Quality-bounded Solutions for Finite Bayesian Stackelberg Games: Scaling up

Manish Jain[+], Christopher Kiekintveld[*], Milind Tambe[+]
[+] Computer Science Department, University of Southern California, Los Angeles, CA. 90089
{manish.jain,tambe}@usc.edu
[*] Department of Computer Science, University of Texas at El Paso, El Paso, Texas. 79968
cdkiekintveld@utep.edu

ABSTRACT

The fastest known algorithm for solving General Bayesian Stackelberg games with a finite set of follower (adversary) types have seen direct practical use at the LAX airport for over 3 years; and currently, an (albeit non-Bayesian) algorithm for solving these games is also being used for scheduling air marshals on limited sectors of international flights by the US Federal Air Marshals Service. These algorithms find optimal randomized security schedules to allocate limited security resources to protect targets. As we scale up to larger domains, including the full set of flights covered by the Federal Air Marshals, it is critical to develop newer algorithms that scale-up significantly beyond the limits of the current state-of-the-art of Bayesian Stackelberg solvers. In this paper, we present a novel technique based on a hierarchical decomposition and branch and bound search over the follower type space, which may be applied to different Stackelberg game solvers. We have applied this technique to different solvers, resulting in: (i) A new exact algorithm called HBGS that is orders of magnitude faster than the best known previous Bayesian solver for general Stackelberg games; (ii) A new exact algorithm called HBSA which extends the fastest known previous security game solver towards the Bayesian case; and (iii) Approximation versions of HBGS and HBSA that show significant improvements over these newer algorithms with only 1-2% sacrifice in the practical solution quality.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance, Experimentation

Keywords

Game Theory, Bayesian Stackelberg Games, Hierarchical Decomposition

1. INTRODUCTION

This paper focuses on Stackelberg games where a leader commits to a mixed strategy, and then a follower selfishly optimizes his own reward, with the knowledge of the mixed strategy chosen by the leader. These models are common for modeling *attacker-defender* scenarios in security domains [15, 9], patrolling domains [1, 3], and are also being applied to network routing [11] and transportation networks [16]. Indeed, these models have seen at least two deployed applications at the Los Angeles International Airport (LAX) and the Federal Air Marshals Service (FAMS) [8].

Uncertainty over player preferences, a key aspect of the real-world, is modeled using a Bayesian extension to Stackelberg games. Bayesian Stackelberg games allow us to explicitly model players as *types*, where each type can have its own preferences. Indeed, the application at LAX uses a Bayesian Stackelberg game. Unfortunately, the problem of finding the Stackelberg equilibrium for Bayesian Stackelberg games has been shown to be NP-Hard [5].

The two chief techniques previously employed for identifying Bayesian Stackelberg equilibrium are: (1) Multiple-LPs [5] that uses the Harsanyi transformation [6] to convert the Bayesian game into a perfect information game, and then analyzes each of the exponential number of combinations of the actions for all follower types independently; (2) DOBSS [15] that analyzes the entire Bayesian game at once without using the Harsanyi transformation by using a mixed-integer linear program, which optimizes against each adversary type independently while keeping the leader strategy fixed across all types. However, these methods fail to scale up beyond 10 types even for 20 actions for the players, or beyond 30 actions for just 5 follower types. Alternatively, sampling techniques have been proposed for Bayesian Stackelberg games with infinite types, but they only provide approximate solutions [10]. Thus, efficient algorithms for Bayesian Stackelberg games need to be developed for the application of game-theoretic techniques to more complex real-world domains.

The focus of this paper is to present a new technique for solving large Bayesian Stackelberg games that decomposes the entire game into many hierarchically-organized, *restricted* Bayesian Stackelberg games; it then utilizes the solutions of these restricted games to guide us to more efficiently solve the larger Bayesian Stackelberg game. In particular, we use this overarching idea of hierarchical structure to improve the performance of branch and bound search for Bayesian Stackelberg games; the solutions obtained for the restricted games at the 'child' nodes are used to provide: (i) pruning rules, (ii) tighter bounds, and (iii) efficient branching heuristics to solve the bigger game at the 'parent' node faster. Such hierarchical techniques have seen little application towards obtaining optimal solutions in Bayesian games (decompositions have been proposed to obtain approximate Nash equilibrium for symmetric games [17]), while Stackelberg settings have not seen any application of such hierarchical decomposition.

We first present HBGS (Hierarchical Bayesian solver for General Stackelberg games), an algorithm that applies such decompo-

Cite as: Quality-bounded Solutions for Finite Bayesian Stackelberg Games: Scaling up, Manish Jain, Christopher Kiekintveld and Milind Tambe, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. 997-1004.

sition techniques to general Bayesian Stackelberg games, and show that we can scale up to 50 types for games where the state-of-the-art algorithms cannot even solve for 10. Secondly, we present HBSA (Hierarchical Bayesian Solver for Security games with Arbitrary schedules), which uses the same key decomposition ideas to solve large scale security domains with arbitrary scheduling constraints. Finally, we show that these algorithms are naturally designed for obtaining quality bounded approximations, and can provide a further order of magnitude scale-up *without significant loss in quality*.

2. BACKGROUND AND NOTATION

We begin by defining a normal-form Stackelberg game. A generic Stackelberg game is a two person bi-matrix game, between a leader and a follower. These players need not represent individuals, but could also be groups like the police force, that cooperate to execute a joint-strategy. Each player has a set of *pure strategies*, and a *mixed strategy* allows the player to play a probability distribution over these pure strategies. Payoffs for each player are defined over all possible joint pure-strategy outcomes. In a Stackelberg game, the follower acts with the full knowledge of the leader's strategy.

Table 1: Bayesian Game Notation

Variable	Definition
Θ	Leader
Ψ	Follower
Λ	Set of follower types, iterated using λ
$G(\Theta, \Psi^{\Lambda})$	Bayesian Game with Λ follower types.
Σ	Set of pure strategies, iterated using σ
σ_Θ	A pure strategy of the leader
σ_Ψ	A pure strategy of the follower, $\sigma_\Psi =< \sigma_\Psi^\lambda >$
p^λ	Probability of facing follower type λ
$\mathcal{U}_\Theta^\lambda(\sigma_\Theta, \sigma_\Psi^\lambda)$	Payoff of leader against follower type λ
$\mathcal{U}_\Psi^\lambda(\sigma_\Theta, \sigma_\Psi^\lambda)$	Payoff of follower type λ
δ	Mixed strategy of the leader
$\delta(\sigma_\Theta)$	Probability of leader playing pure strategy σ_Θ
$\mathcal{V}_\Theta(\delta, \sigma_\Psi)$	Expected utility of the leader
$\mathcal{V}_\Psi(\delta, \sigma_\Psi)$	Expected utility of the follower

The Bayesian extension to the Stackelberg game allows for multiple types of players, with each type associated with its own payoff values. For the games discussed in this paper, we assume that there is only one leader type, although there may be multiple follower types. This is motivated by the real-world deployments: there could be one security force which is facing many types of adversaries like local thieves as well as hard-lined terrorists. Each type is represented by a different and possibly uncorrelated payoff matrix. The leader does not know the follower's exact type, however, the probability distribution over follower types is known.

