12-2021

Building action sets in a deep reinforcement learner

Yongzhao WANG

Arunesh SINHA
Singapore Management University, aruneshs@smu.edu.sg

Sky C.H. WANG

Michael P. WELLMAN

Citation

WANG, Yongzhao; SINHA, Arunesh; WANG, Sky C.H.; and WELLMAN, Michael P.. Building action sets in a deep reinforcement learner. (2021). *Proceedings of the 20th IEEE International Conference on Machine Learning and Applications, Virtual Conference, 2021 December 13-15*. 484-489.
Available at: https://ink.library.smu.edu.sg/sis_research/6785

Building Action Sets in a Deep Reinforcement Learner

Yongzhao Wang
University of Michigan
Ann Arbor, USA
wangyzh@umich.edu

Arunesh Sinha
Singapore Management University
Singapore
aruneshs@smu.edu.sg

Sky CH-Wang
Columbia University
New York, USA
skywang@cs.columbia.edu

Michael P. Wellman
University of Michigan
Ann Arbor, USA
wellman@umich.edu

***Abstract*—In many policy-learning applications, the agent may execute a *set* of actions at each decision stage. Choosing among an exponential number of alternatives poses a computational challenge, and even representing actions naturally expressed as sets can be a tricky design problem. Building upon prior approaches that employ deep neural networks and iterative construction of action sets, we introduce a *reward-shaping* approach to apportion reward to each atomic action based on its *marginal contribution* within an action set, thereby providing useful feedback for learning to build these sets. We demonstrate our method in two environments where action spaces are combinatorial. Experiments reveal that our method significantly accelerates and stabilizes policy learning with combinatorial actions.**

I. Introduction

Deep reinforcement learning (DRL) has had remarkable success in many challenging domains [1]–[3]. These achievements have inspired researchers to use DRL for a broad range of complex problem formulations. One such challenge arises from combinatorial action spaces, that is, where an action taken in any state is a subset of underlying set of atomic actions. Tasks with combinatorial action spaces appear naturally in many areas. For example, a recommendation task may call for proposing a portfolio of suggestions. Besides, in the domain of security, an attacker may be able to strike a set of targets simultaneously, forcing upon the defender a choice of which subset of targets to protect.

Solving tasks with combinatorial action spaces is challenging for DRL because policy networks typically devote a node to every exclusive option, of which there are exponentially many. In order to handle problems with combinatorial action space, existing methods either focus on certain selected actions [4], [5] or transform the problem of choosing a combinatorial action into a sequence of choices of the underlying atomic actions [6]–[8].

In this study, we focus on efficiently learning to build action sets for single-agent DRL problems with combinatorial action space. We are guided by the observation that learning efficiency is impeded in existing methods because reward feedback is associated with the action set as a whole, making it difficult to attribute reward to individual constituent atomic actions.

To mitigate this issue, we propose a general *reward-shaping* scheme that credits each atomic action with its *marginal contribution* (MC) towards the reward of the action set that it belongs to. This marginal contribution is used to update the reward for each atomic action in an *a posteriori* manner. We also consider the relation of our reward-shaping scheme to the concept of Shapley value from coalitional games.

We demonstrate the efficacy and generality of our algorithm in two environments with combinatorial action space: a simple goods-matching problem which represents the essence of combinatorial choice in a one-shot setting, and a complex cyber-security game featuring imperfect information, sequential dependence, and asymmetric players. To verify the flexibility of our approach, we test our scheme in these two environments by combining it with different action-set building methods and DRL algorithms. Our experimental results show that our proposed reward-shaping scheme can significantly accelerate, stabilize, and improve learning performance compared to cases without the marginal contribution signal. Finally, we empirically investigate factors that affect the performance of reward-shaping by analyzing different constitutions of a combinatorial action, which provides an intuitive understanding of how they impede learning in such problems.

The key contributions of this work are: (a) a detailed analysis of issues in DRL with combinatorial action spaces; (b) a general reward-shaping scheme for efficient learning with combinatorial action space in DRL; (c) a detailed demonstration of reward-shaping efficacy in two realistic environments; and (d) experimental analysis of factors that affect the performance of reward-shaping.

