Solving Online Threat Screening Games using Constrained Action Space Reinforcement Learning

Sanket Shah,1 Arunesh Sinha,1 Pradeep Varakantham,1 Andrew Perrault,2 Milind Tambe2
1School of Information Systems, Singapore Management University, {sankets, aruneshs, pradeepv}@smu.edu.sg
2Harvard University, {aperrault@g., milind_tambe@}harvard.edu

Abstract
Large-scale screening for potential threats with limited resources and capacity for screening is a problem of interest at airports, seaports, and other ports of entry. Adversaries can observe screening procedures and arrive at a time when there will be gaps in screening due to limited resource capacities. To capture this game between ports and adversaries, this problem has been previously represented as a Stackelberg game, referred to as a Threat Screening Game (TSG). Given the significant complexity associated with solving TSGs and uncertainty in arrivals of customers, existing work has assumed that screenees arrive and are allocated security resources at the beginning of the time window. In practice, screenees such as airport passengers arrive in bursts correlated with flight time and are not bound by fixed time windows. To address this, we propose an online threat screening model in which screening strategy is determined adaptively as a passenger arrives while satisfying a hard bound on acceptable risk of not screening a threat. To solve the online problem with a hard bound on risk, we formulate it as a Reinforcement Learning (RL) problem with constraints on the action space (hard bound on risk). We provide a novel way to impose a hard bound on risk while simultaneously minimizing delay time, we impose a hard bound on risk while simultaneously minimizing average delay time faced by screenees at no loss to security risk (measured as attacker utility) compared to past work. Further, while past models have used a weight to balance the risk of missing an attacker and average delay time, we show experimentally that this leads to a significantly reduce screenee wait time while guaranteeing a bound on risk.

Introduction
Screening for potential threats entering large safety-sensitive establishments (e.g., airports, seaports, museums) using the right subset of available screening methods (e.g., metal detectors, advanced imaging technology, pat-down) is an important defensive activity undertaken by various agencies around the world. However, the sheer scale of the problem at these large establishments with a large number of screenees and various screening methods makes screening in a timely fashion with limited resources quite challenging. For example, average delays for air passengers in Chicago, USA jumped to 2 hours due to higher passenger volume in the summer of 2016 (Jeffrey Dastin 2016). Additionally, intelligent adversaries can exploit any gaps in screening and cause catastrophic damage to these critical establishments. Screening gaps can arise due to the use of a less effective but faster screening method for high-risk passengers, which can happen as the combination of more passengers and limited resources can result in the unavailability of the “right” screening method.

Online resource allocation is a problem of interest in many domains including transportation (Simao et al. 2009) – allocating taxis to customers, emergency response (Maxwell et al. 2010) – allocating ambulances to emergencies, and airports – allocating terminals to arriving aeroplanes. Threat screening is also an online resource allocation problem, but in the presence of an observing adversary. Given that adversaries can monitor resource allocation strategies and exploit any gaps, we consider robust or risk-averse objectives rather than traditional expected objectives (e.g., expected revenue, expected delay). Hence, game-theoretic models and approaches have been considered for such problems.

One such model is the Threat Screening Game (TSG) model introduced in Brown et al. (2016), where a strategic attacker attempts to enter a secure area, while the screener uses teams of limited capacity screening resources with varying efficacy of catching the attacker to screen the screenees. However, despite enhancements made in subsequent work (McCarthy, Vayanos, and Tambe 2017), this model suffers from a lack of adaptability as screening strategies are fixed for every hour. Further, all versions of the model assume a favourable rate of passenger arrival within each time window such that no screening resource is idle within the hour. We address these shortcomings with a novel online allocation model and a completely new solution approach.

Our first contribution is an online version of the threat screening problem, in which the screening strategy is decided adaptively, based on the current queue lengths, as the screenees arrive. We show experimentally that this leads to a much better characterization and optimization of the average delay time faced by screenees at no loss to security risk (measured as attacker utility) compared to past work. Further, while past models have used a weight to balance the risk of missing an attacker and average delay time, we impose a hard bound on risk while simultaneously minimizing delay. We show that, given uncertain and unknown passenger arrivals, the online model can be solved as a Reinforcement Learning (RL) problem with continuous action
space where the hard bound on risk translates to hard constraints on the action space. We show mathematically that the choice of hard bound on risk is not different from the one where a weighted defender objective is maximized and we can switch back and forth between these two seemingly different optimization goals. Our mathematical analysis also reveals the game-theoretic nature of this formulation.