A Bayesian game between the leader and a set of follower types is represented by $G(\Theta, \Psi^{\Lambda})$ where Θ represents the leader, Λ represents the set of follower types and Ψ represents the follower. The leader, Θ, for the Bayesian Stackelberg games in this paper is always the row player, while the follower Ψ is always the column player. The follower could be of any type λ_i from the set of types Λ. The pure strategies for each player are represented by σ, whereas the set of these pure strategies is represented by Σ. Subscripts Θ and Ψ are used to denote the player, e.g., σ_Θ represent the pure strategies for the leader. The strategy space Σ_Ψ of the follower in the Bayesian game is a cross product of the strategy spaces of all the follower types, $\Sigma_\Psi = \prod_{\lambda \in \Lambda} \Sigma_\Psi^\lambda$, and so the pure strategy σ_Ψ of the follower is represented as a tuple of pure strategies for each follower type, $\sigma_\Psi =< \sigma_\Psi^\lambda >= [\sigma_\Psi^1, \ldots, \sigma_\Psi^{|\Lambda|}]$. The notation is described in Table 1.

The solution concept of interest is a *Strong Stackelberg Equilibrium* (SSE) [13], where the objective for the leader is to find the mixed strategy δ, such that the expected leader utility is maximized given that the follower will choose its action with the complete knowledge of the leader's mixed strategy in its own interest. We limit the follower to play only pure strategies, since their always exists a pure strategy best response for the follower in such Stackelberg games [15]. The expected utility of the leader against follower type λ for strategy profiles δ and σ_Ψ is denoted as $\mathcal{V}_\Theta^\lambda(\delta, \sigma_\Psi^\lambda)$. The expected utility of the leader, $\mathcal{V}_\Theta(\delta, \sigma_\Psi)$, is a weighted combination of the leader expected utility against all follower types:

$$\mathcal{V}_\Theta^\lambda(\delta, \sigma_\Psi) = \sum_{\sigma_\Theta \in \Sigma_\Theta} \delta(\sigma_\Theta) \mathcal{U}_\Theta^\lambda(\sigma_\Theta, \sigma_\Psi^\lambda) \quad (1)$$

$$\mathcal{V}_\Theta(\delta, \sigma_\Psi) = \sum_{\lambda \in \Lambda} p_\lambda \mathcal{V}_\Theta^\lambda(\delta, \sigma_\Psi^\lambda) \quad (2)$$

The expected utility of the follower is defined analogously. Formally, SSE is defined as follows:

1. The leader plays a best response:
$$\mathcal{V}_\Theta(\delta, \sigma_\Psi) \geq \mathcal{V}_\Theta(\delta', \sigma_\Psi) \forall \delta' \quad (3)$$
2. Every follower type plays a best response:
$$\mathcal{V}_\Psi^\lambda(\delta, \sigma_\Psi^\lambda) \geq \mathcal{V}_\Psi^\lambda(\delta, \sigma_\Psi^{'\lambda}) \forall \sigma_\Psi^{'\lambda} \in \Sigma_\Psi^\lambda, \forall \lambda \in \Lambda \quad (4)$$
3. The follower breaks ties in favor of the leader[1]:
$$\mathcal{V}_\Theta^\lambda(\delta, \sigma_\Psi^\lambda) \geq \mathcal{V}_\Theta^\lambda(\delta, \sigma_\Psi^{'\lambda}) \forall \sigma_\Psi^{'\lambda} \in \Sigma_\Psi^{*\lambda}, \forall \lambda \in \Lambda \quad (5)$$
where $\Sigma_\Psi^{*\lambda}$ is the set of pure strategy best responses, satisfying Equation (4).

2.1 Existing Approaches / Related Work

Two main approaches have been proposed in prior work to compute the equilibrium in Bayesian Stackelberg games. DOBSS [15] solves the Bayesian game by solving a mixed-integer linear program that internally decomposes the problem by individual follower types. On the other hand, Multiple-LPs approach [5] works on the Harsanyi transformed version of the game. Harsanyi transformation converts the Bayesian game into a normal form representation, however, with an exponential number of pure strategies. Multiple-LPs thus computes an exponential number of linear programs to find the Stackelberg equilibrium [15].

The follower's pure strategy space Σ_Ψ in the Bayesian Stackelberg game $G(\Theta, \Psi^{\Lambda})$ can be represented using a tree, where each branch corresponds to a pure strategy choice for a follower type. Figure 1 shows an example of such a tree presentation of $G(\Theta, \Psi^{\Lambda})$, where $\Lambda = \{\lambda_1, \lambda_2\}$ with $|\Sigma_\Psi^\lambda| = 2, \lambda \in \Lambda$. Every leaf in this tree represents a pure strategy of the follower; for example, the pure strategy $[\sigma_2^1, \sigma_1^2]$ is represented by the leaf $[2, 1]$. In a game with $|\Lambda|$ types and $|\Sigma_\Psi^\lambda|$ pure strategies per type, the number of leaves in this tree would be $\prod_{\lambda \in \Lambda} |\Sigma_\Psi^\lambda|$. The path from the root to a leaf represents a distinct pure strategy σ_Ψ of the follower. Thus, there are exponentially many *leaves* in $G(\Theta, \Psi^{\Lambda})$; for example, a game with 10 follower types and just 5 actions per type would have $9,765,625$ leaves.

The LP employed by Multiple-LPs algorithm is described in Equations (6) to (9). This LP is executed for all pure strategies

[1]The leader can always induce the follower to break ties in its favor [2].

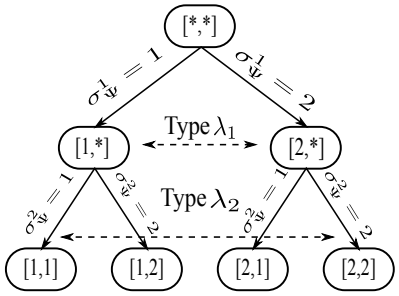


Figure 1: Example tree representing the pure strategy action choices for the follower in a Bayesian Stackelberg game.

