A Unified Method for Handling Discrete and Continuous Uncertainty in Bayesian Stackelberg Games

Zhengyu Yin and Milind Tambe
University of Southern California, Los Angeles, CA 90089, USA
{zhengyuy, tambe}@usc.edu

ABSTRACT

Given their existing and potential real-world security applications, Bayesian Stackelberg games have received significant research interest [3, 12, 8]. In these games, the defender acts as a leader, and the many different follower types model the uncertainty over discrete attacker types. Unfortunately since solving such games is an NP-hard problem, scale-up has remained a difficult challenge.

This paper scales up Bayesian Stackelberg games, providing a novel unified approach to handling uncertainty not only over discrete follower types but also other key continuously distributed real world uncertainty, due to the leader's execution error, the follower's observation error, and continuous payoff uncertainty. To that end, this paper provides contributions in two parts. First, we present a new algorithm for Bayesian Stackelberg games, called HUNTER, to scale up the number of types. HUNTER combines the following five key features: i) efficient pruning via a best-first search of the leader's strategy space; ii) a novel linear program for computing tight upper bounds for this search; iii) using Bender's decomposition for solving the upper bound linear program efficiently; iv) efficient inheritance of Bender's cuts from parent to child; v) an efficient heuristic branching rule. Our experiments show that HUNTER provides orders of magnitude speedups over the best existing methods to handle discrete follower types. In the second part, we show HUNTER's efficiency for Bayesian Stackelberg games can be exploited to also handle the continuous uncertainty using sample average approximation. We experimentally show that our HUNTER-based approach also outperforms latest robust solution methods under continuously distributed uncertainty.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

Algorithms, Security, Performance

Keywords

Game theory, Bayesian Stackelberg Games, Relaxation

1. INTRODUCTION

Due to their significance in real-world security, there has been a lot of recent research activity in leader-follower Stackelberg games, oriented towards producing deployed solutions: ARMOR at LAX [9], IRIS for Federal Air Marshals Service [9], and GUARDS for the TSA [14]. Bayesian extension to Stackelberg game has been used to model the uncertainty over players' preferences [12, 8] by allowing multiple discrete follower types, as well as, by use of sampling-based algorithms, continuous payoff uncertainty [10].

The key idea in this paper is to scale-up Bayesian Stackelberg games, providing a novel unified approach to handling not only discrete follower types but also continuous uncertainty. Scalability of discrete follower types is essential in domains such as road network security [6], where each follower type could represent a criminal attempting to follow a certain path. Scaling up the number of types is also necessary for the sampling-based algorithms to obtain high quality solutions under continuous uncertainty. Unfortunately, such scale-up remains difficult, as finding the equilibrium of a Bayesian Stackelberg game is NP-hard [5]. Indeed, despite the recent algorithmic advancement including Multiple-LPs [5], DOBSS [12], HBGS [8], none of these techniques can handle games with more than ≈ 50 types, even when the number of actions per player is as few as 5: inadequate both for scale-up in discrete follower types and for sampling-based approaches. This scale-up difficulty has led to an entirely new set of algorithms developed for handling continuous payoff uncertainty [10], and continuous observation and execution error [16]; these algorithms do not handle discrete follower types, however.

This paper provides contributions in two parts. In the first part, to address the challenge of discrete uncertainty, we propose a novel algorithm for solving Bayesian Stackelberg games, called HUNTER, combining the following five key ideas. First, it conducts a best-first search in the follower's best-response assignment space, which only expands a small number of nodes (within an exponentially large assignment space). Second, HUNTER computes tight upper bounds to speed up this search using a novel linear program. Third, HUNTER solves this linear program efficiently using Bender's decomposition. Fourth, we show that the Bender's cuts generated in a parent node are valid cuts for its children, providing further speedups. Finally, HUNTER deploys a heuristic branching rule to further improve efficiency. Thus, this paper's contribution is in combining an AI search technique (best-first search) with multiple techniques from Operations Research (disjunctive program and Bender's decomposition) to provide a novel efficient algorithm; the application of these techniques for solving Stackelberg games had not been explored earlier, and thus their application towards solving these games, as well as their particular synergistic combination in HUNTER are both novel. Our experiments show HUNTER dramatically improves the scalability of the number of types over other

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

existing approaches [12, 8].

In the second part of our contribution, we show that via sample average approximation, HUNTER for Bayesian Stackelberg games can be used in handling continuously distributed uncertainty such as the leader's execution error, the follower's observation noise, and both players' preference uncertainty. For comparison, we consider a class of Stackelberg games motivated by security applications, and *enhance* two existing robust solution methods, BRASS [13] and RECON [16] to handle such uncertainty. We again show that HUNTER provides significantly better performance than BRASS and RECON. Our final set of experiments in this paper also illustrates HUNTER's *unique ability* to handle both discrete and continuous uncertainty within a single problem.

2. BACKGROUND AND NOTATION

The first part of the paper is focused on solving Bayesian Stackelberg games with discrete follower types; this background section focuses on such games. A Stackelberg game is a two-person game played by a leader and a follower [15]. Following the recent literature [8], we focus on Stackelberg games where the leader commits to a mixed strategy first, and the follower observes the leader's strategy and responds with a pure strategy, maximizing his utility correspondingly. We generalize this set-up by extending the definition of the leader's strategy space and the leader and follower utilities in two ways beyond what was previously considered [12, 8] and by allowing for compact representation of constraints.

We assume the leader's mixed strategy is an N-dimensional real column vector $\mathbf{x} \in \mathbb{R}^N$, bounded by a polytope $A\mathbf{x} \preceq \mathbf{b}, \mathbf{x} \succeq \mathbf{0}$, which generalizes the traditional constraint of $\sum_i x_i = 1$ and allows for compact strategy representation with constraints as in [9] (although such constraints are not the focus of this paper). Second, given a leader's strategy $\mathbf{x}$, the follower maximizes his utility by choosing from J pure strategies. For each pure strategy $j = 1, \ldots, J$ played by the follower, the leader gets a utility of $\boldsymbol{\mu}_j^{\mathrm{T}}\mathbf{x} + \mu_{j,0}$ and the follower gets a utility of $\boldsymbol{\nu}_j^{\mathrm{T}}\mathbf{x} + \nu_{j,0}$, where $\boldsymbol{\mu}_j, \boldsymbol{\nu}_j$ are real vectors in $\mathbb{R}^N$ and $\mu_{j,0}, \nu_{j,0} \in \mathbb{R}$. This use of $\mu_{j,0}$, $\nu_{j,0}$ terms generalizes the utility functions.

We now define the leader's utility matrix U and the follower's utility matrix V as the following,

$$U = \begin{pmatrix} \mu_{1,0} & \cdots & \mu_{J,0} \\ \boldsymbol{\mu}_1 & \cdots & \boldsymbol{\mu}_J \end{pmatrix}, V = \begin{pmatrix} \nu_{1,0} & \cdots & \nu_{J,0} \\ \boldsymbol{\nu}_1 & \cdots & \boldsymbol{\nu}_J \end{pmatrix}.$$

Then for a leader's strategy $\mathbf{x}$, the leader and follower's J utilities for the follower's J pure strategies are $U^{\mathrm{T}}\begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}$ and $V^{\mathrm{T}}\begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}$.