Our second contribution is a novel method to efficiently impose hard constraints on actions in Deep RL by using what we call \( \alpha \)-projection. In contrast to prior approaches (see Section 2), our approach guarantees that the constraint is never violated (even during training) while also being much more scalable in training as well as execution. The main component of our method is an extremely efficient mapping of infeasible actions to the feasible space specified by the constraints.

Finally, our third contribution is a set of experiments that reveal why and how prior TSG models fail to handle realistic continuous arrival of passengers in bursts. The experiments also show that our approach achieves the same risk as prior models but improves upon the average delay by 100% in the best case and 25% on average. Overall, the realism of our model coupled with a novel scalable RL solution method makes our approach appealing for practical large scale threat screening problems.

### Related Work

**TSG and Security Games**

There have been quite a few papers published on various aspects of threat screening games. The early papers (Brown et al. 2016; Schlenker et al. 2017) make two stringent assumptions—first, they assume perfect prior knowledge of passenger arrivals (for one hour time windows) and second, they implicitly assume that all passengers are screened within the same window in which they arrive, with no screening resource being idle (thus, delay is not an explicit consideration in this work). Both these assumptions are unrealistic in practice as, clearly, there is uncertainty in the number of passengers arriving in any time window and passengers arrive in bursts that are correlated with the flight timings. We show how these assumptions result in sub-optimal outcomes in practice.

Later papers (McCarthy, Vayanos, and Tambe 2017; McCarthy et al. 2018) attempt to account for uncertainty in arrivals across time windows and relax the second assumption by allowing an overflow of passengers from one time window to next. However, their approach is simply unscalable and hence impractical for real-world application. They show solutions for only up to 15 flights for a full day. Additionally, because the solution goal chosen is inspired from robust optimization, the method tries to find a solution that minimizes risk and delay across any realizable sample, which results in a pessimistic solution. The solution approach also makes the approximation of calculating the worst case from a sample of arrivals and, as a result, cannot guarantee that the solution will bound the true worst-case risk. Moreover, a problematic assumption from earlier work about the passengers arriving at a favourable rate within a time window continues to be assumed in this later work. In this paper, we propose a scalable online model without the restrictive assumptions of past work: we guarantee a bound on the worst-case risk and simultaneously minimize the average delay. Our online model allows for fine-grained adaptivity at the level of each passenger arrival as opposed to the hourly time-window adaptations in past work and also scales up to a large number of flights.

In other applications of security games played over multiple time steps, there has been work in which the defender strategy is a policy for an MDP or a sequence of actions (Delle Fave et al. 2014; Bosansky et al. 2015). This work assumes that the MDP or game parameters are known beforehand and is hence a planning problem rather than a learning problem. Additionally, there has been work in this space where the double oracle approach has been used in tandem with Deep RL to compute the equilibrium (Wang et al. 2019; Wright, Wang, and Wellman 2019). In this work, however, there is only a single constraint on actions (actions sum to one), which is readily enforced using a softmax layer. In this paper, however, our main contribution to Deep RL is in enforcing multiple arbitrary linear inequality constraints efficiently. There is also theoretical work on solving security games in an extensive form (Letchford and Conitzer 2010; Kroer, Farina, and Sandholm 2018; Černý, Boıyansky, and Kiekintveld 2018; Basilio, Gatti, and Amigoni 2009) or stochastic game (Letchford et al. 2012) setting. Again, these assume complete knowledge of the game structure including transition functions whereas we focus on learning aspects. Also, whereas learning in the context of security games has appeared in the literature (Balcan et al. 2015; Letchford, Conitzer, and Munagala 2009), this paper introduces an RL-based approach to threat screening for the first time.

