Scalable Game-Focused Learning of Adversary Models: Data-to-Decisions in Network Security Games

Kai Wang, Andrew Perrault, Aditya Mate, Milind Tambe
Harvard University
{kaiwang,aperrault,aditya_mate}@g.harvard.edu,milind_tambe@harvard.edu

ABSTRACT

Previous approaches to adversary modeling in network security games (NSGs) have been caught in the paradigm of first building a full adversary model, either from expert input or historical attack data, and then solving the game. Motivated by the need to disrupt the multibillion dollar illegal smuggling networks, such as wildlife and drug trafficking, this paper introduces a fundamental shift in learning adversary behavior in NSGs by focusing on the accuracy of the model using the downstream game that will be solved. Further, the paper addresses technical challenges in building such a game-focused learning model by i) applying graph convolutional networks to NSGs to achieve tractability and differentiability and ii) using randomized block updates of the coefficients of the defender's optimization in order to scale the approach to large networks. We show that our game-focused approach yields scalability and higher defender expected utility than models trained for accuracy only.

KEYWORDS

Adversarial multi-agent learning; Game theory for practical applications

ACM Reference Format:
Kai Wang, Andrew Perrault, Aditya Mate, Milind Tambe. 2020. Scalable Game-Focused Learning of Adversary Models: Data-to-Decisions in Network Security Games. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 13 pages.

1 INTRODUCTION

Many real-world security problems present the challenge of how to allocate limited resources to large number of important targets, including infrastructure [14], transportation systems [33], urban crime [48], and web security [43]. *Stackelberg security games* (SSGs) are frequently used to study the interaction between defender and attacker and optimally allocate the security resources accordingly. *Network security games (NSGs)* [11, 38, 44], a natural extension of SSGs, describe a strategic adversarial interaction between an attacker and a defender on a graph. The attacker's goal is to take a path from a starting location to a target without being caught by the defender. The defender declares (i.e., attacker surveils) a mixed strategy of how she will deploy her security resources to the edges of the network. NSGs are relevant in many real-world settings such as wildlife conservation [9, 25], infrastructure protection [21], and nuclear material smuggling [29, 34].

One key challenge in applying NSGs in the real world is learning an adversary's behavior from historical data. Past works [1, 5, 30] in security games have shown that constructing bounded rationality adversary models from data greatly improves performance of deployed models because attackers often behave quite differently from how rational models would suggest. A particular motivation for this paper is wildlife smuggling [10, 37, 49], a natural NSG domain where large amounts of historical attack data is available in the form of past seizures.

Almost all previous work on security games approaches the problem of adversary modeling by first building a full adversary model that aims to predict the adversary behavior as accurately as possible [2, 7, 9, 32]. In early work, the judgments of human experts were used to estimate the adversary's preferences [40]. Later, in domains where historical attack data was available, machine learning was used to construct models instead (starting from Letchford et al. [24]). In NSGs, building an adversary model to maximize accuracy has several key limitations. First, the model is selected without any consideration of the impact of errors downstream. Prediction errors on paths that are frequently taken by the adversary have a large impact on defender utility, but are weighted the same as errors on paths that are rarely taken. Secondly, standard adversary models require human feature engineering to apply to NSGs due to a great variety of paths from the attacker's starting location to each potential target [13, 16, 17, 46]. Once the adversary model is determined, the following defender utility maximization problem can be solved by any optimization techniques, including bilevel optimization [25], branch and cut [12], and double oracle [21].

Our approach represents a fundamental shift: we take an end-to-end, game-focused approach, focusing on learning a model that yields a high defender utility. More specifically, we take the downstream defender utility maximizatoin problem into account while learning the adversary model. To that end, we use a graph convolutional neural network architecture to learn the adversary's behavior, which allows us to overcome both of the issues of prior work. First, assuming we can differentiate through the defender's optimization problem, we can train the entire model end-to-end because the predictive model is differentiable, i.e., to take the optimization problem into account while training. Second, the graph convolutional network automatically extracts features from the graph, meaning that hand engineering is not necessary. Nevertheless, several challenges must be overcome to implement this approach: principally, poor scalability of naive end-to-end training and non-convexity of the game-focused objective.

A summary of our contributions is as follows: first, we construct a *graph convolution*-based adversary model for NSGs. This model is fully differentiable, does not require manual selection of path features, and transmits target value information over the network.

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

Second, we develop a randomized block update scheme for differentiating through optimization problems, whose computation time is usually more than quadratic in terms of the number of variables due to the computation of Hessian matrix and matrix inversion. Such computational issue is especially influential for optimization problems with a huge number of variables, which is commonly seen in NSGs as every edge corresponds to one individual decision variable. In these cases, randomized block update can largely reduce the time complexity. We further provide an approximation guarantee relative to the complete derivatives, and we show empirically that our approach greatly improves scalability. We also show that through judicious use of the standard predictive loss as regularization, we can escape local minima in the end-to-end loss function.

Related Work. There is a rich literature on learning adversary behavior models in Stackelberg security games (SSGs) (starting from Letchford et al. [24]), but learning in NSGs has received much less attention. While SSGs generalize NSGs, the scalability concerns are quite different because reducing NSGs to SSGs may create exponentially many targets—one for each path to the target in the NSG. Thus, applying standard attacker bounded rationality models, such as quantal response (QR) [26, 27] and subjective utility quantal response (SUQR) [32] is nontrivial. Yang et al. [46] and Ford et al. [13] reduced NSGs to SSGs by considering each individual path as an attacker pure strategy. Their approach scales poorly, creating exponentially many paths in many networks. It also relies on hand-crafting path features that capture adversary behavior well. Other authors have developed models that use Markovian dynamics to model the attacker. Gutfraind et al. [16] and Abbasi et al. [2] assume the attacker does not receive any information beyond the neighboring nodes—attackers do not make any decisions that are more long term than a single timestep. Gutfraind et al. [17] takes the opposite approach: attackers follow a path that minimizes some cost (such as the risk of being caught) with randomness in the individual decisions. This adds some global information, but requires the model designer to specify the choice of cost function in advance.

Past work in adversary modeling in SSGs has viewed the problem of constructing an adversary model and solving the defender's optimization as completely separate problems and does not consider the impact of errors in the defender model on the quality of the optimization outcome, with a few exceptions. Sinha et al. [39] and Haghtalab et al. [18] relate the predictive accuracy of the learned model to the defender's expected utility. In the case of Haghtalab et al., this view motivates the use of a non-standard loss function to achieve better utility. However, even these papers take a fundamentally two-stage approach: the model is trained independently of any information about the game itself, such as the defender's utilities. Perrault et al. [36] takes a game-focused approach to SSGs, but the issues that arise in NSGs are different and require a greater focus on scalability.

A major challenge in our work is differentiating through the nonconvex defender optimization problem. Recent work has developed general approaches for differentiating convex problems [3]. Perrault et al. [36] present an approach for a limited class of nonconvex problems. Our setting is challenging in two ways. First, we have a decision variable for each edge in the network and these approaches scale poorly (more than quadratically) in the number of variables. Second, our setting is more severely nonconvex than that of Perrault et al.

2 BACKGROUND

Stackelberg Security Games. A *Stackelberg security game (SSG)* [40, 47] is a two-player sequential game. The defender aims to protect a set of targets T with limited budget b which can only protect up to b targets. Each target $t \in T$ is associated with a defender penalty $U^d(t) \leq 0$ and an attacker reward $U^a(t) \geq 0$ when the target is successfully attacked. For simplicity, we assume there is no reward and penalty when the attacker is caught or fails to reach the target. Once the defender commits to her mixed strategy, the attacker can conduct surveillance to observe the defender's mixed strategy and choose one target to attack accordingly. We denote the defender's mixed strategy by $\mathbf{x} \in \mathbb{R}^{|T|}$, where $0 \leq \mathbf{x}_t \leq 1$ denotes the marginal probability that target t is protected. The budget constraint can be written as $\mathbf{1}^\top \mathbf{x} \leq b$. On the attacker side, we use $\mathbf{q}(\mathbf{x}, \xi)$ to represent the attacker's behavior, where $\mathbf{q}_t(\mathbf{x}, \xi)$ (or $\mathbf{q}_t$ if there is no ambiguity) is the probability of attacking target t, and ξ is the available features revealed to both the defender and the attacker, e.g., the attacker payoff value $U^a(t)\ \forall t \in T$ can be considered as a feature. Notice that $\mathbf{q}$ is a function of the defender strategy $\mathbf{x}$ and the feature ξ, which implies that the attacker can be reactive to the defender strategy and select the target based on the underlying feature. We can write the defender's utility function as:

$$\mathrm{DefU}(\mathbf{x}; \mathbf{q}) = \sum\nolimits_{t \in T} \mathbf{q}_t(\mathbf{x}, \xi) U^d(t)(1 - \mathbf{x}_t). \tag{1}$$

This includes the case where the attacker is fully rational, where $\mathbf{q}_t(\mathbf{x}, \xi) = 1$ if $t = \arg\max_{t' \in T}(1 - \mathbf{x}_{t'}) U^a(t')$ else 0.

Bounded Rationality in SSGs. Quantal response (QR) [26] models the attacker's behavior by setting the probability that each target is attacked to be proportional to the exponential of its payoff scaled by a constant. *Subjective utility quantal response (SUQR)* [32], which fits data better than QR in practice, sets the probability proportional to the exponential of a subjective utility or an attractiveness function of the attacker:

$$\mathbf{q}_t(\mathbf{x}, \xi) \propto \exp(-\omega \mathbf{x}_t + \Phi(t, \xi)), \tag{2}$$

where $\omega > 0$ is a constant representing the attacker's risk aversion and $\Phi(t, \xi)$ denotes the subjective utility of target t given feature ξ.

Network Security Games. Network security games (NSGs) [11, 31] are SSGs played on a graph structure. Given an undirected (or directed) graph $G = (V, E)$, the defender allocates a limited number of checkpoints along edges in E, while the attacker tries to find a path from a source to a target without being caught. We divide the set of all vertices V into targets $T = \{t_1, t_2, ..., t_{|T|}\}$ and non-targets $S = \{s_1, s_2, ..., s_{|S|}\}$ (or potential sources). At each time, the attacker appears in one potential source $s \in S$ drawn from a given prior distribution $\pi \in \mathbb{R}^{|S|}$. From the defender's perspective, the defender strategy $\mathbf{x}_e\ \forall e \in E$ is the marginal probability of covering edge e. Similarly, the defender has a limited number of resources b to protect the targets.

We use $\alpha = \{v_1, v_2, ..., v_{|\alpha|}\}$ to denote a path which starts from a source $v_1 \in S$ and ends with a target $v_{|\alpha|} \in T$. We use $\mathcal{A}$

to denote the set of all possible paths from any source to any target, which could be exponentially many or infinitely many when the graph contains any cycle. Similar to SSGs, let $U^d(t)$ be the defender's payoff when the target t is attacked successfully and U^d_{caught} be the defender's payoff when the attacker is caught. Let $U^d = \{U^d(t_1), ..., U^d(t_{|T|}), U^d_{\text{caught}}\} \in \mathbb{R}^{|T|+1}$ denote the defender's payoff vector. In addition, we assume each node $v \in \mathcal{V}$ has a node feature vector $\xi_v \in \mathbb{R}^D$ consisting of characteristics of node v, e.g., the attacker payoff of the current node $U^a(v)$ if $v \in T$. We use $\xi \in \mathbb{R}^{|V| \times D}$ to denote all the node features in graph G.