A Bayesian extension to the Stackelberg game allows multiple types of players, each with its own payoff matrix. We represent a Bayesian Stackelberg game with S follower types by a set of utility matrix pairs $(U^1, V^1), \ldots, (U^S, V^S)$, each corresponding to a type. A type s has a prior probability p^s representing the likelihood of its occurrence. The leader commits to a mixed strategy without knowing the type of the follower she faces. The follower, however, knows his own type s, and plays the best response $j^s \in \{1, \ldots, J\}$ according to his utility matrix V^s. A strategy profile in a Bayesian Stackelberg game is $\langle \mathbf{x}, \mathbf{j} \rangle$, a pair of leader's mixed strategy $\mathbf{x}$ and follower's response $\mathbf{j}$, where $\mathbf{j} = \langle j^1, \ldots, j^S \rangle$ denotes a vector of the follower's responses for all types.

Type 1	*Target1*	*Target2*
Target1	1, -1	-1, 0
Target2	0, 1	1, -1

Type 2	*Target1*	*Target2*
Target1	1, -1	-1, 1
Target2	0, 1	1, -1

Figure 1: Payoff matrices of a Bayesian Stackelberg game.

The solution concept of interest is a *Strong Stackelberg Equilibrium* (SSE) [15], where the leader maximizes her expected utility assuming the follower chooses the best response and breaks ties in favor of the leader for each type. Formally, let $u(\mathbf{x}, \mathbf{j}) = \sum_{s=1}^{S} p^s((\boldsymbol{\mu}_{j^s}^s)^{\mathrm{T}}\mathbf{x} + \mu_{j^s,0}^s)$ denote the leader's expected utility, and $v^s(\mathbf{x}, j^s) = (\boldsymbol{\nu}_{j^s}^s)^{\mathrm{T}}\mathbf{x} + \nu_{j^s,0}^s$ denote the follower's expected utility for a type s. Then, $\langle \mathbf{x}^*, \mathbf{j}^* \rangle$ is an SSE if and only if,

$$\langle \mathbf{x}^*, \mathbf{j}^* \rangle = \arg\max_{\mathbf{x},\mathbf{j}} \{u(\mathbf{x}, \mathbf{j}) | v^s(\mathbf{x}, j^s) \geq v^s(\mathbf{x}, j'), \forall j' \neq j^s\}.$$

As an example, which we will return to throughout the paper, consider a Bayesian Stackelberg game with two follower types, where type 1 appears with probability .84 and type 2 appears with probability .16. The leader (defender) chooses a probability distribution of allocating one resource to protect the two targets whereas the follower (attacker) chooses the best target to attack. We show the payoff matrices in Figure 1, where the leader is the row player and the follower is the column player. The utilities of the two types are identical except that a follower of type 2 gets a utility of 1 for attacking *Target2* successfully, whereas one of type 1 gets 0. The leader's strategy is a column vector $(x_1, x_2)^{\mathrm{T}}$ representing the probabilities of protecting the two targets. Given one resource, the strategy space of the leader is $x_1 + x_2 \leq 1, x_1 \geq 0, x_2 \geq 0$, i.e., $A = (1, 1), \mathbf{b} = 1$. The payoffs in Figure 1 can be represented by the following utility matrices,

$$U^1 = \begin{pmatrix} 0 & 0 \\ 1 & -1 \\ 0 & 1 \end{pmatrix}, V^1 = \begin{pmatrix} 0 & 0 \\ -1 & 0 \\ 1 & -1 \end{pmatrix};$$

$$U^2 = \begin{pmatrix} 0 & 0 \\ 1 & -1 \\ 0 & 1 \end{pmatrix}, V^2 = \begin{pmatrix} 0 & 0 \\ -1 & 1 \\ 1 & -1 \end{pmatrix}.$$

In terms of previous work, Bayesian Stackelberg games have been typically solved via tree search, where we assign one follower type to a pure strategy at each tree level [8]. For example, Figure 2 shows the search tree of the example game in Figure 1. We solve four linear programs, one for each leaf node. At each leaf node, the linear program provides an optimal leader strategy such that the follower's best response for every follower type is the chosen target at that leaf node, e.g., at the leftmost leaf node, the linear program finds the optimal leader strategy such that both type 1 and type 2 have a best response of attacking *Target1*. Comparing across leaf nodes, we obtain the overall optimal leader strategy [5]. In this case, the leaf node where type 1 is assigned to *Target1* and type 2 to *Target2* provides the overall optimal strategy.

Instead of solving an LP for all J^S leaf nodes, recent work uses a branch-and-bound technique to speed up the tree search [8]. The key to efficiency in branch-and-bound is obtaining tight upper and lower bounds for internal nodes, i.e., for nodes shown by circles in Figure 2, where subsets of follower types are assigned to particular targets. For example, in Figure 2, suppose the left subtree has been explored; now if at the rightmost internal node (where type 1 is assigned to *Target2*) we realize that the upper bound on solution quality is 0.5, we could prune the right subtree without even considering type 2. One possible way of obtaining upper bounds is by relaxing the integrality constraints in DOBSS MILP [12]. Unfortunately, when the integer variables in DOBSS are relaxed, the objective can be arbitrarily large, leading to meaningless upper bounds. HBGS [8] computes upper bounds by heuristically utilizing the solutions of smaller restricted games. However, the preprocessing involved in solving many small games can be expensive and the bounds computed using heuristics can again be loose.

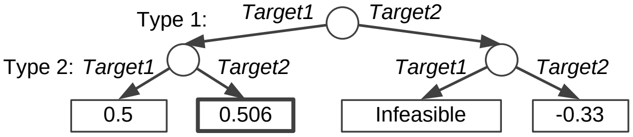


Figure 2: Example search tree of solving Bayesian games.

3. APPROACH

We present HUNTER (**H**andling **UN**cer**T**ainty **E**fficiently using **R**elaxation) based on the five key ideas mentioned in Section 1.

3.1 Algorithm Overview

To find the optimal leader's mixed strategy, HUNTER would conduct a best-first search in the search tree that results from assigning follower types to pure strategies, such as the search tree in Figure 2. Simply stated, HUNTER aims to search this space much more efficiently than HBGS [8]. As discussed earlier, efficiency gains are sought by obtaining tight upper bounds and lower bounds at internal nodes in the search tree (which corresponds to a partial assignment in which a subset of follower types are fixed). To that end, as illustrated in Figure 3, we use an upper bound LP within an internal search node. The LP returns an upper bound UB and a feasible solution $\mathbf{x}^*$, which is then evaluated by computing the follower best response, providing a lower bound LB. The solution returned by the upper bound LP is also utilized in choosing a new type s^* to create branches. To avoid having this upper bound LP itself become a bottleneck, it is solved efficiently using Bender's decomposition, which will be explained below.

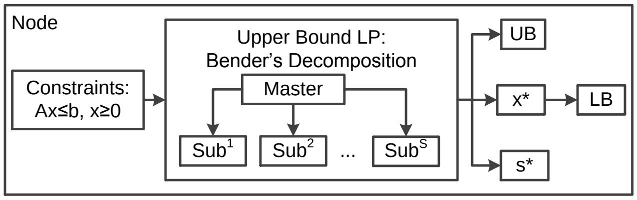


Figure 3: Steps of creating internal search nodes in HUNTER.