II. Preliminaries and Prior Work

Preliminaries.: We work in a single-agent RL setting with an underlying MDP characterized by a state space $\mathcal{S}$ and a set of *atomic actions* $\mathcal{A}$. The action taken in state $s \in \mathcal{S}$ is denoted by A and $A \in 2^{\mathcal{A}}$ is a subset of the atomic actions. Clearly, the action space $2^{\mathcal{A}}$ is combinatorial in nature. The immediate reward is $R(s, A)$ and the goal is to learn a policy $\pi : \mathcal{S} \to 2^{\mathcal{A}}$ that maximizes the value function.

We broadly classify previous approaches for RL problems with combinatorial action space into two categories.

Independent selection.: In independent selection, the RL learner selects each atomic action with only limited consid-

This work was supported by funding from the US Army Research Office (MURI grant W911NF-13- 1-0421) and from DARPA SI3-CMD.

978-1-6654-4337-1/21/$31.00 ©2021 IEEE
DOI 10.1109/ICMLA52953.2021.00081

eration of its joint effect with other selected atomic action. Such selection is usually driven by a ranking of the atomic action or sampling from a very limited sets of atomic actions. [4] address the problem of recommending a subset of K news articles from a large candidate set. They then sample a finite number of candidate subsets and choose the best. [5] focus on another NLP task where actions correspond to fixed-length word sequences. The action space is combinatorial in the length and the size of the dictionary. They present a DRL architecture which prunes the actions available in each state to make the problem tractable. [9] tackle a text-based problem where actions are sets of words. They first learn a dense representation of actions from word embeddings, then apply specialized matching techniques to construct candidate sets. [10] propose a variant of dueling double deep Q-networks with multiple action network branches and a shared decision module. Each branch corresponds to one action component of a combinatorial action and selects action independently.

Sequential selection.: In sequential selection, the selection of a combinatorial action is based on a sequence of choices of the underlying atomic actions and thus avoiding handling the exponentially large combinatorial space directly. For example, [7] tackle a video-game domain where actions are compound combinations of primitive actions. They employ an auto-regressive method based on Asynchronous Advantage Actor Critic to factor the joint distribution over actions into marginal distributions, thus reducing the action dimensionality through a sequential action selection based on the marginal distributions. In the problem studied by [6], a scheduler chooses a set of jobs to run at each time step. To circumvent the combinatorial action space issue, they model the agent as selecting the jobs in sequence within a time step, generating a state transition after each selection. The time step terminates when a void or invalid action is chosen, upon which point the agent receives rewards based on the selected actions. Like [6], [8] build the action set incrementally, in an approach they term *greedy action set building* (GASB). After each selection, they feed the partially constructed action set back to the neural network as modified input, and select the next atomic action. This approach accounts for value dependencies among actions by learning the incremental values for each candidate conditioned on the current partial set.

III. Reward Shaping by Marginal Contribution

As a motivation for our approach, we identify two general causes of inefficient learning in past work on DRL problems with combinatorial action space. First, the reward feedback $R(s, A)$ is associated with the overall action set A, and so does not distinguish the relative benefit or harm of constituent atomic actions in A. In effect, the atomic actions in A are treated the same. We term this the *homogeneous-action* issue. In principle, with sufficient data over a variety of overlapping sets, a reinforcement learner could eventually figure out the effect of each atomic action as well as the dependency among these atomic actions. However, the number of samples required to achieve this could be prohibitive.

A second reason for inefficient learning, which arises in sequential selection approaches, is what we term the *sparse-reward* issue. Since the agent receives feedback $R(s, A)$ from the environment only when the entire action set A is assembled, there are many intermediate steps (selecting individual atomic actions) not associated with any reward. Again, whereas DRL can eventually propagate rewards backward to properly credit earlier decisions, sparseness in intermediate rewards is known to seriously impede efficient learning.