Bounded Rationality in NSGs. In this paper, we assume the attacker to be boundedly rational, where the attacker's behavior is characterized by a function $\mathbf{q}(\mathbf{x}, \xi)$, where $\mathbf{q}_\alpha(\mathbf{x}, \xi)$ represents the probability of choosing path α under coverage $\mathbf{x}$ and feature ξ. Given the coverage $\mathbf{x}$, we can compute the defender expected utility:

$$\text{DefU}(\mathbf{x}; \mathbf{q}) = \sum\nolimits_{\alpha \in \mathcal{A}} \mathbf{q}_\alpha(\mathbf{x}, \xi) U^d(\alpha) \prod\nolimits_{e \in \alpha} (1 - \mathbf{x}_e), \tag{3}$$

where $U^d(\alpha) = U^d(t)$ is the defender utility when the attacker successfully passes through α to attack its target t.

The difference between Equation 1 and 3 is that there are multiple layers of protection along the path α. Therefore the probability of successfully attacking a target is the product of all the success probabilities of crossing each edge e in the path. The defender's optimization problem is generally hard. For example, if the function $\mathbf{q}(\mathbf{x}, \xi)$ is given by full rationality restricted to only polynomial many paths $\mathcal{A}$, the defender optimization problem is NP-hard [21]. Furthermore, the set of all possible paths $\mathcal{A}$ could be exponentially large or infinitely many when there is any cycle.

Graph Convolutional Networks. There has been much recent attention paid to *graph convolutional networks (GCNs)* [19, 22, 28]. Given a graph, the convolutional layers in GCNs can transmit information through message passing, which allows information to propagate to distant nodes and be aggregated in a non-linear fashion. GCNs are much more expressive than hand-crafted features. In this paper, we apply GCNs, parameterized by θ, to map each node $v \in V$ and the entire node features ξ with graph structure to a scalar $\Phi(v, \xi; \theta)$, which represents the extent that the attacker is "pulled" toward that node. The message passing in GCNs is similar to the information gathering conducted by the adversary, where a rough understanding of faraway targets is available to the adversary.

3 ADVERSARY MODEL

Our attacker model is Markovian—the probability of using a path α can be decomposed into the product of transition probabilities:

$$\mathbf{q}_\alpha(\mathbf{x}, \xi) = \prod\nolimits_{e \in \alpha} \mathbf{q}_e(\mathbf{x}, \xi). \tag{4}$$

Motivated by the SUQR model, we propose a **local SUQR** model, which assumes the probability that the attacker moves from u to v using edge $e = (u, v)$ is proportional to $exp(-\omega \mathbf{x}_{u \to v} - \eta \mathbf{y}_v + \Phi(v, \xi; \theta))$ $\forall v \in N_{\text{out}}(u)$. $\Phi(v, \xi; \theta)$ represents the subjective utility or attractiveness of node v parameterized by θ, which can be learned by GCN. $\mathbf{y}_v$, with a weight $\eta \geq 0$, represents the downstream future risk or coverage perceived by the attacker at node v. In other words, the attacker tends to move toward the target with higher attractiveness $\Phi(v, \xi; \theta)$, but avoids using the edge $e = (u, v) \in E$ with higher coverage $\mathbf{x}_{u \to v}$ and avoids moving towards nodes v with higher future risk $\mathbf{y}_v$.

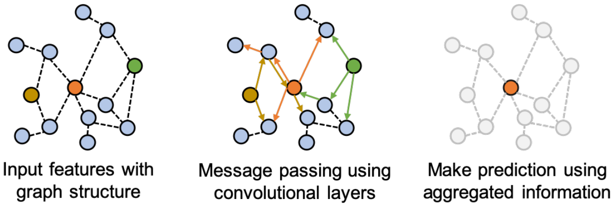

Input features with graph structure
Message passing using convolutional layers
Make prediction using aggregated information

Figure 1: The convolutional layers of GCNs can propagate and aggregate information in a non-linear fashion. In NSGs, such message passing ability corresponds to the attacker's ability of conducting surveillance to neighbor nodes.

Given a defender coverage strategy, there are many heuristic ways to obtain a measure of future risk. For example, we can follow the above Markovian behavior without the effect of the future risk, where the probability of being caught can be analytically computed efficiently. Another heuristic is the shortest distance to any target, as suggested by Gutfraind et al. [17]. The only restriction put on the choice of the future risk is differentiability.

We can compute the transition probability from u to any $v \in N_{\text{out}}(u)$ as:

$$\mathbf{q}_{u \to v}(\mathbf{x}, \xi; \theta) = \frac{exp(-\omega \mathbf{x}_{u \to v} - \eta \mathbf{y}_v + \Phi(v, \xi; \theta))}{\sum_{v' \in N_{\text{out}}(u)} exp(-\omega \mathbf{x}_{u \to v'} - \eta \mathbf{y}_{v'} + \Phi(v', \xi; \theta))}. \tag{5}$$

Unlike previous boundedly rational models [13, 46], we do not need to enumerate all the feasible paths, which could be exponentially large. Unlike the nonreactive Markovian model [16], our model is reactive to the defender's strategy. Unlike Gutfraind et al. [17], we are not limited to noisily following a shortest path.

In local SUQR, the path structure is automatically encoded in the reactive Markovian behavior. Since the edge coverage effect is involved in the transition probability, the probability of taking a path is also exponentially proportional to the total coverage along the path, which is also included in other bounded rational models [13, 46]. The flexibility and the generalizability of the attractiveness function allow us to apply any graph learning algorithms to extract the adversary behavior. Compared to previous hyperparameters tuning models, our model is more expressive and can adapt to a broader range of adversary behavior.

4 PROBLEM STATEMENT

For each instance, a directed graph $G = (V, E)$ with node features ξ is presented to both the defender and the attacker. The attacker has a hidden rationality function $\mathbf{q}^*$, which is a function of node features ξ and the defender coverage $\mathbf{x}$. The defender first chooses a coverage $\{\mathbf{x}_e\}_{e \in E}$ under the budget constraint $\mathbf{1}^\top \mathbf{x} \leq b$. The attacker observes $\mathbf{x}$ and then behaves based on his own rationality function $\mathbf{q}^*$. We assume that the defender has access to historical play between the defender and the attacker, which can be used to form an estimate of the adversary behavior. The goal of the defender is to maximize the received expected reward.

5 TWO-STAGE LEARNING FOR NETWORK SECURITY GAMES

The main comparison of the remainder of the paper is between our GCN-based adversary model implemented as two-stage vs. our game-focused methods. Thus, we briefly describe the two-stage approach that we consider.

Predictive Model. A two-stage approach fits the GCN-based attractiveness function $\Phi(v, \xi) \forall v \in V$ to minimize the difference between predicted behavior $\mathbf{q}$ given by Equation 5 and the corresponding true attacker behavior $\mathbf{q}^*$. Given the attacker behavior $\mathbf{q}^*$ and a prediction $\mathbf{q}$, we can define the loss by either matrix norm or the KL-divergence of the path distribution inferred by two behaviors under previous coverage $\mathbf{x}$ and features ξ. These losses are generally infeasible to compute since there are infinite many possible paths. In practice, however, we often have paths sampled from the true behavior $\mathbf{q}^*$ we can use to approximately compute the KL-divergence between two behaviors. Given the choice of loss function $\mathcal{L}$, we can train a model $\mathbf{q}$ by minimizing the average loss:

$$E_{(\mathbf{x}, \xi, \mathbf{q}^*) \in D} \mathcal{L}(\mathbf{q}^*, \mathbf{q}; \mathbf{x}, \xi) \tag{6}$$

Prescriptive Model. Given a graph G, node features ξ, and predicted attacker behavior q, the defender's goal is to choose an optimal coverage $\mathbf{x}^*$ satisfied the budget constraint to maximize her own objective value.

When the defender strategy $\mathbf{x}$ is chosen, the attacker follows his own Markovian behavior $\mathbf{q}(\mathbf{x}, \xi)$. But due to the allocated coverage, the attacker will be caught with probability $\mathbf{x}_e$ when he passes through edge e. This can be cast as an absorbing Markov chain, where the probability of crossing an edge e is $\mathbf{q}_e(\mathbf{x}, \xi)(1 - \mathbf{x}_e)$, and the rest of the probability the attacker will be caught and turned into a dummy caught state v_{caught}. We also assume that once the attacker reaches either any terminal or caught state v_{caught}, the attacker cannot go back to any other states, i.e., these are absorbing states. Therefore, given a coverage $\mathbf{x}$, we can model the attacker's behavior as an absorbing Markov chain. We can analytically compute the corresponding defender utility. To align with the standard minimization formulation, we denote the *negative* defender utility by $f(\mathbf{x}, \mathbf{q})$. For ease of notation, we omit the presence of node features. The optimization problem is given by:

$$\begin{aligned} \min\nolimits_{\mathbf{x}} \quad & f(\mathbf{x}, \mathbf{q}) \\ \text{s.t.} \quad & \mathbf{1}^\top \mathbf{x} \leq b, \qquad 0 \leq \mathbf{x}_e \leq 1 \quad \forall e \in \mathcal{E} \end{aligned} \tag{7}$$

Unfortunately, the function f is neither convex nor submodular when the attacker is reactive. The standard approach is to apply constrained black-box optimization solvers to solve the problem, e.g., Sequential Least SQuares Programming (SLSQP) [6, 23].

6 NAIVE GAME-FOCUSED LEARNING FOR NETWORK SECURITY GAMES

In general, a good predictive model does not necessarily imply a high defender utility in the second stage. Sometimes a slightly inaccurate prediction might lead to a better final decision. This happens frequently especially when the predictive model cannot perfectly represent the ground truth. For example, in our case, the model relies on the Markovian assumption and SUQR assumption in Equation 5, which might not be able to fully recover the underlying attacker behavior.

Algorithm 1: Naive Game-focused Learning [36]

```
Input: Training data $D$, initialized $\mathrm{GCN}(\cdot, \cdot; \theta) : V \times \xi \to \mathbb{R}$
while until converge do
    for $(G, \mathbf{q}^*, \xi) \in D$ do
        Compute prediction $\mathbf{q}$ in Eq. 5 by $\Phi = \mathrm{GCN}(V, \xi; \theta)$
        Find optimum $\mathbf{x}^{\text{opt}}$ of Optimization 7
        $Q = \frac{\partial^2 f(\mathbf{x}, \mathbf{q})}{\partial \mathbf{x}^2}|_{\mathbf{x}=\mathbf{x}^{\text{opt}}}, p = \frac{\partial f(\mathbf{x}, \mathbf{q})}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}^{\text{opt}}} - Q\mathbf{x}^*$
        Re-solve QP: $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x} \text{ feasible}} \frac{1}{2}\mathbf{x}^\top Q \mathbf{x} + \mathbf{x}^\top p$
        Update $\theta$ by gradient $\frac{df(\mathbf{x}^*, \mathbf{q}^*)}{d\mathbf{x}^*} \frac{d\mathbf{x}^*}{dp} \frac{dp}{d\theta}$
Return: trained model $\mathrm{GCN}(\cdot, \cdot; \theta)$
```

Game-focused learning, instead, can directly optimize the final solution quality by back-propagating from the final solution quality all-the-way back to the predictive model. Game-focused learning has been proven to be able to outperform a standard two-stage learning approach [36], finding a shortcut to better final solution quality. However, the major issue of back-propagation is the non-differentiable optimization layer in the prescriptive state. Amos et al. [4] provides a method to differentiate through the optimization layer when the optimization program is convex; Perrault et al. [36] instead used quadratic function as a surrogate to deal with the case when the optimization program is non-convex.

