Efficient Exploration in Reinforcement Learning Leveraging Automated Planning

Yohei Hayamizu*
University of Electro-Communications
hayamizu@cas.lab.uec.ac.jp

Saeid Amiri
SUNY Binghamton
samiri1@binghamton.edu

Kishan Chandan
SUNY Binghamton
kchanda2@binghamton.edu

Keiki Takadama
University of Electro-Communications
keiki@inf.uec.ac.jp

Shiqi Zhang
SUNY Binghamton
zhangs@binghamton.edu

Abstract

While reinforcement learning (RL) and automated planning algorithms share the objective of accomplishing complex tasks, the research of RL and automated planning has been largely separated due to their different computational modalities. Focusing on improving RL agents' learning efficiency for service robots, we develop Guided Dyna-Q (GDQ) to enable RL agents to reason with action knowledge to avoid exploring less-relevant states. GDQ has been evaluated in simulation and using a real mobile robot conducting navigation tasks in an office environment. Compared with competitive baselines, GDQ reduces the effort in exploration while improving the quality of learned policies.

1 Introduction

Recent advances of artificial intelligence have enabled robots to conduct a variety of service and interaction tasks in human-inhabited environments [8, 10, 20]. When a world model (dynamics and rewards) is available, one can use the Markov decision process (MDP) algorithms to compute action policies [16]. In practice, however, world models are frequently unavailable or tend to change over time due to exogenous changes. Reinforcement learning (RL) algorithms have been used to solve sequential decision-making problems, where agents learn action policies toward maximizing long-term utilities from trial-and-error experiences [18].

There are various types of RL algorithms. Among them, model-based RL enables agents to learn a world model while learning an action policy to achieve long-term goals [1, 13, 23]. There are at least two advantages of model-based RL, c.f., model-free RL. First, one can potentially incorporate models acquired in the process of policy learning, such as world dynamics, into domain knowledge from a human expert. Second, model-based RL is goal-independent in terms of the model construction perspective, rendering the learned world model applicable to other tasks. We are particularly interested in model-based RL in this work, due to the characteristics of service robotics domains, such as widely available domain knowledge (e.g., how rooms are connected through doors), and highly diverse service requests (e.g., requests of guiding visitors to different indoor locations).

*Work conducted while visiting SUNY Binghamton.

NeurIPS 2020 3rd Robot Learning Workshop: Grounding Machine Learning Development in the Real World.

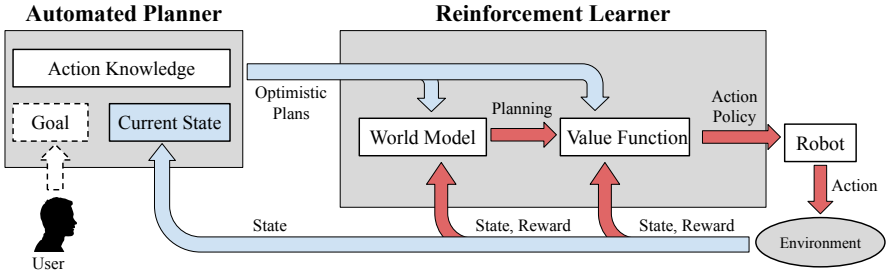


Figure 1: An overview of Guided Dyna-Q (GDQ), where the key is the interplay between an automated planner and a reinforcement learner. The "**red-color loop**" corresponds to the standard control loop of Dyna-Q, where the agent (robot) interacts with the environment to update both its world model, and its Q-value function. GDQ further incorporates an automated planner into the learning process in the "**blue-color loop**," where goal-independent action knowledge (highly sparse, and potentially inaccurate) is used for computing action sequences toward goal achievement. The action sequences are then used for updating both the world model and the Q-value function.