To understand HUNTER's behavior on a toy game instance, see Figure 4, which illustrates HUNTER's search tree in solving the example game from Figure 1 above. To start the best-first search, at the root node, no types are assigned any targets yet; we solve the upper bound LP with the initial strategy space $x_1 + x_2 \leq 1, x_1, x_2 \geq 0$ (Node 1). As a result, we obtain an upper bound of 0.560 and the optimal solution $x_1^* = 2/3, x_2^* = 1/3$. We evaluate the solution returned and obtain a lower bound of 0.506. Using HUNTER's heuristics, type 2 is then chosen to create branches by assigning it to *Target1* and *Target2* respectively. Next, we consider a child node (Node 2) in which type 2 is assigned to *Target1*, i.e., type 2's best response is to attack *Target1*. As a result, the follower's expected utility of choosing *Target1* must be higher than that of choosing *Target2*, i.e., $-x_1 + x_2 \geq x_1 - x_2$, simplified as $x_1 - x_2 \leq 0$. Thus, in Node 2, we impose an additional constraint $x_1 - x_2 \leq 0$ on the strategy space and obtain an upper bound of 0.5. Since its upper bound is lower than the current lower bound 0.506, this branch can be pruned out. Next we consider the other child node (Node 3) in which type 2 is assigned to *Target2*. This time we add constraint $-x_1 + x_2 \leq 0$ instead, and obtain an upper bound of 0.506. Since the upper bound coincides with the lower bound, we do not need to expand the node further. Moreover, since we have considered both *Target1* and *Target2* for type 2, we can terminate the algorithm and return 0.506 as the optimal solution value.

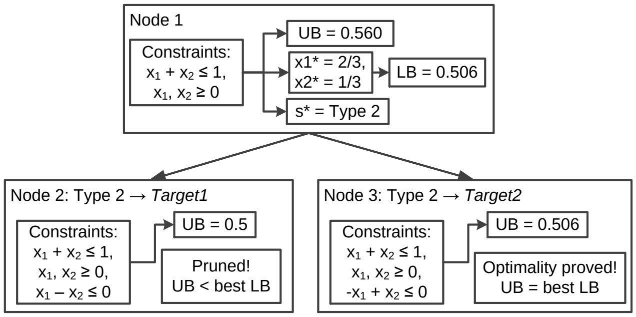


Figure 4: Example of internal nodes in HUNTER's search tree.

We now discuss HUNTER's behavior line-by-line (see Algorithm 1). We initialize the best-first search by creating the root node of the search tree with no assignment of types to targets and with the computation of the node's upper bound (Line 2 and 3). The initial lower bound is obtained by evaluating the solution returned by the upper bound LP (Line 4). We added the root node to a priority queue of open nodes which is internally sorted in a decreasing order of their upper bounds (Line 5). Each node contains information of the partial assignment, the feasible region of $\mathbf{x}$, the upper bound, and the Bender's cuts generated by the upper bound LP. At each iteration, we retrieve the node with the highest upper bound (Line 8), select a type s^* to assign pure strategies (Line 9), compute the upper bounds of the node's child nodes (Line 12 and 14), update the lower bound using the new solutions (Line 15), and enqueue child nodes with upper bound higher than the current lower bound (Line 16). As shown later, Bender's cuts at a parent node can be inherited by its children, speeding up the computation (Line 12).

Algorithm 1: HUNTER

```
1  Initialization;
2  [UB, $\mathbf{x}^*$, BendersCuts] = SolveUBLP($\phi$, $A\mathbf{x} \preceq \mathbf{b}$, $-\infty$);
3  root := ⟨ UB, $\mathbf{x}^*$, $A\mathbf{x} \preceq \mathbf{b}, x \succeq \mathbf{0}$, BendersCuts ⟩ ;
4  LB := Evaluate($\mathbf{x}^*$);
5  Enqueue(queue, root);
6  Best-first Search;
7  while not Empty(queue) do
8      node := pop(queue);
9      $s^*$ := PickType(node);
10     for $j := 1$ to $J$ do
11         NewConstraints := node.Constraints $\cup\{D_j^{s^*}\mathbf{x} + \mathbf{d}_j^{s^*} \succeq \mathbf{0}\}$ ;
12         [NewUB, $\mathbf{x}'$, NewBendersCuts] = SolveUBLP(node.BendersCuts,
           NewConstraints, LB) ;
13         if NewUB > LB then
14             child := ⟨ NewUB, $\mathbf{x}'$, NewConstraints, NewBendersCuts⟩ ;
15             LB := $\max${Evaluate($\mathbf{x}'$), LB} ;
16             Enqueue(queue, child);
17         end
18     end
19 end
```

In the rest of the section, we will 1) present the upper bound LP, 2) show how to solve it using Bender's decomposition, and 3) verify the correctness of passing down Bender's cuts from parent to child nodes, 4) introduce the heuristic branching rule.

3.2 Upper Bound Linear Program

We derive a tractable linear relaxation of Bayesian Stackelberg games to provide an upper bound efficiently at each of HUNTER's internal nodes. For expository purpose, we focus on the root node of the search tree. Applying the results in disjunctive program [2], we first derive the convex hull for a single type. Then we show intersecting the convex hulls of all its types provides a tractable, polynomial-size relaxation of a Bayesian Stackelberg game.

3.2.1 Convex hull of a Single Type

Consider a Stackelberg game with a single follower type (U, V), the leader's optimal strategy $\mathbf{x}^*$ is the best among the optimal solutions of J LPs where each restricts the follower's best response to one pure strategy [5]. Hence we can represent the optimization problem as the following disjunctive program (i.e., a disjunction of "Multiple LPs" [5]),

$$\begin{array}{ll} \max\limits_{\mathbf{x},u} & u \\ s.t. & A\mathbf{x} \preceq \mathbf{b}, \mathbf{x} \succeq \mathbf{0} \\ & \bigvee\limits_{j=1}^{J} \begin{pmatrix} u \le \boldsymbol{\mu}_j^{\mathrm{T}}\mathbf{x} + \mu_{j,0} \\ D_j\mathbf{x} + \mathbf{d}_j \preceq \mathbf{0} \end{pmatrix} \end{array} \tag{1}$$

where D_j and $\mathbf{d}_j$ are given by,

$$D_j = \begin{pmatrix} \boldsymbol{\nu}_1^{\mathrm{T}} - \boldsymbol{\nu}_j^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\nu}_J^{\mathrm{T}} - \boldsymbol{\nu}_j^{\mathrm{T}} \end{pmatrix}, \mathbf{d}_j = \begin{pmatrix} \nu_{1,0} - \nu_{j,0} \\ \vdots \\ \nu_{J,0} - \nu_{j,0} \end{pmatrix}.$$

The feasible set of (1), denoted by H, is a union of J convex sets, each corresponding to a disjunctive term. Applying the results in [2], the closure of the convex hull of H, clconv$H$, is[1],

$$\left\{ \begin{array}{l|l} & \mathbf{x} = \sum\limits_{j=1}^{J} \boldsymbol{\chi}_j, \boldsymbol{\chi}_j \succeq \mathbf{0}, \forall j \\ u \in \mathbb{R} & u = \sum\limits_{j=1}^{J} \psi_j, \psi_j \ge 0, \forall j \\ \mathbf{x} \in \mathbb{R}^n & \sum\limits_{j=1}^{J} \theta_j = 1, \theta_j \ge 0, \forall j \\ & \begin{pmatrix} A & -\mathbf{b} & \mathbf{0} \\ D_j & \mathbf{d}_j & \mathbf{0} \\ -\boldsymbol{\mu}_j^{\mathrm{T}} & -\mu_{j,0} & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{\chi}_j \\ \theta_j \\ \psi_j \end{pmatrix} \preceq \mathbf{0}, \forall j \end{array} \right\}.$$