More specifically, the idea of tackling non-convex function in Perrault et al. [36] is to approximate the non-convex function by a quadratic function around a local minimum $\mathbf{x}^{\text{opt}}$ using Taylor expansion, which can be written as:

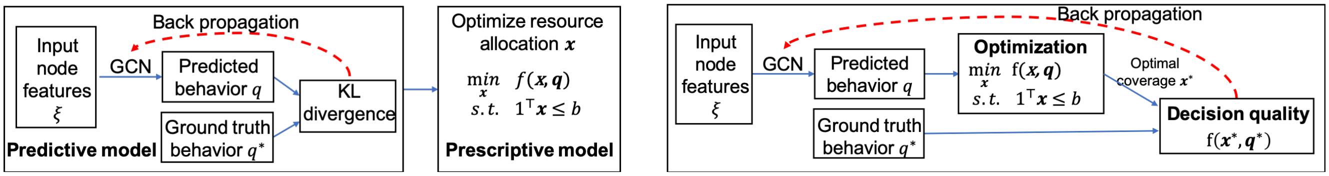
$$f(\mathbf{x}, \mathbf{q}) \approx f(\mathbf{x}^{\text{opt}}, \mathbf{q}) + (\Delta \mathbf{x})^T \frac{\partial f}{\partial \mathbf{x}} + \frac{1}{2} (\Delta \mathbf{x})^T \frac{\partial^2 f}{\partial \mathbf{x}^2} (\Delta \mathbf{x}) \tag{8}$$

where $\Delta \mathbf{x} = \mathbf{x} - \mathbf{x}^{\text{opt}}$. They use this approximate quadratic program (QP) as a surrogate of the non-convex optimization problem, where the optimal solution $\mathbf{x}^*$ of QP matches the local optimum $\mathbf{x}_{\text{opt}}$ computed before. This allows us to differentiate through a QP and compute the gradient of optimal solution $\mathbf{x}^*$ with respect to the linear coefficient $p = \frac{\partial f}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}^{\text{opt}}}$.

$$\frac{df(\mathbf{x}^*, \mathbf{q}^*)}{d\theta} = \frac{df(\mathbf{x}^*, \mathbf{q}^*)}{d\mathbf{x}^*} \frac{d\mathbf{x}^*}{dp} \frac{dp}{d\theta} \tag{9}$$

where $p = \frac{\partial f}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}^{\text{opt}}}$ is a function of $\mathbf{q}$ with $\frac{dp}{d\theta} = \frac{dp}{d\mathbf{q}} \frac{d\mathbf{q}}{d\Phi} \frac{d\Phi}{d\theta}$ can be decomposed and computed. Equation 9 gives us the gradient of the final solution quality with respect to the model parameter θ, which allows us to directly run stochastic gradient descent end-to-end. We apply this approach to our domain. The algorithm is sketched in Algorithm 1 and Figure 2(b).

Issues of Game-focused Learning. Although game-focused learning ideally can achieve better final performance compared to two-stage learning, in this section, we point out two main issues that arise when this game-focused learning is applied to NSGs: scalability and non-convexity.


Back propagation
Input node features ξ
GCN
Predicted behavior q
Ground truth behavior q^*
KL divergence
Predictive model
Optimize resource allocation $\boldsymbol{x}$
$\min_{x} \; f(\boldsymbol{x}, \boldsymbol{q})$
$s.t. \; 1^\top \boldsymbol{x} \leq b$
Prescriptive model
Back propagation
Input node features ξ
GCN
Predicted behavior q
Optimization
$\min_{x} \; f(\boldsymbol{x}, \boldsymbol{q})$
$s.t. \; 1^\top \boldsymbol{x} \leq b$
Optimal coverage x^*
Ground truth behavior q^*
Decision quality
$f(\boldsymbol{x}^*, \boldsymbol{q}^*)$

(a) Two-stage method

(b) game-focused method

Figure 2: Two-stage method trains the behavior model by minimizing the predictive loss, while the game-focused method trains the behavior model by optimizing the final decision quality.

- *Scalability:* In the forward and backward paths of solving QP (Equation 8), we need to solve and be able to back-propagate through the QP, which involves the computation of matrix inverse. Taking matrix inverse grows between quadratic and cubically as the size of the decision variable $\mathbf{x}$ grows. Moreover, in order to compute the Taylor expansion 8, we need to compute the Hessian $\frac{\partial^2 f}{\partial \mathbf{x}^2}$ explicitly, which is usually the major bottleneck of the computation cost when the target function f is complex.
- *Non-convexity:* In the non-convex setting, the objective function $f(\mathbf{x}, \mathbf{q})$ can be non-convex in both $\mathbf{x}$ and $\mathbf{q}$. The gradient-based approaches rely on updating model parameters θ and thus $\mathbf{q}$ to improve the solution quality. However, since the f is non-convex in $\mathbf{q}$, it could create non-convex searching space for gradient-based approaches, which could easily get stuck in local optimum or saddle points. Two-stage methods escape this problem because their loss function $\mathcal{L}(\mathbf{q}, \mathbf{q}^*)$ in Equation 6 is convex, which gradient-based approaches can more easily handle.

7 IMPROVING NAIVE GAME-FOCUSED LEARNING

In this section, we provide a scalable randomized block update approach to resolve the scalability issue, which also suggests a block game-focused algorithm as a scalable version of game-focused learning approach. To resolve the non-convexity issue, we apply the intermediate loss as a regularization, which helps game-focused methods escape local minimums. We further provide theoretical guarantees to link the randomized block update to the naive game-focused learning approach.

7.1 Block Game-focused Learning

Instead of using the entire Taylor expansion (Equation 8) to approximate the objective function locally, we can use a partial Taylor expansion with respect to a subset of variables to approximate it:

$$f(\mathbf{x}, \mathbf{q}) \approx f(\mathbf{x}^*, \mathbf{q}) + (\Delta \mathbf{x}_C)^T \frac{\partial f}{\partial \mathbf{x}_C} + \frac{1}{2} (\Delta \mathbf{x}_C)^T \frac{\partial^2 f}{\partial \mathbf{x}_C^2} (\Delta \mathbf{x}_C), \qquad (10)$$

where $C \subset \{1, 2, ..., |E|\}$ is a subset of indices and $\mathbf{x}_C$ is the corresponding truncation over indices C of the entire variables $\mathbf{x}$. Equation 10 is equivalent to freezing the variables outside of C and applying Taylor expansion to the rest of them. In this formulation, we only need to compute the Hessian with respect to $\mathbf{x}_C$. When the size of C is significantly smaller than the original variable size $|E|$, it can save the computational time of Hessian quadratically. Furthermore, while back-propagating through the KKT conditions, the QP formulation of Equation 10 results in a smaller size of quadratic term, which can reduce the computation of matrix inverse. The block-wise chain rule can be written as:

$$\frac{d f(\mathbf{x}^*, \mathbf{q}^*)}{d\theta} \approx \frac{d f(\mathbf{x}^*, \mathbf{q}^*)}{d \mathbf{x}_C^*} \frac{d \mathbf{x}_C^*}{d p_C} \frac{d p_C}{d\theta} \qquad (11)$$

where $p = \frac{\partial f}{\partial \mathbf{x}_C}|_{\mathbf{x}=\mathbf{x}^{\text{opt}}}$, $\frac{d p_C}{d\theta} = \frac{d p_C}{d \mathbf{q}} \frac{d \mathbf{q}}{d \Phi} \frac{d \Phi}{d\theta}$. When the block size is smaller, the approximation can be more inaccurate. But we will show in the later section that the block gradient is an approximation to the entire gradient.

Algorithm 2: Block Game-focused Learning

```
Input: Training data $D$, initialized $\mathrm{GCN}(\cdot, \cdot; \theta) : V \times \xi \longrightarrow \mathbb{R}$,
  block size $k$
while until converge do
    for $(G, \mathbf{q}^*, \xi) \in D$ do
        Compute prediction $\mathbf{q}$ in Eq. 5 by $\Phi = \mathrm{GCN}(V, \xi; \theta)$
        Find optimum $\mathbf{x}^{\text{opt}}$ of Optimization 7
        Sample $C \subset \{1, 2, ..., |E|\}$ with $|C| = k$
        $Q_{CC} = \frac{\partial^2 f(\mathbf{x}, \mathbf{q})}{\partial \mathbf{x}_C^2}|_{\mathbf{x}=\mathbf{x}^{\text{opt}}}, p_C = \frac{\partial f(\mathbf{x}, \mathbf{q})}{\partial \mathbf{x}_C}|_{\mathbf{x}=\mathbf{x}^{\text{opt}}} - Q_{CC} \mathbf{x}_C^*$
        Re-solve QP: $\mathbf{x}_C^* = \underset{\mathbf{x}_C \text{ feasible}}{\text{argmin}} \; \frac{1}{2} \mathbf{x}_C^\top Q_{CC} \mathbf{x}_C + \mathbf{x}_C^\top p_C$
        Update $\theta$ by gradient $\frac{d f(\mathbf{x}^*, \mathbf{q}^*)}{d \mathbf{x}_C^*} \frac{d \mathbf{x}_C^*}{d p_C} \frac{d p_C}{d\theta}$
Return: trained model $\mathrm{GCN}(\cdot, \cdot; \theta)$
```

All the above reasons suggest a randomized block update algorithm, which is described in Algorithm 2. The algorithm randomly samples a block of variables to compute Hessian and back-propagate accordingly. In comparison, Algorithm 1 requires to compute the entire Hessian matrix $Q = \frac{\partial^2 f(\mathbf{x}, \mathbf{q})}{\partial \mathbf{x}^2}|_{\mathbf{x}=\mathbf{x}^{\text{opt}}}$, which is usually very expensive. Instead, Algorithm 2 only requires the computation of a block Hessian $Q_{CC} = \frac{\partial^2 f(\mathbf{x}, \mathbf{q})}{\partial \mathbf{x}_C^2}|_{\mathbf{x}=\mathbf{x}^{\text{opt}}}$, which can save at least quadratic amount of Hessian computation depending on the block size. It can also reduce the running time of the following quadratic program due to reducing the number of variables.

7.2 Block Selection

In Algorithm 2, the idea of block game-focused learning is to restrict the focus to a subset of variables and to update accordingly. The choice of the sampled block could affect the convergence rate. Here we propose three block selection approaches: i) *random* approach selects block uniformly at random; ii) *coverage*-based approach randomly selects indices with probability proportional to $\mathbf{x}^*$, which guarantees that there is space for the variables in the block to reallocate coverage; iii) *derivative*-based approach selects indices with probability proportional to the magnitude of the derivatives $\frac{df(\mathbf{x}^*,\mathbf{q})}{d\mathbf{x}_i^*}$, which is the weight put on the chain rule.

7.3 Regularization

Another issue associated with the naive game-focused learning method is the non-convex objective function, where gradient-based approaches can encounter issues of local optimums and saddle points. Instead, the two-stage approach optimizes the intermediate loss, which is generally convex in the prediction space. Therefore, we propose to add a weighted two-stage loss as a regularization to smoothify the final objective value. As the training epochs increase, the weight put on the two-stage loss drops exponentially with a decay rate 0.95, pulling the learning back to game-focused methods. This regularization technique helps resolve the non-convexity issue of naive game-focused method, which can achieve better performance afterward.

7.4 Approximation Guarantees

In this section, we will show that both Algorithm 1 and 2 have 0 gradient when the prediction perfectly matches to the ground truth, showing that both algorithms are stable at the global optimum. Later on, we will show that Algorithm 2 is an approximate version of Algorithm 1. This shows that our block game-focused approach can not only achieve scalability due to the reduction in Hessian and QP computation, but it is also aligned with the standard naive game-focused approach with theoretical guarantees.