To deal with these two issues, we propose a general reward-shaping approach that credits individual atomic actions based on their marginal contribution to the *sequential* construction of an action A. We implement the reward shaping in a posteriori manner and update immediate rewards of the atomic actions in an action set by their marginal contributions *after* the action A is built and deployed.

Formally, we represent the constructed action as a sequence $A = (a_1, \ldots, a_n)$ reflecting the order that the atomic actions $a_{i\in[n]}$ are added to the set. We use pass to denote a special dummy action that serves to indicate the end of the iterative building of A; pass is not included explicitly in A as it is not an actual playable action. Let A_k be a subset of A that contains the first k items in A where $0 \leq k < n$. The key step in reward shaping is to decompose immediate reward $R(s, A)$ into marginal contributions $r(a_{k+1} \mid s, S_k)$, crediting the $(k+1)$st action added based on its marginal contribution with respect to the set S_k. To preserve overall reward semantics, this decomposition should obey the following accounting identity:

$$R(s, A) = r(\mathsf{pass} \mid s, A) + \sum_{k=0}^{n-1} r(a_{k+1} \mid s, A_k) \,. \qquad (1)$$

Note that pass, which is considered as a special atomic action, earns a credit like other atomic actions. We found that a proper contribution of pass plays an important role in learning and therefore we discuss the contribution of pass in detail later in the paper.

Example 1. *Goods-Matching Problem*
Consider a middleman who purchases goods from manufacturers in the morning and sells to consumers at the end of the day. The middleman's decision on which goods to purchase forms a combinatorial action. The cost to buy item x is given by $c(x)$, which is independent across items. The sale price depends on the entire bundle X of items acquired, designated by the function $P(X)$, with $P(\emptyset) = 0$. The bundle price may be greater or less than the sum of individual items, depending on whether the goods are complements or substitutes. For example, consider the goods B1, B2, and T, representing two badminton shuttlecocks and a racket. The two shuttlecocks are substitutes: $P(\{B1, B2\}) < P(\{B1\}) + P(\{B2\})$. Bundling a shuttlecock with a racket, in contrast, exhibits complementarity: $P(\{B1, T\}) > P(\{B1\}) + P(\{T\})$.

The definition of marginal contribution for this example domain is straightforward. The reward for adding an item to the set (considered as an atomic action) is the marginal revenue it contributes to the bundle, minus its individual cost.

In general, the definition of marginal contribution will be domain specific, and may be defined in hindsight once the outcome of the action set is known.

Relation to Shapley Value: In coalitional game theory, Shapley value is a well-known and accepted way of distributing value generated from a coalition to the individual players in the coalition [11]. Shapley value for any set R and $a_i \in R$ is defined as $\phi_i(R) = \sum_{T \subset R}((t-1)!(r-t)!/r!)r(a_i|T)$, where $r(a_i|T)$ as defined earlier is the marginal contribution of a_i given T and r, t denote size of sets R, T. We additionally define a conditional Shapley value: $\phi_{i,B}(R) = \sum_{T \subset R}((t-1)!(r-t)!/r!)r(a_i|T \cup B)$ that is conditioned on actions in B always chosen apriori. As a convention $\phi_{i,B}(R) = 0$ if $a_i \notin R$.

Our reward-shaping scheme computes a sampling-based weighted combination of conditional Shapley values, where the samples are generated by the RL agent interacting with the environment. We show this in the first item of the result below (proof in Appendix A) that additionally shows that our approach satisfies desirable properties. In order to state these formally, we make a number of mild assumptions and definitions: (1) the marginal contribution is at most 1, (2) a_i, a_j are complementary if they together contribute marginal value 1 but no value alone, (3) a_i, a_j are substitutes if they provide marginal value 1 on their own without the other action present but slightly lower than 1 together, and (4) ϵ-greedy in our context means choosing the action set given by current policy with probability $1-\epsilon$ and continuing to grow the atomic action set more by random atomic action selection with probability ϵ.