**Constrained Action-Space RL**

Historically, dealing with constraints on the action space in Deep RL has been a challenging task. This is exacerbated by the fact that our action space is continuous. The most common and intuitive technique is to discourage disallowed actions with a penalty. This method does not guarantee that the security risk will be bounded, however, and given that the risk is the worst-case allocation across an episode, the probability and extent of violation increases with scale. Given the adversarial nature of the security problem, this method is unsuitable.

Recently, Pham, De Magistris, and Tachibana (2018) have suggested enforcing constraints by projecting any unconstrained point onto the constrained space by solving an optimization program that minimizes the L2 distance and back-propagating through it to train the network (Amos and Koltzer 2017). This approach is very time consuming as it requires solving a quadratic program (QP) in the forward pass in every training iteration and, as a result, does not scale to problems with large dimensional action spaces (Amos and Koltzer 2017) seen in practical screening problems.

Our RL approach is similar in spirit to Bhatia, Varakantham, and Kumar (2019), which uses a complicated variable-length iterative approximation of the L2 projection to deal
with a specific subset of linear constraints faster than [Pham, De Magistris, and Tachibana (2018)]. The type of linear constraints they can handle are constraints on the sum of sets of variables, where these sets must form a hierarchy. In contrast, the approach that we propose can handle arbitrary linear constraints, that is, in which the coefficients take any real values. Moreover, our approach is simple, can be computed in a single step, and is easy to implement as gradients can be computed using automated symbolic differentiation.

**MDP Model of TSG**

Our main departure from past TSG work is that we determine the screening strategy for a passenger when they arrive. Thus, while the model below reuses various notions from past versions of the TSG model, it models the online nature by formulating the problem as a Markov Decision Process (MDP). The MDP treatment considerably simplifies the problem of dealing with passenger arrival uncertainty by incorporating those in the MDP transition model.

The basic structure of the screening problem stays the same as prior TSG models: every arriving passenger has a category \( c \in C \), which is made up of two parts \( \langle \theta, \kappa \rangle \), where \( \theta \) is the part of the category that the attacker cannot control (risk level determined by screener) and \( \kappa \) is the part that they can control (which flight to take). The screening resource types comprise the set \( R \); for example, \( R \) could be \{X-Ray, Metal Detector, Advanced imaging\}. Each resource type \( r \in R \) has a rate of screening (called capacity in prior work) given by \( f_r \) measured in units of passengers per unit time. The passengers are screened by a team (set) of screening resources where the set of all teams \( T \) is given a priori \( (T \subset 2^R) \). An attacker, apart from choosing a flight, uses an attack method \( m \) (e.g., knife or gun) and each team \( t \) has an effectiveness (probability) \( E_{t,m} \) of detecting attack method \( m \). \( U_{t,m}^\kappa, m \) is the defender’s utility for detecting an attacker with attacker’s choice being \( \kappa, m \) and \( U_{t,m}^- \) is the utility of not doing so. As in previous TSG models, the adversary’s utilities are the negation of these values. The defender has a belief about the attacker’s uncontrollable category \( \theta \) given by \( P_\theta \) with \( \sum_\theta P_\theta = 1 \).

**MDP Model**

Next, we describe our MDP formulation, which prescribes an online screening strategy for each arriving passenger. This is unlike past approaches in which the same randomized screening strategy was used for every passenger of a given category that arrived in the same time window.

- **States:** The state at any given point in time is a combination of 4 quantities \( \langle c, \xi, h, \tau \rangle \). The first, \( c \), is the category of the passenger that has arrived for screening and we have to allocate security resources to. The remaining quantities summarise the history and provide information about the current context. \( \xi \in \mathbb{R}^{|R|} \) encodes the number of passengers (or part thereof) in the queue for each resource at this current point in time. \( h \in \mathbb{Z}_+^{|C|} \) is a summary of the history: it is the number of passengers from every category that have already been screened. \( \tau \) is the wall clock time when the passenger arrives.

