KNOW YOUR ACTION SET: LEARNING ACTION RELATIONS FOR REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Intelligent agents can solve tasks in a variety of ways depending on the action set at their disposal. For instance, while using a toolkit for repair, the choice of tool (the action) closely depends on what other tools are available. Yet, such dependence on other available actions is ignored in conventional reinforcement learning (RL) since it assumes a fixed action set. In this work, we posit that learning the interdependence between actions is crucial for RL agents acting under a varying action set. To this end, we propose a novel policy architecture that consists of an input graph composed of available actions and a graph attention network to learn the action interdependence. We demonstrate that our architecture makes action decisions by correctly attending to the relevant actions in both value-based and policy-based RL. Consequently, it consistently outperforms non-relational architectures on applications where the action space can vary, such as recommender systems and physical reasoning with tools and skills.

1 INTRODUCTION

Imagine you want to hang a picture on the wall. You may start by placing a nail on the wall and then use a hammer to nail it in. However, if you do not have access to a hammer, you would not use the nail. Instead, you would try alternative approaches such as using a hook and adhesive-strips, or a screw and a drill (Figure 1). In general, we solve tasks by choosing actions that interact with each other to achieve a desired outcome in the environment. Therefore, the best action decision depends not only on the environment, but also on what other actions are available to use.

In this work, we address the setting of varying action space in sequential decision-making. Typically reinforcement learning (RL) assumes a fixed action space, but recent work has explored variations in action space when adding or removing actions (Boutilier et al., 2018; Chandak et al., 2020a;b) or for generalization to unseen actions (Jain et al., 2020). These assume that the given actions can be treated independently in decision-making. But this assumption often does not hold. As the picture hanging example illustrates, the optimality of choosing the nail is dependent on the availability of a hammer. Therefore, our goal is to address this problem of learning the interdependence of actions.

Addressing this problem is important for many varying action space applications. These include recommender systems where articles to recommend vary every day and physical reasoning where decision-making must be robust to any given set of tools, objects, or skills. In this paper, we benchmark three such scenarios where learning action interdependence is crucial for solving the task optimally. These are (i) shortcut-actions in grid navigation, whose presence shortens the path to the goal, (ii) co-dependent actions in tool reasoning, where tools need other tools to be useful, and (iii) list-actions or slate-actions (Sunehag et al., 2015) in simulated and real-data recommender systems, where the user satisfaction is a consequence of the collective effect of the recommended list.

There are three key challenges in learning action interdependence for RL with varying action space. First, since the overall actions are not known in advance, the policy framework must be flexible to work with action representations. Second, the action space is now an additional variable component just like state observations in RL. Therefore, the policy framework must incorporate a variably sized set of action representations as part of the input. Finally, an agent's decision for each action's utility (e.g. Q-value or probability) should explicitly model its relationship with other available actions.

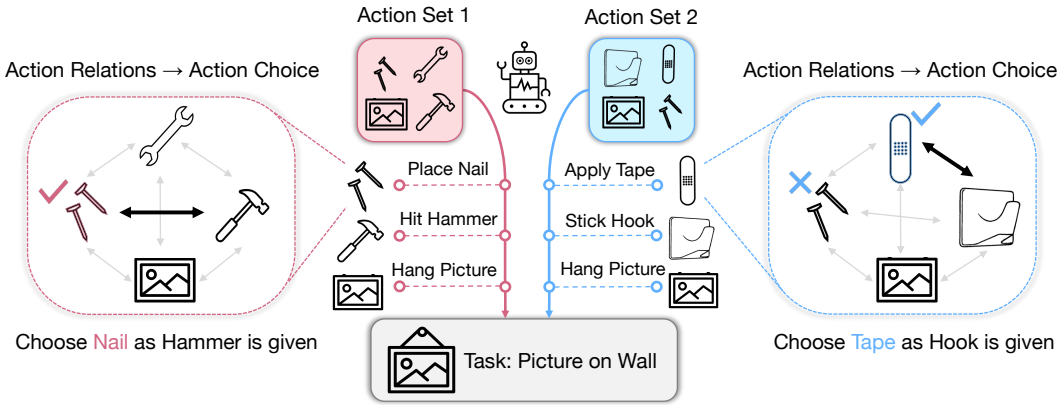


Figure 1: Picture hanging task with different tool sets. Intelligent agents adapt task solutions based on how the given actions relate to each other. (Left) Since hammer is available, the agent starts by choosing the nail to place on the wall. (Right) The nail is no longer a useful tool without a hammer, so using an adhesive tape is better since a hook is provided.

To address these challenges, we propose a novel policy architecture: AGILE, Action Graph for Interdependence Learning. AGILE builds on the utility network proposed in Jain et al. (2020) to incorporate action representations. Its key component is a graph attention network (GAT) (Veličković et al., 2017) over a fully connected graph of actions. This serves two objectives: summarizing the action set input and computing the action utility with relational information of other actions.

Our primary contribution is introducing the problem of learning action interdependence for RL with variable action space. We demonstrate our proposed policy architecture, AGILE, learns meaningful action relations. This enables it to improve optimality in varying action space tasks such as simulated and real-world recommender systems, and reasoning with skills and tools.

2 Related Work

2.1 Stochastic Action Sets

In recent work, Boutilier et al. (2018) provide the theoretical foundations of MDPs with Stochastic Action Sets (SAS-MDPs), where actions are sampled from a known base set of actions. They propose a solution with Q-learning and Chandak et al. (2020a) extend it to policy gradients. However, the assumption of knowing the finite base action set limits the practical applicability of SAS-MDPs. For example, recommender agents often receive unseen items to recommend and a robotic agent does not know beforehand what tools it would encounter in the future. We work with action representations to alleviate this limitation. Furthermore, in SAS-MDPs the action set can vary at any timestep of an episode. Thus, the learned policy will only be optimal on average over all the possible action sets (Qin et al., 2020). Whereas in out setting, the action sets only change at the beginning of a task instance and stays constant over the episode. This is a more practical setup and raises the challenging problem of solving a task optimally with any given action space.