Theorem 1. *Let the current policy be choosing a set $B \subset \mathcal{A}$ of actions and $a_i \notin B$. Let $m = |\mathcal{A} \backslash B|$. With ϵ-greedy exploration, we have*

- *The expected reward assigned for atomic action a_i is $\sum_{j=0}^{m}(m-j+1)^{-1}\binom{m}{j}^{-1}\sum_{R \subset \mathcal{A} \backslash B : |R| = j} \phi_{i,B}(R)$.*
- *The max expected reward of a_i is $(1+1/m)H_{m+1} - 1$, where $H_n = \sum_{i=1}^{n} 1/i$ is the harmonic number.*
- *For complementary a_i, a_k, if $a_k \in B$ then a_i is assigned max expected reward and consequently chosen in future.*
- *For substitutes a_i, a_k, if $a_k \in B$ then a_i is assigned negative reward and is not chosen in future.*

IV. Reward Shaping Instances

We instantiate the general approach from the previous section for two problems: goods-matching (Example 1) and attack-graph games.

A. Goods-Matching Problem

We have already described this problem in Example 1. For sequence of atomic actions A let x_k denote the item added by the kth atomic action a_k, and X_k the set of items added by the first k atomic actions A_k. Then the marginal contribution reward for the single state s is given by $r(a_k \mid s, S_{k-1}) = P(X_k) - P(X_{k-1}) - c(x_k)$. Reducing the telescoping sum reveals that this satisfies Equation (1), with $r(\mathsf{pass} \mid s, S) = 0$. The difficulty for learning in the goods-matching problem is that, first, the price of an item may fluctuate wildly and hence a large number of samples are needed to figure out if purchasing a item would be profitable. Second, the middleman needs to learn the formation of the bundles (i.e. which items are complements or substitutes) to maximize his payoff. As the number of items increases in a bundle, the learning becomes more arduous.

B. Attack-Graph Game

To verify the efficacy and generality of our reward-shaping scheme for real-world problem solving, we consider a complex cyber-security domain, i.e the attack-graph games which have imperfect information and asymmetric players. We solve the attack-graph game following a generalized version of the *double oracle* (DO) procedure, termed the *policy-space response oracle* (PSRO) [12]. The PSRO involves alternate solving of single-agent RL problems for the two players. These RL problems of both players feature combinatorial action space in attack-graph games. In this section, we present how to apply our approach to this problem.

Attack Graph.: Attack graphs are a tool in cyber-security analysis employed to model the paths by which an adversary may compromise a system. Informally, attack graphs are directed acyclic graphs (DAGs) where the nodes represent security conditions (for example, vulnerability or root access on a machine) and edges correspond to exploits that can probabilistically *activate* a security condition. An exploit is considered *feasible* for the attacker only if the origin node of its edge is activated. Nodes are either *AND* or *OR* type, depending on whether all (*AND*) or at least one (*OR*) of the exploits on incoming edges need to be taken to activate it. Figure 1 is an example of an attack graph fragment.

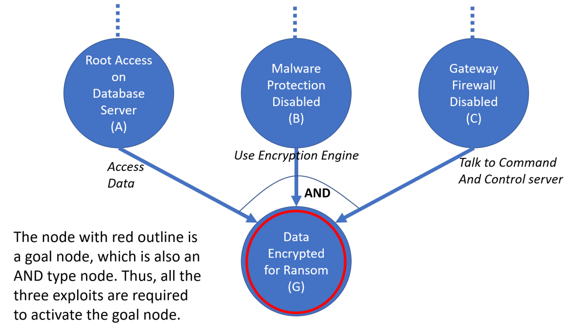

Root Access on Database Server (A)
Malware Protection Disabled (B)
Gateway Firewall Disabled (C)
Access Data
Use Encryption Engine
Talk to Command And Control server
AND
Data Encrypted for Ransom (G)
The node with red outline is a goal node, which is also an AND type node. Thus, all the three exploits are required to activate the goal node.

Fig. 1: Example fragment of an attack graph modeling a ransomware attack.

