{prog}(s,p) = \text{false}$ if $p \notin L(s)$, where $p \in \mathscr{P}$
- $\text{prog}(s,\neg\varphi) = \neg\text{prog}(s,\varphi)$
- $\text{prog}(s,\varphi_1 \wedge \varphi_2) = \text{prog}(s,\varphi_1) \wedge \text{prog}(s,\varphi_2)$
- $\text{prog}(s,\bigcirc\varphi) = \varphi$
- $\text{prog}(s,\Diamond\varphi) = \text{prog}(s,\varphi) \vee \Diamond\varphi$
- $\text{prog}(s,\varphi_1 \mathsf{U} \varphi_2) = \text{prog}(s,\varphi_2) \vee (\text{prog}(s,\varphi_1) \wedge \varphi_1 \mathsf{U} \varphi_2)$

Example:



$\varphi_1 = \Diamond(\texttt{got\_iron} \wedge \Diamond\texttt{used\_factory}) \wedge \Diamond\texttt{got\_gold}$

$\varphi_2 = \Diamond\texttt{used\_factory} \wedge \Diamond\texttt{got\_gold}$

$\varphi_3 = \Diamond\texttt{used\_factory}$

$\varphi_4 =$ true (**+1 reward**)

Off-Policy Learning with LTL

Suppose Luigi has to learn two tasks:

$\varphi_1 =$ **eventually** (`got_iron` and **eventually** `used_factory`) and **eventually** `got_gold`

$\varphi_2 =$ **eventually** ([`got_grass` or `got_wood`] and **eventually** `used_factory`)

Then, all the experience collected while learning to solve φ_1 can also be used to learn a policy for φ_2 using off-policy RL.

LPOPL Overview

Step 1: Decompose tasks into subtasks with LTL progression.

$\varphi_1 =$ **eventually** (`got_iron` and **eventually** `used_factory`) and **eventually** `got_gold`

$\varphi_2 =$ **eventually** ([`got_grass` or `got_wood`] and **eventually** `used_factory`)

$\varphi_3 =$ **eventually** (`got_iron` and **eventually** `used_factory`)

$\varphi_4 =$ **eventually** `used_factory` and **eventually** `got_gold`

$\varphi_5 =$ **eventually** `used_factory`

$\varphi_6 =$ **eventually** `got_gold`

$\varphi_7 =$ true

Step 2: Learn one policy per subtask with off-policy learning.

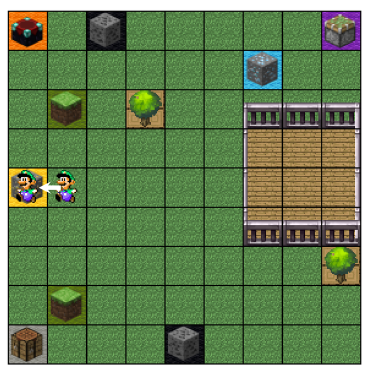
Standard q-learning update given experience (s,a,r,s'):

$$Q(s,a) \xleftarrow{\alpha} r + \gamma \max_{a'} Q(s',a')$$

LPOPL update using q-learning given experience (s,a,s'):

$$Q_\varphi(s,a) \xleftarrow{\alpha} r_\varphi + \gamma \max_{a'} Q_{\varphi'}(s',a')$$

where $\varphi' = \text{prog}(s',\varphi)$ and $r_\varphi = 1$ iff $\varphi \neq \varphi' = \text{true}$.