Researchers have developed algorithms, DARLING to integrate model-free RL and automated planning to avoid taking unreasonable actions in exploration. DARLING leverages action preconditions and effects from human knowledge to avoid risky or useless state visits in RL, and has been applied to mobile robot navigation, and grid world domains [11]. Researchers have developed algorithms to use subgoals to guide RL agents. Those subgoals can be learned and represented using non-monotonic logics [6], or action languages [3]. Those algorithms exploited the flexibility of RL approaches and the accuracy of the declarative knowledge from humans. Very recently, a paper surveyed research on leveraging knowledge to improve RL agents' learning performance [22]. The main difference from the above-mentioned methods is that GDQ (ours) uses model-based RL, whereas they used model-free RL methods that are goal-oriented. Our service robotics domain includes potentially many service requests, rendering goal-independent methods more suitable.

In this paper, we focus on addressing the low sample-efficiency challenge of model-based RL algorithms. We develop Guided Dyna-Q (**GDQ**) that consolidates the two classical paradigms of model-based RL and automated planning to help the agent avoid exploring less-relevant states toward more sample-efficient model learning and decision making. GDQ reasons with declarative action knowledge to optimistically simulate action sequences to "accomplish" tasks. The simulated experience is then used to initialize and update Q-values toward efficient policy learning. In particular, we use Answer Set Programming (ASP) to formulate action knowledge [12, 4], and use Dyna-Q for model-based RL [18].

2 Algorithm

In this section, we present Guided Dyna-Q (GDQ) that leverages goal-independent action knowledge for sample-efficient policy learning by enabling the interplay between a model-based reinforcement learner and an automated planner. Although model-based RL and automated planning have been well developed in AI literature, their development has been largely isolated in the past.

We use $\Pi(\mathcal{S}^A, \mathcal{A}^A, M)$ to represent our automated planner, where $\mathcal{S}^A$ and $\mathcal{A}^A$ are the state and action sets respectively. A task to the automated planner is defined as $M = (s_0^A, s_g^A)$, where $s_0^A, s_g^A \in \mathcal{S}^A$ are the initial state and goal states respectively. For the sake of simplicity, the goal is defined as a single state, though it can be a set of states in practice. Given task M, an automated planning system (e.g., Lifschitz, 2002) can use $\Pi(\mathcal{S}^A, \mathcal{A}^A, M)$ to compute a set of plans, $\mathcal{H}$, where $p \in \mathcal{H}$ is in the form of a sequence of actions denoted as follows: $p = \left\langle \left\langle s_0^A, a_0^A \right\rangle, \left\langle s_1^A, a_1^A \right\rangle, \cdots, \left\langle s_i^A, a_i^A \right\rangle, \cdots, \left\langle s_g^A, a_g^A \right\rangle \right\rangle$. Each action sequence leads to state transitions from the initial state s_0^A all the way to the goal state s_g^A.

In order to enable the reinforcement learner to exploit the plans, we introduce a mapping function, $\mathcal{O}$, that constructs the correspondence between the planner's action space $\mathcal{A}^A$ and the learner's action space $\mathcal{A}$, and the correspondence between their state spaces $\mathcal{S}^A$ and $\mathcal{S}$.

Next, we describe how the plans generated by the automated planner are used for *optimistic initialization*, *policy update*, and their integration, i.e., the GDQ algorithm.

Algorithm 1 Optimistic Initialization: OPTINIT

Input: $\mathcal{S}$, $\mathcal{A}$, $\mathcal{S}^A$, $\mathcal{A}^A$, $M = (s_0^A, s_g^A)$
Parameter: γ, α, R_{max}
Output: π

1: **for** $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ **do**
2: $Q(s,a) \leftarrow 0.0$; $\mathcal{R}(s,a) \leftarrow 0.0$; $\mathcal{T}(s,a,s_0) \leftarrow 1.0$;
3: **end for**
4: $\mathcal{H} \leftarrow \Pi(\mathcal{S}^A, \mathcal{A}^A, M)$
5: **for** p in $\mathcal{H}$ **do**
6: **for** $\langle s^A, a^A \rangle$ in p **do**
7: $s, a \leftarrow \mathcal{O}(s^A)$
8: $\mathcal{R}(s,a) \leftarrow R_{max}$
9: **end for**
10: **end for**
11: $\pi \leftarrow$ random policy
12: **while** $\pi \neq \pi'$ **do**
13: $\pi' \leftarrow \pi$
14: **for** $\forall s \in \mathcal{S}$ **do**
15: $Q(s, \pi(s)) \leftarrow \mathcal{R}(s, \pi(s)) +$
16: $\gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') Q(s', \pi(s'))$
17: $\pi(s) \leftarrow \text{argmax}_a \, Q(s,a)$
18: **end for**
19: **end while**
20: **return** π