The attacker's aim is to reach *goal* nodes, which provides a large benefit R_A for the attacker and a substantial loss L for the defender. The defender's actions are to protect any node (for example, patch vulnerability or deny access). Both offensive and defensive actions are associated with a cost, denoted by c_a with respect to certain action a. The ability of the attacker to choose any subset of feasible exploits and of the defender to defend any subset of the nodes induces action spaces of combinatorial size.

Game Model.: Formally, the attack graph [13] is given by a DAG $\mathcal{G} = (V, E)$, where vertices $v \in V$ represent security conditions and edges $e \in E$ are labelled by exploits. An *attack-graph game* is defined by an attack graph endowed with additional specifications. The game is a *two-player partially observable stochastic game* that is played over a finite number of time steps T. At each time step t, the state of the game is given by the state of the graph, which is simply whether nodes are active or not (i.e., compromised by attacker or not), indicated by $s_t(v) \in \{0, 1\}$. It is assumed to be fully observable for the attacker while the defender receives a noisy observation $o_t(v) \in \{0, 1\}$ of the state, based on commonly known probabilities $P_v(o \mid s)$ for each node v. Positive observations are called *alerts*.

The attacker and defender act simultaneously at each time step t. The defender's atomic action is to defend a node, thus, the atomic actions are simply V. The defender's action space is 2^V, meaning it can choose to defend any subset of the nodes. The attacker's atomic action set varies with the graph state. Exploits on an edge are feasible only if the origin node of the edge is activated. Nodes without parents, called root nodes, can be attacked without preconditions. The attacker's action space at any time step is the power set of feasible atomic actions.

Defender actions override attacker actions, that is, any node v that is defended becomes inactive. Otherwise, active nodes remain active; attacks succeed not with certainty but with given probability.

Each *goal* node, v, carries reward $R_A(v)$ for attacker and penalty $P_D(v)$ for defender for all time steps in which v is active. Any atomic action a of an agent has a cost: $c_{a,D}(v)$ for nodes defended in case of defender; $c_{a,A}(v)$ for *AND* nodes selection and $c_{a,A}(e)$ for edges selection in case of attacker. For simplicity, we omit the argument (v or e) for action a in the notation. When obvious from context we also drop the subscript D and A, simply using v_a and c_a to denote the target node of a and the cost of action a respectively.

The defender's immediate loss (negative reward) is the cost of all its atomic actions (i.e., total cost of nodes defended), plus the penalty for goal nodes active after the moves. The defender's long-term payoff is the discounted expected sum of losses over time. Similarly, the attacker's long-term payoff is the discounted expected sum of immediate payoff, where the immediate payoff is the reward for active goal nodes minus the cost of atomic actions used in that time step.

A policy for either player (pure strategy in the game) maps its observations at any step to a set of actions. For the attacker, the mapping is from states to action sets. The defender only partially observes state, so its policy maps observation histories to action sets. In our implementation, we limit the defender to a fixed length h of past observations for tractability. Solving the game means finding a pair of mixed strategies (distribution over pure strategies), one for each player, that constitutes a NE.

Game-Solving via Single-Agent RL.: Briefly, in the DO framework, each player starts with an initial set of strategies (policies) and then the following steps are repeated: (1) Compute a Nash equilibrium (NE) of the current game; (2) Fix one player's strategy to the NE strategy (possibly mixed) and find a best-responding strategy (policy) of the other player; (3) If a better response policy is found for any player then add it to the strategy (policy) set of corresponding player and go to step 1 or else stop. For the second step, any single-agent RL method for computing a best- (or better-) response policy can be employed. Specifically, we use double DQNs [14] to find an approximate best response here. We demonstrate the benefits of our reward-shaping approach based on the second step wherein defender or attacker plays against various opponent's strategies and then learning performance is measured. We also show that our reward-shaping approach improves overall game learning quality.