2.2 Action Representations

Recent works from Chen et al. (2019b); Chandak et al. (2019); Kim et al. (2019); Tennenholtz & Mannor (2019); Jain et al. (2020) learn continuous representations of discrete actions. Such action representations have been used for learning in large discrete action spaces (Dulac-Arnold et al., 2015; Chandak et al., 2019), exploiting the shared structure among actions for efficient learning and exploration (He et al., 2015; Tennenholtz & Mannor, 2019; Kim et al., 2019), and continual learning (Chandak et al., 2020b). Closely related to our work, Jain et al. (2020) assume a similar setting of varying action space while training to generalize to unseen actions. Following their motivation, we use action representations to avoid making assumptions on the base action set. However, their policy treats each action independently, which we demonstrate leads to a suboptimal performance.

2.3 LIST-WISE ACTION SPACE

In listwise RL (or slate RL), the action space is combinatorial in list size. Commonly it has applications in recommendation systems (Sunehag et al., 2015; Zhao et al., 2017; 2018; Ie et al., 2019b; Gong et al., 2019; Liu et al., 2021; Jiang et al., 2018; Song et al., 2020). In recent work, Chen et al. (2019a) proposed Cascaded DQNs (CDQN) framework which learns a Q-network for every index in the list and train them all with a shared reward. For our experiments on listwise RL, we utilize CDQN as the algorithm and show how it can be used with AGILE as the policy framework.

2.4 RELATIONAL REINFORCEMENT LEARNING

Graph neural networks (Battaglia et al., 2018) have been explored in RL tasks with a rich relational structure, such as morphological control (Wang et al., 2018; Sanchez-Gonzalez et al., 2018; Pathak et al., 2019), multi-agent RL (Tacchetti et al., 2019), physical construction (Hamrick et al., 2018), and structured perception in games like StarCraft (Zambaldi et al., 2018). In this paper, we propose that the set of actions possess a relational structure which enables the actions to interact and solve tasks in the environment. Therefore, we leverage graph attention networks (Veličković et al., 2017) to learn these action relations and show that it can represent meaningful action interactions.

3 PROBLEM FORMULATION

A hallmark of intelligence is the ability to be robust in an ever changing environment. To this end, we consider the setting of RL with a varying action space, where an agent receives a different action set in every task instance. Our key problem is to learn the interdependence of actions so the agent can act optimally with any given action set. Figure 1 illustrates that for a robot with the task of hanging a picture on the wall, starting with a nail is optimal only if it can access a hammer subsequently.

3.1 REINFORCEMENT LEARNING WITH VARYING ACTION SPACE

We consider episodic Markov Decision Processes (MDPs) with discrete action spaces, supplemented with action representations. The MDP is defined by a tuple $\{\mathcal{S}, \mathbb{A}, \mathcal{T}, \mathcal{R}, \gamma\}$ of states, actions, transition probability, reward function, and discount factor, respectively. The base set of actions $\mathbb{A}$ can be countably infinite. To support infinite base actions, we use D-dimensional action representations $c_a \in \mathbb{R}^D$ to denote an action $a \in \mathbb{A}$. These can be image or text features of a recommendable item, or behavior characteristics of a tool, or simply be one-hot vectors for a known and finite action set.

In each instance of the MDP, a subset of actions $\mathcal{A} \subset \mathbb{A}$ is given to the agent, with associated representations $\mathcal{C}$. Thereafter, at each time step t in the episode, the agent receives a state observation $s_t \in \mathcal{S}$ from the environment and acts with $a_t \in \mathcal{A}$. This results in a state transition to s_{t+1} and a reward r_t. The objective of the agent is to learn a policy $\pi(a|s, \mathcal{A})$ that maximizes the expected discounted reward over evaluation episodes with potentially unseen actions, $\mathbb{E}_{\mathcal{A}\subset\mathbb{A}}\left[\sum \gamma^{t-1} r_t\right]$.

3.2 CHALLENGES OF VARYING ACTION SPACE

1. **Using Action Representations**: The policy framework should be flexible to take a set of action representations $\mathcal{C}$ as input, and output corresponding Q-values or probability distribution for RL.
2. **Action Set as part of State:** When action set varies, the original state space $\mathcal{S}$ is not anymore Markovian. For example, the state of a robot hanging the picture is under-specified without knowing if its toolbox contains a hammer or not. The MDP can be preserved by reformulating the state space as $S' = \{s \circ \mathcal{C}_\mathcal{A} : s \in \mathcal{S}, \mathcal{A} \subset \mathbb{A}\}$ to include the representations $\mathcal{C}_\mathcal{A}$ associated with the available actions $\mathcal{A}$ (Boutilier et al., 2018). Thus, the policy framework must support the input of $s \circ \mathcal{C}_\mathcal{A}$, where $\mathcal{A}$ is a variably sized action set.
3. **Interdependence of Actions:** The optimal choice of an action $a_t \in \mathcal{A}$ is dependent on the action choices that would be available in the future steps of the episode, $a_{t'} \in \mathcal{A}$. Recalling Figure 1, a nail should only be picked initially from the toolbox if a hammer is accessible later. Thus, we must explicitly model the relationships between the characteristics of the current action c_{a_t} and the possible future actions $c_{a_i} \forall a_i \in \mathcal{A}$.

