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 cybersecurity 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 \( S \) and a set of atomic actions \( A \). The action taken in state \( s \in S \) is denoted by \( A \) and \( A \in 2^A \) is a subset of the atomic actions. Clearly, the action space \( 2^A \) is combinatorial in nature. The immediate reward is \( R(s, A) \) and the goal is to learn a policy \( \pi : S \rightarrow 2^A \) 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-
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 $r(a_{k+1} | s, S_k)$, 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:

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

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.

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 $\phi_i(R) = \sum_{T \subseteq R} ((t-1)!(r-t)!(r!/t!))r(a_i|T)$, 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: $\phi_{i,B}(R) = \sum_{T \subseteq R} ((t-1)!(r-t)!(r!/t!))r(a_i|T \cup B)$ 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) $\epsilon$-greedy in our context means choosing the action set given the current policy with probability 1 - $\epsilon$ and continuing to grow the atomic action set more by random atomic action selection with probability $\epsilon$.

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

- The expected reward assigned for atomic action $a_i$ is $\sum_{j=0}^{m} (m-j+1)^{-1}(j)^{-1} \sum_{R \subseteq A \setminus B : |R| = j} \phi_{i,B}(R)$.
- The max expected reward of $a_i$ is $(1 + 1/m)H_{m+1} - 1$, where $H_m = \sum_{i=1}^{m} 1/i$ 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 $r(\text{pass} \mid s, S) = 0$. 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 an 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.

![Image of an attack graph](image-url)
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 \( G = (V, E) \), 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 \( e_a \) 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 (polices) 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 ∧ 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 ← 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: \( A \leftarrow A \setminus \{a\} \)
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 (\( \text{defendSet} \) 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 \( A \subseteq \text{defendSet} \) 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
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


APPENDIX

Proof of Theorem 1. Clearly pass is equally probably among left-over actions. Let Pass$_j$ be the even that pass is called when additional $j$ actions (on top of $B$) have been chosen. Thus, $P(\text{Pass}_j) = 1/(m - j + 1)$. Let $X$ be random variable for expected reward assigned of $a_i$. Then, $E[X] = \sum_{j=0}^{m} E[X|\text{Pass}_j] P(\text{Pass}_j)$.

Then, focusing on $E[X|\text{Pass}_j]$, there are $\binom{m}{j}$ subsets of size $j$ that can be chosen, each is equiprobable due to the random choice. But, only those subsets that contains $a_i$ contribute to reward of $a_i$. If $a_i \in R$ then $E[X|R]$ is $\phi_{i,B}(R)$ because there are $j!$ different sequence of choosing $R$ and averaging over the marginal contribution, by definition, we obtain the Shapley value. Thus, $E[X|\text{Pass}_i] = (\binom{m}{j})^{-1} \sum_{R:j(R) = j} \phi_{i,B}(R)$, which gives the first result.

Next, $\phi_{i,B}(R)$ is non-zero only if $a_i \in R$, which happens for $\binom{m}{j-1}$ cases. With max marginal contribution of 1, we have $\phi_{i,B}(R) = 1$ for $a_i \in R$. Thus, the expected reward is $\sum_{j=0}^{m} (m - j + 1)^{-1} \binom{m}{j}^{-1} \binom{m}{j-1}$. This simplifies to $(1/m) \sum_{j=0}^{m} (m - j + 1)/(m - j + 1) - 1 = (1+1/m)H_{m+1} - 1$. By definition of complementary actions $a_i$ has marginal contribution 1 always (since $a_k \in B$). Thus, it obtain max reward and the RL policy will ultimately chose $a_i$ over pass which has 0 reward (given current policy which choose $B$).

Finally, by definition of substitutes, $a_i$ has marginal contribution slightly smaller than 0 always (since $a_k \in B$). Thus, it obtains negative reward and the RL policy will ultimately chose pass before $a_i$.

We follow the attack graph model of [13]. See Figure 1 for an example attack graph fragment. As described in the main paper, an attack graph 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 $P_e(o = 0 | s = 1)$ and $P_e(o = 1 | s = 0)$ 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 for any time step is $2^V$, meaning it can choose to defend any subset of the nodes. The attacker’s action set varies with time and is based on current 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. For the special case of attacking an AND node, the attacker’s atomic action is treated as attacking a node rather than choosing exploits on all incoming edges. Thus, the attacker’s atomic actions can be viewed as selecting edges (feasible exploits) for attacking OR nodes or selecting nodes from AND nodes whose parent nodes are all active. 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; an AND node $v$ that is attacked becomes active with probability $P(v)$, and any OR node becomes active based on the success probabilities $P(e)$ of attacked edges.