- **Actions:** An action \( \pi_t \in \mathbb{R}^{|T|} \) at time step \( t \) is a randomized allocation of the just-arrived passenger to teams. We use the insight that the risk is a function of the policy and does not depend on passenger arrivals to codify the hard bound as risk as constraints on the action space. Only actions with risk less than the specified risk level are allowed. Risk in TSGs is measured as the expected adversary utility, which is the negation of the utility of the defender. This leads to \( m \) different inequalities constraints (one for each attack method) on the action stated in terms of defender utility.

\[
P_\theta \geq \left[ z_m U_{t,m}^\kappa, m \right] \geq -\psi_{\theta} \quad \forall m, \quad (1)
\]

\[
\sum_{t \in T} E_{t,m} \pi_t = z_m \quad \forall \theta, \quad \text{and} \quad \sum_{t} \pi_t = 1
\]

Given a marginal policy \( \pi, z_m \) is the overall probability that one of the teams \( t \in T \) will detect an attack of type \( m \). The last equality constraint enforces that \( \pi_t \) is a probability distribution over teams. In order to explain the first set of inequalities, we first state the defender detection utility explicitly. \( U_{t,m} = z_m U_{t,m}^\kappa, m + (1 - z_m)U_{t,m}^- \) is the expected utility of the defender if the current passenger is an attacker. The utility \( U_\theta = \sum_\kappa U_{\kappa,m} \) is the worst case expected utility when the attacker is one with uncontrollable category \( \theta \). \( \sum_\theta P_\theta U_\theta \) is the overall defender detection expected utility. We wish to impose a lower bound \( -\psi_{\theta} \) on \( P_\theta U_\theta \) which indirectly lower bounds \( \sum_\theta P_\theta U_\theta \) (or in other words, upper bounds risk). It is easy to see that the first set of inequalities is \( P_\theta \geq \left( \min_{\kappa,m} U_{\kappa,m} \right) \geq -\psi_{\theta} \).

- **Transitions:** The transition from \( \langle c, \xi, h, \tau \rangle \) to \( \langle c', \xi', h', \tau' \rangle \) can be decomposed into three parts. The first is how the allocation at the previous step and passage of wall clock time since then affects the queues for each resource. In previous TSG models, passengers in the resources queues are screened according to the screening rate of a given category that have arrived in the same time window.

- **Rewards:** The reward for each time step \( t \) is the negative of the expected wait time of the currently arrived passenger. The wait time is determined by the maximum wait
time over all resources in the realized team allocation. The wait time for each resource \( r \) is determined by the screening rate \( f_r \) and the number of passengers \( \xi_r \) already in queue for that resource: \( \xi_r/f_r \). We use \( U_{o,t} \) to denote the delay reward at time \( t \) (note \( U_{o,t} \) is negative). The value (long term reward) is given by \( V_o = \mathbb{E}([1/N] \sum_{t=1}^{N} U_{o,t}) \), where \( N \) passengers arrive in a day (implicitly conditional on the start state with empty history).

Relationship to Game Theory

In the above constrained RL problem, the defender learns a policy which is a mixed strategy of the defender (mixed since the allocation at each time step is randomized). The adversary observes this policy and chooses an optimal attack \( a \), which is a combination of \( \kappa \) and attack method \( m \). Thus, this is a Stackelberg game setting, similar to prior models of TSG. There are two components of the defender’s value function: (a) the risk of getting, similar to prior models of TSG. The optimal attack method \( \kappa,m \) is determined at each time step is randomized). The above \( \kappa,m \) is determined by the screen-frontier a priori by solving for the optimal policy for many values of \( \psi \) where these values are uniformly spaced and distributed throughout the possible space of \( \psi \) values. Then, when given \( \psi \), the solution will choose one of the specific points for which the output \( \pi \) maximizes \( \sum_{\theta} P_\theta U_\theta + \psi \) over all points considered in the \( \psi \) space.

Overall Solution

To solve the screening problem modelled in Section, we use Reinforcement Learning (RL). We use techniques from RL instead of trying to solve the MDP directly because the exact passenger arrival distribution is unknown. Rather than trying to model the distribution explicitly, we use model-free RL techniques to jointly learn the distribution and the optimal policy.