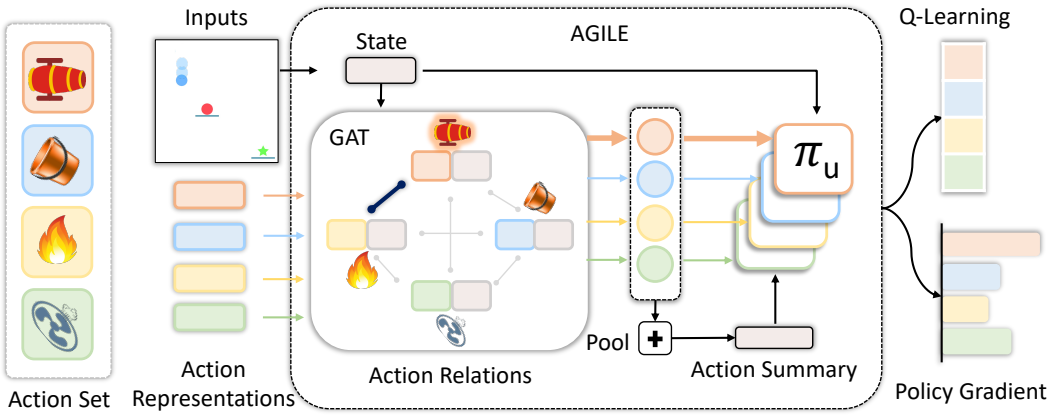


Figure 2: Given an action set, AGILE builds a complete action graph by joining action representations and state encoding. A graph attention network learns to attend to other actions that are relevant for each action in the current state. For example, the attention weight is expected to be high between the cannon and the fire because fire can activate the cannon. The output relational action features are better suited for decision-making via a utility network that computes the value or selection probability of the action. This utility network also takes as input the state and mean-pooled vector of the relational action features.

4 Approach

Our goal is to design a policy framework that is optimal for any given action space by addressing the challenges in Sec. 3.2. To work with action representations, we build on the utility network proposed by Jain et al. (2020). Our key insight is to use graph neural networks for both, summarizing the list of given action representations and learning inter-action interdependence.

4.1 AGILE: Action Graph for Interdependence Learning

We propose AGILE, a relational framework for learning action interdependence in RL with a varying action space. Given a list of action representations $\mathcal{C}$, AGILE builds a fully-connected action graph. This is processed by a graph attention network (GAT) (Veličković et al., 2017) which learns action relations. The resultant nodes are pooled together to form a compact relational summary of the action space. This summary vector and each action's relational representation is used to compute the utility value of the action, which can be used as a Q-value or as a probability distribution for policy gradient methods via a softmax (Figure 2).

Action Graph The input to our policy framework consists of the state s and a list $\mathcal{C} = [c_{a_0}, ..., c_{a_k}]$ of action representations for each action $a_i \in \mathcal{A}$. We build a fully connected action graph $\mathcal{G}$ with vertices corresponding to each action. If certain action relations are predefined in domain knowledge, we could use those as well. The relations that we must infer depend on the current state. For instance, a screwdriver is relevant to a screw while repairing furniture, but a drill machine is more related when the screw is for a wall. Following this insight, we make join state and action representations, $c'_{a_k} = (s, c_{a_k})$ to form the input nodes of the graph. In Sec. 6.3.2, we validate this insight by comparing performance against having graph nodes as action representations only.

Graph Attention Network: The action graph $\mathcal{G}$ is input to a GAT. Since the graph is fully-connected, we choose an attention-based graph architecture which can learn to focus on the most relevant actions in the given set. A similar insight was employed by Zambaldi et al. (2018) where the entities inferred from the visual observation are assumed to form a fully connected graph. To enable propagation of sufficient relational information between actions, we use two graph attention layers with an ELU activation in between (Clevert et al., 2015). We found a residual connection after the second GAT layer was crucial in experiments.

Action Set Summary: The output of the GAT is a list of relational action features $\mathcal{C}^R = \{c^R_{a_0}, ...c^R_{a_k}\}$ which contain the information about presence of other available actions and their re-

lations. Therefore, we compute the action set summary in a compact form by mean-pooling the relational action features, $S_{\mathcal{A}} = \frac{1}{K}\sum_{k=1}^{K} c_{a_k}$.

Action Utility In order to use the relational action representations with RL, we follow the insight of utility network π_u from Jain et al. (2020). It takes the relational action representations, state, and action summary as input for each action and outputs a utility score reflecting how useful the action is for the current state and in relation to the other available actions. The utility score can be used as Q-value directly if the training algorithm is DQN, whereas it can be used as a probability distribution over the available actions as a softmax.

4.2 Training AGILE framework with Reinforcement Learning

The policy architecture of AGILE can be trained with both policy gradient and value-based RL methods.

Policy Gradient (PPO) : For every action decision step, we take the output action utility and use a softmax over all available actions to get a probability distribution. We use PPO (Schulman et al., 2017) to train the policy. PPO requires a value function $V(s')$ as a baseline. We represent $s' = (s, S_{\mathcal{A}})$ as the concatenation of the state encoding and the action set summary $S_{\mathcal{A}}$ inferred from the GAT. An important consideration implementation detail to make the training faster and stable was to *not* share the GAT weights between the action $\pi(a|s')$ and the critic $V(s')$ networks.

Value-based RL (DQN) The output action utility can be directly treated as the $Q(s', a)$ value of the action at the current state with the given available action set $\mathcal{A}$. This can be trained using standard Q-learning Bellman backup.

