Multi-agent Hierarchical Reinforcement Learning in Urban and Search Rescue

Long Le[1] and Dana Hughes[2] and Katia Sycara[2]

***Abstract*— In an urban search-and-rescue (USAR) task, a team of agents cooperates to explore different rooms in the environment, clear rubble, and triage victims. USAR presents a challenging control problem in multi-agent Reinforcement Learning due to its long horizon, sparse and delayed reward, and the large size of the state space. Hierarchical approaches such as options and state abstraction have been proposed as a way to reduce the state space and planning complexity in such problems. In this work, we leverage domain knowledge to (1) train individual low-level (sub)-policies of each agent in smaller subsets of the state space, and (2) then train a team-level policy over a reduced graph representation of the states using those pretrained sub-policies. We show some early-stage results where our approach outperforms two multi-agent algorithms option-critic, and independent Q-learning on a simple environment.**

***Index Terms*— Multi-agent Reinforcement Learning. Hierarchical Planning. Search-and-Rescue.**

I. INTRODUCTION

Reinforcement Learning (RL) is a framework where an agent or multiple agents interact with the environment, receive observations and rewards for their actions. The agents learn through experience how to maximize their future discounted rewards. RL has been applied to solve a diverse array of domains, from game playing [Mnih et al., 2013] to self-driving [Kiran et al., 2021], and provides a natural way for training artificial teams in urban search-and-rescue tasks.

In a USAR task, a fully cooperative team of agents has to navigate and explore the environment, removing obstacles such as rubble along the way, to locate and rescue victims. This is a difficult task in several ways. First, USAR is a *hard-exploration* problem where the environment has very sparse and delayed rewards. That is, because the objective of the team is to rescue victims, agents can only receive extrinsic rewards when a victim is successfully triaged, which is a rare event. Further, the learning is complicated by *delayed* rewards [Arjona-Medina et al., 2018] in that the decision an agent makes in one time step tends to have its consequence revealed much later. For example, when the agent clears the rubble at the entrance of a large hall-way, it might only know if its effort has been worth it when a teammate discovers a victim in the hall-way multiple time steps later. The learning can also suffer from *long horizon*. Coupled with sparse rewards, long horizon means that an agent needs to perform a very long sequence of actions before any reward can be received. Finally, environments for USAR missions are typically large, requiring agents to differentiate and generalize between a vast number of states.

[1]Long Le is with the College of Information and Computer Sciences, University of Massachusetts Amherst. `lnle@umass.edu`

[2]Sycara and Hughes are with the Robotics Insitute, Carnegie Mellon University. {`danahugh, sycara`}`@andrew.cmu.edu`

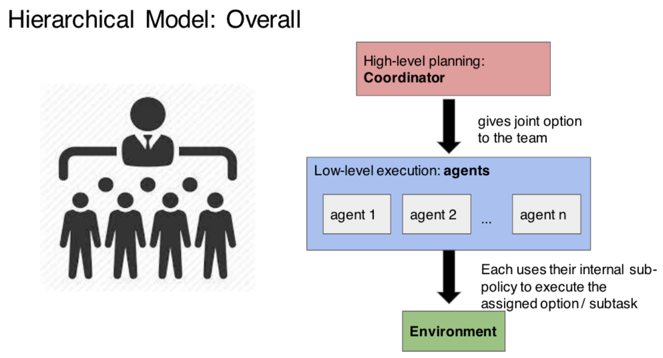

Hierarchical Model: Overall
High-level planning: Coordinator
gives joint option to the team
Low-level execution: agents
agent 1
agent 2
...
agent n
Each uses their internal sub-policy to execute the assigned option / subtask
Environment

Fig. 1: Coordinator-Agent Hierarchical Framework.

To deal with hard-exploration and large state space, hierarchical approaches, both over the action space [Dayan and Hinton, 1992], [Sutton et al., 1999] and the state space [Barry et al., 2011], [Steccanella et al., 2021], [Shang et al., 2019], have been proposed. In this work, we utilize both action and state hierarchical representations to tackle the USAR problem. Specifically, we train each agent in the team to complete some sub-tasks in a completely decentralized manner using the low-level features of subsets of the state space, and then train a centralized team-level coordinator on a high-level graph representation of the environment. In our framework, the coordinator gives command to the team, for example "agent 1 goes to search for victims in room A, and agent 2 goes to room B". The agents execute their assigned sub-tasks and report their rewards to the coordinator. We do not assume an agent's sub-policy to achieve a sub-task is prescribed a priori but instead choose to obtain it through decentralized training. See Fig. 1 for an illustration. This approach reduces the state space since an agent only plans over a small subset of the environment, and the coordinator only plans over a small graph view. It also alleviates the long planning horizon problem since the coordinator avoids micro-management and learns to associate high-level decisions with rewards. We demonstrate that this approach outperforms two multi-agent algorithms option-critic, and independent Q-learning in a simple environment.