Specifically, we use the Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. 2015) algorithm that is a state-of-the-art technique in Deep Reinforcement Learning literature. DDPG is an extension of the standard actor-critic approach that allows the modelling of a continuous action space like the one present in our problem. However, DDPG cannot enforce general action-space constraints as is.

To deal with this, we propose an \( \alpha \)-projection layer in Section that enforces constraints on the output of the previous layer. We then modify the standard DDPG algorithm by adding this \( \alpha \)-projection layer on top of the output layer of the actor network. This ensures that any action produced by the actor satisfies the risk constraints on the action space. This combination of DDPG and \( \alpha \)-projection represents the overall approach that we use to solve the MDP.

Approach for Linear Constraints on Action Space in Deep RL

All prior approaches for imposing hard constraints on the action output of any policy neural network use a layer(s) at the end of the network to map the unconstrained output from intermediate layers to an output in the feasible space. Mathematically, suppose the output must lie in a feasible space \( Y \) defined by linear inequality constraints. Let \( f(x) \) be the output of the intermediate layer, given input \( x \). The last layer(s) define a mapping \( M \) such that \( M(f(x)) \) lies in the feasible space \( Y \). For our problem, \( Y \) is a fixed polytope for a given problem instance (see Equation [1]). Typically, such mappings \( M \) have been some type of \( L_\mu \) projec-
output, thus, any choice of the loss. The purpose of $\alpha$ We propose a very simple mapping $\alpha f$ so that will work since the neural network will appropriately adjust the whole neural network is the function composition $y$.

Figure 1: Left: The red points are produced by an intermediate layer. The $\alpha$-projection is shown with cross marks. If the red point is inside the feasible region, the $\alpha$-projection of the point is the point itself. For one red point (inside the small dashed circle), the L2 projection is shown with the arrow. Right: A zoomed-in version of the dashed circle. This shows that if the L2 projection of $P1$ is the optimal point (green), then the intermediate layer will adjust its output to $P1^*$ to get the optimal point with the $\alpha$-projection.

$\alpha$-projection
We propose a very simple mapping $M$, which we call $\alpha$-projection. We start by finding a feasible point $y_0 \in \text{int}(Y)$, where $\text{int}(Y)$ are all the interior points of $Y$. Then, we find the maximum $\alpha \in [0, 1]$ such that $y = \alpha \cdot f(x) + (1 - \alpha) \cdot y_0$ and $y \in Y$, with the final output being the point $y$, so $M(f(x)) = y$. Intuitively, we join $f(x)$ and $y_0$ with a line and choose the closest point to $f(x)$ on this line that lies in $Y$, which could be $f(x)$ itself if $f(x) \in Y$. See Figure 1.

Forward pass $\alpha$-projection turns out to be very efficient for linear constraints such as those in the TSG-RL problem. For training neural networks, every iteration requires a forward pass for the network also. An optimization layer (such as our $\alpha$-projection layer) requires solving an optimization problem. However, our simple mapping allows obtaining $\alpha$ in a closed-form with no need to solve expensive optimizations. To get to the closed-form, first, lets consider $m$ inequality constraint with the $i$th constraint being $a_i \cdot y \leq b_i$. Using our mapping, this $i$th constraint is $\alpha \cdot (a_i \cdot f(x)) + (1 - \alpha) \cdot (a_i \cdot y_0) \leq b_i$. Thus, $\alpha \cdot (a_i \cdot f(x) - a_i \cdot y_0) \leq b_i - a_i \cdot y_0$. We get a similar upper (or lower, depending on the sign of $a_i \cdot f(x) - a_i \cdot y_0$) bound for $\alpha$ for every inequality constraint and then $\alpha$ is the simply the highest value in $[0, 1]$ that satisfies all these bounds. All these bounds are closed-form formulas, thus, computing $\alpha$ is very efficient for the forward pass, and obtaining $y = \alpha \cdot f(x) + (1 - \alpha) \cdot y_0$ is easy.