Listwise RL (CDQN): To solve the combinatorial action space problem of listwise actions, we follow the Cascaded DQN (CDQN) framework of Chen et al. (2019a). The main challenge is that building the list all at once is not feasible due to the intractably large number of possible lists. Therefore, the key is to build the list incrementally, one action at a time. Thus, each list index can be treated as an individual non-combinatorial action which can be trained with RL. We replace the Q-network of CDQN with AGILE in order to accommodate a varying action space. We share the weights of the cascaded Q-networks.

5 Environments

We evaluate AGILE on three varying action set scenarios requiring learning action interdependence. (i) shortcut actions in goal-reaching, whose presence shortens the optimal path to the goal in a 2D Grid World, (ii) co-dependent actions in tool reasoning, which require other tools to activate their functionality, and (iii) list-actions in simulated and real-data recommender systems where the cumulative list affects the user preference. Figure 3 provides an overview of the tasks, base and varying action space, and an illustration of how action interdependence is important.

5.1 Dig Lava Grid Navigation

We modify the grid world environment (Chevalier-Boisvert et al., 2018) where an agent navigates a 2D maze with two lava rivers to reach a goal. The agent always has access to 4 direction movements and a *turn-left* skill (Figure 5). However, two actions out of 4 special skills: *turn-right, step-forward, dig-orange-lava* and *dig-pink-lava*, are randomly sampled for the agent in every task instance. The agent can walk over lava for one timestep without dying, but it can remove the lava by using the matching *dig-lava* skill. Thus, when available, *dig-lava* skills can be used to build shortcut paths to the goal on the other end, and thus receive a higher reward. We use PPO to train all methods.

5.2 Chain REAction Tool Environment: CREATE

We use the CREATE environment (Jain et al., 2020) as a challenging environment with a large base action space and unseen actions for evaluation. The task is to sequentially select tools from a given toolbox and place them appropriately on the screen with the objective to push the red ball towards the green goal. We deal with the hybrid action space with an auxiliary network that takes the state

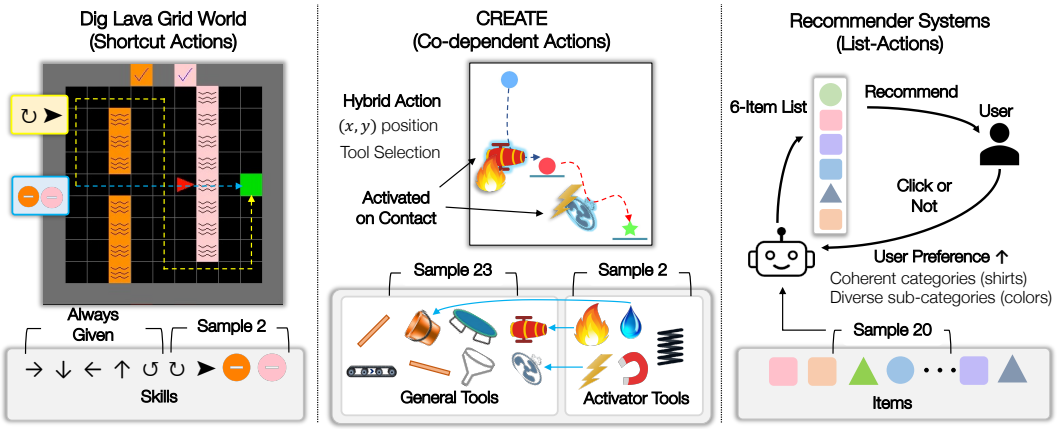


Figure 3: **Environment Setup.** (Left) In the grid world, the red agent must avoid orange and pink lava to reach the green goal. The dig lava skills, when available, enable shortcut paths to the goal. (Middle) In CREATE, tools are selected and placed sequentially with the aim to push the red ball to the green goal. General tools (cannon, fan) require activator tools (fire, electric) to start functioning. The choice of general tools depend on what activators are available and vice-versa. (Right) In simulated and real-data recommender systems, the agent selects a list of items. Lists with coherent categories and complementary sub-categories improve user satisfaction (e.g. click likelihood).

and action set summary as input, following Jain et al. (2020). We simulate the need of action relation learning by associating each tool (e.g. cannon) with an activator tool (e.g. fire). A tool functions only if in contact with the right activator, otherwise the objects pass through it. Thus the tool choice must now consider the current environment state as well as whether its activators are available. We borrow the action representations from the prior work and add extra dimensions to accommodate one-hot vectors for the activator tools.

5.3 Recommender Systems

Recommender system is a natural application of the problem of varying action space where items to recommend change often. Specifically, action interdependence is apparent in the case of listwise actions. For example, a list of diverse ads is more likely to get a user click than ads from the same domain (Zhou et al., 2010). In this work, we experiment with the list metric of Complementary Product Recommendation (CPR) (Hao et al., 2020).

Complementary Product Recommendation (CPR): concerns with the scenario where user satisfaction with a recommended list increases if the list is coherent at a high-level but diverse at a low-level. Specifically, we consider each item to have a primary category, such as all of them are shirts, and a subcategory, such as color of the shirt. We define the CPR of a slate as, $\frac{\text{Entropy of subcategory}}{\text{Entropy of category}}$, i.e. we encourage diversity in subcategories from and similarity in the main category theme. We demonstrate the effect of maximizing CPR in different ways for the simulated and real-data experiments.

5.3.1 Simulated Recommender System: RecSim

We develop on the open-sourced simulator RecSim simulator ((Ie et al., 2019a)) which allows an agent to interact with simulated users by recommending a slate of items. We have a base action set of 500 train items and 500 test items, and in each episode 20 items are given to the agent. It must recommend a list of size 6. We take user preference vector as the state (MDP) and ground truth action representations from the environment. We incorporate CPR by boosting up the probability of user clicking any item proportional to the CPR. This simulates one of the scenarios where user preference model is influenced by the entire list. An optimal agent should identify the most common category in the available action set and try to recommend most items in its list from category. This requires understanding the relation with the other candidate items from the same category. We train CDQN-based models to maximize the number of clicks in a user session.