Each goal node, $v$, carries reward $R_A(v)$ for attacker and penalty $R_D(v)$ 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 $c_a$ and $c_e$ to denote the target node of $a$ and the cost of action $a$ respectively.

The defender’s loss (negative reward) at any time step 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 payoff per time step, where the per time step 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.

There is a considerable literature on game-theoretic analysis of attack graphs [13], [15]–[17]. In particular, [8] apply DRL within a search for Nash equilibria (NE) in this domain. A key challenge for DRL in this setting is dealing with the large action spaces available to both attacker and defender, and the quality of policy learned by DRL bears directly on the stability of game-theoretic solutions. Therefore, in this study, we focus on improving the performance of DRL in such scenario with the goal of facilitating both the local learning and the game analysis.

A. Goods-Matching Problem Experiments

We consider the goods-matching problem of two different scales. One has 12 items in which the cardinality of a bundle is small while another has 42 types and a bundle contains more items. The numbers of possible combinatorial actions of these two scales are $2^{12}$ and $2^{42}$ respectively. We consider baselines from two categories in the Related Work section respectively. First, we find that methods in the independent action selection
could fail when the number of actions is extremely large. For example, searching effective combinatorial actions, i.e., learning superior actions and the dependency among actions, in the $2^{42}$ scale with sampling or action elimination is nearly impossible. Therefore, we focus on the baseline from the second category that is capable to solve problems with large action space. Specifically, we build the action set in an auto-regressive manner following [3] in which the atomic-action selection is based on atomic actions that have been added to the action set. The probability of an atomic action to be added in the order $l$ is given by $\pi_\theta(a_l|a^{<l})$ where $\pi_\theta$ is a policy parametrized by $\theta$ and $a^{<l}$ indicates the set of atomic actions that were added before the order $l$. We implement this baseline with the on-policy algorithm PPO [18] and compare the performance before and after applying our reward-shaping method.

![Figure 3: Goods Matching, 12 items (top), 42 items (bottom).](image)

In Figure 3, the orange line shows the averaged learning curve for each scale enhanced by RS compared to the blue line in which a set of atomic actions is viewed as a whole. In both figures, we observed a fast convergence when crediting atomic actions with their marginal contributions. Moreover, we found that for the problem with 42 items, the DRL learner can barely figure out the matching items and thus resulting in a lower utility. This indicates that when atomic actions are highly correlated, regular DRL learner fails to learn sometimes and may need more samples to understand the interplay among atomic actions.

**B. Example of Reward Shaping in Attack-graph Game**

**Example 2.** Suppose the action comprises the three atomic actions defending the three nodes $A, B, C$ (Figure 1) with cost $10, 20, 30$ respectively. Also, in hindsight, when these atomic actions are taken action $A$ protected a node that the attacker attacked otherwise the penalty for this node would be 100. However, action $B$ and $C$ protected nodes that were not attacked. Further, assume there is only one compromised goal node $Z$ (not shown in the fragment) that led to a 900 penalty for the defender as this node was not protected. Applying the marginal contribution approach to this attack graph example, we get

$$r_A = -c_A = -10 \quad r_B = -c_B - \frac{900}{3} = -320 \quad r_C = -c_C - \frac{900}{3} = -300 \quad r_{\text{pass}} = -\frac{300}{3}$$

**C. Algorithms**

To deal with combinatorial actions in DRL, [8] proposed greedy action set building, as mentioned in Section II. In their method, the DQN representing the policy has inputs for the conditioning state, which in the attack-graph game encodes the network state $s_t$ for the attacker, or alerts over horizon $h$ for the defender. In addition, GASB requires input nodes for each atomic action, to represent the partial action set under construction. GASB works by iteratively building the action set, at each iteration selecting an atomic action to add conditioned on the state and partial action set. It employs a special output option pass to indicate that the set construction is complete. GASB terminates if an illegal (infeasible or already present) atomic action is chosen, or at random in each iteration with a small probability (random pass). The framework of GASB is illustrated in Figure 4 based on the example in Figure 1. In GASB as originally proposed, since the reward feedback is associated with the set as a whole, both homogeneous-action issue and sparse-reward issue exist.

To create a strong baseline for our reward-shaping approach, we 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.