Gradients: Computing the gradient (for backpropagation) of $\alpha$ w.r.t. the input $f(x)$ to the mapping layer is also easy. For readability, we write $s$ instead of $f(x)$. As stated in the previous paragraph $\alpha$ is the minimum of a number of upper bounds (ignoring lower bounds), where these upper bounds are given closed-form functions $b_i(s)$ ($y_0$ is a constant). Thus, $\alpha = \min_{s} (b_1(s), \ldots, b_m(s))$ for some $k < m$. For notational ease, let $b_0(s) = 1$. For any specific $s_0$, if there is a unique index $j$ for which $\alpha = b_j(s)$ then the gradient is simply $\nabla b_j(s)$, which is if $j = 0$. If there is a set of indices $\{J, |J| \geq 1\}$ and $\alpha = b_j(s)$ for all $j \in J$, then the gradient is simply $(1/|J|) \cdot (\sum_{j \in J} \nabla b_j(s))$.

Handling equality constraints: For equality constraints, say $k$ constraints, the general approach would be to eliminate $k$ variables using Gaussian elimination (or any other method) and then deal with the only inequalities in this new space. However, for our TSG problem, we only have one equality constraint, which is a probability simplex constraint that can be easily enforced by a softmax layer. With a slight abuse of notation, we use $s$ to denote the output of the softmax whose input is the unconstrained output $f(x)$, $s$ satisfies the probability simplex constraint. Additionally, we choose $y_0$ such that it also satisfies the probability simplex constraint. Then, the final output $\alpha \cdot s + (1 - \alpha) \cdot y_0$ satisfies the probability simplex constraint and all the inequality constraints. Thus, for our problem, the overall mapping is made of 2 layers: a softmax followed by the $\alpha$-projection layer.

Choosing an Interior Point
The choice of $y_0$ is important. First, $y_0$ should be an interior point. Otherwise, if $y_0$ is on an external face (hyperplane) of the $Y$ polytope, all external points on the side of the face not containing the polytope will map to $y_0$. This violates nonzero gradients property of a feasible mapping, as all points on one side of the hyperplane will have zero gradients. We
also find that we get better performance when $y_0$ is near the centre of the polytope. While there are many different types of centres, we choose the Chebyshev centre of a polytope because: (1) Chebyshev centre can be computed efficiently by solving a linear program and (b) the Chebyshev centre maximizes the minimum distance from the faces of the polytope, that is, making sure the centre is far from bad points. Informally, Chebyshev centre is the centre of the largest ball that fits inside the polytope. Note that we need to compute the Chebyshev centre only once for our polytope $Y$ and that this computation time is of the order of seconds.

Experiments
In line with past work on TSGs, we evaluate the performance of our approach on the airport passenger screening domain. For the most simple head-to-head comparison, we look at the difference in solution quality between our approach and past work within single time-window. Brown et al. [2016] and McCarthy, Vayanos, and Tambe [2017] both have the same optimal solution in this case and the optimal marginal solution can be found by using a simple linear program (LP). When we compare the solutions of the LP to our approach, we control for the risk and measure the corresponding difference in delay. Specifically, we take the difference in risk levels associated with the uncontrollable categories $\psi_\theta$ from the solution of the LP and run our approach using those as the risk threshold in the risk constraints of our approach. We then test both sets of policies (LP and ours) using an online simulator and compare the ratio of average delays obtained.

We construct our problem instances using the description in Brown et al. [2016] and McCarthy, Vayanos, and Tambe [2017]. The attacker utility associated with successfully launching an attack $U^+$ is sampled from a uniform distribution over $[1, 10]$, while the utility of failing to launch an attack $U^-$ is set to 0. The game is zero-sum and, as a result, the defender utilities are the negation of the attacker utilities. There are 3 attack methods $m$, 5 uncontrollable screening resource types $|R|$. The efficacies (probability of detection) of different resources are sampled from a uniform random distribution over $[0, 1]$ for each attack method. We create 10 random 2-sized combinations of resources to represent the teams $T$ and their efficacies are found assuming that the efficacies of their associated resources are independent. We choose a passenger arrival distribution as used in McCarthy, Vayanos, and Tambe [2017], and consider arrivals to be normally distributed over a 3-hour window leading up to the departure of the flight. We combine this with real flight departure times taken from one of the busiest airports in the world to generate a realistic arrival distribution of passengers. The default number of flight types for experiments is 10.