Algorithm 2 Policy Update (POLICYUP)

Input: $\mathcal{S}$, $\mathcal{A}$, $\mathcal{S}^A$, $\mathcal{A}^A$, $M = (s_0^A, s_g^A)$, π, C
Parameter: γ, α, R_{max}, m, N
Output: $\mathcal{R}, \mathcal{T}, \pi$

1: Collect the current world state s from the world
2: $a \leftarrow \pi(s)$, with ϵ exploration rate
3: Initialize R_{sum} with 0
4: Collect state s' and reward r after taking a
5: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s,a)]$
6: $C(s,a,s') \leftarrow C(s,a,s') + 1$
7: $R_{sum}(s,a) \leftarrow R_{sum}(s,a) + r$
8: **if** $\sum_{s'} C(s,a,s') > m$ **then**
9: $\mathcal{T}(s,a,s') \leftarrow C(s,a,s') / \sum_{s''} C(s,a,s'')$
10: $\mathcal{R}(s,a) \leftarrow R_{sum}(s,a) / \sum_{s''} C(s,a,s'')$
11: **end if**
12: $\mathcal{H} \leftarrow \Pi(\mathcal{S}^A, \mathcal{A}^A, M)$
13: **for** n in $\{1 \cdots N\}$ **do**
14: $p \leftarrow$ randomly selected plan in $\mathcal{H}$
15: $\langle s^A, a^A \rangle \leftarrow$randomly selected transition in p
16: $s, a \leftarrow \mathcal{O}(s^A)$
17: Update $Q(s,a)$ using Bellman equation.
18: **end for**
19: $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}, \pi(s) \leftarrow \text{argmax}_a \, Q(s,a)$
20: **return** $\mathcal{R}, \mathcal{T}, \pi$

Optimistic Initialization: The plans computed by the automated planner are referred to as *optimistic* plans, because real-world domain uncertainty is frequently overlooked in building the planners. The goal of *optimistic initialization* (OPTINIT) is to use the plans computed by the automated planner to initialize Q-values, and prevent the agent from exploring less-relevant states.

Algorithm 1 presents our optimistic initialization process. The input includes the state and action spaces of both the reinforcement learner and the automated planner. M is the provided task. The output is an initial policy π, which is generated by the agent interacting with the world. Lines 1-3 are used for initializing the Q-values, as well as the transition and reward functions. Given task $M = (s_0^A, s_g^A)$, our automated planner computes a set of optimistic plans in Line 4. The two for-loops in Lines 5-10 assign the highest reward, R_{max}, to the reward of all state-action pairs that appear in the plans from our automated planner. Lines 12-19 are used for computing an action policy using policy iteration. Finally, the computed policy is returned in Line 20.

Policy Update and GDQ: Given the initialized Q-value function, the agent is able to compute an initial policy, and use this policy to interact with the real world. Here, we describe how the interaction experience, along with the automated planner, is used to update the Q-value function at runtime.

Algorithm 2 presents the runtime policy update process. Its input is the same as Algorithm 1, except that it also includes an action policy and a counter function C. This policy is provided by Algorithm 1. Parameter m denotes how many times a state-action pair has been selected. Parameter N is used for determining how many state-action pairs are simulated using the automated planner. The output includes a policy, the reward, and transition functions. Lines 1-11 are used for interacting with the real world using the current action policy, π while learning the world model. In Line 12, the automated planner generates a set of plans. The planner generates all of the plans that the paths start from the current state s to the target state s_g. Using the generated plans, the learner randomly select one transition from a randomly-selected plan p, and update the Q-value accordingly (repeated for N times in Lines 13-18). Finally, the learned model ($\mathcal{R}, \mathcal{T}$), and updated policy are returned.