The intuition here is that the continuous variables $\boldsymbol{\theta}$, $\sum_{j=1}^{J} \theta_j = 1$ are used to create all possible convex combination of points in H. Furthermore, when $\theta_j \ne 0$, $\langle \frac{\boldsymbol{\chi}_j}{\theta_j}, \frac{\psi_j}{\theta_j} \rangle$ represents a point in the convex set defined by the j-th disjunctive term in the original problem (1). Finally, since all the extreme points of clconv$H$ belong to H, the disjunctive program (1) is equivalent to the linear program:

$$\max_{\mathbf{x},u} \{u | (\mathbf{x}, u) \in \text{clconv}H\}.$$

3.2.2 Tractable Relaxation

Building on the convex hulls of individual types, we now derive the relaxation of a Bayesian Stackelberg game with S types. We write this game with S types as the following disjunctive program,

$$\begin{array}{ll} \max\limits_{\mathbf{x},u^1,\dots,u^S} & \sum_{s=1}^{S} p^s u^s \\ s.t. & A\mathbf{x} \preceq \mathbf{b}, \mathbf{x} \succeq \mathbf{0} \\ & \bigwedge\limits_{s=1}^{S} \left[\bigvee\limits_{j=1}^{J} \begin{pmatrix} u^s \le (\boldsymbol{\mu}_j^s)^{\mathrm{T}}\mathbf{x} + \mu_{j,0}^s \\ D_j^s\mathbf{x} + \mathbf{d}_j^s \preceq \mathbf{0} \end{pmatrix} \right] \end{array} \tag{2}$$

Returning to our toy example, the corresponding disjunctive program of the game in Figure 1 can be written as,

$$\begin{array}{ll} \max\limits_{x_1,x_2,u^1,u^2} & 0.84u^1 + 0.16u^2 \\ s.t. & x_1 + x_2 \le 1, x_1, x_2 \ge 0 \\ & \begin{pmatrix} u^1 \le x_1 \\ x_1 - 2x_2 \le 0 \end{pmatrix} \bigvee \begin{pmatrix} u^1 \le -x_1 + x_2 \\ -x_1 + 2x_2 \le 0 \end{pmatrix} \\ & \begin{pmatrix} u^2 \le x_1 \\ x_1 - x_2 \le 0 \end{pmatrix} \bigvee \begin{pmatrix} u^2 \le -x_1 + x_2 \\ -x_1 + x_2 \le 0 \end{pmatrix} \end{array} \tag{3}$$

Denote the set of feasible points $(\mathbf{x}, u^1, \dots, u^S)$ of (2) by H^*. Unfortunately, to use the results of [2] here and create clconvH^*, we need to expand (2) to a disjunctive normal form, resulting in a linear program with an exponential number ($O(NJ^S)$) of variables. Instead, we now give a much more tractable, polynomial-size relaxation of (2). Denote the feasible set of each type s, $(\mathbf{x}, u^s)$ by H^s, and define $\widehat{H^*} := \{(\mathbf{x}, u^1, \dots, u^S) | (\mathbf{x}, u^s) \in \text{clconv}H^s, \forall s\}$. Then the following program is a relaxation of (2):

$$\max_{\mathbf{x},u^1,\dots,u^s} \left\{ \sum_{s=1}^{S} p^s u^s | (\mathbf{x}, u^s) \in \text{clconv}H^s, \forall s \right\} \tag{4}$$

Indeed, for any feasible point $(\mathbf{x}, u^1, \dots, u^S)$ in H^*, $(\mathbf{x}, u^s)$ must belong to H^s, implying that $(\mathbf{x}, u^s) \in \text{clconv}H^s$. Hence $H^* \subseteq \widehat{H^*}$, implying that optimizing over $\widehat{H^*}$ provides an upper bound on H^*. On the other hand, $\widehat{H^*}$ will in general have points not belonging to H^* and thus the relaxation can lead to an overestimation.

For example, consider the disjunctive program in (3). $(x_1 = \frac{2}{3}, x_2 = \frac{1}{3}, u^1 = \frac{2}{3}, u^2 = 0)$ does not belong to H^* since $-x_1 + x_2 \le 0$ but $u^2 \not\le -x_1 + x_2 = -\frac{1}{3}$. However the point belongs to $\widehat{H^*}$ because: i) $(x_1 = \frac{2}{3}, x_2 = \frac{1}{3}, u^1 = \frac{2}{3})$ belongs to $H^1 \subseteq$ clconvH^1; ii) $(x_1 = \frac{2}{3}, x_2 = \frac{1}{3}, u^2 = 0)$ belongs to clconvH^2, as it is the convex combination of two points in H^2: $(x_1 = \frac{1}{2}, x_2 = \frac{1}{2}, u^2 = \frac{1}{2})$ and $(x_1 = 1, x_2 = 0, u^2 = -1)$,

$$(\frac{2}{3}, \frac{1}{3}, 0) = \frac{2}{3} \times (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}) + \frac{1}{3} \times (1, 0, -1).$$

The upper bound LP (4) has $O(NJS)$ number of variables and constraints, and can be written as the following two-stage problem by explicitly representing clconvH^s:

$$\begin{array}{ll} \max\limits_{\mathbf{x}} & \sum\limits_{s=1}^{S} p^s u^s(\mathbf{x}) \\ s.t. & A\mathbf{x} \preceq \mathbf{b}, \mathbf{x} \succeq \mathbf{0} \end{array} \tag{5}$$

where $u^s(\mathbf{x})$ is defined to be the optimal value of,

$$\begin{array}{ll} \max\limits_{\boldsymbol{\chi}_j^s,\psi_j^s,\theta_j^s} & \sum\limits_{j=1}^{J} \psi_j^s, \ \ \psi_j^s \ge 0, \forall j \\ s.t. & \sum\limits_{j=1}^{S} \boldsymbol{\chi}_j^s = \mathbf{x}, \ \ \boldsymbol{\chi}_j^s \succeq \mathbf{0}, \forall j \\ & \sum\limits_{j=1}^{S} \theta_j^s = 1, \ \ \theta_j^s \ge 0, \forall j \\ & \begin{pmatrix} A & -\mathbf{b} & \mathbf{0} \\ D_j^s & \mathbf{d}_j^s & \mathbf{0} \\ -(\boldsymbol{\mu}_j^s)^{\mathrm{T}} & -\mu_{j,0}^s & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{\chi}_j^s \\ \theta_j^s \\ \psi_j^s \end{pmatrix} \preceq \mathbf{0}, \forall j \end{array} \tag{6}$$

Although written in two stages, the above formulation is in fact a single linear program, as both stages are maximization problems and combining the two stages will not produce any non-linear

[1] To use the results in [2], we assume $u \ge 0$ for convenience. In the case where u can be negative, we can replace u by $u^+ - u^-$, with $u^+, u^- \ge 0$.

terms. We display formulations (5) and (6) in order to reveal the block structure for further speedup as explained below.