Training high-performant agents not only can contribute towards the future development of autonomous robotic teams for USAR missions but can also serve to give advice to human teams in order to improve their performance.

The organization of this paper is as follows. Section II reviews single-agent and multi-agent Reinforcement Learning. Section III discusses some previous work to obtain hierarchical structures for action and state spaces. Our approach in modeling the search-and-urban task using hierarchical planning is given in Section IV. The result in given in Section V. Section VII is the conclusion and future work.

II. PRELIMINARIES

A. Markov Decision Process

In single-agent RL, a Markov Decision Process (MDP) is often assumed. MDP is a tuple $(\mathcal{S}, \mathcal{A}, p_0, p, r, \gamma, L)$, where $\mathcal{S}$ is a set of states ($s \in \mathcal{S}$), $\mathcal{A}$ the action space ($a \in \mathcal{A}$), $p_0(s_0)$ the initial state distribution, $p(s_{t+1}|s_t, a_t)$ the environment dynamics, and $r(s_t, a_t, s_{t+1})$ the reward function, γ the discount factor, and $L \in \mathbb{N} \cup \{\infty\}$ the horizon. In words, the agent appears initially at state s_0 drawn from p_0. At any time step t, the agent is in state s_t, takes an action a_t and transition to the next state s_{t+1} drawn from $p(.|s_t, a_t)$, and receives a discounted reward of $\gamma^t r(s_t, a_t, s_{t+1})$. In finite-horizon problems such as ours, the agent keeps interacting with the environment until the time step reaches the horizon L, after which the episode is terminated. A policy $\pi(a_t|s_t)$ is a conditional probability distribution over actions, dictating which action the agent will take in a given state. The goal of the agent in RL is to learn a policy π that maximizes the expected total discounted reward

$$\mathbb{E}\left[\sum_{t=0}^{L} \gamma^t r(s_t, a) \mid s_0, \pi\right].$$

B. Partially Observable Markov Decision Process

A partially observable decision process (POMDP) is an extension of MDP to situations where the agent does not have full access to the environment state s but rather to only observation $o \in \mathcal{O}$ drawn from the observation probability model $O(o \mid s)$. In a USAR mission, an agent does not have perfect information about the environment, and can only access the observations. For example, the agent does not know where the victims are initially but can observe its local surroundings.

C. Decentralized Partially Observable Markov Decision Process

A Decentralized Partially Observable Markov Decision Process (Dec-POMDP) is a generalization of POMDP to multi-agent systems. Dec-POMDP allows for heterogeneous teams of agents by enabling agents to have different action sets and observations. Formally, an agent i can takes action $a^i \in \mathcal{A}_i$ and receives observation $o^i \in \mathcal{O}^i$. The state transition probabilities $p(s'|s, a^1, ..., a^n)$ and rewards $r(s, a^1, ..., a^n, s')$, where n is the number of agents, depend on the joint action $(a^1, ..., a^n)$ of all agents. Note that the reward r in this case is shared among all agents. This is appropriate for fully-cooperative team settings such as ours. Markov games [Littman, 1994] extend Dec-POMDP to mixed cooperative-competitive settings by allowing different agents to have different rewards. We will assume a Dec-POMDP in our work.

III. RELATED WORK

A. Action Abstraction

Two popular frameworks for hierarchical RL over the action space are options [Sutton et al., 1999] and Feudal learning [Dayan and Hinton, 1992], [Vezhnevets et al., 2017].

In the former approach, the concept of "macro-actions" or options is introduced. An option is a high-level action that consists of primitive actions and is *temporally extended i.e.* the option might take a variable number of time steps to complete. For example, the option of "going to the airport" consists of a sequence of primitives such as steering the car wheel, and might take 30 or 45 minutes depending on the traffic condition. Formally, an option ω consists of an initiation set $\mathcal{I}_\omega \subset \mathcal{S}$, a sub-policy $\pi_\omega(a|s)$ and a termination probability distribution $\beta_\omega(s)$. In state s, the agent use a meta-control policy $\mu(\omega|s)$ to picks an option ω available, execute the option using π_ω until completion determined by β_ω. There are two levels to be learned μ and $\{\pi_{\omega_i}\}_{i=1}^m$ where m is the number of options. Options can be explicitly defined and learned using subgoals and pseudo-rewards as in [Sutton et al., 1999], or both levels can be learned simultaneously from end-to-end using option-critic in [Bacon et al., 2016]. Distributed option-critic [Chakravorty et al., 2019] extends the option-critic framework to multi-agent systems via common-belief and communication.