GDQ integrates the two sub-procedures for optimistic initialization (Algorithm 1) and repeatedly conducting runtime policy update (Algorithm 2), which identifies the main contribution of this

research. Informally, Algorithm 1 helps the agent avoid the near-random exploration behaviors through a "warm start" enabled by our automated planner, and Algorithm 2 guides the agent to only try the actions that can potentially lead to the ultimate goal states.

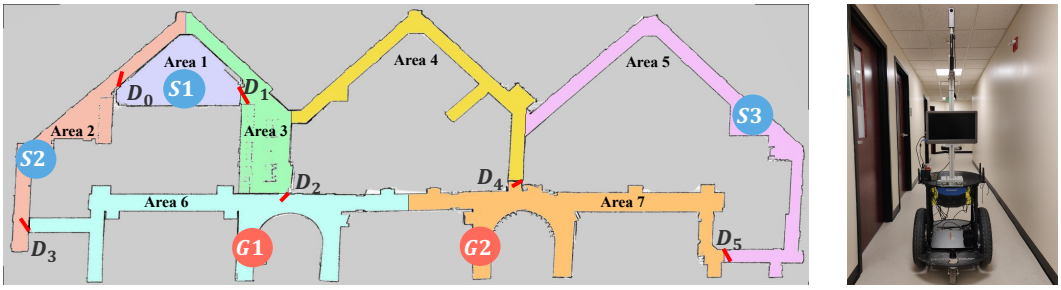
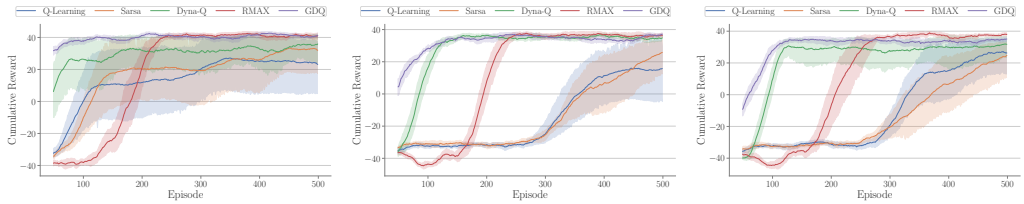


Figure 2: An occupancy-grid map (**Left**) of an indoor office environment with more than 20 rooms, where the map was built using a mobile robot running simultaneous localization and mapping (SLAM) algorithms [15, 19] and each pixel is labeled in color with its semantic meaning; and our Segway-based mobile robot platform (**Right**) that was used for building the map and evaluations of the GDQ algorithm.



(a) Task A: $M_A = (S3, G2)$ (b) Task B: $M_B = (S1, G1)$ (c) Task C: $M_C = (S1, G2)$

Figure 3: Average cumulative reward over ten runs (each run includes a row of 500 episodes), while the robot working on different navigation tasks in simulation. GDQ produced the best performance in all three tasks.

3 Experiment

In this section, we focus on experimentally evaluating the following two hypotheses: I) GDQ performs better than existing RL methods from the literature in cumulative reward (**Hypothesis-I**); and II) GDQ can help the robot avoid visiting "irrelevant" areas (**Hypothesis-II**). An area is deemed **relevant** to a navigation task, if there exists one optimal plan that requires the robot navigating that area. GDQ has been compared with the methods of model-free RL (Q-learning and SARSA), and model-based RL (Dyna-Q and RMAX) [18, 1]. We conduct the experiments in a 2D simulation, which is shown in Figure 2. The domain consists of 7 areas where include 3 states and 6 doors. We assume that $D0$, $D2$, and $D5$ are difficult doors, and $D1$, $D3$, and $D4$ are easy doors to go through.