Action-Selection Procedure.: To construct an action set for the single-agent RL problems, we follow the GASB method by [8] (see Section II) endowed with reward shaping. For completeness, a summary of GASB is in Appendix B. We first propose a variant of GASB where infeasible actions are filtered. This variant, which we call *Masking GASB* (MGASB), improves performance when available actions vary with states—as they do for the attacker in the attack-graph game; MGASB is same as GASB for defender. We choose MGASB without reward shaping as our baselines.

Reward Shaping Details.: We elaborate reward shaping details for both players due to the considerable literature on applications of attack graphs [13], [15], [16].

Algorithm 1 shows our reward-shaping approach for the attacker. In particular, if an attacker's atomic action a successfully compromises a goal node v_a (line 4), its immediate reward r_a (marginal contribution) is equal to the sum of negative cost $-c_a$ and reward of that goal node $R_A(v_a)$ (line 5). Otherwise, it is only equal to the negative cost (line 7). At each time step, the attacker also receives a reward, denoted by R_{ex}, from goal nodes compromised in previous time steps and which are not yet defended. R_{ex} can be zero if there are no goal nodes already compromised from the last time step or if a previously compromised node was defended. This extra reward is independent of the current action, and the attacker gets this reward irrespective of its current action. As the atomic action pass is present with all actions, we assign this extra reward to pass.

The dummy action pass should be chosen by the agent when there is nothing to gain by including more atomic actions. To enable the agent to learn when to choose pass, it needs to be associated with an appropriate immediate reward. In general, pass should get a zero reward, indicating that adding any other atomic action would have negative impact. Our attack-graph game has specific complications, as agents may receive extra rewards from actions in prior time steps. For example, the attacker continues to receive rewards R_{ex} (and defender a corresponding penalty) from compromised goal nodes until the goal node is defended. Conceptually, the immediate reward

Algorithm 1 Attacker reward shaping with M.C.

```
1: Build $attackSet$ and $defendSet$.
2: Both players deploy their action sets.
3: for action $a$ in $attackSet$ do
4:     if $a$ succeeds $\wedge\, v_a$ is a goal node then
5:         $r_a \leftarrow R_A(v_a) - c_a$
6:     else
7:         $r_a \leftarrow -c_a$
8: $r_{pass} \leftarrow R_{ex}$
```

Algorithm 2 Defender reward shaping with M.C.

```
1: Build $attackSet$ and $defendSet$; $A \leftarrow defendSet$
2: Both players deploy their action sets.
3: Calculate defender's penalty $L$ based on graph states.
4: for atomic action $a$ in $defendSet$ do
5:     if $v_a$ is a compromised node or under attack then
6:         $r_a \leftarrow -c_a$
7:         $A \leftarrow A \backslash \{a\}$
8: for atomic action $a$ in $A \cup \{\text{pass}\}$ do
9:     $r_a \leftarrow -c_a - \frac{L}{|A|+1}$
```

assigned to pass should reflect this excess because if an empty atomic action set was chosen (that is, pass was chosen first), the immediate reward of pass should indicate the reward of staying in current state given opponent's actions. Though this R_{ex} immediate reward for pass does not provide information about the relative contribution of atomic actions at the current time step, it is important as it is required for proper inter-temporal accounting of rewards and action selection given previous states.

Algorithm 2 shows our reward shaping for the defender. For the defender, the reward of an action depends both on nodes it protects through its atomic actions (*defendSet* in line 1) and on compromised goal nodes that it fails to protect through an absence of atomic actions. In order to account for the penalty from the absence of atomic actions (L in the algorithm), we distribute the penalty from such unprotected nodes to specific atomic actions present in the action (set $A \subseteq defendSet$ in the algorithm). In particular, any atomic action a that does protect a compromised node or prevent a successful attack (line 5) is excluded from A (line 7) and the immediate reward for such atomic actions is just a negative cost $-c_a$ (line 6). Then, the penalty L is equally distributed among the atomic actions in A and pass (line 9).