In the latter approach of Feudal learning, there is a hierarchy of managers (lords) and sub-managers (serfs). The managers at different levels in the hierarchy observe different scales of resolution of the environment, assign tasks to sub-managers or workers one level below them. A manager at a level receives an intrinsic reward from their superior based on how well they execute their assigned task. This framework is different from the option's in that it focuses on state space abstraction between different levels of hierarchy rather than temporal abstraction [Riemer et al., 2019].

Our approach is inspired by both of the mentioned frameworks. We use sub-goals to train agents' sub-policies, and use a coarser resolution of time and space for the coordinator (manager) for the team-level policy.

B. State Abstraction

In some domains, it is possible to reduce the state space by "factoring" the MDP [Boutilier et al., 2000]. The idea is that some parts of the MDP state are independent and thus can be exploited by a more compact Bayesian graphical representation. [Barry et al., 2011] presents an algorithm to cluster states of a factored MDP into "macro-states". In the present work, we have not attempted to represent the states in factored form.

The authors in [Steccanella et al., 2021] learn a hierarchical representation of the environment by partitioning the states into a set of "abstract states". A "compress function", mapping primitive states to abstract states, is learned from data. The idea is to first collect a set of trajectories by letting the agent wander through the environment. Then, a compress function is optimized using the heuristic that consecutive states in a trajectory should usually belong to the same abstract state [1]. They showed that the learned abstract states correspond to different rooms in a grid-world experiment. Our current work assumes that the abstract states are already given.

The paper [Shang et al., 2019] is another approach to obtain a high-level view of the environment by identifying the "pivotal states", where pivotal states are defined to be those most useful to predict the agent's actions. A variational autoencoder is used to learn the pivotal states from collected trajectories. They show that the pivotal states correspond to hall-way junctions in 2-D mazes. This is an interesting direction for our future work.

Note that in both [Steccanella et al., 2021] and [Shang et al., 2019] approaches, we need to run some exploration trajectories to build a state abstraction before actual learning can occur. In this paper, we take a shortcut and use a pre-specified state abstraction instead.

IV. Approach

A. Two-rooms Environment

The two-rooms environment consists of a 10x15 2-D grid with two victims in the top and bottom rooms, and two medics in a hall-way (see Figure 2). The goal of the agents is to rescue both victims in the shortest amount of time possible. Each agent location is a tuple (x, y, dir), where (x, y) is a coordinate on the 2-D grid and dir is one of four possible directions NORTH, SOUTH, WEST, EAST, representing the direction that the agent is currently facing. The navigation actions include FORWARD, TURN-LEFT and TURN-RIGHT. An agent also has a special action TRIAGE that only takes effect if the agent is facing a victim; otherwise, it is a no-op. The environment also has walls that the agents cannot pass through. In other words, a FORWARD action when facing a wall does nothing.

For simplicity, we assume a fully observable MDP. The horizon is set to 100, and γ to 0.99.

B. Graph-level View and Sub-policies Training

We handcraft a graph representation of the environment where each node is a section of the map (see Figure 3). The agents learn to navigate between neighboring nodes of the graph through pseudo-rewards. For example, to train a navigation strategy for the bottom agent from its position to the top room, we remove the other agents and victims from the map. Then, we give a synthetic reward of +100 for reaching any locations in the top room, and a reward of -1 with every passing time step to encourage the agent to reach the sub-goal as fast as possible. Within each room, we also train the agents to triage the victims by giving a reward of +100 for a successful triage and -1 time reward as before. Note that this triage sub-policy is learned assuming that an

[1]One naive abstract function is to map all states to the same abstract state. [Steccanella et al., 2021] avoids this by having another loss term to make sure that all abstract states are somewhat equally likely over the trajectories.