σ_Ψ of the follower (i.e. for all the *leaves* of Figure 1). It takes σ_Ψ as input, and then maximizes the leader expected utility $\mathcal{V}_\Theta$ under the constraint that the best response of the follower of type λ will be σ_Ψ^λ. The follower strategy σ_Ψ is labeled *infeasible* if it can never be the best response of the follower for any defender strategy δ. The optimal leader strategy is one that gives the leader the maximum expected utility across all these linear programs.

$$\max_{\delta} \quad \mathcal{V}_\Theta(\delta, \sigma_\Psi) \tag{6}$$

$$\text{s.t.} \quad \mathcal{V}_\Psi^\lambda(\delta, \sigma_\Psi^\lambda) \geq \mathcal{V}_\Psi^\lambda(\delta, \sigma_\Psi^{'\lambda}) \quad \forall \sigma_\Psi^{'\lambda} \in \Sigma_\Psi^\lambda, \lambda \in \Lambda \tag{7}$$

$$\sum_{\sigma \in \Sigma_\Theta} \delta(\sigma_\Theta) = 1 \tag{8}$$

$$\delta \in [0, 1] \tag{9}$$

3. HBGS OVERVIEW

The exponential number of linear programs that are solved by Multiple-LPs approach does not allow it to scale well with increasing number of follower types. Indeed, if the optimal solution could be obtained by solving only a few of these linear programs, the performance could be improved significantly — even significantly better than DOBSS. Specifically, if we could construct a smaller tree of the follower's action choices in the first place, or obtain bounds on solution quality to perform branch and bound search, significant speed-ups would be obtained. This is the intuition behind HBGS: HBGS reduces the number of linear programs that need to be solved using two main insights: (1) *Feasibility* rules that help eliminate *infeasible* follower strategies in the Bayesian game; and (2) *Bounds* that help prune the follower action space using branch and bound search. HBGS constructs a hierarchical tree of restricted games, the solutions of which provide such feasibility and bounds information. We first discuss the hierarchical structure of HBGS, and then describe the feasibility and bounding techniques.

3.1 Hierarchical Type Trees

As mentioned above, HBGS constructs a hierarchical structure of restricted games to obtain the feasibility sets Σ_Ψ^λ per follower type, and corresponding upper bounds $\mathcal{B}^\lambda$ for every pure strategy for every follower type. For this purpose, the Bayesian Stackelberg game $G(\Theta, \Psi^\Lambda)$ is decomposed into many smaller restricted games, $G(\Theta, \Psi^{\Lambda_i})$ by partitioning the set of types, Λ, into subsets Λ_i.[2] Any partition of Λ into subsets Λ_i is applicable, such that:

$$\cup_i \Lambda_i = \Lambda \tag{10}$$

$$\Lambda_i \cap \Lambda_j = \emptyset \qquad \forall i, \forall j, j \neq i \tag{11}$$

[2]The probability distribution over types, $p^\Lambda =< p^\lambda >$, is renormalized for each restricted sub-game.

These restricted games are smaller and are much easier to solve (the number of follower pure strategies in these restricted games is exponentially smaller as compared to the entire Bayesian game).

Once a partition has been established, a hierarchical type tree is constructed where the root node corresponds to the entire Bayesian game $G(\Theta, \Psi^\Lambda)$, and its children correspond to the restricted games, $G(\Theta, \Psi^{\Lambda_i})$. While any partitioning is valid, we present and experimentally evaluate two partitions in this paper: (1) a *depth-one* partition, and (2) a *fully branched binary tree* (where children can then be hierarchically decomposed into even more restricted games). An example game of depth-one partitioning with 4 types is shown in Figure 2(a). Here, each restricted game solves for exactly one type such that the total depth of the tree is one. On the other hand, Figure 2(b) shows fully branched binary partitioning, where the entire problem is broken down into two restricted games of two types each, which are again broken down into two sub-games themselves.

All the nodes in the constructed hierarchical tree are visited such that the children are evaluated before the parent. Every node is evaluated using Algorithm 1 (discussed next), and the feasible pure strategies $\Sigma_\Psi^{\Lambda_i}$ with corresponding bounds $\mathcal{B}^{\Lambda_i}$ obtained at the i^{th} child are propagated up to the parent. These are then used when the parent is evaluated, again using Algorithm 1. This process continues until the root node is solved and the optimal solution for the entire game $G(\Theta, \Psi^\Lambda)$ is obtained.

3.2 Pruning a Bayesian Game

If a parent in the HBGS tree obtains feasibility and bounds information from its children, how can it use it to improve its efficiency of processing the Bayesian game?

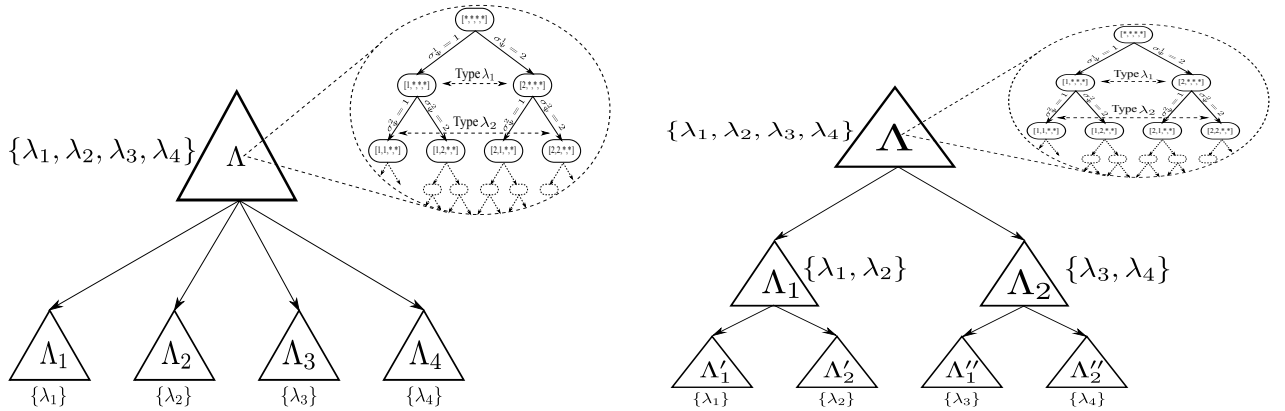
(1) *Feasibility:* HBGS uses the following theorem to reduce the strategy space Σ_Ψ^Λ of the follower.

THEOREM 1. *The follower's pure strategy $\sigma_\Psi = [\sigma_\Psi^\lambda]$ is infeasible in the Bayesian game $G(\Theta, \Psi^\Lambda)$ if the strategy σ_Ψ^λ is infeasible for the follower of type λ in a restricted game, $G(\Theta, \Psi^{\Lambda'})$, where the follower can only be of type λ (that is, $\Lambda' = \{\lambda\}$).*

PROOF. Suppose that the pure strategy σ_Ψ containing σ_Ψ^λ is feasible in the Bayesian game with δ being the corresponding defender mixed strategy. Thus, the best response of the follower of type λ to the leader strategy δ is σ_Ψ^λ, as stated in Equation (4). Therefore, the pure strategy σ_Ψ^λ is feasible in the restricted game $G'(\Theta, \Psi^{\Lambda'})$, which is a contradiction. □

Theorem 1 states that if σ_Ψ^λ can never be the best response of follower type λ in the restricted game $G(\Theta, \Psi^{\Lambda'}), \Lambda' = \{\lambda\}$ (that is, a game with only the follower of type λ), then a pure strategy containing σ_Ψ^λ can never be the best-response of the follower in *any* Bayesian game $G(\Theta, \Psi^\Lambda), \Lambda = \{\lambda_1, \lambda_2, \ldots\}$. In other words, if some *branches* in the follower action tree (Figure 1) are infeasible, no *leaves* in the subtree connected by that branch need to be evaluated. The theorem can easily be extended to restricted games with $\Lambda' \subseteq \Lambda$ by considering Λ' as one *hyper-type*. This implies that a pure strategy σ_Ψ can be removed from the Bayesian game if any of its components σ_Ψ^λ is infeasible in the corresponding restricted game. Thus, such pure strategies need not be reasoned over, thereby reducing the computational burden significantly.