Figures 3 presents the cumulative rewards collected from the robot conducting Tasks A, B, and C. We observe that GDQ performed the best in learning rate in comparison to the four baselines, which supports Hypothesis-I. Looking into Task-B (Figure 3b), there are the following valid routes that can lead to the goal position while producing different costs and success rates: $[1 \rightarrow 2 \rightarrow 6]$, $[1 \rightarrow 3 \rightarrow 6]$, $[1 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 6]$, $[1 \rightarrow 3 \rightarrow 2 \rightarrow 6]$, $[1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 6]$, where each number corresponds to the index of an area. The shortest routes are $[1 \rightarrow 2 \rightarrow 6]$, and $[1 \rightarrow 3 \rightarrow 6]$. However, these two routes have doors of $D0$ and $D2$ on the way. In comparison, the route of $[1 \rightarrow 3 \rightarrow 2 \rightarrow 6]$ provides the best trade-off between traveling distance and door difficulty, and is the best solution. GDQ enabled the robot to converge to this solution earlier than all other baseline methods.

An additional experiment in support of Hypothesis II, as well as an experiment involving a real mobile platform, are described in the supplementary material.

4 Conclusions

In this paper, we develop Guided Dyna-Q (GDQ) for bridging the gap between model-based RL, and automated planning. The goal is to help the agent (robot) avoid exploring less-relevant states toward

speeding up the learning process. GDQ has been demonstrated and evaluated both in simulation and using a real robot conducting navigation tasks in an indoor office environment. From the experimental results, we see that, using the widely available action knowledge, GDQ performed significantly better than competitive baseline methods, demonstrating the best performance in learning efficiency.

References

[1] Brafman, R. I. and M. Tennenholtz (2002). R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research 3*(Oct), 213–231.

[2] Brewka, G., T. Eiter, and M. Truszczyński (2011). Answer set programming at a glance. *Communications of the ACM 54*(12), 92–103.

[3] Efthymiadis, K. and D. Kudenko (2013). Using plan-based reward shaping to learn strategies in starcraft: Broodwar. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8. IEEE.

[4] Erdem, E., M. Gelfond, and N. Leone (2016). Applications of answer set programming. *AI Magazine 37*(3), 53–68.

[5] Fikes, R. E. and N. J. Nilsson (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence 2*(3-4), 189–208.

[6] Furelos-Blanco, D., M. Law, A. Russo, K. Broda, and A. Jonsson (2020). Induction of subgoal automata for reinforcement learning. In *AAAI*, pp. 3890–3897.

[7] Gelfond, M. and Y. Kahl (2014). *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press.

[8] Hawes, N., C. Burbridge, F. Jovan, L. Kunze, B. Lacerda, L. Mudrova, J. Young, J. Wyatt, D. Hebesberger, T. Kortner, et al. (2017). The strands project: Long-term autonomy in everyday environments. *IEEE Robotics & Automation Magazine 24*(3), 146–156.

[9] Jiang, Y., S. Zhang, P. Khandelwal, and P. Stone (2019). Task planning in robotics: an empirical comparison of pddl- and asp-based systems. *Frontiers of Information Technology & Electronic Engineering 20*(3), 363–373.

[10] Khandelwal, P., S. Zhang, J. Sinapov, M. Leonetti, J. Thomason, F. Yang, I. Gori, M. Svetlik, et al. (2017). Bwibots: A platform for bridging the gap between ai and human–robot interaction research. *The International Journal of Robotics Research 36*(5-7), 635–659.

[11] Leonetti, M., L. Iocchi, and P. Stone (2016). A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artif. Intell. 241*, 103–130.

[12] Lifschitz, V. (2002). Answer set programming and plan generation. *Artificial Intelligence 138*(1-2), 39–54.

[13] Mann, T. A. and Y. Choe (2011). Scaling up reinforcement learning through targeted exploration. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

[14] McDermott, D., M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins (1998). Pddl-the planning domain definition language.

[15] Montemerlo, M., S. Thrun, D. Koller, and B. Wegbreit (2002). Fastslam: a factored solution to the simultaneous localization and mapping problem. In *Eighteenth national conference on Artificial intelligence*, pp. 593–598.

[16] Puterman, M. L. (2014). *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

[17] Sutton, R. S. (1991, July). Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull. 2*(4), 160–163.

[18] Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction*.

