DinerDash Gym: A Benchmark for Policy Learning in High-Dimensional Action Space

Siwei Chen, Xiao Ma, David Hsu
National University of Singapore
{siwei-15, xiao-ma, dyhsu}@comp.nus.edu.sg

***Abstract*—It has been arduous to assess the progress of a policy learning algorithm in the domain of hierarchical task with high dimensional action space due to the lack of a commonly accepted benchmark. In this work, we propose a new light-weight benchmark task called Diner Dash for evaluating the performance in a complicated task with high dimensional action space. In contrast to the traditional Atari games that only have a flat structure of goals and very few actions, the proposed benchmark task has a hierarchical task structure and size of 57 for the action space and hence can facilitate the development of policy learning in complicated tasks. On top of that, we introduce Decomposed Policy Graph Modelling (DPGM), an algorithm that combines both graph modelling and deep learning to allow explicit domain knowledge embedding and achieves significant improvement comparing to the baseline. In the experiments, we have shown the effectiveness of the domain knowledge injection via a specially designed imitation algorithm as well as results of other popular learning algorithms.**

I. Introduction

Imitation learning (IL) and reinforcement learning (RL) have shown remarkable success in completing complex and challenging tasks [8, 18, 15, 2, 13, 23]. The most representative algorithms such as Behaviour Cloning (BC) [17] and Generative Adversarial Imitation Learning (GAIL) is widely studied and used in many works [10, 9, 12]. Real-world decision-making problems often have a high-dimensional and highly structured action space, e.g., traffic light control with thousands of actions [25]. However, both imitation learning and reinforcement learning algorithms do not always generalize to tasks with high dimensional state space and high dimensional action space. For example, the maximum entropy inverse reinforcement learning (MaxEnt IRL) [26], which matches the expert trajectories by shaping a reward function, works well in a small state space task but not a high dimensional state space task.

Efficient simulation for evaluating IL / RL algorithms in high-dimensional action space is, however, hard to acquire. Most of the existing benchmarks have a simple action space, e.g., Atari games with up to 6 actions and Mujoco [7] with less than 20 dimensions of actions. Recent works proposed solutions to specific tasks with high-dimensional action spaces, e.g., real-world YouTube recommendation system [4] and StartCraft Learning Environment [22]. However, the simulation cost of these tasks is very high, and the training difficulty is not only about the action space itself but also with the complexity of the games.

In this work, we proposed a benchmark task called Diner Dash for high dimensional action space with hierarchical structure in the Open-AI Gym environment [3]. The benchmark task is light-weight and fast to run in a gym environment. Existing training environments like StarCraft is too complicated and is resource-consuming to train. Other existing benchmarks like Atari games are light-weight, however, cannot provide a high dimensional action space environment. DinerDash is hence a suitable and balanced benchmark task for evaluating policy in a task with high dimensional action space. Furthermore, we have implemented a simple imitation learning algorithm DPGM to study how existing domain knowledge can help with the performance. DPGM leverages on the decomposition and models each decomposed task as a factor graph. Following this approach, DPGM significantly beat the naively implemented baselines such as behaviour cloning (BC). Finally, We compare with popular imitation learning algorithms such as Behaviour cloning and GAIL as well as popular RL methods such as PPO [20] with their results reported in the experiments.

Fig. 1: Diner Dash Simulator. The task is called Diner Dash, where the player runs a restaurant and serves the customer as many as possible. The difficulty of this task is overwhelming customers and a long planning horizon. Each group of customers takes multiple steps to serve and have to serve multiple customers at the same time. The agent has to make the right choice for every step; otherwise, the customer will run away and brings in large negative rewards.

II. Related Work

There are many benchmark works available, and some of them target on specific problems. [6] introduced Acrobot and [14] brought out mountain car model. Modern benchmark works including RL-Toolbox, Beliefbox, RLLib and RLLab [7, 16, 11, 1] provide a good training platform with tasks ranging from Cart-Pole Balancing, Mountain Car, Atari games to locomotion tasks and partially observable tasks.