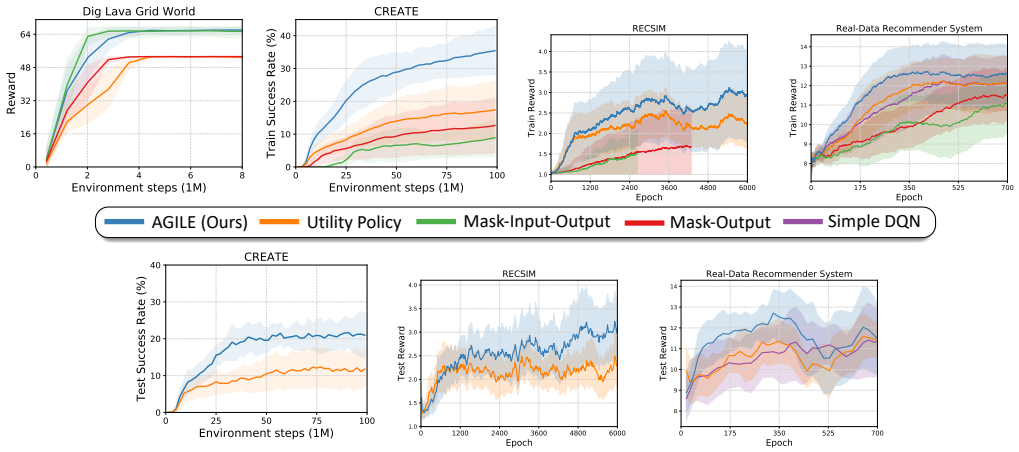


Figure 4: We evaluate AGILE against baselines on both training and testing actions (except Grid World). The results are averaged over 5 random seeds and the seed variance is shown with shading. AGILE outperforms all baselines which either do no consider the overall action set as input or cannot deal with unseen actions.

5.3.2 REAL-DATA RECOMMENDER SYSTEM

We collect interaction data for one month for a listwise online campaign recommender system. Users are represented by attributes such as age, occupation, and localities. Items attributes are also given such as text features, image features, and reward points of campaigns. We simulate a representative RL environment by training a reward model on a set of 68,775 users and 57 items to estimate the click likelihood of the users(Appendix A.2). We augment the user reward model with the CPR reward to incentivize the agent to recommend high-CPR user-relevant items. We train a VAE (Kingma & Welling, 2013) to learn action representations, which are given to the RL agent as input. The test data consists of 82,445 users and 58 items, about 30 of which are shared with the training items. We report the test reward for models trained with CDQN algorithm.

6 EXPERIMENTS

Our experiments are designed to answer the following questions in the context of RL with varying action space: (1) How effective is AGILE compared to prior works that treat actions independently or assume a fixed action set? (2) How effective are AGILE's relational action representations for computing action set summary and action utilities? (3) Does the attention in AGILE capture meaningful action relations? (4) Is attention necessary in the graph network of AGILE? (5) Is learning state-dependent action relations necessary for solving general varying action space tasks?

6.1 EFFECTIVENESS OF AGILE IN VARYING ACTION SPACES

We evaluate AGILE against baselines from prior work in varying action space, that either assume a fixed action set or work without the knowledge of the other available actions. As ablations, we study the contributions of relational action features in decision-making and summarizing of input action set. All methods share the same training framework (PPO or CDQN) depending on the environment.

6.1.1 BASELINES

- **Mask-Output** (No representations, No input action set): Assuming a fixed action space in the output, q-values or policy probabilities are masked out for unavailable actions. It represents the architectures used in SAS-MDP Boutilier et al. (2018); Chandak et al. (2020a).
- **Mask-Input-Output** (No representations): Augments the binary availability mask of given actions, 1s for available indices and 0s for unavailable, to the state input of *Mask-Output*.
- **Utility-Policy** (No input action set): Proposed by Jain et al. (2020), action representations are used independently for computing each action's utility, ignoring any interdependence.

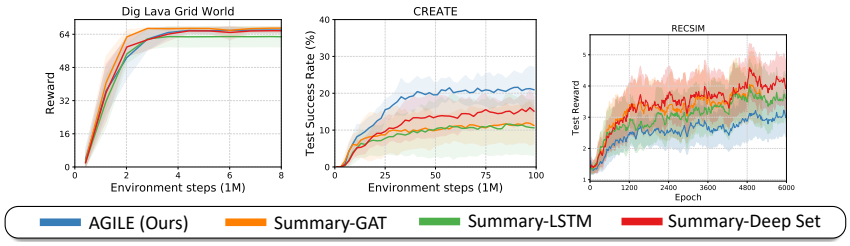


Figure 5: Comparison against ablations.

- **Simple DQN** (No cascade, No input action set): For listwise RL specifically, we include the DQN baseline that simply selects top-K items, instead of reasoning about the overall list. Thus, it ignores two kinds of action interdependence: (i) on other available actions, (ii) on other items in the list.

6.1.2 Ablations

- **Summary-LSTM**: A Bi-LSTM (Huang et al., 2015) encodes the list of action representations.
- **Summary-Deep Set**: A deep set architecture (Zaheer et al., 2017) with mean pooling is used to aggregate the input action list.
- **Summary-GAT**: AGILE where action set summary is used, but relational action feature output of GAT is not used for computing an action's utility. Instead, raw action representations are used. This delineates the importance of using action interdependence for decision-making.