As an example of the gain in performance, consider a sample problem with five follower types ($|\Lambda| = 5$), such that there are ten pure strategies for follower of each type ($|\Sigma_\Psi^\lambda| = 10, \lambda \in \Lambda$). Thus, the total number of pure strategies for the follower in the Bayesian Stackelberg game are 10^5. If an oracle could inform us *a-priori* that two particular pure strategies can be discarded for every type of



(a) Depth-One Partitioning. (b) Full Binary Partitioning.

Figure 2: Examples of possible hierarchical type trees generated in HBGS. Each node is a restricted Bayesian game in itself.

the follower, the strategy space would reduce to 8^5 pure strategies, which is approximately only 33% of the initial problem.

HBGS identifies the infeasible strategies of the restricted games, and then applies Theorem 1 to prune out infeasible strategies from $G(\Theta, \Psi^\Lambda)$. This process is applied recursively in the hierarchical tree (refer Figure 2) to obtain effective pruning at the root node.

(2) *Bounds:* A pure strategy for the follower needs not be evaluated if the upper bound on the maximum leader expected utility for the corresponding pure strategy is available, and if this upper bound is not better than the best solution known so far. A naïve upper bound is $+\inf$ which leads to no pruning, and would lead to the conventional Multiple-LPs approach. However, HBGS uses novel techniques for obtaining tighter upper bounds on the maximum leader expected utility, which are based on Theorem 2.

THEOREM 2. *The maximal leader payoff is upper bounded by* $\sum_{\lambda\in\Lambda} p^\lambda \mathcal{B}(\sigma^\lambda_\Psi)$ *when the follower chooses a pure strategy* $\sigma_\Psi =< \sigma^\lambda_\Psi >$*, where* $\mathcal{B}(\sigma^\lambda_\Psi)$ *is the upper bound on the leader utility in the restricted game* $G'(\Theta, \Psi^{\Lambda'})|\Lambda' = \{\lambda\}$ *when the follower of type* λ *is induced to choose pure strategy* σ^λ_Ψ*.*

PROOF. $\mathcal{B}(\sigma^\lambda_\Psi)$ upper-bounds the maximum utility of the leader for any strategy that induces the follower of type λ to choose σ^λ_Ψ as the best response. Thus, the leader utility against follower of type λ for any strategy δ is no more than $\mathcal{B}(\sigma^\lambda_\Psi)$. Therefore, $\mathcal{V}^\lambda_\Theta(\delta, \sigma^\lambda_\Psi) \leq \mathcal{B}(\sigma^\lambda_\Psi)$. Applying Equation (2),

$$\mathcal{V}_\Theta(\delta, \sigma_\Psi) \leq \sum_{\lambda\in\Lambda} p^\lambda \mathcal{B}(\sigma^\lambda_\Psi) \quad \forall\delta \tag{12}$$

which proves the theorem.[3] □

These bounds are generated for all children and then propagated up the hierarchical tree (Figure 2), where they are used by the parent to prune out branches from its own Bayesian game (Figure 1).

3.3 HBGS Description

HBGS solves each node of the hierarchical tree using Algorithm 1. A tree representing the follower actions, as in Figure 1, is constructed which is then solved using an efficient branch-and-bound search. Only the pure strategies in the cross-product of the feasible set of strategies of individual types need to be evaluated for the follower (Theorem 1). Σ^* represents this maximal set, as given in Line number 2 (and updated later in Line 10). $\mathcal{B}^*$ represents the bounds for all these strategies, and is obtained in Line 3 (and updated later in Line 9). Lines 2 to 5 are initialization; $\Sigma^\lambda_\Psi(i)$ represents the i^{th} pure strategy in the set Σ^λ_Ψ. The main loop of the algorithm starts after Line 6, where one pure strategy (*leaf*) is evaluated after another. The function `solve` (Line 7) in HBGS

Algorithm 1 HBGS$(\Lambda, \Sigma_\Theta, \Sigma^\Lambda_\Psi, \mathcal{B}^\Lambda, U_\Theta, U_\Psi)$

```
// initialize
// Σ^Λ_Ψ: pruned feasible pure strategy set for all follower types
// B^Λ: bounds for all pure strategies for all follower types
1. FT := construct-Follower-Action-Tree(Σ^Λ_Ψ)
2. Σ* := leaves-of(FT)   //feasible pure strategies of Ψ
3. B*(σ_Ψ) := getBounds(σ_Ψ, B^Λ)  ∀σ_Ψ ∈ ∏_λ Σ^λ_Ψ
4. sort(Σ*, B*(σ_Ψ)) // sort σ_Ψ in descending order of B*(σ_Ψ)
5. σ_Ψ := [Σ^1_Ψ(1), Σ^2_Ψ(1), ..., Σ^|Λ|_Ψ(1)]  // left-most leaf
6. r* := −inf      //r*: current best known solution
// start
repeat
   7. (feasible, δ, r) := solve(Σ_Θ, σ_Ψ) // Equations 6-9
   if feasible then
      if r > r* then
         // update current best solution
         8a. r* := r
         8b. δ* := δ
      9. B*(σ_Ψ) := r      //update bound
   else
      10. Σ* := Σ* − σ_Ψ    //remove infeasible strategy
   11. σ_Ψ := getNextStrategy(σ_Ψ, r*, Σ^Λ_Ψ, B^Λ)
until σ_Ψ <> NULL
return (δ*, r*, Σ*, B*)
```

solves the LP given in Equations (6) to (9). The follower pure strategy σ_Ψ is feasible if this LP has a feasible solution. The maximal leader reward r and the corresponding leader mixed strategy δ are also obtained from the LP (Line 7). If the pure strategy is feasible, the bounds $\mathcal{B}^*$ are updated (Line 9). Otherwise, the strategy σ_Ψ is removed from the pure strategy set Σ^* of the follower (Line 10).