Note that so far, we have only derived the relaxation for the root node of HUNTER's search tree, without assigning any type to a pure strategy. This relaxation is also applied to other internal nodes in HUNTER's search tree. For example, if type s is assigned to pure strategy j, the leader's strategy space is further restricted by the addition of constraints of $D^s_j\mathbf{x} + \mathbf{d}^s_j \preceq \mathbf{0}$ to the original constraints $A\mathbf{x} \preceq \mathbf{b}, \mathbf{x} \succeq \mathbf{0}$. That is, we now have $A'\mathbf{x} \preceq \mathbf{b}', \mathbf{x} \succeq \mathbf{0}$, where $A' = \begin{pmatrix} D^s_j \\ A \end{pmatrix}$ and $\mathbf{b}' = \begin{pmatrix} -\mathbf{d}^s_j \\ \mathbf{b} \end{pmatrix}$.

3.3 Bender's Decomposition

Although much easier than solving a full Bayesian Stackelberg game, solving the upper bound LP can still be computationally challenging. Here we invoke the block structure of (4) we observed above, which partitioned it into (5) and (6), where, (5) is a master problem and (6) for $s = 1, \ldots, S$ are S subproblems. This block structure allows us to solve the upper bound LP efficiently using multi-cut Bender's Decomposition [4]. Generally speaking, the computational difficulty of optimization problems increases significantly with the number of variables and constraints. Instead of considering all variables and constraints of a large problem simultaneously, Bender's decomposition partitions the problem into multiple smaller problems, which can then be solved in sequence. For completeness, we now briefly describe the technique.

In Bender's decomposition, the second-stage maximization problem (6) is replaced by its dual minimization counterpart, with dual variables $\boldsymbol{\lambda}^s_j, \boldsymbol{\pi}^s, \eta^s$ for $s = 1, \ldots, S$:

$$\begin{aligned} u^s(\mathbf{x}) = & \min_{\boldsymbol{\lambda}^s_j \succeq \mathbf{0}, \boldsymbol{\pi}^s, \eta^s} (\boldsymbol{\pi}^s)^{\mathrm{T}}\mathbf{x} + \eta^s \\ s.t. & \begin{pmatrix} A^{\mathrm{T}} & (D^s_j)^{\mathrm{T}} & -\boldsymbol{\mu}^s_j \\ -\mathbf{b}^{\mathrm{T}} & (\mathbf{d}^s_j)^{\mathrm{T}} & -\mu^s_{i,0} \\ \mathbf{0}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix} \boldsymbol{\lambda}^s_j + \begin{pmatrix} \boldsymbol{\pi}^s \\ \eta^s \\ -1 \end{pmatrix} \succeq \mathbf{0}, \forall i \end{aligned} \tag{7}$$

Since the feasible region of (7) is independent of $\mathbf{x}$, its optimal solution is reached at one of a finite number of extreme points (of the dual variables). Since $u^s(\mathbf{x})$ is the minimum of $(\boldsymbol{\pi}^s)^{\mathrm{T}}\mathbf{x} + \eta^s$ over all possible dual points, we know the following inequality must be true in the master problem,

$$u^s \le (\boldsymbol{\pi}^s_k)^{\mathrm{T}}\mathbf{x} + \eta^s_k, \;\; k = 1, \ldots, K \tag{8}$$

where $(\boldsymbol{\pi}^s_k, \eta^s_k), k = 1, \ldots, K$ are all the dual extreme points. Constraints of type (8) for the master problem are called optimality cuts (infeasibility cuts, another type of constraint, turn out not to be relevant for our problem).

Since there are typically exponentially many extreme points for the dual formulation (7), generating all constraints of type (8) is not practical. Instead, Bender's decomposition starts by solving the master problem (5) with a subset of these constraints to find a candidate optimal solution $(\mathbf{x}^*, u^{1,*}, \ldots, u^{S,*})$. It then solves S dual subproblems (7) to calculate $u^s(\mathbf{x}^*)$. If all the subproblems have $u^s(\mathbf{x}^*) = u^{s,*}$, the algorithm stops. Otherwise for those $u^s(\mathbf{x}^*) < u^{s,*}$, the corresponding constraints of type (8) are added to the master program for the next iteration.

3.4 Reusing Bender's Cuts

We can further speed up the upper bound LP computation at internal nodes of HUNTER's search tree by not creating all of the Bender's cuts from scratch; instead, we can reuse Bender's cuts from the parent node in its children. Suppose $u^s \le (\boldsymbol{\pi}^s)^{\mathrm{T}}\mathbf{x} + \eta^s$ is a Bender's cut in the parent node. This means u^s cannot be greater than $(\boldsymbol{\pi}^s)^{\mathrm{T}}\mathbf{x} + \eta^s$ for any $\mathbf{x}$ in the feasible region of the parent node. Because a child node's feasible region is always more restricted than its parent's, we can conclude u^s cannot be greater than $(\boldsymbol{\pi}^s)^{\mathrm{T}}\mathbf{x} + \eta^s$ for any $\mathbf{x}$ in the child node's feasible region, i.e., $u^s \le (\boldsymbol{\pi}^s)^{\mathrm{T}}\mathbf{x} + \eta^s$ must also be a valid cut for the child node.

3.5 Heuristic Branching Rules

Given an internal node in the search tree of HUNTER, we must decide on the type to branch on next, i.e., the type for which J child nodes will be created at the next lower level of the tree. As we show in Section 5 below, the type selected to branch on has a significant effect on efficiency. Intuitively, we should select a type whereby the upper bound at these children nodes will decrease most significantly. To that end, HUNTER chooses the type whose $\boldsymbol{\theta}^s$ returned by (6) violates the integrality constraint the most. Recall that $\boldsymbol{\theta}^s$ is used to generate convex combinations. The motivation here is that if all $\boldsymbol{\theta}^s$ returned by (6) are integer vectors, the solution of the upper bound LP (5) and (6) is a feasible point of the original problem (2), implying the relaxation already returns the optimal solution. More specifically, as suggested in [7], HUNTER chooses type s^* whose corresponding $\boldsymbol{\theta}^{s^*}$ has the maximum entropy, i.e., $s^* = \arg\max_s - \sum_{j=1}^J \theta^s_j \log \theta^s_j$.

4. CONTINUOUS UNCERTAINTY IN STACKELBERG GAMES

This section extends HUNTER to handle continuous uncertainty via the *sample average approximation* technique [1]. We first introduce the uncertain Stackelberg game model with continuously distributed uncertainty in leader's execution, follower's observation, and both players' utilities. Then we show the uncertain Stackelberg game model can be written as a two-stage mixed-integer stochastic program, to which existing convergence results of the sample average approximation technique apply. Finally, we show the sampled problems are equivalent to Bayesian Stackelberg games, and consequently could also be solved by HUNTER.