Algorithm 3 presents the procedure for generating the attacker’s action using MGASB. The attacker policy DQN $\phi_{\text{att}}$ outputs a choice $x$ based on the current attacker DQN input vector $\text{attObs}$. $x$ can represent pass or an atomic action $a$ to add to the $\text{attackSet}$ and then $\text{attObs}$ gets updated with the updated $\text{attackSet}$. $x$ is legal only if it is pass or a feasible atomic action and it is not already in the $\text{attackSet}$. We revise the original GASB to add a Mask (line 3), which filters out infeasible actions given a state by subtracting a large number from the Q-values of infeasible actions (keeping those of legal actions unchanged), in both the loss function and output layer of the DQN. Similar masking approaches have also been employed in previous DRL work [7]. As a further variation on GASB, we remove the random termination (i.e, selection of pass with a fixed minimum probability) to encourage more exploration. Since the defender does not have...
Infeasible actions in our problem, MGASB for the defender is just GASB without random pass.

**Algorithm 3 Attacker’s Masking GASB**

Require: attObs

1: attackSet ← ∅
2: do
3: \( x \leftarrow \text{Mask}(\phi_{\text{att}}(\text{attObs})) \)
4: if \( \neg \text{isDup} \land x \neq \text{pass} \) then
5: \( \text{attackSet} \leftarrow \text{attackSet} \cup \{x\} \)
6: update(\text{attObs}, \text{attackSet})
7: while \( \neg \text{isDup} \land x \neq \text{pass} \)
8: return \text{attackSet}

Fig. 4: Flowchart describing the GASB. The input includes the current action set with 4 bits. Initially the bits are set to zeros to indicate that the action set is empty. Then the bits flip if corresponding atomic actions were added.

### D. Input Representation

We summarize the DQN input vectors for both players in Table I. The attacker’s input first includes its current observation where an observation bit indicates whether a node is active. Then we denote \( A_{\text{att}} \) as the set that contains all exploits: \( \text{AND} \) nodes plus edges to \( \text{OR} \) nodes. For each exploit, one bit indicates whether it is feasible and another indicates whether it is currently in the attack set. The defender’s input includes data from its previous observations. For each node and each time step, one bit indicates if an alert was observed on this node, one bit whether the node was defended last time, and one more bit to indicate if the node is in the current defense set. Finally, both attacker and defender have an input element counting how many time steps are left.

<table>
<thead>
<tr>
<th>Attacker Feature</th>
<th>Defender Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>isActive</td>
<td>hadAlert</td>
</tr>
<tr>
<td>canAttack</td>
<td>inDefenseSet</td>
</tr>
<tr>
<td>inAttackSet</td>
<td>wasDefended</td>
</tr>
<tr>
<td>timeStepsLeft</td>
<td>timeStepsLeft</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#Entry</th>
<th>#Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>V</td>
</tr>
<tr>
<td>(</td>
<td>A_{\text{att}}</td>
</tr>
<tr>
<td>(</td>
<td>A_{\text{att}}</td>
</tr>
</tbody>
</table>

TABLE I: DQN input vectors with entry counts in GASB.

### E. Reward Shaping for the Attacker

We test the attacker’s performance under RS on a random graph under two situations: high and low variance in the cost \( c_a \) of atomic action \( a \) (an exploit). If the scale of the cost for different exploits is large then the attacker needs to be more prudent in action selection; otherwise, it may suffer a loss even if a goal node is reached. Figure 5 and Figure 6 show the attacker’s performance against a uniform defender’s policy with high and low cost variances, respectively. Analogous to the defender’s performance under RS, the attacker’s performance exhibits a steady improvement and converges faster in both scenarios. We notice that with high cost variance the performance of RS becomes more stable, as the difference among atomic actions is more distinct. We further find that the homogeneous-action issue dominates the sparse-reward issue on our attack graphs, which we show in extra experiments in Appendix F.

Fig. 5: RG: Att R.S. vs Uni. Def.

Fig. 6: RG with low cost variances.

For a three-layer graph with \( \text{AND} \) nodes, even an uniform defender can force a learning attacker to choose not to attack at all. Thus, leveraging the advantage of graph structure, defender reaches similar performance using both MGASB and our method as can be seen in Figure 7. Hence, RS does not offer additional advantage in this setting where any set of atomic actions is good enough.