[19] Thrun, S., W. Burgard, and D. Fox (2005). *Probabilistic Robotics*. MIT Press.

[20] Veloso, M. M. (2018, May). The increasingly fascinating opportunity for human-robot-ai interaction: The cobot mobile service robots. *ACM Transactions on Human-Robot Interaction 7*(1).

[21] Yang, F., P. Khandelwal, M. Leonetti, and P. H. Stone (2014). Planning in answer set programming while learning action costs for mobile robots. In *2014 AAAI Spring Symposium Series*.

[22] Zhang, S. and M. Sridharan (2020). A survey of knowledge-based sequential decision making under uncertainty. *arXiv preprint arXiv:2008.08548*.

[23] Łukasz Kaiser, M. Babaeizadeh, P. Miłos, B. Osiński, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski (2020). Model based reinforcement learning for atari. In *International Conference on Learning Representations*.

5 Additional Experiments in 2D simulations

Exploration Aiming to evaluate Hypothesis-II on exploration, we manually provided the ground truth relevance information, where we introduce function IRR that maps a task to a set of irrelevant areas:

$$\textit{irrelevant areas} \leftarrow IRR(\textit{task})$$

Back to our testing domain, the irrelevant areas to each task are: $\{1, 2, 3, 6\} \leftarrow IRR(A)$, $\{4, 5, 7\} \leftarrow IRR(B)$, $\{2, 5\} \leftarrow IRR(C)$.

Table 1 shows the results in evaluating the performances in exploration. The blue color highlights the method that produced the least visits, and we say *the robot successfully avoids the area using this method.* Consider the first five rows that correspond to Task-B. We see that GDQ enabled the robot to visit Area-4 for as few as only 106 times, which is much lower than the numbers of visits required by the other methods (say Sarsa requires 4936.2 visits), while still produced the best performance in policy quality. This observation is consistent with our prior knowledge that Areas 4, 5, and 7 are less-relevant to Task-B. In these tasks, Task-A, Task-B, and Task-C, the robot successfully avoided the irrelevant areas (see the highlighted areas in blue color and the listed irrelevant areas in column *IRR*), supporting Hypothesis-II.

Table 1: This table shows how many times the robot visited each area using five different methods (including GDQ) in conducting the two different tasks. The goal is to show that GDQ is able to help the robot avoid visiting areas that are less-relevant to the given task. Italic and blue color indicates the fewest visits among the five methods.