4.1 Uncertain Stackelberg Game Model

We consider the following types of uncertainty in Stackelberg games with known distributions. First, similar to [10], we assume there is uncertainty in both the leader and the follower's utilities U and V. Second, similar to [16], we assume the leader's execution and the follower's observation are noisy. In particular, we assume the executed strategy and observed strategy are linear perturbations of the intended strategy, i.e., when the leader commits to $\mathbf{x}$, the actual executed strategy is $\mathbf{y} = F^{\mathrm{T}}\mathbf{x} + \mathbf{f}$ and the observed strategy by the follower is $\mathbf{z} = G^{\mathrm{T}}\mathbf{x} + \mathbf{g}$, where $(F, \mathbf{f})$ and $(G, \mathbf{g})$ are uncertain. Here $\mathbf{f}$ and $\mathbf{g}$ are used to represent the execution and observation noise that is independent on $\mathbf{x}$. In addition, F and G are $N \times N$ matrices allowing us to model execution and observation noise that is linearly dependent on $\mathbf{x}$. Note that G and $\mathbf{g}$ can be dependent on F and $\mathbf{f}$. For example, we can represent an execution noise that is independent of $\mathbf{x}$ and follows a Gaussian distribution with $\mathbf{0}$ mean using $F = I_N$ and $\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \Sigma)$, where I_N is the $N \times N$ identity matrix. We assume U, V, F, $\mathbf{f}$, G, and $\mathbf{g}$ are random variables, following some known continuous distributions. We use a vector $\xi = (U, V, F, \mathbf{f}, G, \mathbf{g})$ to represent a realization of the above inputs, and we use the notation $\xi(\omega)$ to represent the corresponding random variable.

We now show the uncertain Stackelberg game can be written as a two-stage mixed-integer stochastic program. Let $Q(\mathbf{x}, \xi)$ be the leader's utility for a strategy $\mathbf{x}$ and a realization ξ, assuming the follower chooses the best response. The first stage maximizes the expectation of leader's utility with respect to the joint probability distribution of $\xi(\omega)$, i.e., $\max_{\mathbf{x}} \{\mathbb{E}[Q(\mathbf{x}, \xi(\omega))] | A\mathbf{x} \preceq \mathbf{b}, \mathbf{x} \succeq \mathbf{0}\}$. The second stage computes $Q(\mathbf{x}, \xi)$[2]:

$$\begin{aligned} Q(\mathbf{x}, \xi) = & \quad \boldsymbol{\mu}_{i^*}^{\mathrm{T}}(F^{\mathrm{T}}\mathbf{x} + \mathbf{f}) + \mu_{i^*,0} \\ \text{where} \qquad i^* = & \quad \arg\max_{i=1}^{m} \boldsymbol{\nu}_i^{\mathrm{T}}(G^{\mathrm{T}}\mathbf{x} + \mathbf{g}) + \nu_{i,0}. \end{aligned} \tag{9}$$

4.2 Sample Average Approximation

Sample average approximation is a popular solution technique for stochastic programs with continuously distributed uncertainty [1]. It can be applied to solving uncertain Stackelberg games as follows. First, a sample $\xi^1, \ldots, \xi^S$ of S realizations of the random vector $\xi(\omega)$ is generated. The expected value function $\mathbb{E}[Q(\mathbf{x}, \xi(\omega))]$ can then be approximated by the sample average function $\frac{1}{S}\sum_{s=1}^{S} Q(\mathbf{x}, \xi^s)$. The sampled problem is given by,

$$\max_{\mathbf{x}} \left\{ \sum_{s=1}^{S} \frac{1}{S} Q(\mathbf{x}, \xi^s) | A\mathbf{x} \preceq \mathbf{b}, \mathbf{x} \succeq \mathbf{0} \right\}. \tag{10}$$

The sampled problem provides tighter and tighter statistical upper bound of the true problem with increasing number of samples [11]; the number of samples required to solve the true problem to a certain accuracy grows linearly in the dimension of $\mathbf{x}$ [1].

In the sampled problem, each sample ξ corresponds to a tuple $(U, V, F, \mathbf{f}, G, \mathbf{g})$. The following proposition shows ξ is equivalent to some $\hat{\xi}$ where $\hat{F} = \hat{G} = I_N$ and $\hat{\mathbf{f}} = \hat{\mathbf{g}} = \mathbf{0}$, implying the sampled execution and observation noise can be handled by simply perturbing the utility matrices.

PROPOSITION 1. *For any leader's strategy* $\mathbf{x}$ *and follower's strategy* j*, both players get the same expected utilities in two noise realizations* $(U, V, F, \mathbf{f}, G, \mathbf{g})$ *and* $(\hat{U}, \hat{V}, I_N, \mathbf{0}, I_N, \mathbf{0})$*, where,*

$$\hat{U} = \begin{pmatrix} 1 & \mathbf{f}^{\mathrm{T}} \\ \mathbf{0} & F \end{pmatrix} U, \hat{V} = \begin{pmatrix} 1 & \mathbf{g}^{\mathrm{T}} \\ \mathbf{0} & G \end{pmatrix} V.$$

PROOF. We calculate both players' expected utility vectors for both noise realizations to establish the equivalence:

$$\hat{U}^{\mathrm{T}} \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} = U^{\mathrm{T}} \begin{pmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \mathbf{f} & F^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} = U^{\mathrm{T}} \begin{pmatrix} 1 \\ F^{\mathrm{T}}\mathbf{x} + \mathbf{f} \end{pmatrix}.$$

$$\hat{V}^{\mathrm{T}} \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} = V^{\mathrm{T}} \begin{pmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \mathbf{g} & G^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} = V^{\mathrm{T}} \begin{pmatrix} 1 \\ G^{\mathrm{T}}\mathbf{x} + \mathbf{g} \end{pmatrix}. \quad \square$$

A direct implication of Proposition 1 is that the sampled problem (10) and (9) is equivalent to a Bayesian Stackelberg game of S equally weighted types, with utility matrices $(\widehat{U^s}, \widehat{V^s}), s = 1, \ldots, S$. Hence, via sample average approximation, HUNTER could be used to solve Stackelberg games with continuous payoff, execution, and observation uncertainty.

4.3 A Unified Approach

Applying sample average approximation in Bayesian Stackelberg games with discrete follower types, we are able to handle both discrete and continuous uncertainty simultaneously using HUNTER. The idea is to replace each discrete follower type by a set of samples of the continuous distribution, converting the original Bayesian Stackelberg game to a larger one. The resulting problem could again be solved by HUNTER, providing a solution robust to both types of uncertainty.

5. EXPERIMENTAL RESULTS

Since none of the existing algorithm can handle both discrete and continuous uncertainty in Stackelberg games, we conduct three sets of experiments considering i) only discrete uncertainty, ii) only continuous uncertainty, and iii) both types of uncertainty . The utility matrices were randomly generated from a uniform distribution between -100 and 100. Results were obtained on a standard 2.8GHz machine with 2GB main memory, and were averaged over 30 trials.

5.1 Handling Discrete Follower Types

For discrete uncertainty, we compare the runtime of HUNTER with DOBSS [12] and HBGS [8] (specifically, HBGS-F, the most efficient variant), the two best known algorithms for general Bayesian Stackelberg games. We compare these algorithms, varying the number of types and the number of pure strategies per player. The tests use a cutoff time of one hour for all three algorithms.

Figure 5(a) shows the performance of the three algorithms when the number of types increases. The games tested in this set have 5 pure strategies for each player. The x-axis shows the number of types, while the y-axis shows the runtime in seconds. As can be seen in Figure 5(a), HUNTER provides significant speed-up, of orders of magnitude over both HBGS and DOBSS[3](the line depicting HUNTER is almost touching the x-axis in Figure 5(a)). For example, we find that HUNTER can solve a Bayesian Stackelberg game with 50 types in 17.7 seconds on average, whereas neither HBGS nor DOBSS can solve an instance in an hour. Figure 5(b) shows the performance of the three algorithms when the number of pure strategies for each player increases. The games tested in this set have 10 types. The x-axis shows the number of pure strategies for each player, while the y-axis shows the runtime in seconds. HUNTER again provides significant speed-up over both HBGS and DOBSS. For example, HUNTER on average can solve a game with 13 pure strategies in 108.3 seconds, but HBGS and DOBSS take more than 30 minutes.