The function `getNextStrategy()` moves from one *leaf* (pure strategy) to another of this follower action tree: it is the branching heuristic (Line 11). For example, it would iterate through all the 4 leaves in Figure 1 one by one if no leaf was pruned. The leader strategy δ^* to the maximal corresponding leader reward r^* is the

[3]This theorem can also be generalized to restricted games where $\Lambda' \subseteq \Lambda$, just like Theorem 1.

optimal leader strategy for this Bayesian game. Additionally, Algorithm 1 also returns the set of feasible pure strategies, Σ^*, and the corresponding bounds, $\mathcal{B}^*$. This feasible strategy set Σ^* is a subset of the cross-product of Σ^λ_Ψ, the feasible strategies per type, since it does not contain the strategies that were computed and found to be infeasible.[4] Σ^* and $\mathcal{B}^*$ are the feasibility sets and bounds that are propagated up the hierarchical tree; however, we first discuss the branch and bound heuristic used in Algorithm 1.

Branch and Bound Heuristics: HBGS sorts $\sigma^\lambda_\Psi \in \Sigma^\lambda_\Psi$, the pure strategies per type, in decreasing order of their bounds $\mathcal{B}(\sigma^\lambda_\Psi)$ before the tree in Figure 1 is constructed. The branching heuristic is that the leaf which can generate the higher leader expected utility is preferred. The bounds on each leaf are a direct application of Theorem 2. The function `getBounds` computes the weighted sum of the bounds per follower type $\mathcal{B}(\sigma^\lambda_\Psi)$[5] to generate the bound $\mathcal{B}(\sigma_\Psi)$ for this leaf.

Tree Traversal and Pruning: Algorithm 2 formally defines the tree-traversal strategy. The algorithm traverses the leaves of the follower action tree from left to right (*lexicographic* order) with the objective to find the first leaf (pure strategy) whose bound is higher than the current best solution r^*. If no such leaf exists, the optimal solution has been achieved and HBGS can be successfully terminated. This tree is constructed keeping the child nodes sorted in descending order from left to right in every sub-tree. For example, in Figure 1, $\mathcal{B}(\Sigma^1_\Psi(1)) \geq \mathcal{B}(\Sigma^1_\Psi(2))$ (children of root) and $\mathcal{B}(\Sigma^2_\Psi(1)) \geq \mathcal{B}(\Sigma^2_\Psi(2))$ where $\Sigma^\lambda_\Psi(i)$ represents the i^{th} pure strategy for follower type λ. The leaves are evaluated from left to right, that is, the leaf $[1,1]$ is evaluated first and leaf $[2,2]$ last.

If the bound $\mathcal{B}$ for any leaf σ_Ψ is smaller than the best solution obtained thus far, that leaf need not be evaluated. Additionally, *right* siblings of this leaf σ_Ψ need not be evaluated either, given the sorted nature of every sub-tree. For example, in Figure 1, if the bound of leaf $[2,1]$ is worse than the solution at $[1,2]$, then the leaf $[2,2]$ does not need to evaluated as well. Algorithm 2 accomplishes this type of pruning of branches as well.

Algorithm 2 `getNextStrategy`(σ_Ψ, r^*,Σ^Λ, $\mathcal{B}^\Lambda$)

for $\lambda = |\Lambda|$ to 1 Step -1 **do**
  j := `index-of`$(\Sigma^\lambda_\Psi, \sigma^\lambda_\Psi)$
  // *Fix the pure strategies of parents:* $\sigma^i_\Psi, i < \lambda$
  // *Update the pure strategy of type* λ: $\Sigma^\lambda_\Psi(j+1)$
  // *Children choose their best pure strategy:* $\Sigma^i_\Psi(1), i > \lambda$
  $\sigma_\Psi := [\sigma^1_\Psi, \ldots, \sigma^{\lambda-1}_\Psi, \Sigma^\lambda_\Psi(j+1), \Sigma^{\lambda+1}_\Psi(1), \ldots, \Sigma^{|\Lambda|}_\Psi(1)]$
  if $r^* <$ `getBounds`$(\sigma_\Psi, \mathcal{B}^\Lambda)$ **then**
    return σ_Ψ
return NULL

HBGS Summary: The leaves of the hierarchical type tree are solved to identify infeasible strategies and obtain upper bounds on every follower strategy. This information is propagated up the tree, and the procedure repeated for every node until the optimal solution is obtained at the root. While HBGS does incur the overhead of solving many smaller restricted games, it outperforms all existing techniques in the overall performance, as shown in Section 6.

4. HBSA OVERVIEW

Applications with complex scheduling constraints have inspired new algorithms to take advantage of structure in domains with extremely large strategy spaces for the leader. One example of such a domain is the scheduling problem faced by FAMS where the air marshals (*defender*) need to *cover* flights (*targets*) from a terrorist (*adversary*). Scheduling even 10 air marshals over 100 flights leads to approximately 1.7e13 joint schedules for the defender, so new algorithms like ASPEN [7] based on large scale optimization techniques like *column generation* have been proposed. However, no Bayesian extensions exist.

We first extend the ASPEN algorithm to handle arbitrary scheduling constraints in the presence of multiple follower types. We then present HBSA, which like HBGS, solves the Bayesian game hierarchically. We show that the key ideas of hierarchical decomposition can also be applied to Bayesian games in such domains.

Security problems with arbitrary scheduling constraints (SPARS) were first introduced by Jain et. al [7]. These problems are known to be NP-Hard in general [12]. The defender in the SPARS problem needs to protect a set T of targets from the adversary. The pure strategy of the defender is a joint schedule $\mathbf{P_j}$, which is an allocation of all its resources to a set of schedules S that agree with the scheduling constraints given in the SPARS problem. The pure strategy space of the adversary is the set of targets T; the adversary can choose to attack any target. The adversary succeeds if the target being attacked is not covered by the defender. The payoffs $\mathcal{U}$ are defined for both the players (refer Table 2). For example, consider a SPARS game modeling FAMS with 5 targets (flights), $T = \{t_1, \ldots, t_5\}$, and two air marshals. Let the set of feasible schedules be $S = \{\{t_1,t_2\},\{t_2,t_3\},\{t_3,t_4\},\{t_4,t_5\},\{t_1,t_5\}\}$. The set of all feasible joint schedules is shown below (1 implies that the target t is being *covered* by joint schedule $\mathbf{P_j}$), where each column represents a joint schedule:

$$\mathbf{P} = \begin{array}{c} \\ t_1: \\ t_2: \\ t_3: \\ t_4: \\ t_5: \end{array} \begin{array}{c} \begin{array}{ccccc} \mathbf{P_1} & \mathbf{P_2} & \mathbf{P_3} & \mathbf{P_4} & \mathbf{P_5} \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

The pure strategy space of the defender in such domains is so large that all the joint schedules cannot even be represented in memory all at once. ASPEN handles such large pure strategy spaces by using *column-generation*, a technique for large scale optimization where the "useful" joint schedules (or columns) are generated iteratively. The LP formulation of ASPEN is decomposed into a *master* problem and a *slave* problem to facilitate the application of column generation [7]. The master problem solves for the defender strategy $\mathbf{x}$, given a restricted set of columns (*joint schedules*) $\mathbf{P}$. The slave is designed to identify the best new column (i.e., joint schedule) to add to the master problem, while ensuring that the proposed joint schedule conforms to all the scheduling constraints of the domain. The objective function for the slave is updated based on the solution of the master using *reduced costs* from the solution of the master[6]. Column generation terminates if no column can improve the defender expected utility. We now first introduce the Bayesian extension to ASPEN.