Theorem 7.1. *When the intermediate prediction matches the ground truth, i.e.,* $\mathbf{q}(\cdot,\cdot;\theta^*) = \mathbf{q}^*$*, we have* $\frac{df(\mathbf{x}^*,\mathbf{q}^*)}{d\theta}|_{\theta=\theta^*} = 0$ *for both Algorithm 1 and Algorithm 2 with any block* C*.*

This theorem implies that if the predictive model is rich enough and able to reach the ground truth, then the gradient computed in both algorithms is equal to 0 at the ground truth. So if we can avoid getting stuck by local optimum, then both algorithms will be able to learn the ground truth. This is also true for the two-stage learning when the loss is defined as any convex norms.

Theorem 7.2. *The quadratic programs in Algorithm 1 and Algorithm 2 share the same primal solutions on the block C. They also share the same dual solution on the non-degenerate constraints containing at least one variable in the block.*

When restricting to variables inside the block, there are some degenerate constraints containing only variables outside of the block, which are always satisfied in the block QP. Thus, there is no restriction put on the dual variable corresponding to these degenerative constraints, which we have no control on them. But in this theorem, we prove that the dual solution to the other valid constraints will match to the dual solution given by the QP in Algorithm 1.

Theorem 7.3. *Given the primal solution* $\mathbf{x}^*$ *and the dual solution* λ^* *of the quadratic program in Algorithm 1 with linear constraints* G, h, A, b*, the Hessian* $Q = \frac{\partial^2 f}{\partial \mathbf{x}}$*, linear coefficient* $p = \frac{\partial f}{\partial \mathbf{x}}$*, and the sampled indices* $C \subset \{1, 2, ..., |E|\}$*, the gradient* $\frac{d\mathbf{x}_C^*}{dp_C} \in \mathbb{R}^{|C|\times|C|}$ *computed in Algorithm 2 is an approximation to the block component of the gradient* $\frac{d\mathbf{x}^*}{dp} \in \mathbb{R}^{|E|\times|E|}$ *computed in Algorithm 1. More specifically,*

$$\left\| \left(\frac{d\mathbf{x}^*}{dp}\right)_{CC} - \frac{d\mathbf{x}_C^*}{dp_C} \right\| \leq \frac{\Delta + \Delta_C}{\mu_{min}(Q)} \max(\lambda^*, 1) \left\|K_{CC}^{-1}\right\| \left\| \left(\frac{d\mathbf{x}^*}{dp}\right)_{CC} \right\| \tag{12}$$

where $\Delta = \left\|G^\top G + A^\top A\right\|, \Delta_C = \left\|Q_{C\bar{C}}^\top Q_{\bar{C}C}\right\|$*, and* $\mu_{min}(Q)$ *is the smallest eigenvalue of positive definite matrix* Q*.* K_{CC} *is the KTT matrix given by the quadratic program in Algorithm 2.*

The Δ in the numerator is a constant that only depends on the constraint matrices. The other term Δ_C depends on the choice of block C, which measures the magnitude of the off-diagonal elements of the Hessian matrix Q. This is usually a small term when the Hessian Q is diagonally dominant. Another interesting finding is that this bound depends on the convexity of the Hessian Q. When the Hessian is more convex, then the smallest eigenvalue of Q is also larger, giving a stronger bound in Theorem 7.3. The last term K_{CC}^{-1} measures the stability of the KKT matrix K_{CC}. We can get a good bound if the KKT matrix K_{CC} is far from singular. Greif et al. [15] provides various bounds on the eigenvalues of the KKT matrix. However, in general, poor constraints can still lead to a KKT matrix close to singular. It also indicates that a good choice of C can imply a more stable KKT matrix, leading to a better estimate in Theorem 7.3.

Theorem 7.3 also implies an alternative explanation to Algorithm 2, where the gradient in Algorithm 2 is an approximation to the partial gradient with indices C in Algorithm 1:

$$\frac{df(\mathbf{x}^*,\mathbf{q}^*)}{d\mathbf{x}_C^*} \left(\frac{d\mathbf{x}^*}{dp}\right)_{CC} \frac{dp_C}{d\theta} \approx \frac{df(\mathbf{x}^*,\mathbf{q}^*)}{d\mathbf{x}_C^*} \frac{d\mathbf{x}_C^*}{dp_C} \frac{dp_C}{d\theta} \tag{13}$$

which implies that Algorithm 2 can be thought as an approximate block-wise gradient descent of Algorithm 1, which relates to the literature of block coordinate gradient descent [35, 42].

8 EXPERIMENTS

In this section, we compare *two-stage (TS), naive game-focused (naive-GF) mentioned in Section 6, block game-focused (block-GF), and regularized block game-focused (reg-block-GF)* methods on synthetic data to show that our block game-focused and regularized block game-focused methods can achieve better performance especially in larger instances. These two methods are also able to scale up to large instances, where the naive game-focused method cannot. Lastly, we study the convergence and scalability of the block game-focused and regularized block game-focused methods with different block sizes and block sampling methods. This allows us to choose the right block size to balance between solution quality and scalability.

8.1 Synthetic Data Generation

8.1.1 Graph and features: we first randomly generate a graph G with various node sizes, 5 random sources with uniform initial distribution π, and 5 random targets with defender penalties $U^d(t)\ \forall t \in T$ drawn from $[-10, -5]$ uniformly at random. We focus on stochastic block model [20] and geometric graphs [45], which can respectively model community structures and physical road networks[1]. For each node in the graph, we draw an attractiveness value, depending on the shortest distance to the targets plus a uniform noise, as the attacker's unbiased preference. We also randomly generate the past coverage $\mathbf{x}$ subject to budget constraints. To generate the node features ξ, we feed the private attractiveness values to a randomly initialized GCN, where the GCN will output a fixed size vector per node as our node features ξ. A different level of Gaussian noise was added to the features to model the noise in the real-world scenario.

8.1.2 Attacker behavior: we choose $\omega = 4$ as the risk aversion parameter suggested by Perrault et al. [36] and Abbasi et al. [1], and set $\eta = 0$ to ignore the future risk factor for the sake of simplicity. For each instance with given attractiveness and the defender coverage, we simulate 100 attacks by initializing the attacker at one of the sources and following the localized SUQR behavior described in Section 3 until the attacker reaches to one of the targets. These sampled paths Λ are used to reconstruct a Markovian behavior: $\mathbf{q}^*_{u\to v}(\mathbf{x}, \xi) := \frac{|\{e=(u,v), e\in\alpha, \alpha\in\Lambda\}|}{|\{e=(u,w), e\in\alpha, \alpha\in\Lambda, w\in N(u)\}|}$ [41], which is then used as our ground truth to evaluate the solution quality[2]. Each instance is composed of the graph G, past coverage $\mathbf{x}$, node features ξ, the attacker behavior $\mathbf{q}^*$, and the sampled paths Λ (only used in two-stage method).

8.1.3 Training, validating, and testing: we generate 50 instances $(G, \mathbf{q}^*, \mathbf{x}^*, \xi)$ as our entire dataset, which are randomly separated into training, validating, testing set with size 35, 5, 10. The model is trained on the training set for 100 epochs, where the best model is chosen from the 100 epochs with the highest score in the validation set. In the following experiments, to achieve statistical significance, for every method and different setup, we ran 50 independent trials and recorded the average results on the testing set.

8.2 Solution Quality

In this section, we compare the solution quality of all methods on stochastic block models and geometric graphs. We generate a set of random graphs with features as described in Section 8.1.1, where Gaussian noise with std. of 0.2 is added to the features to model noisy real-world data. We set $b = 2$. As our goal is efficient approaches for adversary models in large-scale NSGs, the focus of this paper is then on experimenting with many different settings (graph sizes and types), techniques (different variations of game-focused learning), noise, and other variables in building an adversary model. In addition, since we care more about how much defender utility that various learning approaches can improve, we focus on the *counterfactual regret*, which is defined as the gap between the defender utility of our solution and the true optimum when the ground truth is given in advance. Smaller regret implies that the solution is closer to the actual optimum.

In Figure 3(a) and 3(b), we can see that our regularized block game-focused method outperforms two-stage method (note that all of the improvements in the average regret reported by the reg-block-GF method over the two-stage method are statistically significant with $p < 0.05$). When the instance gets larger, the difference between two approaches also gets larger, showcasing the limit of the standard two-stage behavior learning approach. In Figure 4(a) and 4(b), we compare the solution quality of different game-focused methods. Due to the computational issue, the naive game-focused method can only scale up to graphs with 40 nodes. The block game-focused method can scale up to larger instances but it sacrifices some solution quality compared to the naive game-focused approach. Finally, the regularized block game-focused method can achieve both scalability and solution quality by using the block update and regularization term.

8.3 The Impact of Noise

Figure 5(a) and 5(b) compare the performances under different level of noise, where a noise with std. of r is added to the normalized features. We can see that the more noise implies larger regret and poorer performance. But we can also notice that the gap between regularized block game-focused method and the two-stage method gets larger when more noise is introduced. This is probably due to the mismatch between the low intermediate loss and the good final solution quality when the feature is noisy. This also explains why regularized block game-focused method can outperform two-stage in Figure 3 when the features are noisy.

8.4 Scalability

Figure 6(a) and 6(b) show the scalability of all game-focused methods. We limit the training time to be up to 48 hours. Any programs last more than that were cut and the corresponding results were recorded. Naive game-focused method can only handle graphs with up to 40 nodes and it scales extremely poorly. Our proposed methods, block game-focused and regularized block game-focused with a block size #nodes/2, can scale to much larger instances.

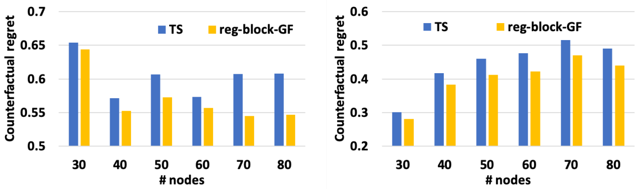
8.5 Block Size Selection

To study the effect of block size, we select various block sizes proportional to the total number of variables and run the block game-focused learning and regularized block game-focused methods to compare the convergence. In Figure 7(a), we can see that for the block-game-focused method, the convergence and the final performance are better when the block size is larger. Figure 7(b) shows the convergence of regularized block game-focused method with different block sizes. In this case, a larger block size still helps, but the difference is relatively tiny.

Figure 7(c) shows the running time of the forward (lines 4-5) and backward path (lines 6-9 in Algorithm 2) for the block game-focused method with various block sizes, where forward path solves prescriptive stage with black-box optimization and the backward path requires computing the Hessian and solving the quadratic program to back-propagate. In practice, we would like to select a block size such that the running time of forward and backward paths are of

[1] For stochastic block model, we separate nodes into communities with 10 nodes in each community, then connect nodes within the same community with probability 0.4 and nodes not in the same community with probability 0.1. For geometric graph, we randomly places nodes in a unit square and connects nodes with distance smaller than 0.2.

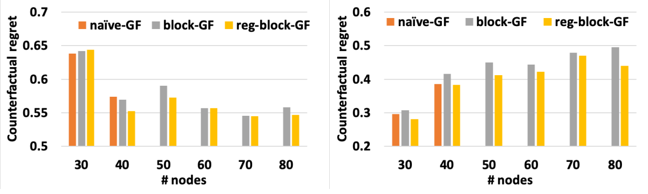
[2] The reason of using sampled paths instead of the actual generated attractiveness values as our ground truth is to align with the real-world data, where it is almost impossible to have access to the underlying attacker preference or Markovian behavior; instead, we generally have access to the paths or edges where illegal activities have been found, which can be used as sampled paths or edges and used to reconstruct the Markovian behavior as we did here.