We now turn to analyzing the contributions of HUNTER's key components to its performance. First, we consider the runtime of HUNTER with two search heuristics, best-first (BFS) and depth-first (DFS), when the number of types is further increased. We set the pure strategies for each player to 5, and increase the number of types from 10 to 200. In Table 1, we summarize the average runtime and average number of nodes explored in the search process. As we can see, DFS is faster than BFS when the number of types is small, e.g., 10 types. However, BFS always explores significantly fewer number of nodes than DFS and is more efficient when the number types is large. For games with 200 types, the average runtime of BFS based HUNTER is 20 minutes, highlighting its scalability to a large number of types. Such scalability is achieved by efficient pruning – for a game with 200 types, HUNTER explores on average 5.3×10^3 nodes with BFS and 1.1×10^4 nodes with DFS, compared to a total of $5^{200} = 6.2 \times 10^{139}$ possible leaf nodes.

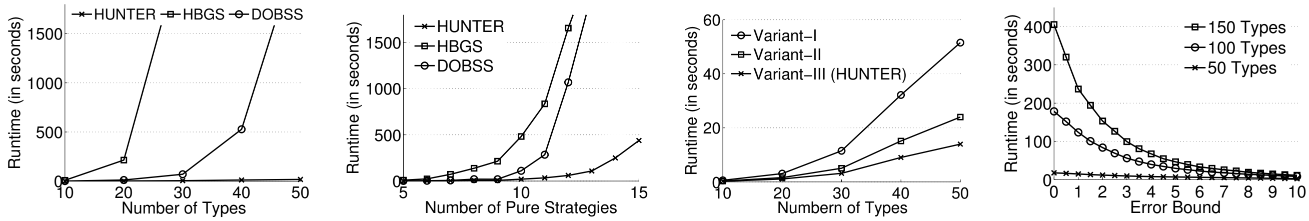
#Types	10	50	100	150	200
BFS Runtime (s)	5.7	17.7	178.4	405.1	1143.5
BFS #Nodes Explored	21	316	1596	2628	5328
DFS Runtime (s)	4.5	29.7	32.1	766.0	2323.5
DFS #Nodes Explored	33	617	3094	5468	11049

Table 1: Scalability of HUNTER to a large number of types

Second, we test the effectiveness of the two heuristics: inheritance of Bender's cuts from parent node to child nodes and the

[2] (9) can be formulated as a mixed-integer linear program as in [12]

[3] The runtime results of HBGS and DOBSS are inconsistent with the results in [8] because we use CPLEX 12 for solving mixed integer linear program instead of GLPK which is used in [8].



(a) Scaling up types. (b) Scaling up pure strategies. (c) Effectiveness of heuristics. (d) Finding approximate solutions.

Figure 5: Experimental analysis of HUNTER and runtime comparison against HBGS, and DOBSS.

branching rule utilizing the solution returned by the upper bound LP. We fix the number of pure strategies for each agent to 5 and increase the number of types from 10 to 50. In Figure 5(c), we show the runtime results of three variants of HUNTER: i) Variant-I does not inherit Bender's cuts and chooses a random type to create branches; ii) Variant-II does not inherit Bender's cuts and uses the heuristic branching rule; iii) Variant-III (HUNTER) inherits Bender's cuts and uses the heuristic branching rule. The x-axis represents the number of types while the y-axis represents the runtime in seconds. As we can see, each individual heuristic helps speed up the algorithm significantly, showing their usefulness. For example, it takes 14.0 seconds to solve an instance of 50 types when both heuristics are enabled (Variant-III) compared to 51.5 seconds when neither of them is enabled (Variant-I).

Finally, we consider the performance of HUNTER in finding quality bounded approximate solutions. To this end, HUNTER is allowed to terminate once the difference between the upper bound and the lower bound decreases to η, a given error bound. The solution returned is therefore an approximate solution provably within η of the optimal solution. In this set of experiment, we test 30 games with 5 pure strategies for each player and 50, 100, and 150 types with varying error bound η from 0 to 10. As shown in Figure 5(d), HUNTER can effectively trade off solution quality for further speedup, indicating the effectiveness of its upper bound and lower bound heuristics. For example, for games with 100 types, HUNTER returns within 30 seconds a suboptimal solution at most 5 away from the optimal solution (the average optimal solution quality is 60.2). Compared to finding the global optimal solution in 178 seconds, HUNTER is able to achieve six-fold speedup by allowing at most 5 quality loss.

5.2 Handling Continuous Uncertainty

For continuous uncertainty, ideally we want to compare HUNTER with other algorithms that handle continuous execution and observation uncertainty in general Stackelberg games; but no such algorithm exists. Hence we restrict our investigation to the more restricted security games [9], so that two previous robust algorithms BRASS [13] and RECON [16] can be used in such a comparison. To introduce the uncertainty in these security games, we assume the defender's execution and the attacker's observation uncertainty follows independent uniform distributions. That is, for an intended defender strategy $\mathbf{x} = \langle x_1, \ldots, x_N \rangle$, where x_i represents the probability of protecting target i, we assume the maximum execution error associated with target i is α_i, and the actual executed strategy is $\mathbf{y} = \langle y_1, \ldots, y_N \rangle$, where y_i follows a uniform distribution between $x_i - \alpha_i$ and $x_i + \alpha_i$ for each i. Similarly, we assume the maximum observation error for target i is β_i, and the actual observed strategy is $\mathbf{z} = \langle z_1, \ldots, z_N \rangle$, where z_i follows a uniform distribution between $y_i - \beta_i$ and $y_i + \beta_i$ for each i. The definition of maximum error $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ is consistent with the definition in [16].

We use HUNTER with 20 samples and 100 samples to solve the problem above via sample average approximation as described in Section 4. For each setting, we repeat HUNTER 20 times with different sets of samples and report the best solution found (as shown below, HUNTER's competitors also try 20 settings and choose the best). Having generated a solution with 20 or 100 samples, evaluating its actual quality is difficult in the continuous uncertainty model – certainly any analytical evaluation is extremely difficult. Therefore, to provide an accurate estimation of the actual quality, we draw $10,000$ samples from the uncertainty distribution and evaluate the solution using these samples.

For comparison, we consider two existing robust solution methods BRASS [13] and RECON [16]. As experimentally tested in [10], when its parameter ϵ is chosen carefully, BRASS strategy is one of the top performing strategy under continuous payoff uncertainty. RECON assumes a maximum execution error $\boldsymbol{\alpha}$ and a maximum observation error $\boldsymbol{\beta}$, computing the risk-averse strategy for the defender that maximizes the worst-case performance over all possible noise realization. To provide a more meaningful comparison, we find solutions of BRASS / RECON repeatedly with multiple settings of parameters and report the best one. For BRASS, we test 20 ϵ settings, and for RECON, we set $\boldsymbol{\alpha} = \boldsymbol{\beta}$ and test 20 settings.