Other benchmark works have also focused on the high dimensional action space. For example, [19] uses a 16-DOF humanoid robot, [24] introduces 17-DOF humanoid robot task for crawling and [5] introduces a 20-link pole balancing task. Other tasks such the RoboCup Keepaway [21] introduces a multi-agent task which has high dimensional actions. However, most of the benchmark tasks above have a relatively small action space comparing to the real problem, which has thousands of actions such as traffic light control. The proposed benchmark task Diner Dash works on this to provide a more realistic training environment with high dimensional action space to quantify the process of a learning algorithm.

III. Task Description

The game in Figure 1 called Diner Dash is proposed as the benchmark task. The task has high dimensional state space, 40 dimensions to be exact, and 57 actions for the action space. The player is running a restaurant by controlling a waitress to serve customers as many as possible. As shown in the picture, the restaurant has six tables with different sizes and up to 7 waiting groups, on the left side and with different sizes, to be served. For each group of people, the player needs to allocate a table for them, collect orders, submit orders, pick up food, serve food, collect bills, clean table and finally return the dish to the dish collection point. There is a happiness value of each group of people, represented in the form of hearts, and the happiness value will decrease if they wait too long. Once the happiness value reaches zero, the customer runs away, and the player loses one life. There is a maximum of 5 lives of each player, and the game ends when the player loses five groups of customers.

Given all the properties above of the task, Diner Dash is a challenging task, with high dimensional action space, high dimensional state space, infinite horizon, hierarchical structure and requires sub-tasks to be completed in parallel. Such a tough task gives a better training environment which is closer to the real-world problems, for example, traffic light control, comparing to typical RL benchmark tasks such as Atari games.

IV. Domain Knowledge Embedding

In this section, we proposed a simple imitation learning algorithm that combines both graph modelling and deep learning to allow explicit domain knowledge injection. The purpose of this algorithm is to provide a baseline for people who want to study how existing domain knowledge can affect the final performance of a policy.

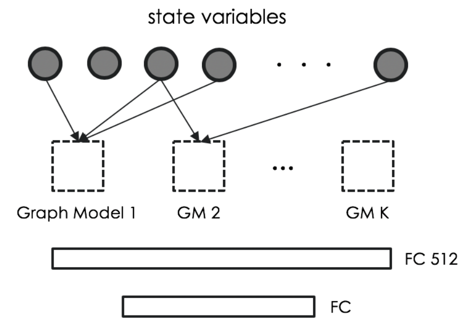

Fig. 2: Decomposed Policy Graph Modeling (DPGM) Structure. The input is the current state vector, and the output is a vector of action values. Instead of learning the entire policy based on the complex state space, we can decompose the problem into K sub-problems to learn a simple policy for each particular action. Each small policy is modelled by a graph model in the form of the factor graph, which allows the expert to inject prior knowledge easily. After the graph modelling, we apply another two fully connected layers to re-weight the importance of each action with supervised learning.

A. Overview

In a closer view, our approach first decompose the entire state space into K small tasks and model each task by a factor graph, see figure 2. More formally, the original task can be formulated with a standard Markov decision process (MDP) $M = \{S, A, T, R, \gamma\}$ with state space S, action space A, state transition probability distribution T, reward function R, and discount factor γ. After decomposition, it becomes $M = \{\{S_1, S_2, ..., S_K\}, A, T, R, \gamma\}$ with $S_k \subseteq S$.

Our goal is to train the K graph models $g_{\theta_k}(s_k)$ to imitate the expert on each action and output the final re-weighted result by the fully connected layers f_α, see equation 1.

$$\alpha^*, \theta^* = arg \min_{\alpha,\theta} \sum_{(s,a)\in D} L(a,\ \ f_\alpha(g_{\theta_1}(s_1), ..., g_{\theta_k}(s_k))) \quad (1)$$

B. Task Decomposition Along Action Space

Figure 3 is an example of task decomposition along with each action, wherein this task, the agent needs to allocate a table for a group of people. Even though the full state space has 40 variables, the key factors the agent needs to pay attention are only 4 variables: the table status, the table size, the group size and the group happiness. Other irrelevant state variables are discarded, leaving a much lesser state space. However, the task decomposition introduces an underlining assumption that the new sub-state space should be sufficient for decision making.

Assumption 1 The decomposed state space S_k of action is sufficient for decision making policy $\pi_k(s_k)$, where π_k only has one action and learns that action from the expert demonstration.