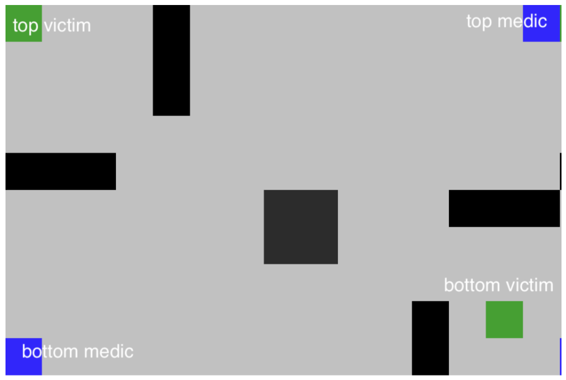


Fig. 2: The two-rooms environment. The walls are in black. Two green squares are two victims. Two blue squares are two agents.

agent is already in the room where a victim might be located. Thus, the sub-policy has no guarantee for good performance when, for example, an agent is asked to execute the option "RESCUE TOP VICTIM" when the agent is not in the top room but rather is in the hall-way.

Note that the agents' sub-policies are local in the sense that they require local information for execution. For example, to execute the option of going from the hall-way to the top-room, an agent needs not know what is happening in the bottom room. Note that in our current work, since we assume a MDP, this discussion about local information is less relevant. However, in a Dec-POMDP version of the problem, sub-policies dependency on only local information would reduce the state representation needed for learning those policies.

Further, the approach of local sub-policies provides an implicit curriculum training [Bengio et al., 2009] for the agents, lessening the problem of sparse rewards and long horizon. For instance, when an agent is already in the top-room, it is much more likely that the agent stumbles upon triaging the top victim, thereby providing more frequent rewards in the initial exploration phase of an RL algorithm, and thus speed up learning.

The description of options is given in Table I. In general, we should train 4 navigation sub-policies: TOP-to-HALLWAY, HALLWAY-to-TOP, HALLWAY-to-BOTTOM, and BOTTOM-to-HALLWAY. Then, to navigate from the bottom room to the top room, an agent would chain together its BOTTOM-to-HALLWAY and HALLWAY-to-TOP options. For this simple environment, we take a shortcut and only train 2 navigation sub-policies as in Table I with the understanding that the agents are always initialized in the hall-way during the full search-and-rescue mission. Also, there is no need to jump 2 hops in the graph here (e.g. going from the bottom to the top room) since an optimal team-level policy is to have the top medic go to the bottom room, and the bottom medic to the top room.

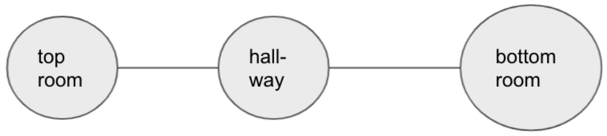


Fig. 3: The graph representation for the two-rooms environment.

C. Coordinator's Policy Training

Once we have had the trained subpolicies, we proceed to train the coordinator's policy over the joint option using the graph representation. That is, the coordinator sees a state (n_1, n_2, v_1, v_2) where $n_i \in$ {TOP-ROOM, HALL-WAY, BOTTOM-ROOM} indicates the position of agent i, and $v_i \in \{0, 1\}$ indicates whether the i^{th} victim has been triaged. The coordinator's meta-policy is over the joint options (ω_1, ω_2) where each ω_i can take on 1 of 4 possible options (see Table I).

The coordinator selects a joint option and commands its agents to use their sub-policies to execute that joint option. Once one of the agents completes its option, the coordinator terminates the options of all agents, re-evaluates the situation, and select another joint option. We also enforce the termination conditions of options by giving a large negative reward to the coordinator whenever it tries to select an unavailable option. For example, if the top agent is in the hall-way and the coordinator commands it to execute "TRIAGE-BOTTOM-VICTIM", the coordinator will receive a large negative reward and learn to avoid selecting invalid options in the future.

V. Result

Both the training of sub-policies and the coordinator's meta-policy are done by Deep Q networks with experience replay and target nets [Mnih et al., 2013].

We compare our result against two multi-agent algorithms: the option-critic [Sutton et al., 1999], and independent Q-learning.

For each algorithm, we run 8 trials and plot the results in Figures 4, 5, 6. Besides rewards, we also plot the number of time steps taken in each episode. The lower the number of steps, the quicker the agents were able to achieve their goals. If the agent was not able to accomplish its goal, then the number of steps would be equal to the horizon. For every small number of training episodes, we also performed a test episode. DQN-based models sometimes suffer from "catastrophic forgetting" [Roderick et al., 2017] when the agent's performance degrades after achieving optimal behavior. One reason is that after achieving a decent policy, the agent is only used to seeing "good" states. When some "bad" states are encountered by chance, the agent's Q-value prediction is completely off, incurring a high error and changing the network weights unfavorably. The simple solution in [Mnih et al., 2013] is to regularly test the model during training and save the model resulting in the best test performance so far.