6.1.3 Results

Figures 4 and 5 show the results on baselines (both train and test actions) and ablations (test only, except Grid World). For Grid World, all methods learn to reach the goal with perfect success. The difference in reward is due to the length of path taken. The methods which do not take given action set as input, *Mask-Output* and *Utility Policy*, simply resort to the safe strategy of going around the lava rivers. This is because to take a shortcut, the agent must enter the lava and then dig it. So the movement action right before entering the lava must be aware of whether that *dig-lava* skill is available. Further, since the action space and action relations are relatively simple, having the action summary is enough to solve the task optimally.

For CREATE, Figure 4 show that AGILE clearly outperforms the baselines. This shows that the knowledge of dependence on other available actions is crucial for optimal policy. Further, *Utility policy* outperforms the other baselines by virtue of ease of learning with action representations. For generalization to unseen actions (bottom), the trend still holds. *Mask* methods do not support generalization due to fixed action set assumption. Among ablations, the *Summary-Deep Set* method performs reasonably well due to its simple architecture. This shows a limitation of training GAT-based architectures in sparse reward RL.In RecSim and real-data recommender system, the reward is coming from user clicks which is a noisy signal. Despite this, AGILE outperforms or is competitive with the baselines based, all based on CDQN framework. The ablations show that Deep set summarization achieves faster convergence, due to its simplicity.

6.2 Does the Attention in AGILE Learn Meaningful Action Relations?

In Figure 6, we analyze the agent performance qualitatively. In CREATE, at $t = 0$, the selected action *spring* in AGILE's GAT attends to various other tools, especially covers all the tools that get activated with spring, such as trampoline. At $t = 1$, the trampoline tool is selected with a strong attention on spring. This shows that the for selecting trampoline, the agent checks for the presence of spring, so it is possible to place it after the trampoline, if not before. In Grid World, we visualize the *Summary-GAT* ablation to see if summary also relies on attention. We consider the case where both $dig - lava$ skills are available. The agent goes right, digs the orange lava, and is about to enter the pink lava. At this point, the *Right* action attends with a high weight to $Dig - Pink$ skill, checking for its presence before making an irreversible decision. Finally, in RecSim, we observe

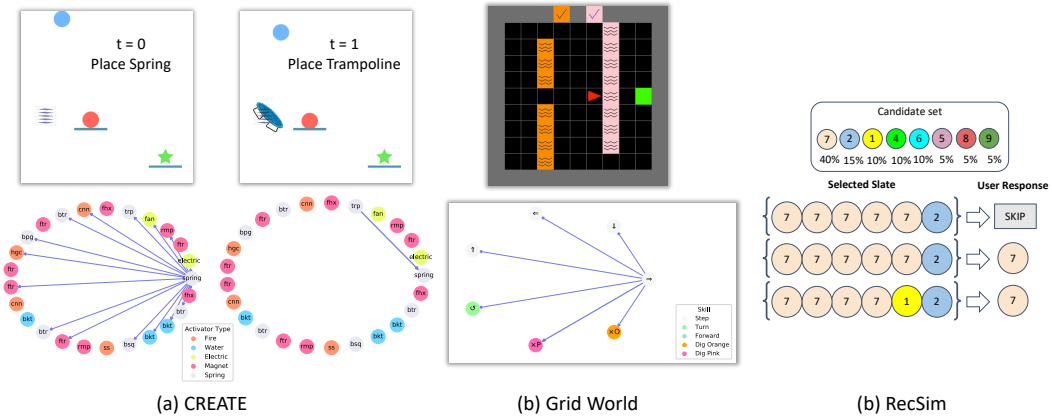


Figure 6: Qualitative analysis of environments with attention map visualization.

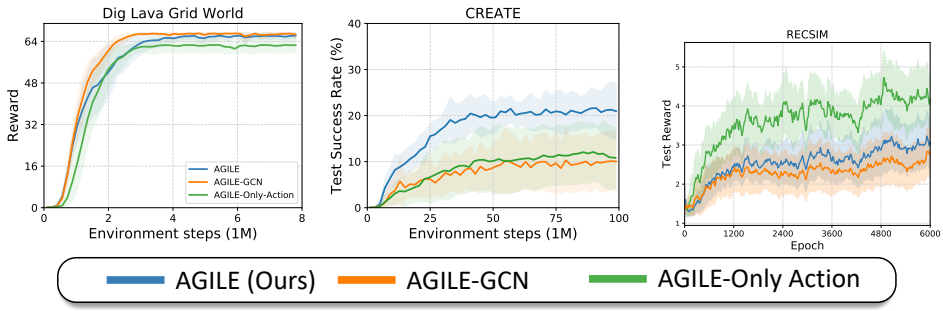


Figure 7: Additional Analyses (i) GAT v/s GCN, and (ii) state-action graph v/s action-only graphs

that the agent is able to maximize the CPR score by selecting 5 out of 6 items in the list from the same primary category. We provide qualitative video results for Grid World and CREATE on the project page `https://sites.google.com/view/varyingaction`.

6.3 Additional Analyses

6.3.1 Importance of Attention in the Graph Network

We validate the choice of using graph attention network as the relational architecture. In Figure 7, we compare GAT against GCN for AGILE. We observe that for the simple grid world environment, GCN achieves optimal performance. However, it suffers in CREATE where the action space is large and thus a fully-connected GCN would be difficult to train with sparse RL reward only. We believe that the attention in GAT makes the graph sparse and makes learning easier.

6.3.2 Importance of state-dependent learning of action relations

We compare the version of AGILE where the GAT only receives action features as input and not the state features. Thus, the decision-choice is still aware of other actions, but the relations are not dependent on the state. Figure 7 shows consistent drop in performance across all environments, because actions relate to other actions differently depending on the context.