$Q_{\varphi_1}(s,a) \xleftarrow{\alpha} \gamma \max_{a'} Q_{\varphi_3}(s',a')$

$Q_{\varphi_2}(s,a) \xleftarrow{\alpha} \gamma \max_{a'} Q_{\varphi_2}(s',a')$

$Q_{\varphi_3}(s,a) \xleftarrow{\alpha} \gamma \max_{a'} Q_{\varphi_3}(s',a')$

$Q_{\varphi_4}(s,a) \xleftarrow{\alpha} \gamma \max_{a'} Q_{\varphi_5}(s',a')$

$Q_{\varphi_5}(s,a) \xleftarrow{\alpha} \gamma \max_{a'} Q_{\varphi_5}(s',a')$

$Q_{\varphi_6}(s,a) \xleftarrow{\alpha} 1$

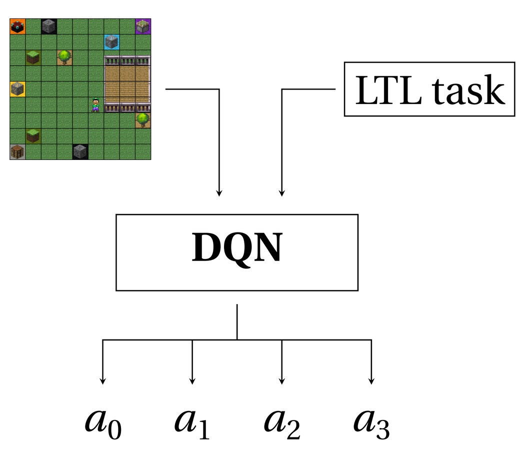
Theorem: LPOPL using tabular q-learning converges to an optimal policy.

Experiments

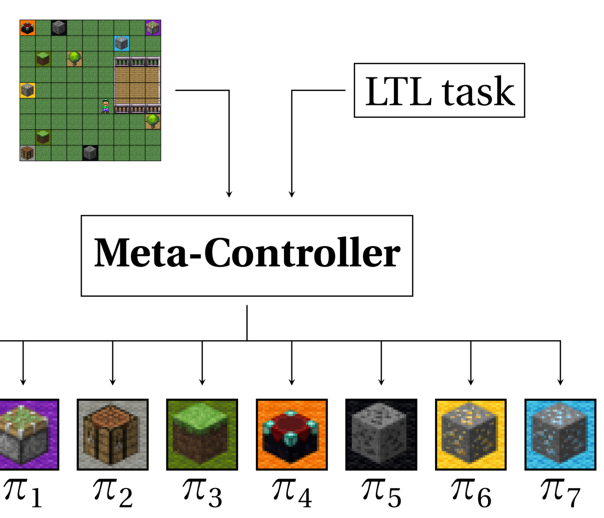
Goals: study LPOPL + DQN; compare with standard RL and with alternative decomposition methods