Each of our sub-policy takes at most around 600 episodes to obtain optimal solutions. In the best trial, they take around 1000 episodes in total but they can be trained concurrently. The coordinator's meta-policy trained on those sub-policies takes about 30 episodes to reach optimality. The best trial of the option-critic takes around 1,500 episodes to reach optimality. For independent Q-learning, the variance in performance is small but the algorithm was not able to rescue both victims at all. The best trial can learn to triage one victim after 450 episodes. The performance of various algorithms is summarized in Table II. We compare the best performance of each algorithm against each other since the average metric might be deceptive. As discussed, there is the problem of catastrophic forgetting so if there are two trials that achieve optimality at different times, for example the test rewards of the two trials are $[0, 100]$ and $[100, 0]$, then their average might seem to suggest that the agent has not achieved the goal at all: the average is $[50, 50]$.

The independent Q-learning does not touch on the multi-agent aspect of the problem since each agent learns on its own, and is not aware of the existence of other agents. Thus, this approach suffers from non-stationarity as other agents change their policies over time.

The distributed option-critic overcomes the non-stationarity issue by giving the centralized critic access to all observations of agents. It also improves the MADDPG algorithm [Lowe et al., 2017] by adding action abstraction to this approach by using options. However, there is no state abstraction, and the options are learned, which might not be as good as hand-specified options using domain knowledge.

VI. CONCLUSION AND FUTURE WORK

In this work, we adopt hierarchical abstraction over the state and action spaces to build a team of autonomous agents for the urban search-and-rescue task. We present our preliminary results for a simple two-rooms environment, and compare them to those by common end-to-end multi-agent algorithms.

In the future, we would like to extend this framework to a more complicated environment of bigger size, with heterogeneous agents with different capabilities, rubble, and partial observability. For a bigger environment, we plan to use a Graph Neural Network (GNN) to encode the environment information and formulate the Reinforcement Learning problem as a node-level decision-making task (e.g. see [Gammelli et al., 2021]). In the case of partial observability, there is a problem: the agents do not know whether a room contains a victim, and if so how many and where they are. Here, intrinsic curiosity-driven exploration [Pathak et al., 2017] and multi-object search [Wandzel et al., 2019] can help. Also, we would like to automate the learning as much as possible, for example by learning the state representation through initial exploration as proposed by [Steccanella et al., 2021] and [Shang et al., 2019]. We would also like to experiment with transfer learning as discussed in [Steccanella

Option	Initiation set	Termination condition
NAVIGATE-TO-TOP-ROOM	agent is in the hall-way	agent reaches the top room
NAVIGATE-TO-TOP-ROOM	agent is in the hall-way	agent reaches the bottom room
TRIAGE-TOP-VICTIM	agent is in the top room	top victim is triaged
TRIAGE-BOTTOM-VICTIM	agent is in the bottom room	bottom victim is triaged

TABLE I: The agent's options in the two-rooms environment.

Algorithms	Best number of eps til optimality in 8 trials
Graph + options (ours)	1030
Option-critic	1,500
Independent Q-learning	$> 5,000$

TABLE II: The performances of different multi-agent control algorithms.

et al., 2021] by randomizing the victim locations. Intermittent communication where the coordinator only has access to a "common belief" [Chakravorty et al., 2019] updated by broadcasting between agents is also possible.

Object-oriented model-based Reinforcement Learning [Diuk et al., 2008], [Wandzel et al., 2019] is also an attractive alternative to model-free RL like DQN we are currently using. Model-based RL is generally more sample-efficient, more generalizable and transferable than model-free RL when domain knowledge is available. In object-oriented RL, the environment consist of objects of different classes, and the experience with one object can be generalized to that with another object of the same class. For example, an object can be a wall in our case. The agent might learn that it cannot pass through a wall in Room 1. This experience can also be referenced when the agent encounters another wall in Room 2 under the model-based framework. In model-free, on the other hand, the agent does not have the concept of a wall. What it learns is that I cannot perform a certain move at a specific location in Room 1. When encountering the object of the same class in Room 2, it has to re-learn the fact that it cannot move through wall again.

Lastly, we would like to consider agent advising [Torrey and Taylor, 2013] and hierarchical active imitation learning [Le et al., 2018]. Under these frameworks, a teacher, which is a trained high-performant team of agents, would observe a student to provide feedback in order to accelerate the student's learning. Under the agent advising paradigm, the teacher predicts the student's next action, and intervene whenever it believes that the student is going to make a mistake. Under the hierarchical imitation approach, the teacher observes the student's mistake after the fact and provides feedback, still in an online manner.