4.1 Bayesian-ASPEN Column Generation

Bayesian-ASPEN also generates a tree of the pure strategies of the follower, as in Figure 1. Every leaf of the tree is evaluated using Bayesian-ASPEN. To that end, master and slave problems in ASPEN are extended for the Bayesian case.

Master Problem for Bayesian-ASPEN: The defender and the adversary optimization constraints from ASPEN need to be ex-

[4]Some of the strategies in Σ^* that were not computed may still be infeasible; Algorithm 1 ensures no feasible strategy is removed.

[5]The bounds are weighted by the distribution p^λ over types.

[6]Reduced costs, widely used in OR literature, measure the impact of a column (or variable) on the objective.

Table 2: SPARS Game Notation

Variable	Definition
$\mathbf{P}$	Mapping between Targets T and Joint Schedules J
$\mathbf{x}$	Distribution over J (mixed strategy of the defender)
$\mathbf{a}_\lambda$	Attack vector (pure strategy of the attacker type λ)
d_λ	Defender reward against type λ (analogous to $\mathcal{V}^\lambda_\Theta$)
k_λ	Reward of adversary type λ (analogous to $\mathcal{V}^\lambda_\Psi$)
$\mathbf{d}_\lambda$	Column vector of d_λ
$\mathbf{k}_\lambda$	Column vector of k_λ
$\mathcal{U}^u_{\lambda,\Theta}$	Utility for defender when target is uncovered
$\mathcal{U}^c_{\lambda,\Theta}$	Utility for defender when target is covered
$\mathbf{D}_\lambda$	Diag. matrix of $\mathcal{U}^c_{\lambda,\Theta}(t) - \mathcal{U}^u_{\lambda,\Theta}(t)$
$\mathbf{A}_\lambda$	Diag. matrix of $\mathcal{U}^c_{\lambda,\Psi}(t) - \mathcal{U}^u_{\lambda,\Psi}(t)$
$\mathbf{U}^u_{\lambda,\Theta}$	Vector of values $\mathcal{U}^u_\Theta(t)$, similarly for Ψ
M	Huge Positive constant

tended over all adversary types, in accordance with Equation (4). The master problem, given in Equations (13) to (17), solves for the probability vector $\mathbf{x}$ that maximizes the defender reward.[7] Equations (14) and (15) enforce the SSE conditions for the defender and adversary of each type, such that the players choose mutual best-responses to each other. The defender expected utility for protecting target t against adversary type λ is given by the t^{th} component of the column vector $\mathbf{D}_\lambda\mathbf{P}\mathbf{x} + \mathbf{U}^{\lambda,u}_\Theta$ (the adversary payoff is defined analogously). The notation is described in Table 2.

$$\max \quad \sum_{\lambda\in\Lambda} d_\lambda p_\lambda \tag{13}$$

$$\text{s.t.} \quad \mathbf{d}_\lambda - (\mathbf{D}_\lambda\mathbf{P}\mathbf{x} + \mathbf{U}^u_{\lambda,\Theta}) \le (\mathbf{1} - \mathbf{a}_\lambda)M \quad \forall\lambda \in \Lambda \tag{14}$$

$$0 \le \mathbf{k}_\lambda - (\mathbf{A}_\lambda\mathbf{P}\mathbf{x} + \mathbf{U}^u_{\lambda,\Psi}) \le (\mathbf{1} - \mathbf{a}_\lambda)M \quad \forall\lambda \in \Lambda \tag{15}$$

$$\sum_{j\in J} x_j = 1 \tag{16}$$

$$\mathbf{x}, \mathbf{a} \ge 0 \tag{17}$$

Slave Problem: The slave problem finds the best column to add to the current columns in $\mathbf{P}$. This is done using *reduced cost*, which captures the total change in the defender payoff if a candidate column is added to $\mathbf{P}$. The candidate column with minimum reduced cost improves the objective value the most [4]. The reduced cost $\bar{c}_j$ of variable x_j, associated with column $\mathbf{P_j}$, calculated using standard techniques, is given in Equation (18), where $\mathbf{w}_\lambda, \mathbf{y}_\lambda, \mathbf{z}_\lambda$ and h are dual variables of master constraints (14),(15-rhs),(15-lhs) and (16) respectively.

$$\bar{c}_j = \sum_{\lambda\in\Lambda} (\mathbf{w}^T_\lambda(\mathbf{D}_\lambda\mathbf{P_j}) + \mathbf{y}^T_\lambda(\mathbf{A}_\lambda\mathbf{P_j}) - \mathbf{z}^T_\lambda(\mathbf{A}_\lambda\mathbf{P_j})) - h \tag{18}$$

Reduced costs $\bar{c}_j$ are decomposed into $\hat{c}_t$, reduced costs per target:

$$\hat{c}_t = \sum_{\lambda\in\Lambda} (w_{\lambda,t}D_{\lambda,t} + y_{\lambda,t}A_{\lambda,t} - z_{\lambda,t}A_{\lambda,t}) \tag{19}$$

The column with the least reduced cost is identified using the same minimum cost network flow slave formulation as presented in ASPEN [7], using the newly computed $\hat{c}_t$.

4.2 HBSA Description

HBSA also decomposes the Bayesian-SPARS problem into many restricted Bayesian-SPARS games, constructing a hierarchical type tree, just like HBGS, and passing up infeasibility and bounds. However, HBSA uses Bayesian-ASPEN to *solve* each node of the follower action tree (refer Figure 1).

[7]The actual algorithm minimizes the negative of the defender reward for correctness of reduced cost computation; we show maximization of defender reward for expository purposes.

5. APPROXIMATIONS

The objective of these algorithms is to maximize the defender expected utility. Thus, the best known solution at any time during the execution of the algorithm is a lower bound to the optimal leader utility in the Bayesian Stackelberg games. Additionally, the upper bounds are determined using $\mathcal{B}$ (as described in Section 3.3) and are also available at all times during the algorithm's execution. The bounds are used to obtain approximate solutions with quality guarantees, the algorithm can be terminated as soon as the distance between lower and upper bounds is smaller than pre-defined approximation ϵ. Allowing for even 1% approximation in these algorithms can provide an order of magnitude speed-up in practice without any significant loss in solution quality (refer Section 6), where as no polynomial time algorithm can guarantee a *factor*-$|\Lambda|^{1-\epsilon}$ approximation for any $\epsilon > 0$ [14].