V. Experiments

We exhibit experimental results for large goods-matching problems and complex attack-graph games. For goods-matching problems, we apply our reward-shaping scheme to the popular auto-regressive approach for building action set [3] in environments with two scales. We show that the learning performance improves dramatically with our reward shaping scheme. We put experimental details in the Appendix A. In the remaining of this section, we show and analyze experimental details for attack-graph games.

For attack-graph games, we consider variations of experimental settings from two perspectives: players' rationality and environmental structures. We first select two types of opponent's strategies, e.g., a uniform strategy and a NE strategy, representing different degree of rationality of opponent. We demonstrate our approach is stable under different opponent's strategies.

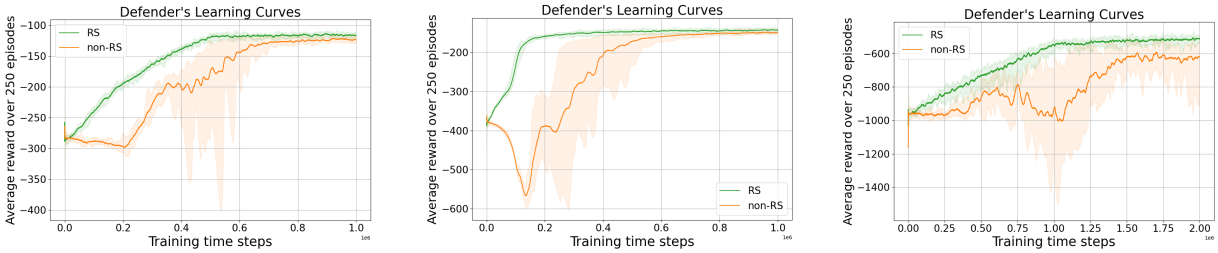
We test our RS scheme on three types of graphs with different structures: random graphs (RG), separate-layer graphs with *OR* nodes (SEP), and separate-layer graphs with *AND* nodes (SEPA).

These graphs represent different degrees of difficulty for defense and attack. Specifically, the separate-layer graph with *OR* nodes is the easiest graph to attack since a node can be activated if any one of its preconditions is satisfied. Correspondingly defense on such graph is the hardest. In contrast, the separate-layer graph with *AND* nodes is the easiest to defend on and the most difficult to attack. The difficulty of attacking a random graph lies in the middle as the attacker needs to attack a reasonable number of nodes to reach a goal node.

Results on Random Graphs.: As a base case, Figure 2a shows the defender's learning curve when training against an attacker with uniform random policy, where shadows indicate the maximum and minimum expected payoff in each epoch. The defender's performance under reward RS shows a steady improvement and a faster convergence. The expected payoffs of both methods are similar on convergence, though much more samples are needed with MGASB to reach this state. This highlights the contribution of our RS method towards improving sample efficiency.

Figure 2b shows the defender's performance against a strategic attacker who plays a NE policy (strategy) of the *combined game*. A combined game is the game with a strategy set that includes all strategies from different iteration of the DO. Comparing Figure 2a and 2b, we can observe that a strategic attacker lowers the defender's expected payoff, as expected. In this scenario, RS is shown to be even more helpful in the initial rounds of training, resulting in much faster convergence compared to MGASB. Also, in order to compute NE, learning a best-response policy in every iteration of DO happens in a different environment (due to changed opponent strategy). Thus, setting proper hyperparameters that guarantee convergence in all iterations of DO is not trivial in case of MGASB, more so because the learning curve in a single iteration has very high variance (as seen in the figure). Our RS scheme mitigates this problem to a large extent.

Results on Separate-Layer Graphs.: Figure 2c is for a three-layer graph with *OR* nodes and a uniform random policy attacker. As stated earlier, this is a difficult setting for the defender and we observe that convergence takes more training rounds. RS leads to much faster convergence, on average requiring 500,000 fewer time steps and with much less variance across runs. In contrast with the results on random



(a) RG: Def RS vs Uni. Att

(b) RG: Def RS vs NE Att

(c) SEP: Def RS vs Uni. Att

Fig. 2: Experimental Results

graphs, we observe an improved expected utility. We find that the set of atomic actions that the defender needs to build is large in separate-layer graphs, which exacerbates the two issues that we identified with combinatorial actions. Hence, the performance fluctuates tremendously for MGASB.