Finally, for runtime, all past methods have used non-gradient based optimization methods and have reported runtimes for programs that have run on CPUs. As has been observed in other domains, GPUs offer a huge advantage due to the immense parallelization of matrix operations for neural network optimization. However, to perform a fair comparison to past work, we run all our experiments on a CPU. Thus, while our scalability results show the runtime trend

Figure 2: Static vs. Adaptive: We measure the ratio of average delays of the LP to average delays of our online approach (so higher is better) across different problem sizes. With increasing problem size, the absolute wall clock time can be much better with GPUs.

Static vs. Adaptive
First, we show that our online approach outperforms past window-based approaches even when the problem size is large. The experiments here are evaluated for 30 random game instances and averaged across 100 samples of passenger arrival sequence. The results can be seen in Figure 2.

For the one time-window problem, improvement in solution quality comes from the fact that past work has a static policy within one time-window, whereas our solution can adapt based on the actual number of passenger arrivals. As a result, our approach can exploit the structure present within a time-window. The reason our improvement decreases with an increase in the number of flights is because, as the problem size increases, the structure present in a randomly generated problem decreases. For example, with many overlapping Gaussian distribution of passenger arrivals, the overall arrival is almost uniform which we show leads to a lower gain (see Section).

Moreover, the performance improvement within one time-window is a lower bound on the amount of improvement we can achieve over past methods. The solution quality associated with past methods deteriorates with an increase in the number of windows (details in Section) while there is no notion of time windows in our model and hence no degradation over long periods.

Delay vs. Risk
In Figure 3, we show the inherent trade-off between risk and average delay by varying $\psi$ (keeping the ratio of $\psi_\theta$ fixed across $\theta$) and measuring its effect on the delay. To do this, we take the optimal risk level obtained by solving the LP and then measure the impact on average delay as we relax it. The results show average delay as a fraction of the delay obtained at the LP optimal risk. The resulting curve can be seen as the Pareto frontier described in Section restricted to the case where the ratio of $\psi_\theta$ fixed across $\theta$. As expected, less stringent risk requirements result in lower average delay.
Figure 3: **Delay vs. Risk**: We measure how the delay decreases (as fraction of delay at LP optimal risk) with increasing risk allowance.

Figure 4: **Scalability in training steps**: We measure the actor loss of our DDPG network as a function of the number of mini-batches that are fed to it as input.

**Scalability**

In Figure 4, we show how training time is affected by the size of the game instance. Given that neural networks are not guaranteed optimal, measuring this is slightly challenging but we use the metric of the time at which our DDPG’s actor network converges to measure how long the training time takes. As we see in the figure, the number of steps to convergence is about the same, regardless of the number of flights. Based on Figure 4, we choose 10,000 training steps as the number of steps for convergence.

Of course, as the input size increases with increasing number of flights, the time taken per training step increases, thus, the actual wall clock time to get to 10,000 steps for different number of flights varies. Figure 5 show the actual wall clock time for convergence (to 10,000 steps) with varying number of flights. The increase appears linear showing the scalability of our approach (as a reminder these results are not even using GPUs). In contrast, past work (Brown et al. 2016; McCarthy, Vayanos, and Tambe 2017) scale highly non-linearly with number of flights and have shown solutions only up to 50 flight and 15 flights respectively.

**Delay vs. Variance**

In Figure 6, we look at how the variance associated with the passenger arrival distribution affects our gain over the time window based solutions. Here the x-axis is measured using $2\sigma$ because of the intuition that 95% of passengers of a flight arrive within a 2 standard deviation window around the mean. This graph can be interpreted as the effect that changing the width of the arrival window (of 95% passengers) has on solution quality. We vary it from 0 to 5 hours.

We find that as the variance associated with arrivals increases, the gain obtained by using an online approach as ours decreases. We believe that this is because the amount of structure present in the problem decreases as the variance increases. In the limit, when the variance is infinity, the arrivals are uniformly distributed, memory-less, and resemble a Poisson process. As a result, there is no information to be gained whenever the next passenger arrives, hence the per-passenger adaptive solution would do as good as one per-time window adaptive solution in this limiting case. Conversely, if the same number of flights arrive over a longer
duration, say 24 hours, our algorithm would do consider-
ably better in terms of the average delay since the arrival
windows are less likely to overlap, resulting in smaller over-
all variance.