VII. ACKNOWLEDGEMENT

The author (Long Le) would like to thank Rachel Burcin, John Dolan, Jennie Piotrzkowski and all members of the RISS cohort for making RISS a wonderful experience. The author would like to thank their mentors Katia Sycara and Dana Hughes. The author also benefited from the insights in multi-agent and hierarchical RL from Tejus Gupta and Sophie Guo.

REFERENCES

[Arjona-Medina et al., 2018] Arjona-Medina, J. A., Gillhofer, M., Widrich, M., Unterthiner, T., and Hochreiter, S. (2018). RUDDER: return decomposition for delayed rewards. *CoRR*, abs/1806.07857.

[Bacon et al., 2016] Bacon, P.-L., Harb, J., and Precup, D. (2016). The option-critic architecture.

[Barry et al., 2011] Barry, J. L., Kaelbling, L., and Lozano-Perez, T. (2011). Deth*: Approximate hierarchical solution of large markov decision processes. In *IJCAI*.

[Bengio et al., 2009] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA. Association for Computing Machinery.

[Boutilier et al., 2000] Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1):49–107.

[Chakravorty et al., 2019] Chakravorty, J., Ward, P. N., Roy, J., Chevalier-Boisvert, M., Basu, S., Lupu, A., and Precup, D. (2019). Option-critic in cooperative multi-agent systems. *CoRR*, abs/1911.12825.

[Dayan and Hinton, 1992] Dayan, P. and Hinton, G. E. (1992). Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, page 271–278, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Diuk et al., 2008] Diuk, C., Cohen, A., and Littman, M. L. (2008). An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 240–247, New York, NY, USA. Association for Computing Machinery.

[Gammelli et al., 2021] Gammelli, D., Yang, K., Harrison, J., Rodrigues, F., Pereira, F. C., and Pavone, M. (2021). Graph neural network reinforcement learning for autonomous mobility-on-demand systems.

[Kiran et al., 2021] Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., and Pérez, P. (2021). Deep reinforcement learning for autonomous driving: A survey.

[Le et al., 2018] Le, H. M., Jiang, N., Agarwal, A., Dudík, M., Yue, Y., and III, H. D. (2018). Hierarchical imitation and reinforcement learning. *CoRR*, abs/1803.00590.

[Littman, 1994] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann.

[Lowe et al., 2017] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

[Oroojlooyjadid and Hajinezhad, 2019] Oroojlooyjadid, A. and Hajinezhad, D. (2019). A review of cooperative multi-agent deep reinforcement learning. *CoRR*, abs/1908.03963.

[Pathak et al., 2017] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. *CoRR*, abs/1705.05363.

[Riemer et al., 2019] Riemer, M., Liu, M., and Tesauro, G. (2019). Learning abstract options.

[Roderick et al., 2017] Roderick, M., MacGlashan, J., and Tellex, S. (2017). Implementing the deep q-network.

[Shang et al., 2019] Shang, W., Trott, A., Zheng, S., Xiong, C., and Socher, R. (2019). Learning world graphs to accelerate hierarchical reinforcement learning. *CoRR*, abs/1907.00664.

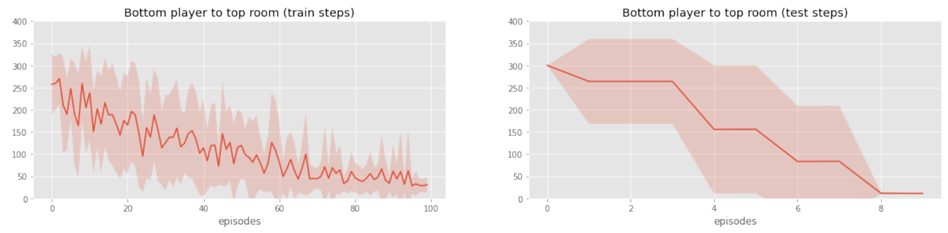
[Steccanella et al., 2021] Steccanella, L., Totaro, S., and Jonsson, A. (2021). Hierarchical representation learning for markov decision processes.

[Sutton et al., 1999] Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1–2):181–211.