(a) Stochastic block model

(b) Geometric graphs

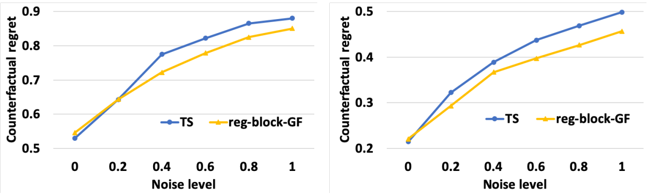
Figure 3: Solution quality comparison between two-stage and regularized block game-focused method. The difference in solution quality gets larger when the graph size increases.



(a) Stochastic block model

(b) Geometric graphs

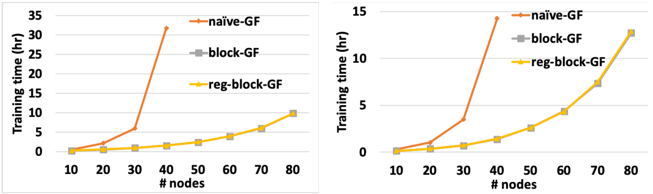
Figure 4: Solution quality comparison between game-focused methods. Randomized block update can improve scalability while the regularization can improve the solution quality.



(a) Stochastic block model

(b) Geometric graphs

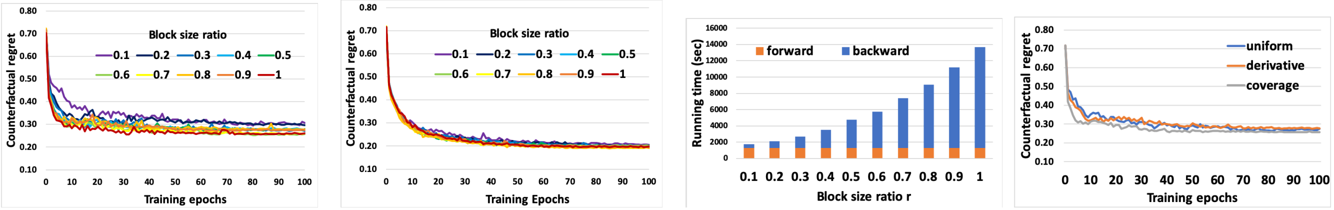
Figure 5: The figures show the effect of noise to all the methods, where regularized block game-focused method is more resilient to noise in the features.



(a) Stochastic block model

(b) Geometric graphs

Figure 6: Naive game-focused method can only scale up to 40 nodes. Instead, block game-focused and regularized block game-focused can solve larger instances with 80 nodes.



(a) Block game-focused method

(b) Regularized block game-focused

(c) Running time

(d) Block selection

Figure 7: Figure (a) and (b) show the convergence rate of different block sizes. Figure (c) shows the running time of backward path for different block sizes, which grows significantly more than linear. Figure (d) shows the effect of different block sampling methods. All methods converge with slightly different speed, where coverage-based sampling is the best and it is also what we use in other experiments.

the same order to balance between the convergence and scalability, which explains the reason that we eventually choose block size = #nodes/2 for all other experiments. Lastly, Figure 7(d) compares different block selections mentioned in Section 7.2, where convergence speed differs but mostly lead to the same point. Coverage-based selection converges the most quickly, and thus we use it throughout the other experiments.

9 CONCLUSIONS

In this paper, we introduce a fundamentally different behavior learning approach, game-focused learning, to network security games, placing the downstream defender utility maximization problem into the loop of behavior learning. We propose a novel local SUQR model as our adversary model, where GCNs can be applied to automatically handle the information propagation in the graph. We further identify two existing issues of game-focused learning method: scalability and non-convexity, which are addressed by our block game-focused and by regularizing respectively. Block game-focused method can largely reduce the computational cost while maintaining the focus on the final solution quality as naive game-focused learning does. We also provide theoretical guarantees on the block game-focused method. In the experimental section, we run extensive experiments to verify the reduction on the training time and show an improvement in terms of solution quality. The block game-focused method reduces the training time, but sacrifices a little solution quality, while regularized block game-focused can achieve both speed and performance.

Acknowledgments. This research was supported by MURI Grant Number W911NF-17-1-0370 and W911NF-18-1-0208.

REFERENCES

[1] Yasaman Abbasi, Debarun Kar, Nicole Sintov, Milind Tambe, Noam Ben-Asher, Don Morrison, and Cleotilde Gonzalez. 2016. Know Your Adversary: Insights for a Better Adversarial Behavioral Model.. In *CogSci*.

[2] Yasaman Dehghani Abbasi, Martin Short, Arunesh Sinha, Nicole Sintov, Chao Zhang, and Milind Tambe. 2015. Human adversaries in opportunistic crime security games: evaluating competing bounded rationality models. In *Proc. of Advances in Cognitive Systems*.

[3] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. 2019. Differentiable Convex Optimization Layers. In *NeurIPS-19*. Vancouver.

[4] Brandon Amos and J. Zico Kolter. 2017. OptNet: Differentiable optimization as a layer in neural networks. In *ICML-17*. Sydney.

[5] Jana Arsovska and Panos A Kostakos. 2008. Illicit arms trafficking and the limits of rational choice theory: the case of the Balkans. *Trends in Organized Crime* 11, 4 (2008), 352–378.

[6] Dimitri P. Bertsekas and John. N. Tsitsiklis. 1996. *Neuro-dynamic Programming*. Athena, Belmont, MA.

[7] Sarah Cooney, Kai Wang, Elizabeth Bondi, Thanh Nguyen, Phebe Vayanos, Hailey Winetrobe, Edward A Cranford, Cleotilde Gonzalez, Christian Lebiere, and Milind Tambe. 2019. Learning to Signal in the Goldilocks Zone: Improving Adversary Compliance in Security Games. In *ECMLPKDD-19*. Würzburg.

[8] Priya Donti, Brandon Amos, and J. Zico Kolter. 2017. Task-based end-to-end model learning in stochastic optimization. In *NIPS-17*. Long Beach, 5484–5494.

[9] Fei Fang, Peter Stone, and Milind Tambe. 2015. When Security Games Go Green: Designing Defender Strategies to Prevent Poaching and Illegal Fishing. In *IJCAI-15*. Buenos Aires, 2589–2595.

[10] Peyton Ferrier et al. 2009. *The economics of agricultural and wildlife smuggling*. Technical Report. Springer.

[11] Matteo Fischetti, Ivana Ljubic, Michele Monaci, and Markus Sinnl. 2016. *Interdiction games and monotonicity*. Technical Report. Technical Report, DEI, University of Padova.

[12] Matteo Fischetti, Ivana Ljubić, Michele Monaci, and Markus Sinnl. 2019. Interdiction games and monotonicity, with application to knapsack problems. *INFORMS Journal on Computing* (2019).

[13] Benjamin Ford, Thanh Nguyen, Milind Tambe, Nicole Sintov, and Francesco Delle Fave. 2015. Beware the soothsayer: From attack prediction accuracy to predictive reliability in security games. In *GameSec-15*. 35–56.

[14] Jiarui Gan, Bo An, and Yevgeniy Vorobeychik. 2015. Security Games with Protection Externalities. In *AAAI-15*. Austin, 914–920.

[15] Chen Greif, Erin Moulding, and Dominique Orban. 2014. Bounds on eigenvalues of matrices arising from interior-point methods. *SIAM Journal on Optimization* 24, 1 (2014), 49–83.

[16] Alexander Gutfraind, Aric Hagberg, and Feng Pan. 2009. Optimal interdiction of unreactive Markovian evaders. In *CPAIOR-09*. Pittsburgh, 102–116.

[17] Alexander Gutfraind, Aric A Hagberg, David Izraelevitz, and Feng Pan. 2011. Interdiction of a Markovian evader. In *Proc. of INFORMS Computing Society*. Monterey, CA.

[18] Nika Haghtalab, Fei Fang, Thanh Hong Nguyen, Arunesh Sinha, Ariel D Procaccia, and Milind Tambe. 2016. Three Strategies to Success: Learning Adversary Models in Security Games. In *IJCAI-16*. New York, 308–314.

[19] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS-17*. Long Beach, 1024–1034.

[20] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. 1983. Stochastic blockmodels: First steps. *Social Networks* 5, 2 (1983), 109–137.

[21] Manish Jain, Dmytro Korzhyk, Ondřej Vaněk, Vincent Conitzer, Michal Pěchouček, and Milind Tambe. 2011. A double oracle algorithm for zero-sum security games on graphs. In *AAMAS-11*. Taipei, 327–334.

[22] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR-17*. Toulon.

[23] Dieter Kraft. 1985. On converting optimal control problems into nonlinear programming problems. In *Computational mathematical programming*. Springer, 261–280.

[24] Joshua Letchford, Vincent Conitzer, and Kamesh Munagala. 2009. Learning and Approximating the Optimal Strategy to Commit To. In *Algorithmic Game Theory*, Marios Mavronicolas and Vicky G. Papadopoulou (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 250–262.

[25] Sara Marie Mc Carthy, Milind Tambe, Christopher Kiekintveld, Meredith L Gore, and Alex Killion. 2016. Preventing illegal logging: Simultaneous optimization of resource teams and tactics for security. In *AAAI-16*. New York.

[26] Richard D McKelvey and Thomas R Palfrey. 1995. Quantal response equilibria for normal form games. *Games and Economic Behavior* 10, 1 (1995), 6–38.

[27] John Morgan and Felix Vardy. 2004. An experimental study of commitment and observability in Stackelberg games. *Games and Economic Behavior* 49, 2 (2004), 401–423.

[28] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI-19*. Honolulu, 4602–4609.

[29] David P Morton, Feng Pan, and Kevin J Saeger. 2007. Models for nuclear smuggling interdiction. *IIE Transactions* 39, 1 (2007), 3–14.

[30] Padmanabhan Murugan and Biniam Abebaw. 2014. Factors Contributing to Human Trafficking, Contexts of Vulnerability and Patterns of Victimization: the case of stranded victims in Metema, Ethiopia. *Ethiopian Journal of the Social Sciences and Humanities* 10, 2 (2014), 75–105.

[31] Michael Victor Nehme. 2009. Two-person games for stochastic network interdiction: models, methods, and complexities. (2009).

[32] Thanh Hong Nguyen, Rong Yang, Amos Azaria, Sarit Kraus, and Milind Tambe. 2013. Analyzing the Effectiveness of Adversary Modeling in Security Games. In *AAAI-13*. Bellevue, Washington.

[33] Steven Okamoto, Noam Hazon, and Katia Sycara. 2012. Solving non-zero sum multiagent network flow security games with attack costs. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 879–888.

[34] Feng Pan, William S Charlton, and David P Morton. 2003. A stochastic program for interdicting smuggled nuclear material. In *Network interdiction and stochastic integer programming*. Springer, 1–19.

[35] Andrei Patrascu and Ion Necoara. 2015. Efficient random coordinate descent algorithms for large-scale structured nonconvex optimization. *Journal of Global Optimization* 61, 1 (2015), 19–46.

[36] Andrew Perrault, Bryan Wilder, Eric Ewing, Aditya Mate, Bistra Dilkina, and Milind Tambe. 2020. End-to-end Game-focused Learning of Adversary Behavior in Security Games. In *AAAI-2020*. New York.

[37] Gail Emilia Rosen and Katherine F Smith. 2010. Summarizing the evidence on the international trade in illegal wildlife. *EcoHealth* 7, 1 (2010), 24–32.