Following this approach, the original task now can be decomposed into several small tasks along with each action. Each

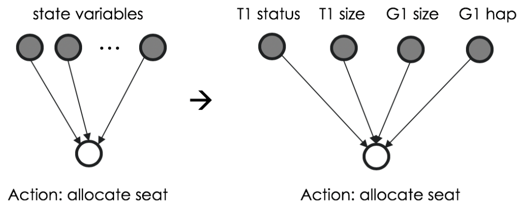

Fig. 3: Task Decomposition Along with Action Space. Although, a task may have a high dimensional state space, the actual related state for each action can be very small. The above shows an example, in a restaurant daily operation, the action to allocate a seat for a waiting customer group to a particular table may only need to consider whether the table is ready and whether the table size fits the customers. Other table status are not relevant to this particular action.

small task has only one action output. Such decomposition process reduces the state space heavily from $O(M^{40})$ to about $O(M^4)$, where M is the average number of state values for each dimension.

C. Decomposed Policy Graph Modeling (DPGM)

After decomposition, it is necessary to reason out why the expert chooses the action by modelling the action with a factor graph.

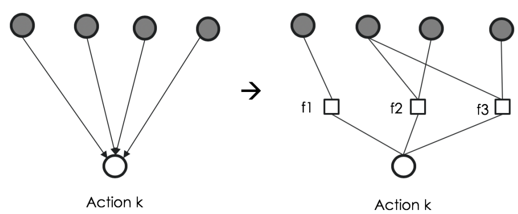

Fig. 4: Policy Graph Modeling as a Factor Graph.

Figure 4 is an example of modelling the correlations between the input variable and the action output. In this example, the action output is modelled by three factors shown on the right side. The original correlations, shown on the left side in Figure 4, is a direct graph with a joint distribution involving all the input variables mingle with each other. While the right side is a factor graph with groups of input variables factorized. By modelling with the factor graph, we can have lesser parameters to train and hence is more data-efficient. The structure of the factor graph is provided by the expert who has the domain knowledge, understanding the correlations between input variables and the probability to choose this action. If the internal correlation groups are not clear to the expert, we can always use one big factor or a neural network to model the distribution.

Moreover, this factor graph structure allows the expert to group input states as factors and allows the expert to specify some known correlations. For example, the factor f_1 can be modelled as a correlation between table happiness and the the action whether to serve that table as the expert knows the lower the happiness of the table the higher priority to serve that table.

The joint distribution of the factor graph is shown below:

$$p(\boldsymbol{x}, y) = \frac{1}{Z} \prod_i \varphi_i(\boldsymbol{x_i}, y) \tag{2}$$
$$Z = \sum_{\boldsymbol{x},y} \prod_i \varphi_i(\boldsymbol{x_i}, y)$$

φ_i is the ith potential function corresponding to factor f_i in the factor graph. S_k is the mapped sub-state space and $\boldsymbol{x_i} \subseteq S_k$ are the input variables required by factor φ_i. Therefore, the joint distribution is the product of all the factors and then normalized by the partition function Z. Given the joint distribution, the goal of the graph model is to model how the expert selects this action. The inference equation is shown below.

$$\begin{aligned} p_k(\boldsymbol{x}) &\equiv p(y=1|\overline{\boldsymbol{x}}) \\ &= \frac{p(\overline{\boldsymbol{x}}, y=1)}{\sum_y p(\overline{\boldsymbol{x}}, y)} \\ &= \frac{\frac{1}{Z}\prod_i \varphi_i(\overline{\boldsymbol{x_i}}, y=1)}{\sum_y \frac{1}{Z}\prod_i \varphi_i(\overline{\boldsymbol{x_i}}, y)} \\ &= \frac{\prod_i \varphi_i(\overline{\boldsymbol{x_i}}, y=1)}{\prod_i \varphi_i(\overline{\boldsymbol{x_i}}, y=0) + \prod_i \varphi_i(\overline{\boldsymbol{x_i}}, y=1)} \end{aligned}$$

$\overline{\boldsymbol{x}}$ means observed input variables. The agent selects actions based on the soft-max of each state-action values. Since the scale of each action is different, it is hard to choose the action based on the absolute values. Therefore, another layer of fully connected layers will be added after the graph modelling layers to re-weight the importance of each action.