Baselines

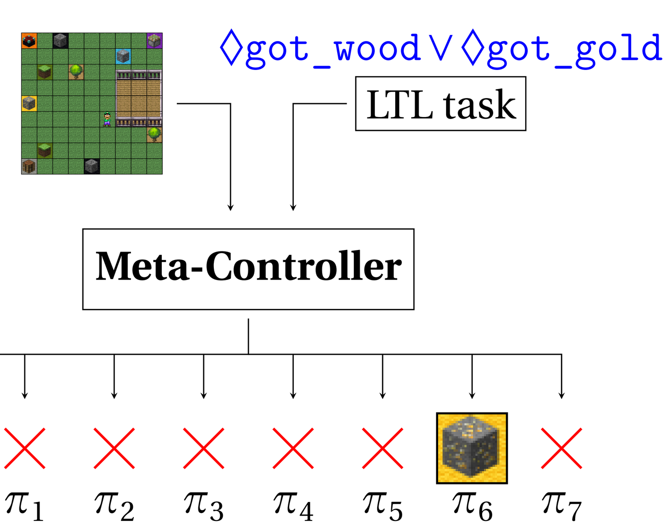
DQN-L



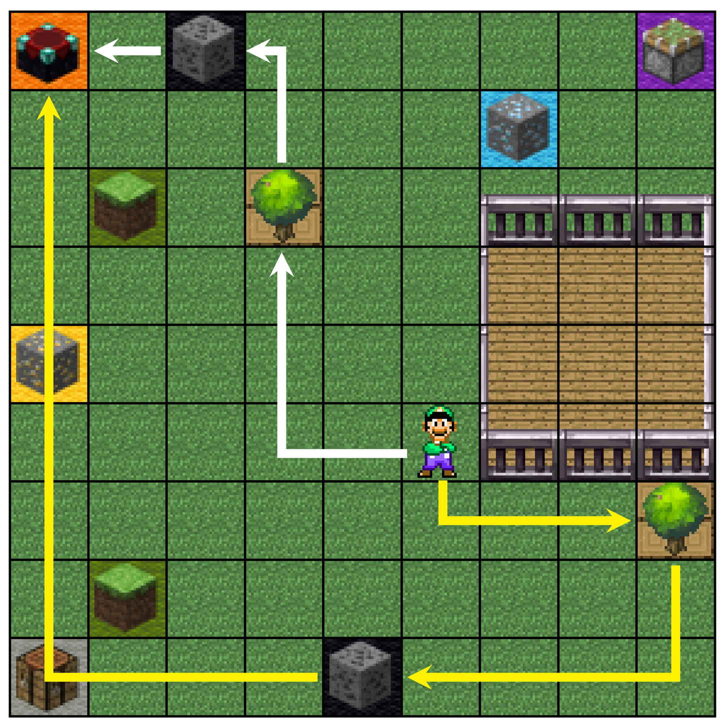
HRL-E



HRL-L



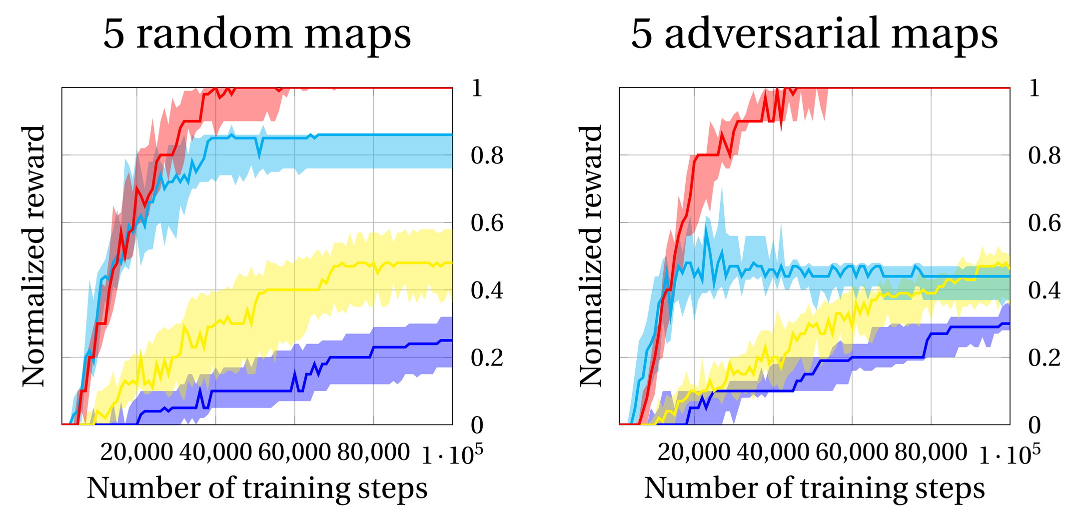
Note: Hierarchical methods may find suboptimal policies.