[38] Sankardas Roy, Charles Ellis, Sajjan Shiva, Dipankar Dasgupta, Vivek Shandilya, and Qishi Wu. 2010. A survey of game theory as applied to network security. In *2010 43rd Hawaii International Conference on System Sciences*. IEEE, 1–10.

[39] Arunesh Sinha, Debarun Kar, and Milind Tambe. 2016. Learning adversary behavior in security games: A PAC model perspective. In *AAMAS-16*. Singapore, 214–222.

[40] Milind Tambe. 2011. *Security and game theory: algorithms, deployed systems, lessons learned*. Cambridge University Press.

[41] Iuliana Teodorescu. 2009. Maximum likelihood estimation for Markov Chains. *arXiv preprint arXiv:0905.4131* (2009).

[42] Paul Tseng. 2001. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications* 109, 3 (2001), 475–494.

[43] Satya Gautam Vadlamudi, Sailik Sengupta, Marthony Taguinod, Ziming Zhao, Adam Doupé, Gail-Joon Ahn, and Subbarao Kambhampati. 2016. Moving target defense for web applications using bayesian stackelberg games. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1377–1378.

[44] Alan Washburn and Kevin Wood. 1995. Two-person zero-sum games for network interdiction. *Operations Research* 43, 2 (1995), 243–251.

[45] Bernard M Waxman. 1988. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications* 6, 9 (1988), 1617–1622.

[46] Rong Yang, Fei Fang, Albert Xin Jiang, Karthik Rajagopal, Milind Tambe, and Rajiv Maheswaran. 2012. Designing better strategies against human adversaries in network security games. In *AAMAS-12*. Valencia.

[47] Zhengyu Yin, Dmytro Korzhyk, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. 2010. Stackelberg vs. Nash in security games: Interchangeability, equivalence, and uniqueness. In *AAMAS-10*. Toronto, 1139–1146.

[48] Chao Zhang, Arunesh Sinha, and Milind Tambe. 2015. Keeping pace with criminals: Designing patrol allocation against adaptive opportunistic criminals. In *AAMAS-15*. Istanbul, 1351–1359.

[49] Mara E Zimmerman. 2003. The black market for wildlife: Combating transnational organized crime in the illegal wildlife trade. *Vand. J. Transnat'l L.* 36 (2003), 1657.

SUPPLEMENTARY MATERIAL FOR AAMAS 2020

10 COMPUTATION OF DEFENDER UTILITY

Given coverage $\mathbf{x}$, if we sort the vertices out by intermediate states then absorbing states, then the transition matrix induced by behavior $\mathbf{q}(\mathbf{x}, \xi)$ can be written as: $P = \begin{bmatrix} Q & R \\ 0 & I \end{bmatrix}$, where I is an identity matrix representing once the attacker reaches any absorbing states, he would never transit to other states. Q, R are both functions of $\mathbf{x}$ and ξ.

The absorbing probability can be computed by $B = (I - Q)^{-1} R \in \mathbb{R}^{|S| \times (|T|+1)}$, where the entry B_{ij} indicates the probability that the attacker initiates from state i and ends up being in absorbing state j. Since we also know the distribution $\pi \in \mathbb{R}^{|S|}$ that the attacker will appear and the defender utility $U^d \in \mathbb{R}^{|T|+1}$ including the reward of catching the attacker, the defender utility can be given by $\pi^\top B U^d$, where the function f is defined by the negative defender utility:

$$f(\mathbf{x}, \mathbf{q}) = -\pi^\top B U^d = -\pi^\top (I - Q(\mathbf{x}, \mathbf{q}))^{-1} R(\mathbf{x}, \mathbf{q}) U^d \tag{14}$$

which is still a function of $\mathbf{x}$ and $\mathbf{q}$. We can also compute the derivatives of this function f. But since the computation of f involves matrix inversion, the computation of derivatives will also involve matrix inversions and multiplications, which can be very expensive especially for higher order derivatives.

11 THE CHOICES OF LOSS FUNCTION

Given two transition matrices $M, M' \in \mathbb{R}^{|V| \times |V|}$, we can align with the any standard definition of matrix norm:

$$\mathcal{L}(M, M') = \left\| M - M' \right\| \tag{15}$$

Another choice of loss function definition is to compute the KL-divergence or cross entropy of the path distribution inferred by these two transition matrices. Since there are absorbing states, the path can eventually terminate when it reaches to any of the absorbing state. However, there could be loop in the graph, which might incur infinitely many possible paths, making the path distribution infinitely dimensional. Another issue is that we do not have a close form of the path distribution, which can prevent us from computing the KL-divergence between two implicit distributions.

In our domain, we usually have samples from the ground truth Markov chain, which can be used as an empirical path distribution. By considering those samples as the empirical distribution Λ, we can compute the KL-divergence between Λ and the predicted Markov chain M:

$$\mathcal{L}(\Lambda, M) = \sum\nolimits_\alpha \text{prob}(\alpha \in \Lambda) \log \frac{\text{prob}(\alpha \in \Lambda)}{\text{prob}(\alpha \in M)} \tag{16}$$

which can be efficiently computed since Λ is finite and all the probability can be analytically computed. This serves as an alternative for us to compute the KL-divergence between the ground truth and our prediction.

12 DIFFERENTIABLE QUADRATIC PROGRAM (AMOS ET AL. [4])

Given a quadratic program:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2} \mathbf{x}^T Q \mathbf{x} + p^T \mathbf{x} \\ \text{s.t.} \quad & G\mathbf{x} \leq h \\ & A\mathbf{x} = b \end{aligned} \tag{17}$$

According to [4, 8], we can compute the derivative of the optimal solution $\mathbf{x}^*$ with respect to each parameters in the quadratic program, e.g., Q, p, G, h, A, b. In our case, we only consider the derivative with respect to p, where G, h, A, b are constants and we ignore the derivative of the Hessian Q since its derivative is of third order.

After solving the quadratic program, we can get the solution $\mathbf{x}^*$ with dual variables λ^*, ν^* respectively for the inequality constraints and equality constraints. All the variables $\mathbf{x}^*, \lambda^*, \nu^*$ are all functions of prediction $\mathbf{q}$. In [4, 8], they proposed that if we write the KKT conditions:

$$\begin{aligned} Q\mathbf{x}^* + p + A^\top \nu^* + G^\top \lambda^* &= 0 \\ A\mathbf{x}^* - b &= 0 \\ D(\lambda^*)(G\mathbf{x}^* - h) &= 0 \end{aligned}$$

where $D(\lambda^*)$ is the diagonal matrix with λ^* on the diagonal. We can consider the total derivative of the above equations, which yields:

$$\begin{aligned} dQ\mathbf{x}^* + Qd\mathbf{x}^* + dp + dA^\top \nu^* + A^\top d\nu^* + dG^\top \lambda^* + G^\top d\lambda^* &= 0 \\ dA\mathbf{x}^* + Ad\mathbf{x}^* - db &= 0 \\ D(G\mathbf{x}^* - h)d\lambda^* + D(\lambda^*)(dG\mathbf{x}^* + Gd\mathbf{x}^* - dh) &= 0 \end{aligned}$$

Since here we assume Q, G, h, A, b are all constants, so we can ignore the derivatives of there terms and get:

$$\begin{aligned} Qd\mathbf{x}^* + dp + A^\top d\nu^* + G^\top d\lambda^* &= 0 \\ Ad\mathbf{x}^* &= 0 \\ D(G\mathbf{x}^* - h)d\lambda^* + D(\lambda^*)Gd\mathbf{x}^* &= 0 \end{aligned}$$

which can be further turned into matrix form:

$$\begin{aligned} & \begin{bmatrix} Q & G^\top & A^\top \\ D(\lambda^*)G & D(G\mathbf{x}^* - h) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} d\mathbf{x}^* \\ d\lambda^* \\ d\nu^* \end{bmatrix} = \begin{bmatrix} -dp \\ 0 \\ 0 \end{bmatrix} \\ \Leftrightarrow \quad & \begin{bmatrix} Q & G^\top & A^\top \\ D(\lambda^*)G & D(G\mathbf{x}^* - h) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{d\mathbf{x}^*}{dp} \\ \frac{d\lambda^*}{dp} \\ \frac{d\nu^*}{dp} \end{bmatrix} = \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix} \\ \Leftrightarrow \quad & \begin{bmatrix} \frac{d\mathbf{x}^*}{dp} \\ \frac{d\lambda^*}{dp} \\ \frac{d\nu^*}{dp} \end{bmatrix} = \begin{bmatrix} Q & G^\top & A^\top \\ D(\lambda^*)G & D(G\mathbf{x}^* - h) & 0 \\ A & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix} \end{aligned} \tag{18}$$

This allows us to compute the gradients $\frac{d\mathbf{x}^*}{dp}$ by solving the corresponging linear equation.

We can also combine the chain rule $\frac{df}{dp} = \frac{df}{d\mathbf{x}^*}\frac{d\mathbf{x}^*}{dp}$ by:

$$\frac{df}{dp} = \begin{bmatrix} \frac{d\mathbf{x}^*}{dp} \\ \frac{d\lambda^*}{dp} \\ \frac{dv^*}{dp} \end{bmatrix}^\top \begin{bmatrix} \frac{df}{d\mathbf{x}^*} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix}^\top \begin{bmatrix} Q & D(\lambda^*)G^\top & A^\top \\ G & D(G\mathbf{x}^* - h) & 0 \\ A & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} \frac{df}{d\mathbf{x}^*} \\ 0 \\ 0 \end{bmatrix}$$

Or equivalently, define

$$\begin{bmatrix} d\mathbf{x} \\ d\lambda \\ dv \end{bmatrix} = \begin{bmatrix} Q & D(\lambda^*)G^\top & A^\top \\ G & D(G\mathbf{x}^* - h) & 0 \\ A & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} -\frac{df}{d\mathbf{x}^*} \\ 0 \\ 0 \end{bmatrix} \tag{19}$$

then $\frac{df}{dp} = d\mathbf{x}$, which is the derivative of the objective function f with respect to the linear coefficient p of the quadratic program.

13 PROOF OF THEOREM 7.1

Theorem 7.1. *When the intermediate prediction matches the ground truth, i.e., $\mathbf{q}(\cdot, \cdot; \theta^*) = \mathbf{q}^*$, we have $\frac{df(\mathbf{x}^*, \mathbf{q}^*)}{d\theta}|_{\theta=\theta^*} = 0$ for both Algorithm 1 and Algorithm 2 with any block C.*

Proof. We first prove for the Algorithm 1 case. Our goal is to prove that this $\frac{df}{dp}|_{\mathbf{q}=\mathbf{q}^*} = d_{\mathbf{x}}$ is exactly 0 at $\mathbf{q}^*$, meaning there is no gradient at the true optimal prediction $\mathbf{q}^*$.