Algorithm 1 Decomposed Policy Function Modeling

Input: Domain Knowledge and full state S

Gather demonstrations $D = \{s_1, a_1 ..., s_t, a_t\}$
Map each state $s_t \to s_{k,t}$ by domain knowledge
Save mapped data as D_k
for each action k **do**
    Factorize and group variables into a factor graph $\mathcal{G}$
    Joint distribution to be $p(\boldsymbol{x}, y) = \frac{1}{Z}\prod_i \varphi_i(\boldsymbol{x_i}, y)$
    Update $\min_\theta L = \sum_{\boldsymbol{x},y \in D_{a_i}} (\, p(y=1|\overline{\boldsymbol{x_t}}) - y_t)^2$
end for
Train the re-weighting layer with the expert action.
Update the fully connect layers f_α

D. Training Method

For each action, with the graph model, we can update the graph parameters by the equation 3. The training data comes from the demonstration and is in the form of $(\overline{\boldsymbol{x_t}}, y_t)$, where $\overline{\boldsymbol{x_t}}$ is the observed sub-state variables and y_t is a binary value whether the expert selects the action. For example, there are actions A and B, sub-state space $\boldsymbol{x_A}$ and $\boldsymbol{x_B}$. If the expert selects A at time step t, then $(\overline{\boldsymbol{x}_{\boldsymbol{A},\boldsymbol{t}}}, 1)$ is the training data for action A and $(\overline{\boldsymbol{x}_{\boldsymbol{B},\boldsymbol{t}}}, 0)$ for action B.

$$\min_{\theta} L = \sum_{t} (\, p(y = 1 | \overline{\boldsymbol{x}_{\boldsymbol{t}}}) - y_t)^2 \tag{3}$$

V. Experiment

A. Collecting Demonstration Data

Collecting demonstration data is a time consuming and even costly process at some times. To address the issue, we used a heuristic policy function together to be the expert for demonstration data collection. The simulator wrapped by Gym Environment is used to collect the demonstration data. In this experiment, a total of 274 trajectories with 163120 state-action pairs, are collected for the expert demonstration.

B. Algorithms

We compare the proposed DPGM with two commonly used imitation learning algorithms on DinerDash, standard behaviour cloning (BC) [17] and one of the SOTA methods, GAIL [10] which is the representative of trajectory matching approaches in IL. We also evaluate the SOTA on-policy reinforcement learning method, PPO [20, 12] and demonstrate the challenges standard RL algorithms would face in DinerDash.

For the standard BC, we used a naive implementation of BC, which has two fully connected layers and one dropout layer in between.

C. DPGM Training Pipeline

The detailed algorithm can be found in Algorithm 1. Firstly, the expert decomposes the task and selects the relevant states regarding each action. This can be done given assumption 1, where the teacher knows well of the task and is able to decide a sub-state space which is sufficient for decision making. Each action is then modelled by a factor graph where the expert can easily inject the prior knowledge.

In this experiment, action 1 to 6 corresponds to moving tables 1 to 6 and action 15 to 57 corresponding to allocating group x to table y are modelled by graph models. The other actions adopt the memorizing approach, which is to memorize the expert's actions. This is because the sub-state space of the other actions is small, and all the choices can be easily memorized. The graph model parameters are learned from the demonstration data based on equation 3.

D. Results

The experiment results can be found in Figure 5. To truly identify the performance of all the agents, the difficulty of Diner Dash is adjusted to a higher level comparing to the original version. Total of 274 expert trajectories from the heuristic policy is collected for demonstration and the performance is the average score based on 100 episodes. In this experiment, we are comparing with standard behaviour cloning, which is a common baseline in imitation learning. The result shows that DPGM can achieve near-optimal performance in a complicated task such as Diner Dash with 40 dimensions of the state space and 57 actions in the action space.

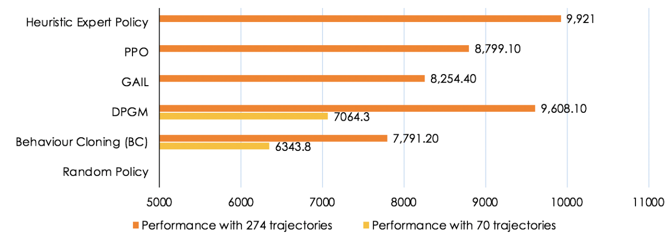

Fig. 5: Final Performance in the Diner Dash Simulator. The top row is the heuristic policy which is also the expert that generates the training data. The second row is the proposed approach, named decomposed policy graph modelling (DPGM). The last row is a random agent and the score of the random agent is -1245.8, which cannot be displayed in the figure.