6. EXPERIMENTAL RESULTS

We provide three sets of experimental results. First, we compare the performance of DOBSS, Multiple-LPs and HBGS for generic Bayesian Stackelberg games. Second, we compare the scale-up performance of HBSA for security games with scheduling constraints. Third, we show speedups via approximations. The payoffs for both players for all test instances were randomly generated, and were consistent with the definition of security games [9] for experiments with HBSA. Results were obtained on a standard 2.8GHz machine with 2GB main memory, and are averaged over 30 trials.

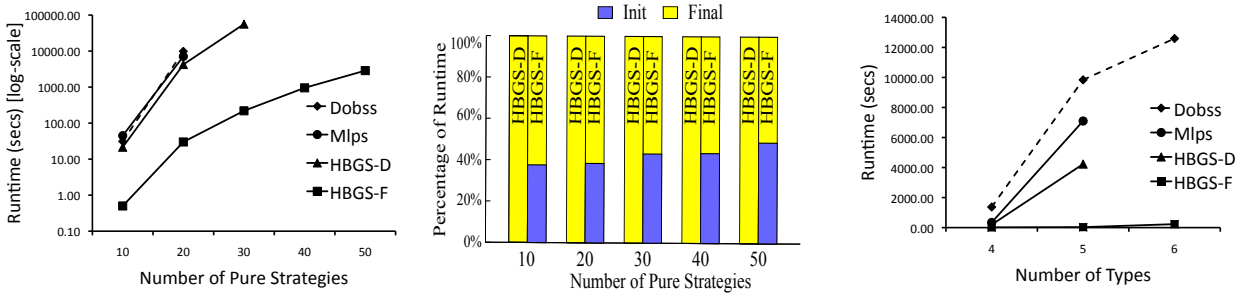
6.1 HBGS Scale-up

We compare the runtime of HBGS against the runtime of DOBSS and Multiple-LPs, the two chief algorithms for general Bayesian Stackelberg games. We use two variants of HBGS: (1) the first variant, denoted HBGS-D constructed a hierarchical tree of a fixed depth of one where as many restricted games were generated as the number of follower types. (2) The second variant, HBGS-F, constructed maximally branched binary trees such that each Bayesian game was decomposed into two restricted games with half as many types, until the leaves solved a restricted game with exactly one type. We compared the performance of these algorithms when the number of targets and the number of types were increased. We also show the speed ups obtained when approximation was allowed.

Scale-up of number of strategies: Figure 3(a) shows how the performance of the four algorithms scales when the strategy spaces are increased. These tests were done for 5 types. The x-axis shows the number of pure strategies for both players, while the y-axis shows the runtime in seconds on a log scale. For example, for 30 actions and 5 types, Multiple-LPs would solve $30^5 = 2.43e7$ linear programs. The experiments had a time cut-off of 24 hours.

The figure shows that while both variants of HBGS can successfully compute for 5 types and 30 pure strategies, DOBSS and Multiple-LPs cannot. Furthermore, HBGS-F with its fully balanced binary tree scales better than HBGS-D. This is because it solves a much smaller problem at the root node, even though it solves many more restricted problems. Each restricted game provides more pruning (infeasible combinations of follower actions will not be propagated up the tree) and potentially tighter bounds.

Figure 3(b) shows an analysis of time required by HBGS-D and HBGS-F in solving all the restricted Bayesian games before the root node of hierarchical type tree is solved. The x-axis shows



(a) Scaling Up Pure Strategies (5 types) (b) Initialization Time versus Total Time (c) Scaling Up Types (30 pure strategies)

Figure 3: This plot shows the comparisons in performance of the four algorithms when the size of the input problem is scaled.

the number of pure strategies for both the players and the y-axis shows the percentage of runtime. It shows that while HBGS-D spends almost no time in initialization ('Init'), HBGS-F spends almost 40% of its runtime in solving the restricted games. On the other hand, HBGS-F decomposes the problem more finely and thus spends more time solving more of the restricted games. This is because the number of restricted games generated by HBGS-F are more than the corresponding number in HBGS-D.However, the total time required by HBGS-F is considerably smaller (Figure 3(a)) which shows that hierarchical decompositions obtain more pruning and generate better bounds than depth-one hierarchical trees.

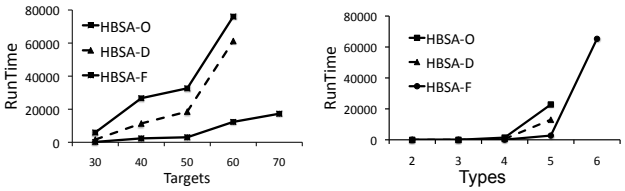
Scale-up of number of types: For these experiments, both the row and the column player had 30 pure strategies.The x-axis shows the number of types, whereas the y-axis shows the runtime in seconds. Again, the experiments were terminated after a cut-off time of 24 hours. We can see that HBGS-F scales extremely well as compared to the other algorithms; for example, HBGS-F solved a problem with 6 types in an average 231 seconds whereas DOBSS took an average of 12593.8 for the same problem instances. The other two algorithms didn't even finish their execution in 24 hours. While DOBSS and Multiple-LPs do not scale beyond a few number of types, HBGS-F provides scale-up by an order of magnitude. In Table 3, we present the runtime results of HBGS-F for up to 50 types. The experiments in this case had 5 pure strategies for both players (the other algorithms can not solve any instance with more than 20 types in 24 hours). This shows that DOBSS is no longer the fastest Bayesian Stackelberg game solution algorithm, and HBGS-F provides scale-up by *an order of magnitude*.

Table 3: Scaling up types (30 pure strategies per type)

Types	Follower Pure Strategy Combinations	Runtime (secs)
10	9.7e7	0.41
20	9.5e13	16.33
30	9.3e20	239.97
40	9.1e27	577.49
50	8.9e34	3321.681

6.2 HBSA Scale-up

In this section, we compare the performance of HBSA for Bayesian-SPARS games. Since no previous algorithms existed to solve such Bayesian security games with scheduling constraints, we compare the performance of variants of HBSA. We tested three different variants: (1) the first, HBSA-D, analogous to HBGS-D, uses a hierarchical tree with a depth of one, such that each leaf solves a restricted game with exactly one follower type. (2) The second, HBSA-F, analogous to HBGS-F, uses a fully branched binary tree. (3) The third, HBSA-O, also constructs a depth-one tree like HBSA-D, but uses ORIGAMI-S [7] to obtain bounds and branching heuristic from the restricted games. ORIGAMI-S is used since it is polynomial time, and has been shown to be an effective heuristic to generate bounds and branching rules for SPARS games [7].



(a) Scaling Up Targets (b) Scaling Up Types

Figure 5: This plot shows the comparisons in performance of the three algorithms when the input problem is scaled.