To prove this, we directly show that $d_{\mathbf{x}} = 0, d_\lambda = \mathbf{1}_{\{\lambda^* \neq 0\}}, d_\nu = \nu^*$ is a solution. When the KKT matrix in Equation 18 is non-singular, this implies that $d_{\mathbf{x}} = 0$ is the unique solution. When the KKT matrix is singular, $d_{\mathbf{x}} = 0$ is a subgradient. Furthermore, if we follow the implementation in [8], they remove the dependent rows of the KKT matrix 18 such that the matrix is non-singular, which again implies that $d_{\mathbf{x}} = 0$ is the unique solution. To verify this, we can compute:

$$\begin{aligned} &\begin{bmatrix} Q & G^\top D(\lambda^*) & A^\top \\ G & D(G\mathbf{x}^* - h) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \mathbf{1}_{\{\lambda^* \neq 0\}} \\ \nu^* \end{bmatrix} \\ = &\begin{bmatrix} G^\top D(\lambda^*)\mathbf{1}_{\{\lambda^* \neq 0\}} + A^\top \nu^* \\ D(G\mathbf{x}^* - h)\mathbf{1}_{\{\lambda^* \neq 0\}} \\ 0 \end{bmatrix} = \begin{bmatrix} G^\top \lambda^* + A^\top \nu^* \\ 0 \\ 0 \end{bmatrix} \end{aligned}$$

Notice that the KKT condition of the quadratic program implies:

$$\begin{aligned} & Q\mathbf{x}^* + p + A^\top \nu^* + G^\top \lambda^* = 0 \\ \Leftrightarrow \quad & Q\mathbf{x}^* + \frac{\partial f}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}^*, \mathbf{q}=\mathbf{q}^*} - Q\mathbf{x}^* + A^\top \nu^* + G^\top \lambda^* = 0 \\ \Leftrightarrow \quad & \frac{\partial f}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}^*, \mathbf{q}=\mathbf{q}^*} + A^\top \nu^* + G^\top \lambda^* = 0 \end{aligned}$$

$$\begin{aligned} &\begin{bmatrix} Q & G^\top D(\lambda^*) & A^\top \\ G & D(G\mathbf{x}^* - h) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \mathbf{1}_{\{\lambda^* \neq 0\}} \\ \nu^* \end{bmatrix} \\ = &\begin{bmatrix} G^\top \lambda^* + A^\top \nu^* \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{df}{d\mathbf{x}^*} \\ 0 \\ 0 \end{bmatrix} \end{aligned}$$

This verifies that $d_{\mathbf{x}} = 0, d_\lambda = \mathbf{1}_{\{\lambda^* \neq 0\}}, d_\nu = \nu^*$ is a solution of Equation 19. This concludes the proof of Algorithm 1.

To prove for Algorithm 2, we consider the following equation:

$$\begin{bmatrix} d\mathbf{x}_C \\ d\lambda \\ dv \end{bmatrix} = \begin{bmatrix} Q_C & D(\lambda^*)G_C^\top & A_C^\top \\ G_C & D(G_C\mathbf{x}_C^* - h_C) & 0 \\ A_C & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} -\frac{df}{d\mathbf{x}_C^*} \\ 0 \\ 0 \end{bmatrix} \tag{20}$$

where $G = \begin{bmatrix} G_C & G_{\overline{C}} \end{bmatrix}, A = \begin{bmatrix} A_C & A_{\overline{C}} \end{bmatrix}$ that G_C, A_C correspond to the coefficients of indices C. $h_C = h - G_{\overline{C}}\mathbf{x}^*_{\overline{C}}$ corresponds to the modified linear inequalities without the effect of terms $\mathbf{x}_{\overline{C}}$.

We can also verify that $\frac{df}{dp_C}|_{\mathbf{q}=\mathbf{q}^*} = d\mathbf{x}_C = 0$ is a solution in Equation 20. By setting $d\mathbf{x}_C = 0, d\lambda = \mathbf{1}_{\{\lambda^* \neq 0\}}, d_\nu = \nu^*$, we can find that this also satisfies the Equation 20.

All of these imply that $\frac{df}{dp_C}|_{\mathbf{q}=\mathbf{q}^*} = 0$ (or at least a feasible subderivative). By applying Equation 9 of Algorithm 1 or Equation 11 of Algorithm 2, we can get $\frac{df(\mathbf{x}^*, \mathbf{q}^*)}{d\theta}|_{\theta=\theta^*} = 0$ where θ^* is the optimal model parameter that gives the correct prediction $\mathbf{q}^*$. □

14 PROOF OF THEOREM 7.2

Theorem 7.2. *The quadratic programs in Algorithm 1 and Algorithm 2 share the same primal solutions on the block C. They also share the same dual solution on the non-degenerate constraints containing at least one variable in the block.*

Proof. Since both algorithms are derived from Taylor expansion around a local optimum, the Hessian is always positive definite. Therefore, the solution given by the quadratic program is exactly the same as the local optimum previously computed, which is shared for both algorithms. So both of them share the same primal solutions at indices C.

For the dual solutions, we can write down the quadratic programs 17 for Algorithm 1 by:

$$\begin{aligned} \min\nolimits_{\mathbf{x}} \quad & \frac{1}{2}\mathbf{x}^\top Q\mathbf{x} + p^\top \mathbf{x} \\ \text{s.t.} \quad & G\mathbf{x} \leq h \\ & A\mathbf{x} = b \end{aligned} \tag{21}$$

with $Q = \frac{\partial^2 f}{\partial \mathbf{x}^2}|_{\mathbf{x}=\mathbf{x}^*}, p = \frac{\partial f}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}^*} - Q\mathbf{x}^*$. The KKT stationary condition can be given by:

$$\begin{aligned} & Q\mathbf{x}^* + p + G^\top \lambda^* + A^\top \nu^* = 0 \\ \Leftrightarrow \quad & Q\mathbf{x}^* + \frac{\partial f}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}^*} - Q\mathbf{x}^* + G^\top \lambda^* + A^\top \nu^* = 0 \\ \Leftrightarrow \quad & \frac{\partial f}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}^*} + G^\top \lambda^* + A^\top \nu^* = 0 \end{aligned} \tag{22}$$

Similarly for Algorithm 2 in the case there is no degenerative constraint, we have:

$$\begin{aligned} \min\nolimits_{\mathbf{x}_C} \quad & \frac{1}{2}\mathbf{x}_C^\top Q_{CC}\mathbf{x}_C + p_C^\top \mathbf{x}_C \\ \text{s.t.} \quad & G_C\mathbf{x}_C \leq h_C = h - G_{\overline{C}}\mathbf{x}_{\overline{C}} \\ & A_C\mathbf{x}_C = b_C = b - A_{\overline{C}}\mathbf{x}_{\overline{C}} \end{aligned} \tag{23}$$

where $Q_{CC} = \frac{\partial^2 f}{\partial \mathbf{x}_C^2}|_{\mathbf{x}=\mathbf{x}^*}, p_C = \frac{\partial f}{\partial \mathbf{x}_C}|_{\mathbf{x}=\mathbf{x}^*} - Q_{CC}\mathbf{x}_C^*$, and constraints $G = \begin{bmatrix} G_C & G_{\overline{C}} \end{bmatrix}, A = \begin{bmatrix} A_C & A_{\overline{C}} \end{bmatrix}$. The KKT stationary condition can be given by:

$$\begin{aligned} & Q_{CC}\mathbf{x}^*_C + p_C + G_C^\top\lambda^* + A_C^\top\nu^* = 0 \\ \Leftrightarrow\quad & Q_{CC}\mathbf{x}^*_C + \frac{\partial f}{\partial \mathbf{x}_C}|_{\mathbf{x}=\mathbf{x}^*} - Q_{CC}\mathbf{x}^*_C + G_C^\top\lambda^* + A_C^\top\nu^* = 0 \\ \Leftrightarrow\quad & \frac{\partial f}{\partial \mathbf{x}_C}|_{\mathbf{x}=\mathbf{x}^*} + G_C^\top\lambda^* + A_C^\top\nu^* = 0 \end{aligned} \tag{24}$$

Comparing Equation 22 and Equation 24, we can find that Equation 24 is just a subset of Equation 22, or more specifically the equations at indices C. Similarly, they also share the same primal, dual feasibility conditions, and complementary slackness conditions. Therefore, the dual solutions of the KKT conditions of quadratic program 21 is also a solution of the KKT conditions of 23.

When there are degenerative constraints, for example, some rows $\overline{R}$ of the constraints G_C are degenerative and thus be all 0 after truncating by block C, i.e., $G_{\overline{R},C} = 0$. In this case, $G_C^\top\lambda^* = G^\top_{R,C}\lambda^*_R + G^\top_{\overline{R},C}\lambda^*_{\overline{R}} = G^\top_{R,C}\lambda^*_R$, where there is no constraint posted on $\lambda^*_{\overline{R}}$, which can be arbitrary here. Similarly, some rows $\overline{L}$ of equality constraints A_C might also be degenerative, i.e., $A_{\overline{L},C} = 0$. But if we only consider the non-degenerative constraints $G_{R,C}$ and $A_{L,C}$, we can re-write the KKT stationary conditions in Equation 24 by:

$$\begin{aligned} & \frac{\partial f}{\partial \mathbf{x}_C}|_{\mathbf{x}=\mathbf{x}^*} + G_C^\top\lambda^* + A_C^\top\nu^* = 0 \\ \Leftrightarrow\quad & \frac{\partial f}{\partial \mathbf{x}_C}|_{\mathbf{x}=\mathbf{x}^*} + G_{R,C}^\top\lambda^* + A_{L,C}^\top\nu^* = 0 \end{aligned} \tag{25}$$

In this case, the entire KKT condition with non-degenerative dual variables is non-singular, which imposes a unique solution to the dual variables. But we have shown that the dual solution of Equation 22 is also a solution to Equation 24, which is again a solution to Equation 25. By uniqueness, this solution of Equation 25is also a solution of Equation 24 on the non-degenerative constraints $G_{R,C}, A_{L,C}$, thus a solution to the Equation 22, which concludes the proof. □

15 PROOF OF THEOREM 7.3

Theorem 7.3. *Given the primal solution $\mathbf{x}^*$ and the dual solution λ^* of the quadratic program in Algorithm 1 with linear constraints G, h, A, b, the Hessian $Q = \frac{\partial^2 f}{\partial \mathbf{x}}$, linear coefficient $p = \frac{\partial f}{\partial \mathbf{x}}$, and the sampled indices $C \subset \{1, 2, ..., |E|\}$, the gradient $\frac{d\mathbf{x}^*_C}{dp_C} \in \mathbb{R}^{|C|\times|C|}$ computed in Algorithm 2 is an approximation to the block component of the gradient $\frac{d\mathbf{x}^*}{dp} \in \mathbb{R}^{|E|\times|E|}$ computed in Algorithm 1. More specifically,*

$$\left\| \left(\frac{d\mathbf{x}^*}{dp}\right)_{CC} - \frac{d\mathbf{x}^*_C}{dp_C} \right\| \leq \frac{\Delta + \Delta_C}{\mu_{min}(Q)} \max(\lambda^*, 1) \left\|K_{CC}^{-1}\right\| \left\| \left(\frac{d\mathbf{x}^*}{dp}\right)_{CC} \right\| \tag{12}$$

where $\Delta = \left\|G^\top G + A^\top A\right\|, \Delta_C = \left\|Q^\top_{C\overline{C}} Q_{\overline{C}C}\right\|$, and $\mu_{min}(Q)$ is the smallest eigenvalue of positive definite matrix Q. K_{CC} is the KTT matrix given by the quadratic program in Algorithm 2.