	Task-A (IRR: 1, 2, 3, 6)				
Area	*GDQ*	*Dyna-Q*	*RMAX*	*Q-Learning*	*Sarsa*
Area 1	*0.0* (*0.0*)	451.5 (2.2)	2128.2 (7.6)	2631.2 (7.0)	1602.3 (4.9)
Area 2	*41.4* (*0.2*)	590.5 (2.9)	3947.5 (14.1)	4992.7 (13.3)	3171.9 (9.6)
Area 3	*109.8* (*0.6*)	1093.7 (5.3)	3824.2 (13.6)	7789.9 (20.7)	5727.8 (17.4)
Area 4	9701.3 (51.0)	8968.3 (43.5)	8295.3 (29.6)	11350.8 (30.2)	11920.0 (36.2)
Area 5	3453.4 (18.2)	4324.8 (21.0)	4143.6 (14.8)	6900.8 (18.4)	6373.8 (19.3)
Area 6	*7.9* (*0.0*)	73.0 (0.4)	1609.9 (5.7)	853.0 (2.3)	132.3 (0.4)
Area 7	5710.7 (30.0)	5123.8 (24.8)	4097.1 (14.6)	3072.5 (8.2)	4025.5 (12.2)
	Task-B (IRR: 4, 5, 7)				
Area	*GDQ*	*Dyna-Q*	*RMAX*	*Q-Learning*	*Sarsa*
Area 1	10787.6 (32.8)	11975.7 (33.9)	11815.9 (31.8)	25247.5 (46.8)	25517.3 (46.8)
Area 2	9986.7 (30.3)	9363.2 (26.5)	8326.5 (22.4)	8253.4 (15.3)	8143.9 (14.9)
Area 3	6515.3 (19.8)	7056.9 (20.0)	7293.5 (19.6)	11748.2 (21.8)	11975.3 (22.0)
Area 4	*106.0* (*0.3*)	1243.0 (3.5)	3021.1 (8.1)	4895.2 (9.0)	4936.2 (9.0)
Area 5	*21.0* (*0.1*)	367.2 (1.0)	1281.3 (3.4)	1937.1 (3.6)	2084.1 (3.9)
Area 6	5522.2 (16.8)	4960.4 (14.0)	4000.8 (10.8)	1634.8 (3.0)	1569.7 (2.9)
Area 7	*3.4* (*0.0*)	382.5 (1.0)	1475.4 (4.0)	260.6 (0.5)	329.0 (0.6)
	Task-C (IRR: 2, 5)				
Area	*GDQ*	*Dyna-Q*	*RMAX*	*Q-Learning*	*Sarsa*
Area 1	11712.7 (33.6)	12212.3 (33.3)	11835.3 (32.0)	24730.6 (47.2)	26233.5 (47.2)
Area 2	*2715.9* (*7.8*)	4402.7 (12.0)	3476.9 (9.4)	5763.0 (11.0)	6822.6 (12.3)
Area 3	6812.9 (19.5)	7189.3 (19.6)	7308.0 (19.8)	11106.4 (21.2)	12068.7 (21.7)
Area 4	7202.2 (20.6)	5853.2 (16.0)	7876.1 (21.3)	6626.1 (12.7)	6177.7 (11.1)
Area 5	*269.1* (*0.8*)	895.8 (2.4)	1227.7 (3.3)	1677.4 (3.2)	2127.7 (3.9)
Area 6	1529.3 (4.4)	2143.9 (5.9)	1184.8 (3.2)	704.1 (1.3)	982.3 (1.8)
Area 7	4651.3 (13.3)	3943.7 (10.8)	4096.3 (11.1)	1769.1 (3.4)	1155.1 (2.1)

6 Real Robot Experiments

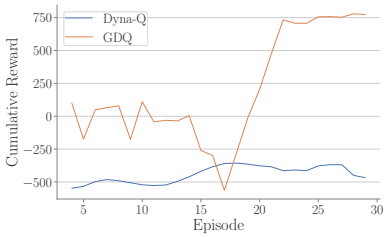
We have conducted experiments using a Segway-based mobile robot platform. In the real world, the robot has to ask people to help open doors, where the action cost and success rate are noisy and out of our control. We intentionally forbade the robot from entering Area-5, because it is a long corridor, and navigating through that area takes a very long time. The following parameters are used in the real-robot experiment: $R_{max} = 1000$, $\alpha = 0.5$, $\gamma = 0.95$, and $\epsilon = 0.1$. The robot's task is M_X = (S2, G1), referred to as Task-X.

Different from simulation experiments, we use the time to measure the cost of navigation and door-opening actions (instead of a predefined fixed value). A maximum of 10 steps is allowed, i.e., if the robot cannot complete Task-X in 10 steps, the corresponding trial will be deemed unsuccessful. We have conducted a total of 30 trails using the real robot.

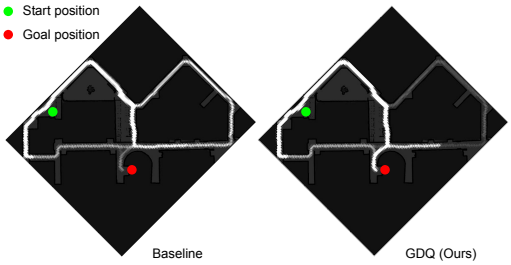
Each trial took up to 30 minutes to complete. The Segway-base robot runs out of battery in about five hours, and the experiments were conducted on three consecutive days (5 hours a day, and 15 hours in total). Due to the long time required for each trial (especially in the early learning phase), we only compared GDQ with one baseline of Dyna-Q.