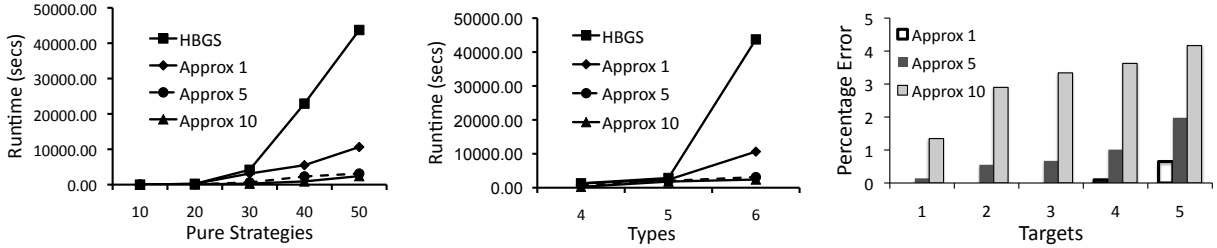
Scale-up in number of targets: In these experiments, the number of targets was varied while keeping the number of adversary types fixed to 5. The number of defender resources was set so cover 10% of the total number of targets. The results are shown in Figure 5(a) where the x-axis shows the number of targets and the y-axis shows the runtime in seconds. The graph shows that HBSA-F is fastest, and scales much better compared to the HBSA-O and HBSA-D variants. The simulations were terminated if they didn't finish in 24 hours. For example, HBSA-D and HBSA-O did not finish in 24 hours for the case with 70 targets, while HBSA-F was able to solve the problem instance in less than 5 hours.

Scale-up in number of types: These experiments varied the number of types, while keeping the number of targets fixed to 50. The number of resources was set to 5, so as to cover 10% of the total number of targets. The x-axis shows the number of types whereas the y-axis shows the runtime in seconds. The graph again shows that HBSA-F is the fastest algorithm. Again, the cut-off time for the experiments was 24 hours, and for example, HBSA-D and HBSA-O could not solve for 6 types in 24 hours.

6.3 Approximations

This section discusses the performance scale-ups that can be achieved when the algorithm was allowed to return approximation solutions. Three parameter settings of approximations were allowed: 1 unit, 5 unit and 10 units[8]. The approximations were

[8]The maximum reward in the matrix was 100 units, and these were chosen as 1%, 5% and 10% of the maximum possible payoff.



(a) Scaling Up Targets (b) Scaling Up Types (c) Solution Quality

Figure 4: This plot shows the comparisons in solution of the HBGS and its approximation variants.

tried on HBGS-F (with fully branched binary trees) since that prior experiments had shown it to be the fastest algorithm.

The number of types was fixed to 6 and the number of pure strategies was varied for the results shown in Figure 4(a). The number of targets here is shown on the x-axis, whereas the y-axis shows the runtime in seconds. Similarly, Figure 4(b) shows the results when the number of types was increased while fixing the strategy space to 50 pure strategies for the leader and all follower types. These figures show that the approximation variants of HBGS scale significantly better. For example, while HBGS-F took 43,727 seconds to solve a problem instance with 50 pure strategies and 6 types, the 1,5 and 10 unit approximations were able to solve the same problem in 10639, 3131 and 2409 seconds respectively, which is up to 18 times faster.

We also analyzed the difference in solution quality when the approximations were allowed, which is shown in Figure 4(c). The y-axis shows the *percentage error* in the *actual* solution quality of the approximate solution while the x-axis shows the number of targets. Lower bar implies lower error. For example, the maximum error in all settings for HBGS with an allowed approximation of five units was less than two percent. These results show that allowing for approximate solutions can dramatically increase the scalability of the algorithms without significant loss in the solution quality.

7. CONCLUSIONS

Algorithms for Stackelberg games have already seen limited applications in real-world domains; the capability to handle uncertainty using Bayesian models is an important avenue of research to facilitate further deployments. We present a new hierarchical algorithm that is able to provide scale-ups by orders of magnitude over the state-of-the-art. We apply this algorithm not only to general Bayesian Stackelberg games but also show how the key ideas can be applied to the latest algorithms for security games.

8. ACKNOWLEDGEMENTS

This research is supported by the United States Department of Homeland Security through Center for Risk and Economic Analysis of Terrorism Events (CREATE). We would like to thank the reviewers for helpful comments and suggestions.

9. REFERENCES

[1] N. Agmon, V. Sadov, G. A. Kaminka, and S. Kraus. The Impact of Adversarial Knowledge on Adversarial Planning in Perimeter Patrol. In *AAMAS*, volume 1, pages 55–62, 2008.

[2] R. Avenhaus, B. von Stengel, and S. Zamir. Inspection Games. In R. J. Aumann and S. Hart, editors, *Handbook of Game Theory*, volume 3, chapter 51, pages 1947–1987. North-Holland, Amsterdam, 2002.

[3] N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS*, pages 500–503, 2009.

[4] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1994.

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

[6] J. Harsanyi and R. Selten. A generalized nash solution for two-person bargaining games with incomplete information. In *Management Science*, volume 18, pages 80–106, 1972.

[7] M. Jain, E. Kardes, C. Kiekintveld, F. Ordonez, and M. Tambe. Security games with arbitrary schedules: A branch and price approach. In *AAAI*, pages 792–797, 2010.

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

[9] C. Kiekintveld, M. Jain, J. Tsai, J. Pita, M. Tambe, and F. Ordonez. Computing optimal randomized resource allocations for massive security games. In *AAMAS*, pages 689–696, 2009.

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

[11] M. Kodialam and T. Lakshman. Detecting network intrusions via sampling: A game theoretic approach. In *INFOCOM*, pages 1880–1889, 2003.

[12] D. Korzhyk, V. Conitzer, and R. Parr. Complexity of computing optimal stackelberg strategies in security resource allocation games. In *AAAI*, pages 805–810, 2010.

[13] G. Leitmann. On generalized Stackelberg strategies. *Optimization Theory and Applications*, 26(4):637–643, 1978.

[14] J. Letchford, V. Conitzer, and K. Munagala. Learning and approximating the optimal strategy to commit to. In *SAGT*, pages 250–262, 2009.

[15] P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus. Playing games with security: An efficient exact algorithm for Bayesian Stackelberg games. In *AAMAS-08*, pages 895–902, 2008.

[16] J. Tsai, S. Rathi, C. Kiekintveld, F. Ordonez, and M. Tambe. IRIS: a tool for strategic security allocation in transportation networks. In *AAMAS (Industry Track)*, pages 37–44, 2009.

[17] M. P. Wellman, D. M. Reeves, K. M. Lochner, S.-F. Cheng, and R. Suri. Approximate strategic reasoning through hierarchical reduction of large symmetric games. In *AAAI*, pages 502–508, 2005.