7 Conclusion

We present AGILE, a policy architecture for leveraging action relations for reinforcement learning with varying action spaces. AGILE builds a complete graph of available actions' representations and utilizes a graph attention network to learn the interdependence between actions. We demonstrate that utilizing the relational knowledge of actions is crucial for optimal decision-making in 4 environments, including a real-data recommendation task.

REFERENCES

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Craig Boutilier, Alon Cohen, Amit Daniely, Avinatan Hassidim, Yishay Mansour, Ofer Meshi, Martin Mladenov, and Dale Schuurmans. Planning and learning with stochastic action sets. *arXiv preprint arXiv:1805.02363*, 2018.

Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip Thomas. Learning action representations for reinforcement learning. In *International Conference on Machine Learning*, pp. 941–950. PMLR, 2019.

Yash Chandak, Georgios Theocharous, Blossom Metevier, and Philip Thomas. Reinforcement learning when all actions are not always available. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3381–3388, 2020a.

Yash Chandak, Georgios Theocharous, Chris Nota, and Philip Thomas. Lifelong learning with a changing action set. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3373–3380, 2020b.

Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. Generative adversarial user model for reinforcement learning based recommendation system. In *International Conference on Machine Learning*, pp. 1052–1061. PMLR, 2019a.

Yu Chen, Yingfeng Chen, Yu Yang, Ying Li, Jianwei Yin, and Changjie Fan. Learning action-transferable policy with action embedding. *arXiv preprint arXiv:1909.02291*, 2019b.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

Yu Gong, Yu Zhu, Lu Duan, Qingwen Liu, Ziyu Guan, Fei Sun, Wenwu Ou, and Kenny Q Zhu. Exact-k recommendation via maximal clique optimization. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 617–626, 2019.

Jessica B Hamrick, Kelsey R Allen, Victor Bapst, Tina Zhu, Kevin R McKee, Joshua B Tenenbaum, and Peter W Battaglia. Relational inductive bias for physical construction in humans and machines. *arXiv preprint arXiv:1806.01203*, 2018.

Junheng Hao, Tong Zhao, Jin Li, Xin Luna Dong, Christos Faloutsos, Yizhou Sun, and Wei Wang. P-companion: A principled framework for diversified complementary product recommendation. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 2517–2524, 2020.

Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with a natural language action space. *arXiv preprint arXiv:1511.04636*, 2015.

Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging, 2015.

Eugene Ie, Chih-wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. Recsim: A configurable simulation platform for recommender systems. *arXiv preprint arXiv:1909.04847*, 2019a.

Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. Slateq: A tractable decomposition for reinforcement learning with recommendation sets. 2019b.

Ayush Jain, Andrew Szot, and Joseph Lim. Generalization to new actions in reinforcement learning. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4661–4672. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/jain20b.html.

Ray Jiang, Sven Gowal, Timothy A Mann, and Danilo J Rezende. Beyond greedy ranking: Slate optimization via list-cvae. *arXiv preprint arXiv:1803.01682*, 2018.

Hyoungseok Kim, Jaekyeom Kim, Yeonwoo Jeong, Sergey Levine, and Hyun Oh Song. EMI: Exploration with mutual information. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3360–3369, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/kim19a.html.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Shuchang Liu, Fei Sun, Yingqiang Ge, Changhua Pei, and Yongfeng Zhang. Variation control and evaluation for generative slate recommendations. *Proceedings of the Web Conference 2021*, Apr 2021. doi: 10.1145/3442381.3449864. URL http://dx.doi.org/10.1145/3442381.3449864.

Deepak Pathak, Chris Lu, Trevor Darrell, Phillip Isola, and Alexei A. Efros. Learning to control self-assembling morphologies: A study of generalization via modularity, 2019.

Zhiwei Qin, Xiaocheng Tang, Yan Jiao, Fan Zhang, Zhe Xu, Hongtu Zhu, and Jieping Ye. Ride-hailing order dispatching at didi via reinforcement learning. *INFORMS Journal on Applied Analytics*, 50(5):272–286, 2020.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pp. 4470–4479. PMLR, 2018.

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

Junshuai Song, Zhao Li, Chang Zhou, Jinze Bai, Zhenpeng Li, Jian Li, and Jun Gao. Co-displayed items aware list recommendation. *IEEE Access*, 8:64591–64602, 2020. doi: 10.1109/ACCESS.2020.2984543.

Peter Sunehag, Richard Evans, Gabriel Dulac-Arnold, Yori Zwols, Daniel Visentin, and Ben Coppin. Deep reinforcement learning with attention for slate markov decision processes with high-dimensional states and actions. *arXiv preprint arXiv:1512.01124*, 2015.

Andrea Tacchetti, H. Francis Song, Pedro A. M. Mediano, Vinicius Zambaldi, János Kramár, Neil C. Rabinowitz, Thore Graepel, Matthew Botvinick, and Peter W. Battaglia. Relational forward models for multi-agent learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJlEojAqFm.

Guy Tennenholtz and Shie Mannor. The natural language of actions. *arXiv preprint arXiv:1902.01119*, 2019.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018.

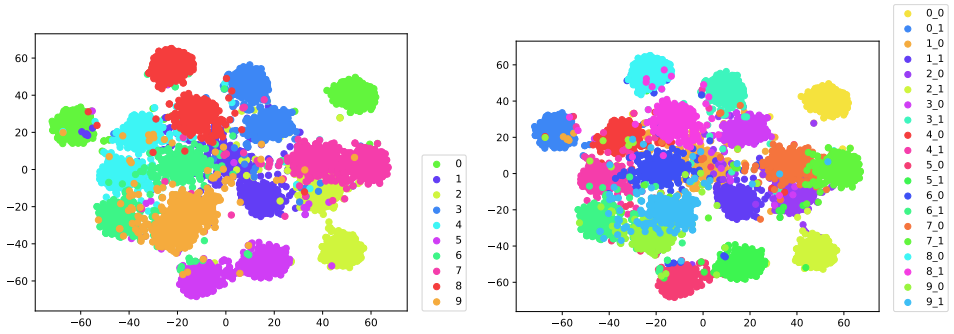
Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations*, 2018.