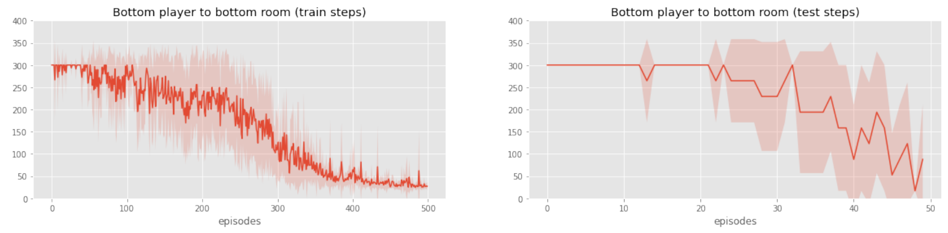
[Torrey and Taylor, 2013] Torrey, L. and Taylor, M. (2013). Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '13, page 1053–1060, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Vezhnevets et al., 2017] Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. *CoRR*, abs/1703.01161.

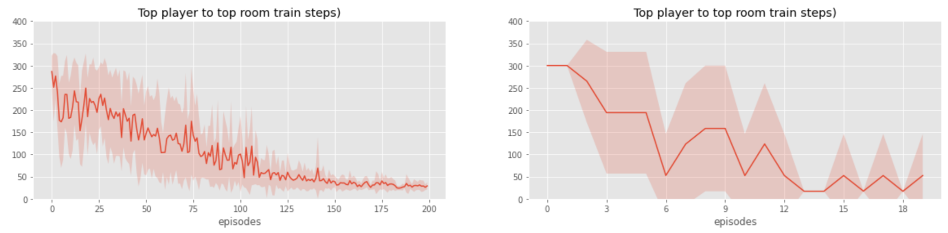
[Wandzel et al., 2019] Wandzel, A., Oh, Y., Fishman, M., Kumar, N., Wong, L. L., and Tellex, S. (2019). Multi-object search using object-oriented pomdps. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7194–7200.



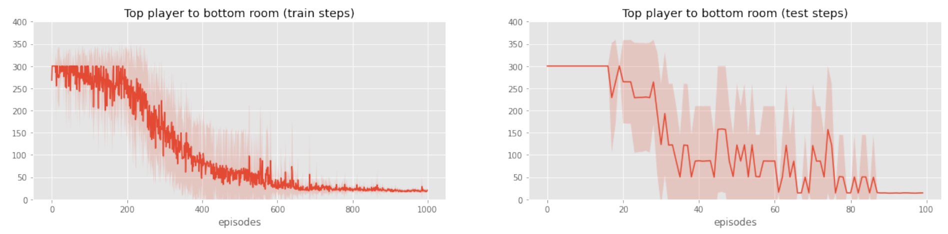
(a) 1 test episode every 10 train episodes



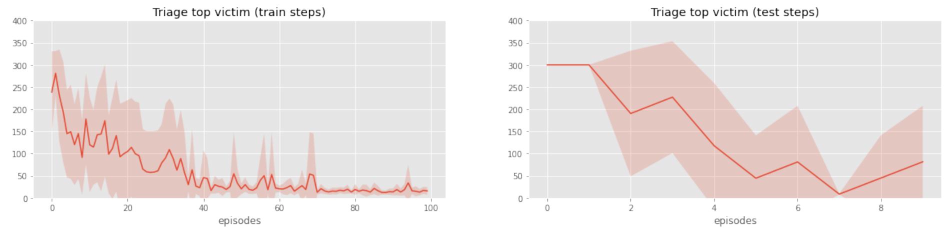
(b) 1 test episode every 10 train episode



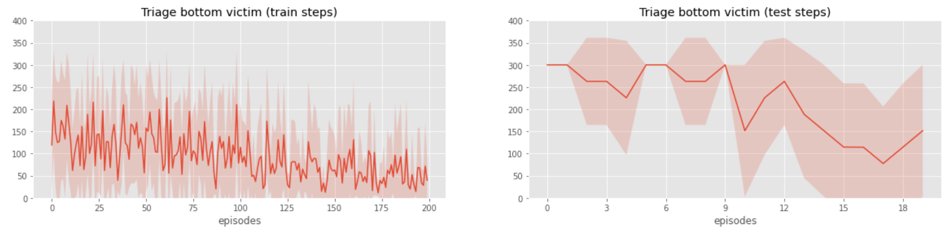
(c) 1 test episode every 10 train episode



