Optimal Interdiction of Attack Plans

Joshua Letchford*
jcl@cs.duke.edu
Duke University
Durham, NC

Yevgeniy Vorobeychik
yvorobe@sandia.gov
Sandia National Laboratories†
Livermore, CA

ABSTRACT

We present a Stackelberg game model of security in which the defender chooses a mitigation strategy that interdicts potential attack actions, and the attacker responds by computing an optimal attack plan that circumvents the deployed mitigations. First, we offer a general formulation for deterministic plan interdiction as a mixed-integer program, and use constraint generation to compute optimal solutions, leveraging state-of-the-art partial satisfaction planning techniques. We also present a greedy heuristic for this problem, and compare its performance with the optimal MILP-based approach. We then extend our framework to incorporate uncertainty about attacker's capabilities, costs, goals, and action execution uncertainty, and show that these extensions retain the basic structure of the deterministic plan interdiction problem. Introduction of more general models of planning uncertainty require us to model the attacker's problem as a general MDP, and demonstrate that the MDP interdiction problem can still be solved using the basic constraint generation framework.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Multi-agent systems*

General Terms

Economics, Security, Algorithms

Keywords

Game theory, security, planning, plan interdiction

1. INTRODUCTION

Interdiction seems by its very nature an adversarial act, one perpetrated, if you will, by "bad guys". For example, an attacker may interdict the power flow on an electric power grid, resulting in widespread blackouts [16], or interdict a transportation or a supply network [9, 8]. Indeed, even when the "good guys" engage in network interdiction, they do so with the goal of preventing activities, such as drug smuggling, by the criminals or adversaries. Conceptually, therefore, we deviate from that model: we argue that the nature of *defense* is fundamentally that of *interdicting attack plans*. In congruence with this point, we view the attacker as a planning agent that starts with a set of initial capabilities, and plans towards a set of specific goals. The defender's goal is therefore to develop mitigation strategies that optimally interdict an attacker who actively seeks to circumvent deployed mitigations. In solving the interdiction problem, the defender takes into account both the cost of mitigation strategies, and their benefit in terms of preventing the attacker from reaching a subset of goals. Crucially, our model is general-sum: the attacker and defender may have different priorities, and additionally, the attacker's decision problem involves planning costs, while the defender is concerned with the cost of mitigations.

We formulate the optimal attack plan interdiction problem as a large-scale integer linear program, and offer several oracle-based approaches for solving it. We then proceed to generalize the model to capture two kinds of uncertainty: first, uncertainty about the attacker's capabilities, costs, and goals, and second, uncertainty about whether a particular planning action is executed or not. We show that both of these aspects can be naturally folded into the interdiction problem based on classical planning. Finally, we consider an attacker model in which the attacker is solving an MDP, and show how the corresponding interdiction problem can be formulated and solved. The generality of MDPs comes at a cost: we now have to explicitly enumerate the entire problem state space, significantly reducing scalability.

2. RELATED WORK

Not surprisingly, graph, planning, and game theoretic approaches to cyber security and attacker modeling have a long history. Rather than tracing it in painstaking detail, we highlight some of the most related themes. The first important theme involves graph and planning-based approaches to attacker modeling. Its oldest incarnation is in the form of attack trees or graphs. The main limitation of the attack graph representation is that the state space, and, thus, the graph grow exponentially in the number of state variables. While much work, and usual industry practice, is to construct attack graphs by hand (by experts), there are a num-

*Partially supported by NSF award IIS-0953756

†Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013),* Ito, Jonker, Gini, and Shehory (eds.), May 6–10, 2013, Saint Paul, Minnesota, USA.

ber of efforts to automate their construction [18, 17]. Along similar lines, many have recognized the scalability limitation of constructing complete attack graphs, and focus only on generating specific sequences of attack steps (e.g., attack plans) that achieve a desired goal [21, 15, 1, 4, 10]. In most of the literature on attack graph analysis, mitigation is rather an afterthought; a few discuss heuristic approaches, such as computing a minimum set of attacks that must be prevented to ensure that the attacker fails to reach his goal [17, 1], as well as approximations to it. It is also a rarity to consider uncertainty in this line of work, though Poolsappasit et al. [14] is an important exception.