In our experiments, we test on 30 random generated security games with five targets and one resource. The maximum execution and observation error is set to $\boldsymbol{\alpha} = \boldsymbol{\beta} = 0.1$. The utilities in the game are drawn from a uniform distribution between -100 and $+100$. Nonetheless, the possible optimal solution quality lies in a much narrower range. Over the 30 instances we tested, the optimal solution quality we found by any algorithm varies between -26 and +17. In Table 2, we show the solution quality of HUNTER compared to BRASS and RECON respectively. #Wins shows the number of instances out of 30 where HUNTER returns a better solution than BRASS / RECON. Avg. Diff. shows the average gain of HUNTER over BRASS (or RECON), and the average solution quality of the corresponding algorithm (shown in the parentheses). Max. Diff. shows the maximum gain of HUNTER over BRASS (or RECON), and the solution quality of the corresponding instance and algorithm (shown in the parentheses). As we can see, HUNTER with 20 and 100 samples outperforms both BRASS and RECON on average. For example, RECON on average returns a solution with quality of -5.1, while even with 20 samples, the average gain HUNTER achieves over RECON is 0.6. The result is statistically significant with a paired t-test value of 8.9×10^{-6} and 1.0×10^{-3} for BRASS and RECON respectively. Indeed, when the number of samples used in HUNTER increases to 100, HUNTER is able to outperform both BRASS and RECON in every instance tested. Not only is the average difference in this case statistically significant, but the actual solution quality found by HUNTER– as shown by max difference – can be significantly better in practice than solutions found by BRASS and RECON.

	HUNTER-20 vs.		HUNTER-100 vs.	
	BRASS	RECON	BRASS	RECON
#Wins	27	24	30	30
Avg. Diff.	0.7(-5.2)	0.6(-5.1)	0.9(-5.2)	0.8(-5.1)
Max. Diff.	2.4(7.6)	4.0(-16.1)	3.31(7.6)	4.4(-16.1)

Table 2: Quality gain of HUNTER against BRASS and RECON under continuous execution and observation uncertainty.

5.3 Handling Both Types of Uncertainty

In our last experiment, we consider Stackelberg games with both discrete and continuous uncertainty. Since no previous algorithm can handle both, we only show the runtime results of HUNTER. We test on security games with five targets and one resource, and with multiple discrete follower types whose utilities are randomly generated. For each type, we use the same utility distribution and the same execution and observation uncertainty as in Section 5.2. Table 3 summarizes the runtime results of HUNTER for 3, 4, 5, 6 follower types, and 10, 20 samples per type. As we can see, HUNTER can efficiently handle both uncertainty simultaneously. For example, HUNTER spends less than 4 minutes on average to solve a problem with 5 follower types and 20 samples per type.

#Discrete Types	3	4	5	6
10 Samples	4.9	12.8	29.3	54.8
20 Samples	32.4	74.6	232.8	556.5

Table 3: Runtime results (in seconds) of HUNTER for handling both discrete and continuous uncertainty.

6. CONCLUSIONS

With increasing numbers of real-world security applications of leader-follower Stackelberg games, it is critical that we address uncertainty in such games, including discrete attacker types and continuous uncertainty such as the follower's observation noise, the leader's execution error, and both players' payoffs uncertainty. Previously, researchers have designed specialized sets of algorithms to handle these different types of uncertainty, e.g. algorithms for discrete follower types [8, 12] have been distinct from algorithms that handle continuous uncertainty [10]. However, in the real-world, a leader may face all of this uncertainty simultaneously, and thus we desire a single unified algorithm that handles all this uncertainty.

To that end, this paper provides a novel unified algorithm, called HUNTER, that handles discrete and continuous uncertainty by scaling up Bayesian Stackelberg games. The paper's contributions are in two parts. First it proposes the HUNTER algorithm. The novelty of HUNTER is in combining AI search techniques (e.g. best first search and heuristics) with techniques from Operations Research (e.g. disjunctive programming and Bender's decomposition). None of these are out-of-the-box techniques, however, and most of these techniques had not been applied earlier in the context of Stackelberg games even in isolation. Our novel contributions are in algorithmically specifying how these can be applied (and applied in conjunction with one another) within the context of Stackelberg games. The result is that HUNTER provides speedups of orders of magnitude over existing algorithms.

Second, we show that via sample average approximation, HUNTER handles continuously distributed uncertainty. While no algorithm other than HUNTER exists to handle such continuous uncertainty in general Stackelberg games, we find, even in restricted settings of security games, HUNTER performs better than competitors focusing on robust solutions [16, 13]. Finally, the paper illustrates HUNTER's unique ability to handle both discrete and continuous uncertainty simultaneously within a single problem.

7. ACKNOWLEDGEMENT

This research was supported by the United States Department of Homeland Security through the Center for Risk and Economic Analysis of Terrorism Events (CREATE). We thank Matthew P. Johnson for detailed comments.

8. REFERENCES

[1] S. Ahmed, A. Shapiro, and E. Shapiro. The sample average approximation method for stochastic programs with integer recourse. *SIAM Journal of Optimization*, 12:479–502, 2002.

[2] E. Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1-3):3 – 44, 1998.

[3] N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS*, 2009.

[4] J. R. Birge and F. V. Louveaux. A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operational Research*, 34(3):384 – 392, 1988.

[5] V. Conitzer and T. Sandholm. Computing the optimal strategy to commit to. In *ACM EC-06*, pages 82–90, 2006.

[6] J. P. Dickerson, G. I. Simari, V. S. Subrahmanian, and S. Kraus. A graph-theoretic approach to protect static and moving targets from adversaries. In *AAMAS*, 2010.

[7] A. Gilpin and T. Sandholm. Information-theoretic approaches to branching in search. *Discrete Optimization*, 8(2):147 – 159, 2011.

[8] M. Jain, C. Kiekintveld, and M. Tambe. Quality-bounded solutions for finite bayesian stackelberg games: Scaling up. In *AAMAS*, 2011.

[9] M. Jain, J. Tsai, J. Pita, C. Kiekintveld, S. Rathi, M. Tambe, and F. Ordóñez. Software Assistants for Randomized Patrol Planning for the LAX Airport Police and the Federal Air Marshals Service. *Interfaces*, 40:267–290, 2010.

[10] C. Kiekintveld, J. Marecki, and M. Tambe. Approximation methods for infinite Bayesian Stackelberg games: Modeling distributional payoff uncertainty. In *AAMAS*, 2011.

[11] W.-K. Mak, D. P. Morton, and R. K. Wood. Monte carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, 24(1-2):47 – 56, 1999.

[12] P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordóñez, and S. Kraus. Playing games with security: An efficient exact algorithm for Bayesian Stackelberg games. In *AAMAS*, 2008.

[13] J. Pita, M. Jain, F. Ordóñez, M. Tambe, S. Kraus, and R. Magori-cohen. Effective solutions for real-world Stackelberg games: When agents must deal with human uncertainties. In *AAMAS*, 2009.

[14] J. Pita, M. Tambe, C. Kiekintveld, S. Cullen, and E. Steigerwald. Guards - game theoretic security allocation on a national scale. In *AAMAS (Industry Track)*, 2011.

[15] B. von Stengel and S. Zamir. Leadership with commitment to mixed strategies. Technical Report LSE-CDAM-2004-01, CDAM Research Report, 2004.

[16] Z. Yin, M. Jain, M. Tambe, and F. Ordonez. Risk-averse strategies for security games with execution and observational uncertainty. In *AAAI*, 2011.