The behaviour cloning baseline shows us precisely what is the performance without using any existing prior knowledge and purely relies on the demonstration data. Eventually, It fails to converge to a near-optimal performance due to the insufficient data. This also shows that DPGM has a higher data sampling efficiency by decomposition and factor graph modelling.

On top of that, we have anther comparison between BC and DPGM with only 70 trajectories to study the sample efficiency of the proposed method. The performance of DPGM is still higher than BC, and this is reasonable due to all the optimizations to make use of the prior knowledge.

The results of PPO suggests that even the SOTA of RL has some difficulties to converge to the optimal performance after 3×10^7 steps. It is hard for RL policies to discover the correlation that is crucial, sparse and long-delayed in high dimensional action space task like Diner Dash. For example, for every customer lost, the agent will receive a sizeable negative reward. However, the game will end after 5 customers lost, resulting in a sparse signal. On top of that, the result of customer loss is due to the actions of not serving them along the trajectories many steps ago. The sizeable negative penalty reward tells the policy not to select the actions that are selected just now. Nonetheless, the problem is that there are so many other actions and trying and error become more difficult than tasks with small action space. Such sparse and high dimensional action space problem poses a challenge to the RL policy, making the RL policies harder to converge.

VI. Conclusion and Future Work

We introduce DinerDash, a challenging light-weight benchmark for IL / RL algorithms with high-dimensional action space. DinerDash poses significant challenges to state-of-the-art IL / RL algorithms. We also introduce DPGM that decomposes policy space using factor graph with expert domain knowledge and outperforms all baselines.

However, DPGM relies on expert domain knowledge and is hard to generalize to less-structured tasks. We leave it for future study.

REFERENCES

[1] Saminda Abeyruwan. Rllib: Lightweight standard and on/off policy reinforcement learning library (c++), 2013. *URL http://web. cs. miami. edu/home/saminda/rilib. html*.

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[4] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[5] Christoph Dann, Gerhard Neumann, Jan Peters, et al. Policy evaluation with temporal differences: A survey and comparison. *Journal of Machine Learning Research*, 15:809–883, 2014.

[6] Gerald DeJong and Mark W Spong. Swinging up the acrobot: An example of intelligent control. In *Proceedings of 1994 American Control Conference-ACC'94*, volume 2, pages 2158–2162. IEEE, 1994.

[7] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[8] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58, 2016.

[9] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

[10] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[11] Mykel Kochenderfer. Jrlf: Java reinforcement learning framework, 2006. *URL http://mykel. kochenderfer. com/jrlf*.

[12] Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.

[13] Jeffrey Mahler and Ken Goldberg. Learning deep policies for robot bin picking by simulating robust grasping sequences. In *Conference on robot learning*, pages 515–524, 2017.

[14] Andrew William Moore. Efficient memory-based learning for robot control. 1990.

[15] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746, 2006.

[16] G Neumann. A reinforcement learning toolbox and rl benchmarks for the control of dynamical systems. *Dynamical principles for neuroscience and intelligent biomimetic devices*, page 113, 2006.

[17] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[18] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[19] Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. Pybrain. *Journal of Machine Learning Research*, 11(24):743–746, 2010.

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[21] Peter Stone, Gregory Kuhlmann, Matthew E Taylor, and Yaxin Liu. Keepaway soccer: From machine learning testbed to benchmark. In *Robot Soccer World Cup*, pages 93–105. Springer, 2005.

[22] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[23] Dequan Wang, Coline Devin, Qi-Zhi Cai, Philipp Krähenbühl, and Trevor Darrell. Monocular plan view networks for autonomous driving. *arXiv preprint arXiv:1905.06937*, 2019.

[24] Akihiko Yamaguchi and Tsukasa Ogasawara. Skyai: Highly modularized reinforcement learning library. In *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pages 118–123. IEEE, 2010.

[25] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3562–3573, 2018.

[26] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.