(d) 1 test episode every 10 train episode

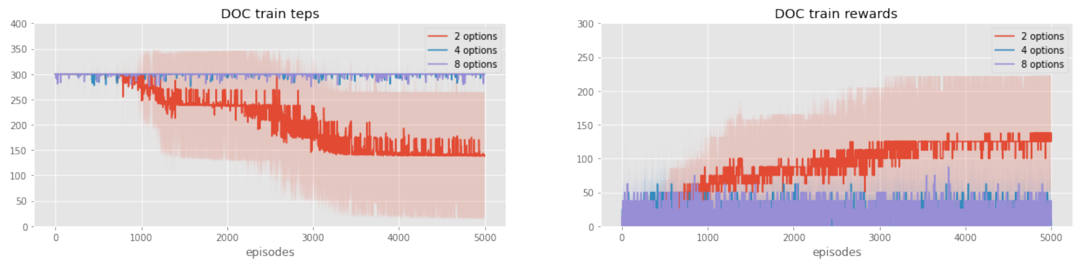


(e) 1 test episode every 10 train episode

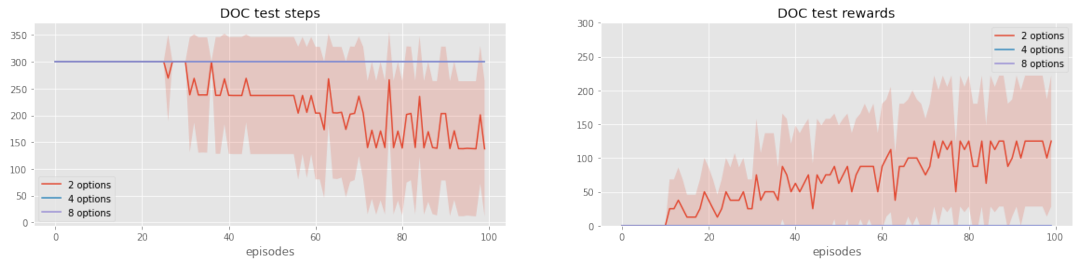


(f) 1 test episode every 10 train episode. For this sub-policy to succeed consistently across trials, we had to lower γ to 0.9. Since the bottom room is very small, having small γ prevented the agent from wandering to other parts of the environment. The usual $\gamma = 0.99$ still worked for one trial. In practice, it would not be necessary to adjust γ since we would save and use the best model in terms of test performance.

Fig. 4: Training sub-policies.

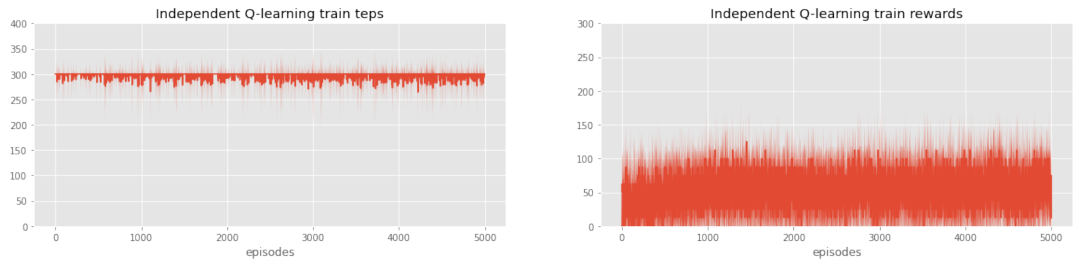


(a) Training performance of DOC

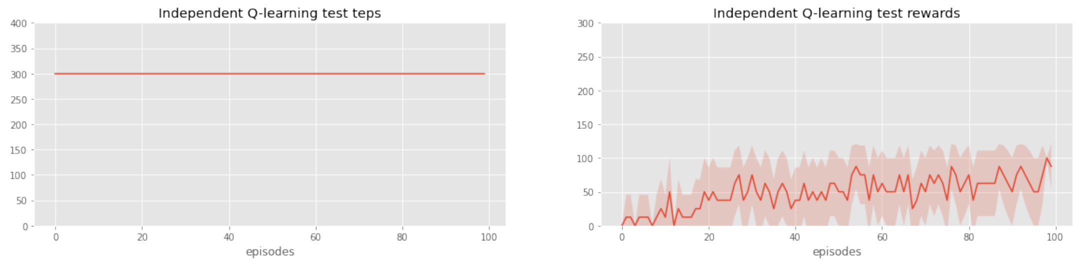


(b) Testing performance of DOC. For every 50 training episodes, we perform one test episode using a greedy-Q policy.

Fig. 5: Performance of the simplified distributed option-critic (DOC) algorithm. The number of options is a hyper-parameter in DOC. We set it to 2,4 and 8 since our approach uses 4 options.



(a) Training performance of Independent Q-learning.



(b) Testing performance of Independent Q-learning. For every 50 training episodes, we perform one test episode using a greedy-Q policy.

Fig. 6: Performance of independent Q-learning.