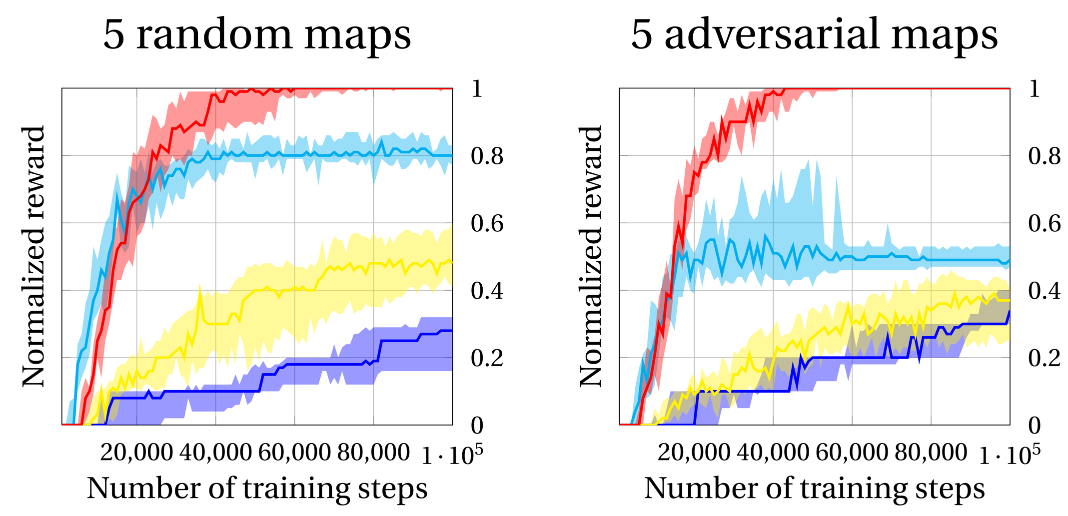
Results

The first experiment considers 10 tasks defined as sequences of subgoals (Andreas et al., 2017), e.g., get iron, then get wood, then use factory. The later experiments take advantage of the expressiveness of LTL to describe partially ordered tasks and safety constraints.

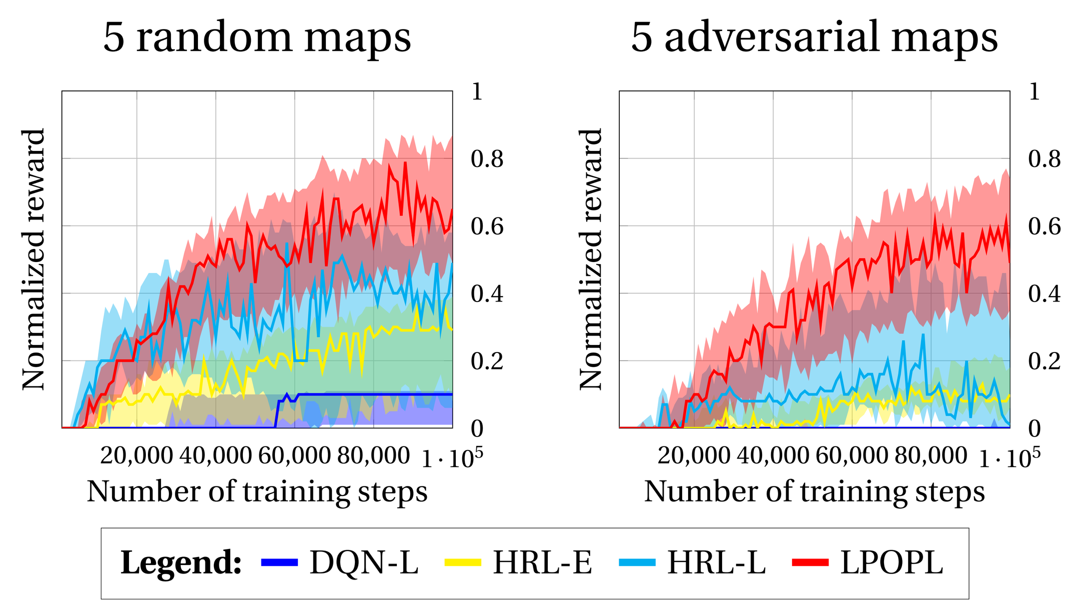
Experiment 1: Sequences of subtasks



Experiment 2: Interleaving subtasks



Experiment 3: Safety constraints



Conclusion

- LPOPL takes tasks defined with LTL, decomposes them using LTL progression, and learns the subtasks
- LPOPL outperformed DQN and HRL over various tasks
- code at `https://bitbucket.org/RToroIcarte/lpopl`

References

[1] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 166–175, 2017.

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba. Hindsight experience replay. In *Proceedings of the 30th Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 5048–5058, 2017.

[3] M. Hasanbeig, A. Abate, and D. Kroening. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

[4] X. Li, C. I. Vasile, and C. Belta. Reinforcement learning with temporal logic rewards. In *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3834–3839, 2017.

[5] M. L. Littman, U. Topcu, J. Fu, C. Isbell, M. Wen, and J. MacGlashan. Environment-independent task specifications via GLTL. *arXiv preprint arXiv:1704.04341*, 2017.