### F. The Effect of Homogeneous Actions

In previous sections, we identified the homogeneous-action and sparse-reward issues in DRL problems with combinatorial
action space. One natural question is about the interplay of these issues and what would happen if one of these two issues is mitigated. First, note that these two issues co-exist in attack-graph games. For example, in the experiment with separate layers (Figure 2c), the defender holds a large action set (sparse-reward issue) and atomic actions are viewed as homogeneous in GASB (homogeneous-action issue). Our reward shaping approach addresses both issues simultaneously without special consideration for each issue. Second, the homogeneous-action issue is influenced by the sparse-reward issue. For instance, the sparse-reward issue nearly disappears when only one atomic action is allowed in an action set and at the same time the homogeneous-action issue is also largely mitigated because players can easily identify the effect of the only atomic action in an action set.

To answer the question we raised in this sub-section, we design an artificially constrained experiment based on the random graph of Figure 6 where we keep both the scale and variance of the costs for different exploits small and rewards of different goal nodes similar. This design increases the proportion of atomic actions that are truly homogeneous for the attacker because of two effects. One is that from the reward-shaping perspective, the rewards assigned to different atomic actions are similar. Another is that many atomic actions have similar Q-values. For example, in the attack graph shown in Figure 8 where node E and F are goal nodes, given a uniform defender and node A and B are compromised, attacks on node C and node D have similar Q-values and hence are considered truly homogeneous.

Figure 9 shows the attacker’s learning curves in this setting. As discussed above, small scale and variance of the costs should reduce the benefit of reward shaping since atomic actions become more indistinguishable. Indeed, we observe that the two averaged learning curves are closer to each other compared to Figure 6. This implies when atomic actions become truly homogeneous, the advantage of reward shaping over GASB is reduced. This also demonstrates that the homogeneous-action issue is critical to address for our problem. Note that we cannot draw a general conclusion about which issue contributes more towards the unstable learning in DRL problems with combinatorial action space as that depends on the problem setting, for example, sparse-reward issue may become critical when the size of action set is large. But in terms of the size and structures of our attack graphs, the homogeneous-action issue is dominant.

G. Masking GASB

We compare the performance of Masking GASB with GASB for attacker, averaged over 10 runs against a fixed uniform defender. Figure 10 shows the expected payoffs of attacker with different GASB variants (vanilla GASB, GASB without random pass and MGASB). The attacker’s expected payoff improves after masking out infeasible atomic actions and removing random pass. These improvements enable the attacker to construct more effective actions in the attack graph game and cause defense against such to be harder. We show the efficacy of our reward-shaping scheme for defending against this powerful attacker next.
TABLE II: Parameters of Goods-Matching Problem

<table>
<thead>
<tr>
<th>#items</th>
<th>12/42</th>
</tr>
</thead>
<tbody>
<tr>
<td>#items in bundles</td>
<td>3/≥3</td>
</tr>
<tr>
<td>Price of item</td>
<td>Sample from Normal($\mu_i$, $\sigma$)</td>
</tr>
<tr>
<td>$\mu_i$</td>
<td>Uniform(3,6)</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>0.9</td>
</tr>
<tr>
<td>Cost of item</td>
<td>$\mu_i - 0.5$</td>
</tr>
<tr>
<td>Complements Bonus</td>
<td>5</td>
</tr>
</tbody>
</table>

TABLE III: Random Graph (RG) Parameters

| #Nodes | 30 |
| #Edges | 100 |
| Costs | Uniform in [0, 1] for attacker; Uniform in [2, 4] for defender |
| Rewards | Uniform in [10, 20] |
| Penalties | Uniform in [7, 10] |
| #Goal Nodes | 6 |
| Activation Prob. | Uniform in [0.6, 1] |
| False Alarm Prob. | Uniform in [0, 0.2] |

TABLE IV: Separate-Layer Graph (SEP) Parameters

| #Nodes | 30 |
| #Edges | 100 |
| Costs | Uniform in [0, 1] for low scale; Uniform in [2, 5] for high variance; Fixed to 3 for low variance |
| Rewards | Uniform in [10, 20] |
| Penalties | Uniform in [7, 10] |
| #Goal Nodes | 3 |
| Activation Prob. | Uniform in [0.6, 1] |
| False Alarm Prob. | Uniform in [0, 0.2] |

TABLE V: Investigation on the Variance of Costs

J. DQN Hyperparameters

| hidden layers | 2 FC layers, size 256 |
| activations | tanh per hidden layer |
| learning rate | $5 \times 10^{-5}$ |
| training discount factor | 0.99 |
| batch size | 32 |
| $\epsilon$ in exploration | linear 1.0 to 0.03 |
| Training Steps | RG: 700K for attacker; SEP: 2M for defender |
| DQN options | no prioritized replay; no param noise |

TABLE VI: Deep RL hyperparameters