Xiangyu Zhao, Liang Zhang, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. Deep reinforcement learning for list-wise recommendations. *arXiv preprint arXiv:1801.00209*, 2017.

Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. Deep reinforcement learning for page-wise recommendations. *Proceedings of the 12th ACM Conference on Recommender Systems*, Sep 2018. doi: 10.1145/3240323.3240374. URL http://dx.doi.org/10.1145/3240323.3240374.

Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.



(a) Items with main category

(b) Items with sub category

Figure 8: t-SNE visualization of synthetically generated items in RecSim.

A Environments

A.1 RecSim

RecSim simulator allows us to flexibly change the behavior of users as well as representation of items and users. So, at each time-step, an agent receives the action set and needs to output the list of items(aka slate) for a user. Then, the selected slate will be evaluated by the user choice model together with the user preference embedding from the user sampler. Finally, the selected item will be used to update the user preference embedding which influences the state observation visible to the agent for the next time-step. In this work, in order to demonstrate the efficacy of our agents, we have made the following changes to RecSim;

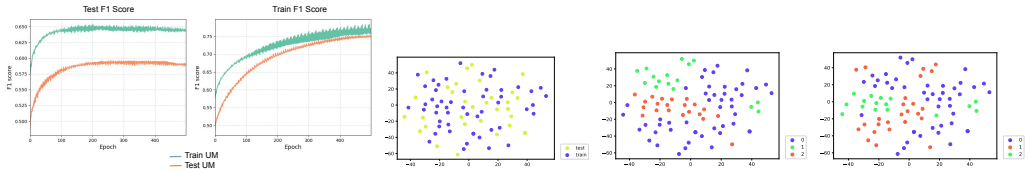
Slate Dependent User Choice Model: The original user choice model computes the pairwise similarity using the dot product of a user embedding and an item embedding in the presented slate as the likelihood of click of that item, which means that the user choice doesn't account for other items in the same slate. For this project, we want the user choice to depend on the slate instead of individual items. So, we have implemented the new two features to evaluate the click probability depending not only on the individual items but on the other items in the same slate, e.g., the common slate-wise Metrics: Diversity / Specificity etc.

Semantic item representations: The original discrete representation of items using the item-category lacks semantics. So, in order to enhance the interpretability of items, we have implemented the continuous item representation sampled from the GMM based on the item category and the sub-category for items. Fig. 8 shows the t-SNE plot of t-SNE visualization of 500 items sampled from the GMMs. There are 10 main categories that has two sub-categories individually. (a) Items sampled from the GMMs based on the main category. (c) Items sampled from the GMMs based on the sub category that there are two sub-category for each main category.

Candidate-set resampling methods: In order to simulate the varying action space environment, we have implemented several distributions to sample from with different characteristics. (1) **Random**; candidate items are sampled randomly from the set of train/test items. So that the candidate set can be missing the theme in the candidate-set. (2) **Uniform**; candidate items are sampled uniformly from each main category such that we sample an equal number of items from each category. (3) **Skewed**; candidate items are sampled in a way that we, first, randomly sample the main category of a candidate-set then we sample a large number of items from that category followed by filling the remaining candidate-set with the items which have been sampled from other categories.

A.2 Real-world recommender system

The biggest issue of training any RL recommender systems is the lack of a simulator which can be accurately approximate the user responses to serves as an environment guiding the agent's learning process over the course of training.



(a) (b) (c) (d)

Figure 9: (a) F1 Score for Training/test user models (b) t-SNE visualization of Train/Test items (c) Item distribution of main category. (d) Item distribution of sub category.

In order to simulate the real-world setting accurately, in which when we train a recommender system, we don't have access to online users, we employed the two phase experiment scheme, namely, offline training and online evaluation using two different pretrained reward models. Note that the use of pretrained user model to evaluate the agents is common practice in recommender system community (Chen et al., 2019a).

We collected the dataset from two different periods, called offline and online periods, respectively. Then we conducted the following scheme to train and evaluate our agents. **Offline Training:** Prior to the training of RL agents, we pretrained a VAE to prepare the item-embedding that will be used for training a reward model and the agents too. So, in this offline setting, an agent needs to learn to generalize over any users through interacting with the offline reward model. **Online Evaluation:** using the pretrained VAE, we construct the test item embedding. Then we trained the reward model on the online dataset to evaluate the trained agents. The training results and item representations can be found in 9.

B EXPERIMENTS

B.1 HYPERPARAMETERS

The hyperparameters for our action graph are shown in Table 1 and the environment/policy-specific hyperparameters are described in Table 2.

Hyperparameter	Value
Action Graph	
number of GAT layers	2
leakyReLU alpha	0.2
graph node dimension	32
attention coefficient	1.0

Table 1: Action Graph-specific Hyperparameters

Hyperparameter	Grid world	RecSim	CREATE	Real RecSys
Environment				
action representation size	16	32	128	32
num. of processes	64	16	64	16
epochs	10000	10000	10000	5000
Policy				
observation space	81	10	$84 \times 84 \times 3$	$32 \times 32 \times 3$
actions per episode	13	20	50	20
total environment steps	4×10^7	3×10^6	6×10^7	8×10^5
max. episode length	10	15	30	10
batch size	128	128	128	128

Table 2: Environment/Policy-specific Hyperparameters