Attacker's Results.: Analogous to the defender's performance under RS, the attacker's performance exhibits a steady improvement and converges faster in both scenarios. We described attacker's learning performance and our analysis in Appendix E, where we also shed light on which among the two issues affects the performance of reward-shaping more and show other results on SEPA graph.

Better Learning Yields Stable NE.: To measure the contribution of RS in learning a stable NE at the end of DO iterations, we compare the performance of the resulting strategies at NE by measuring their regrets with respect to the NE of the combined game. The defender's mean regret with RS is 27.68 whereas for MGASB it is 19.04, which indicates that RS encourages strategy exploration and improves NE stability. We observe that without reward shaping GASB fails to learn a better response that should have been learned at certain rounds, which decreases the stability of NE.

VI. Conclusion

We studied efficient building of action sets with DRL for problems with combinatorial action space. We identify the *homogeneous-action* and *sparse-reward* issues in this domain, and propose a novel reward-shaping approach that credits each action with its marginal contribution towards the total reward of an action set. Our results show substantial improvement of policy learning where primitive actions have a diverse level of contributions to optimal action sets.

References

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[2] OpenAI, "OpenAI Five," https://openai.com/blog/openai-five/, 2018, accessed 2018-06-25.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, pp. 350–354, 2019.

[4] J. He, M. Ostendorf, X. He, J. Chen, J. Gao, L. Li, and L. Deng, "Deep reinforcement learning with a combinatorial action space for predicting popular Reddit threads," in *Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 1838–1848.

[5] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor, "Learn what not to learn: Action elimination with deep reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 3562–3573.

[6] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *15th ACM Workshop on Hot Topics in Networks*, 2016, pp. 50–56.

[7] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser *et al.*, "StarCraft II: A new challenge for reinforcement learning," arXiv, Tech. Rep. 1708.04782v1, 2017.

[8] M. Wright, Y. Wang, and M. P. Wellman, "Iterated deep reinforcement learning in games: History-aware training for improved stability," in *20th ACM Conference on Economics and Computation*, 2019, pp. 617–636.

[9] C. Tessler, T. Zahavy, D. Cohen, D. J. Mankowitz, and S. Mannor, "Action assembly: Sparse imitation learning for text based games with combinatorial action spaces," *arXiv preprint arXiv:1905.09700*, 2019.

[10] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," in *32nd AAAI Conference on Artificial Intelligence*, 2018, pp. 4131–4138.

[11] L. S. Shapley, "A value for n-person games," *Contributions to the Theory of Games*, vol. 2, no. 28, pp. 307–317, 1953.

[12] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel, "A unified game-theoretic approach to multiagent reinforcement learning," in *31st Annual Conference on Neural Information Processing Systems*, 2017, pp. 4190–4203.

[13] E. Miehling, M. Rasouli, and D. Teneketzis, "Optimal defense policies for partially observable spreading processes on Bayesian attack graphs," in *2nd ACM Workshop on Moving Target Defense*, 2015, pp. 67–76.

[14] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *30th AAAI Conference on Artificial Intelligence*, 2016, pp. 2094–2100.

[15] T. H. Nguyen, M. Wright, M. P. Wellman, and S. Singh, "Multi-stage attack graph security games: Heuristic strategies, with empirical game-theoretic analysis," *Security and Communication Networks*, vol. Article ID 2864873, 2018.

[16] K. Durkota, V. Lisỳ, B. Bošanskỳ, and C. Kiekintveld, "Optimal network security hardening using attack graph games," in *24th International Joint Conference on Artificial Intelligence*, 2015, pp. 526–532.

[17] K. Durkota, V. Lisỳ, C. Kiekintveld, B. Bošanskỳ, and M. Pěchouček, "Case studies of network defense with attack graph games," *IEEE Intelligent Systems*, vol. 31, no. 5, pp. 24–30, 2016.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.