Proof. Denote $K = \begin{bmatrix} Q & G^\top & A^\top \\ D(\lambda^*)G & D(G\mathbf{x}^* - h) & 0 \\ A & 0 & 0 \end{bmatrix}$ to be the KKT matrix 18 of the quadratic program 21 given by Algorithm 1. We can also denote $K_{CC} = \begin{bmatrix} Q_{CC} & G_C^\top & A_C^\top \\ D(\lambda^*)G_C & D(G_C\mathbf{x}^*_C - h_C) & 0 \\ A_C & 0 & 0 \end{bmatrix}$ to be the KKT matrix of the quadratic program 23 given by Algorithm 2. Notice that K_{CC} is in fact a block of K since they share the same primal and dual solution. According to Equation 18, we can write down the gradient $\frac{d\mathbf{x}^*}{dp}$ and $\frac{d\mathbf{x}^*_C}{dp_C}$ respectively in Algorithm 1 and Algorithm 2 by:

$$\frac{d\mathbf{x}^*}{dp} = \begin{bmatrix} \mathbf{I} \\ 0 \\ 0 \end{bmatrix}^\top K^{-1} \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix}, \quad \frac{d\mathbf{x}^*_C}{dp_C} = \begin{bmatrix} \mathbf{I} \\ 0 \\ 0 \end{bmatrix}^\top K_{CC}^{-1} \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix}$$

If we use block form to represent the KKT matrix K, we can get:

$$K = \begin{bmatrix} K_1 & K_2 \\ K_3 & K_4 \end{bmatrix}$$

where we can apply the block matrix inversion technique and get:

$$\begin{aligned} & K^{-1} \\ = & \begin{bmatrix} K_1^{-1} + K_1^{-1}K_2(K_4 - K_3K_1^{-1}K_2)^{-1}K_3K_1^{-1} & -K_1^{-1}K_2(K_4 - K_3K_1^{-1}K_2)^{-1} \\ -(K_4 - K_3K_1^{-1}K_2)^{-1}K_3K_1^{-1} & (K_4 - K_3K_1^{-1}K_2)^{-1} \end{bmatrix} \end{aligned} \tag{26}$$

where K_1 needs to be invertible here.

Set $K_1 = Q_{\overline{CC}}, K_2 = \begin{bmatrix} Q_{C\overline{C}} & G_{\overline{C}}^\top & A_{\overline{C}}^\top \end{bmatrix}, K_3 = \begin{bmatrix} Q_{\overline{C}C} \\ D(\lambda^*)G_{\overline{C}} \\ A_{\overline{C}} \end{bmatrix}, K_4 = K_{CC}$, where $K_1 = Q_{\overline{CC}}$ is positive definite therefore also invertible. We can see that $K_1 \in \mathbb{R}^{|\overline{C}|\times|\overline{C}|}$ and the sizes of K_2, K_3, K_4 depend on the size of the block C and the size of the constraints G_C, A_C, which can probably help visualize the size of the block matrix.

If we truncate the gradient $\frac{d\mathbf{x}^*}{dp}$ to its C block, it is equivalent to remove the $\overline{C}$ part from K^{-1}, which gives us:

$$\left(\frac{d\mathbf{x}^*}{dp}\right)_{CC} = \begin{bmatrix} \mathbf{I} \\ 0 \\ 0 \end{bmatrix}^\top \left(K^{-1}\right)_{CC} \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{I} \\ 0 \\ 0 \end{bmatrix}^\top (K_4 - K_3K_1^{-1}K_2)^{-1} \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix}$$

Therefore, the difference between $(\frac{d\mathbf{x}^*}{dp})_{CC}$ and $\frac{d\mathbf{x}^*_C}{dp_C}$ can be bounded by:

$$\begin{aligned} & \left(\frac{d\mathbf{x}^*}{dp}\right)_{CC} - \frac{d\mathbf{x}^*_C}{dp_C} = \begin{bmatrix} \mathbf{I} \\ 0 \\ 0 \end{bmatrix}^\top (K_4 - K_3K_1^{-1}K_2)^{-1} \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{I} \\ 0 \\ 0 \end{bmatrix}^\top K_{CC}^{-1} \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix} \\ = & \begin{bmatrix} \mathbf{I} \\ 0 \\ 0 \end{bmatrix}^\top ((K_4 - K_3K_1^{-1}K_2)^{-1} - K_{CC}^{-1}) \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix} \\ = & \begin{bmatrix} \mathbf{I} \\ 0 \\ 0 \end{bmatrix}^\top (K_4 - K_3K_1^{-1}K_2)^{-1}(I - (K_4 - K_3K_1^{-1}K_2)K_{CC}^{-1}) \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix} \\ = & \begin{bmatrix} \mathbf{I} \\ 0 \\ 0 \end{bmatrix}^\top (K_4 - K_3K_1^{-1}K_2)^{-1}(K_3K_1^{-1}K_2K_{CC}^{-1}) \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix} \end{aligned} \tag{27}$$

where the last equality comes from the choice $K_4 = K_{CC}$, thus the identity matrix is canceled out. We can then bound the matrix norm of $K_3 K_1^{-1} K_2 K_{CC}^{-1}$ by:

$$\begin{aligned}\left\|K_3 K_1^{-1} K_2 K_{CC}^{-1}\right\| &\leq \|K_3 K_2\| \left\|Q_{\overline{CC}}^{-1}\right\| \left\|K_{CC}^{-1}\right\| \\ &\leq \frac{\max(\lambda^*, 1) \left\|K_2^\top K_2\right\|}{\mu_{\min}(Q_{\overline{CC}})} \left\|K_{CC}^{-1}\right\| \\ &\leq \frac{\Delta + \Delta_C}{\mu_{\min}(Q_{\overline{CC}})} \max(\lambda^*, 1) \left\|K_{CC}^{-1}\right\| \\ &\leq \frac{\Delta + \Delta_C}{\mu_{\min}(Q)} \max(\lambda^*, 1) \left\|K_{CC}^{-1}\right\| \qquad (28)\end{aligned}$$

where the second inequality is from the fact that K_3 is a matrix multiplication of $K_2^\top$ and a diagonal matrix with 1 and λ^* on the diagonal. The matrix norm can be bounded by the matrix norm of the diagonal matrix, thus $\max(\lambda^*, 1)$, and the remaining matrix multiplication $K_2^\top K_2$. The third inequality is due to the singular value $\left\|K_2^\top K_2\right\| = \left\|K_2 K_2^\top\right\| = \left\|Q_{C\overline{C}} Q_{\overline{C}C} + G_{\overline{C}}^\top G_{\overline{C}} + A_{\overline{C}}^\top A_{\overline{C}}\right\| \leq \left\|Q_{C\overline{C}} Q_{\overline{C}C} + G^\top G + A^\top A\right\| \leq \Delta + \Delta_C$, where the middle inequality is due to the fact that all these individual terms are positive semi-definite, so adding new postive semi-definite terms such that they become $G^\top G, A^\top A$ only increases the norm value.

Taking matrix norm to Inequality 27 and using Inequality 28 to substitute $\left\|K_3 K_1^{-1} K_2 K_{CC}^{-1}\right\|$, we can get:

$$\begin{aligned}&\left\|\left(\frac{d\mathbf{x}^*}{dp}\right)_{CC} - \frac{d\mathbf{x}_C^*}{dp_C}\right\| \\ &\leq \frac{\Delta + \Delta_C}{\mu_{\min}(Q)} \left\|K_{CC}^{-1}\right\| \left\|\begin{bmatrix}\mathbf{I} \\ 0 \\ 0\end{bmatrix} (K_4 - K_3 K_1^{-1} K_2)^{-1} \begin{bmatrix}-\mathbf{I} \\ 0 \\ 0\end{bmatrix}\right\| \\ &= \frac{\Delta + \Delta_C}{\mu_{\min}(Q)} \left\|K_{CC}^{-1}\right\| \left\|\left(\frac{d\mathbf{x}^*}{dp}\right)_{CC}\right\| \qquad (29)\end{aligned}$$

which concludes the proof. □

15.1 Discussion of Singularity of KKT Matrix

One biggest concern is whether the KKT matrices K and K_{CC} are singular. If the chosen K_{CC} is singular, then the bound provided in Theorem 7.3 becomes useless.

As discussed in the appendix of [8], in Equation 18, due to KKT condition, at least one of λ_i^* and $(G\mathbf{x}^* - h)_i$ is 0. Also as they discussed, when both of them are 0, the whole i-th row in $D(\lambda^*)G$ and $G\mathbf{x}^* - h$ is all 0. We can either impose new constraint or just remove the row to make the matrix non-singular.

If $\lambda_i^* = 0$ with $(G\mathbf{x}^* - h)_i > 0$, then in the i-th row, there is only the term $(G\mathbf{x}^* - h)_i$ being nonzero. Thus we can solve the equation in the i-th row by setting $(\frac{d\lambda^*}{dp})_i = 0$ and remove the row and the variable $(\frac{d\lambda^*}{dp})_i$ from the linear equation. Therefore, the linear equation and the matrix can be simplified by:

$$\begin{bmatrix} Q & G_I^\top & A^\top \\ D(\lambda_I^*)G_I & D(G\mathbf{x}^* - h)_I & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{d\mathbf{x}^*}{dp} \\ \frac{d\lambda_I^*}{dp} \\ \frac{d\nu^*}{dp} \end{bmatrix} = \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix}$$

where $I = \{i : \lambda_i^* \neq 0\}$. But notice that $(G\mathbf{x}^* - h)_i = 0$ due to the KKT conditions and the assumption of I. So we can simply write:

$$\begin{bmatrix} Q & G_I^\top & A^\top \\ D(\lambda_I^*)G_I & 0 & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{d\mathbf{x}^*}{dp} \\ \frac{d\lambda_I^*}{dp} \\ \frac{d\nu^*}{dp} \end{bmatrix} = \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix} \qquad (30)$$

Notice that we can assume $\begin{bmatrix} D(\lambda^*)G_I \\ A_I \end{bmatrix}$ to have a full row rank now. Equivalently, we can also assume $\begin{bmatrix} G_I^\top & A_I^\top \end{bmatrix}$ to have a full column rank.

15.2 Singularity of Block KKT Matrix

Given a simplified version of the non-singular full KKT matrix in Equation 30, we can write down the block KKT matrix as:

$$\begin{bmatrix} Q_{CC} & G_{I,C}^\top & A_C^\top \\ D(\lambda_I^*)G_{I,C} & 0 & 0 \\ A_C & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{d\mathbf{x}_C^*}{dp_C} \\ \frac{d\lambda_I^*}{dp_C} \\ \frac{d\nu^*}{dp_C} \end{bmatrix} = \begin{bmatrix} -\mathbf{I} \\ 0 \\ 0 \end{bmatrix} \qquad (31)$$

where $G_I = \begin{bmatrix} G_{I,C} & G_{I,\overline{C}} \end{bmatrix}, A_I = \begin{bmatrix} A_{I,C} & A_{I,\overline{C}} \end{bmatrix}$. In order to make the block KKT matrix non-singular, we need to select C such that $\begin{bmatrix} G_C \\ A_C \end{bmatrix}$ remains full row rank. In this case, the block KKT matrix will remain non-singular and thus invertible.

In practice, we cannot access to the dual varialbe λ^* before solving the QP and choosing the block C. But we can compute the slack variables $G\mathbf{x}^{\text{opt}} - h$ since $\mathbf{x}^{\text{opt}}$ is given. We need to make sure to make $G_{I,C}$ nonzero for $I = \{i : \lambda_i^* \neq 0\} \subset \{i : (G\mathbf{x}^{\text{opt}} - h)_i = 0\}$, or equivalently the indices of tight constraints.

Some choices of block C might make Equation 31 singular but still solvable. That is due to some dependent rows in $\begin{bmatrix} G_{I,C} \\ A_C \end{bmatrix}$, which admit to each others since the right hand side is all 0. This allows us to remove the redundant constraints and re-solve the linear equation by applying matrix inversion. But in this case, the block KKT matrix will not contain all the constraints, which leaves some constraints out of the block. Algorithm 2 still works but the K_1 in the proof of Theorem 7.3 is not just $Q_{\overline{CC}}$ but contains some additional terms from constraints excluded by the block quadratic program. The bound will also vary since we need to estimate the eigenvalue of K_1, which depends on the added constraints and does not have a simple form here.