Figure 4a reports the results collected from the real-robot experiment. Looking at the very left of the two curves, the "jump start" of GDQ shows that *ptimistic initialization* helped the robot successfully avoid the "random" exploration behaviors in the early phase. Once the robot started interacting with the real world, we can see the cumulative reward of GDQ is consistently higher than Dyna-Q, except for only the $17th$ episode. After that, GDQ soon found the optimal solution. In comparison, Dyna-Q could not find a meaningful solution within a total of 30 episodes.

Figure 4b visualizes the frequency of our robot visiting different locations, where a light gray color represents a lower frequency of visits. We see that GDQ enabled the robot to focus more on the left side of the subarea, whereas, using the baseline approach, the robot traversed the right subarea (irrelevant) more. We have generated a video for the demonstration of GDQ's performance on a real robot – see the supplementary materials*.



(a) Task $M_X = (S2, G1)$ on a real robot. GDQ enabled the robot to find the optimal path in 22 trials, while Dyna-Q could not find a meaningful solution in 30 trials.



(b) Heatmaps of a subarea of our office domain for visualizing where the robot visited using the Dyna-Q baseline (Left) and GDQ (Right).

7 Background

In this section, we briefly summarize the concepts of reinforcement learning, and automated planning.

7.1 Reinforcement Learning

Within the MDP context, when world models (reward functions, transition functions, or both) are not known, the agent must learn action policies from trial-and-error experiences. For instance, Q-learning is a model-free RL algorithm, and its Q-value function can be updated as below.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

where r is the immediate reward after taking action a in state s. This update procedure enables the agent to incrementally learn from every single (s, a, s', r) sample.

Model-based RL algorithms, on the other hand, learn the world model, including $\mathcal{R}(s,a)$ and $\mathcal{T}(s,a,s')$, and then use planning algorithms to compute the action policy. Dyna-Q [17] is a model-based RL framework, and includes the two primary components of model-free RL (Q-learning) and probabilistic planning (e.g., value iteration). The real-world interaction experience is used for two purposes in Dyna-Q: world model learning, and action policy learning. Besides, Dyna-Q is able to generate extra (simulation) experience through interacting with the learned world model, which further speeds up the policy learning process. We use declarative action knowledge to prevent the Dyna-Q agent from exploring less-relevant states.

*https://www.dropbox.com/s/btu1ghi10e4mhov/NeurIPSWS2020_GDQ.mp4?dl=0

7.2 Automated Planning

Automated planning methods aim at computing a sequence of actions toward accomplishing complex tasks. One has to declaratively encode action knowledge into an automated planner, including actions' preconditions and effects. Since the development of STRIPS [5], many action languages have been developed for formally representing action knowledge. The following shows an example of using STRIPS to formulate action `stack`, where preconditions include the robot arm holding object `X` and object `Y` being clear. Executing this action causes the hand to be empty, and object `Y` not clear anymore.

```
operator(stack(X,Y),
         Precond [holding(X),clear(Y)],
         Add [handempty,on(X,Y),clear(X)],
         Delete [holding(X),clear(Y)])
```

There are a number of action languages that can be used for encoding action knowledge. PDDL was initially developed for the International Planning Competition (IPC), and has been maintained by the IPC community [14]. Answer set programming (ASP) is a general-purpose knowledge representation and reasoning paradigm [7, 2], and supports automated planning [12]. We use ASP in this research mainly because it performs better in knowledge-intensive planning domains [9], c.f., PDDL. For instance, in navigation tasks, the robot needs to reason about potentially many rooms and their connections and ASP allows us to implement default reasoning while PDDL does not.

The action knowledge we use in this paper is simple and publicly available [21, 9], so we do not discuss knowledge acquisition in this paper.

8 Acknowledgement

A portion of this work has taken place in the Autonomous Intelligent Robotics (AIR) Group at SUNY Binghamton. AIR research is supported in part by grants from the National Science Foundation (IIS-1925044 and REU Supplement), Ford Motor Company (two URP Awards), OPPO (Faculty Research Award), and SUNY Research Foundation. The authors thank collaborators on related research that fed into the development of the ideas described in this paper.