Conclusion
In summary, we proposed a novel model for threat screen-
ing that captures inherent features of the problem such as
continuous arrival of screenees. We then provided an RL-
based method to solve the model which includes the novel
α-projection method for imposing hard constraints on ac-
tions. We believe these advances make our approach for
threat screening realistic and applicable in practice.

Acknowledgement
This research was supported by the Singapore Ministry of
Education Academic Research Fund (AcRF) Tier 2 grant
MOE2016-T2-1-174 and Ministry of Education Academic
Research Fund (AcRF) Tier 1 grant 19-C220-SMU-011.

References
[Abadi et al. 2015] Abadi, M.; Agarwal, A.; Barham, P.;
Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.;
Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp,
A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.;
Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.;
Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.;
Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasude-
van, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.;
Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-
scale machine learning on heterogeneous systems. Software
available from tensorflow.org.
[Amos and Kolter 2017] Amos, B., and Kolter, J. Z. 2017.
Optnet: Differentiable optimization as a layer in neural net-
works. In Proceedings of the 34th International Conference
on Machine Learning—Volume 70, 136–145. JMLR. org.
[Balcan et al. 2015] Balcan, M.-F.; Blum, A.; Haghtalab, N.;
and Procaccia, A. D. 2015. Commitment without regrets:
Online learning in stackelberg security games. In Proce-
ceedings of the sixteenth ACM EC, 61–78. ACM.
[Basilico, Gatti, and Amigoni 2009] Basilico, N.; Gatti, N.;
and Amigoni, F. 2009. Leader-follower strategies for robotic
patrolling in environments with arbitrary topologies.
In Proceedings of The 8th International Conference on Au-
tonomous Agents and Multiagent Systems—Volume 1, 57–64.
International Foundation for Autonomous Agents and Mul-
tiagent Systems.
[Bhatia, Varakantham, and Kumar 2019] Bhatia, A.;
Varakantham, P.; and Kumar, A. 2019. Resource
constrained deep reinforcement learning. In Proceed-
ings of the International Conference on Automated Planning
and Scheduling, volume 29, 610–620.
[Bosansky et al. 2015] Bosansky, B.; Jiang, A. X.; Tambe,
M.; and Kiekintveld, C. 2015. Combining compact rep-
resentation and incremental generation in large games with
sequential strategies. In Twenty-Ninth AAAI Conference on
Artificial Intelligence.
[Pham, De Magistris, and Tachibana 2018] Pham, T.-H.; De Magistris, G.; and Tachibana, R. 2018. Optlayer-practical constrained optimization for deep reinforcement learning in the real world. In 2018 IEEE International Conference on Robotics and Automation (ICRA), 6236–6243. IEEE.

[Schlenker et al. 2017] Schlenker, A.; Xu, H.; Guirguis, M.; Kiekintveld, C.; Sinha, A.; Tambe, M.; Sonya, S. Y.; Balderas, D.; and Dunstatter, N. 2017. Don’t bury your head in warnings: A game-theoretic approach for intelligent allocation of cyber-security alerts. In IJCAI.

[Simao et al. 2009] Simao, H. P.; Day, J.; George, A. P.; Gifford, T.; Nienow, J.; and Powell, W. B. 2009. An approximate dynamic programming algorithm for large-scale fleet management: A case application. Transportation Science 43(2):178–197.

[Wang et al. 2019] Wang, Y.; Shi, Z. R.; Yu, L.; Wu, Y.; Singh, R.; Joppa, L.; and Fang, F. 2019. Deep reinforcement learning for green security games with real-time information. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, 1401–1408.

[Wright, Wang, and Wellman 2019] Wright, M.; Wang, Y.; and Wellman, M. P. 2019. Iterated deep reinforcement learning in games: History-aware training for improved stability. In Proceedings of the 2019 ACM Conference on Economics and Computation, 617–636. ACM.