There are a number of explicit game-theoretic approaches to attacker-defender interactions in the attack graph framework [3, 22]. However, these approaches require a full specification of the attack graph, and do not scale beyond very small instances. There are several lines of research in game theoretic security games more broadly that have a bearing on our work. The first is the literature on network interdiction [9, 8, 12, 16], in which the attacker typically plays the role of the interdictor (or, the defender may wish to interdict, say, drug traffic or border penetration). In most cases, the problem is formulated as a zero-sum game, which is then cast as a bi-level mathematical program at the core of which is some variant of a network flow problem. One important exception closely connected to our effort is Brown et al. [5], which offers a bi-level programming formulation for a nuclear weapons project interdiction. This effort, however, ultimately retains the minimax flavor of other network interdiction problems, is focused rather narrowly on maximally extending the project length, and assumes that task dependencies are given; in our model, attack goals are much more generic, the game admits arbitrary payoffs for the defender and attacker, and task dependencies are computed dynamically as a part of the attacker's planning problem.

Finally, our effort is connected to the literature on computing Strong Stackelberg equilibria in leader-follower security games [11, 13]. Indeed, one view to take of our work is that we offer an approach to model circumvention games [13] by viewing an attacker as a planning agent, allowing us to reason explicitly about attacker circumvention space, rather than abstracting it into a single circumvention action.

3. INTEGER PROGRAMMING FOR CLASSICAL PLANNING

Our end goal is to create a highly scalable mathematical programming approach for optimal interdiction of attack plans. We begin from the ground up by reviewing an integer programming approach for classical and partial satisfaction planning, which ultimately forms the core of our own framework. While our starting point is the model of an attacker as a deterministic planner, we incrementally relax this assumption later on, attempting to retain as much of the classical planning structure (and relative tractability) in the process.

Formally, a classical planning problem is a tuple $P = \{L, A, I, G\}$, where L is the set of literals which capture all the information about the state of the world relevant for the planning problem, A is the set of actions, I is the set of literals which are initially true (i.e., the initial state of the world), and G is the set of goals. A plan action $a \in A$ is characterized by a set of *preconditions*, that is, the set of literals that must be true in the current state for the action to be applicable, and a set of *effects*, which are comprised of *add effects*, or literals that are added to the state if a is executed, and *delete effects*, which are the literals that a deletes (i.e., makes false) after execution. Our starting point is actually an extension to the classical planning framework termed *partial satisfaction planning*, which assigns each goal literal a value, and each plan action a cost. Formally, we let V_l denote the value of a goal literal $l \in G$, let $V_l = 0$ for all $l \notin G$, and let $C_a \geq 0$ denote the cost of action $a \in A$.

The problem of finding an optimal plan given a fixed number of time-steps has a known integer programming (IP) formulation [20, 19] which will provide the basis for our own techniques. The objective of this IP is to find a plan which maximizes utility (i.e., value of achieved goals less plan execution costs):

$$\max \sum_{l \in L} V_l s_l - \sum_{a \in A} C_a \sum_{t \in T} y_{a,t}, \tag{1}$$

where $y_{a,t} = 1$ if and only if action a is executed at time t and $s_l = 1$ if and only if the literal l is satisfied by the computed plan. For our purposes below, this will be the *attacker's utility function*. The planning IP introduces the following set of meta-variables to capture how actions modify the state of each literal l at every time step t:

- $x^{pa}_{l,t}$: 1 iff an action is executed in timestep t that has l as a precondition but does not delete it
- $x^{pd}_{l,t}$: 1 iff an action is executed in timestep t that has l as a precondition and deletes it
- $x^{add}_{l,t}$: 1 iff an action is executed in timestep t that does not have l as a precondition and adds it
- $x^{del}_{l,t}$: 1 iff an action is executed in timestep t that does not have l as a precondition but deletes it
- $x^{m}_{l,t}$: 1 iff l is true at timestep $t-1$ and no action in timestep t deletes it, adds it or has it as a precondition

The importance of these meta-variables is that they allow the set of constraints that enforce plan validity to be much more concise than if we were to reason about plan actions directly.

The constraints of the planning IP can be loosely categorized into three classes: constraints that build the above meta-variables, constraints that use these meta-variables to ensure that the computed plan is valid, and constraints that capture the initial conditions and goals. As an example, computing x^{pa} requires the following constraints:

$$\forall_{l,t} \sum_{a \in pre_l \setminus del_l} y_{a,t} \geq x^{pa}_{l,t} \tag{2}$$

$$\forall_{l,t,a \in pre_l \setminus del_l}\ y_{a,t} \leq x^{pa}_{l,t} \tag{3}$$

where pre_l represents the set of actions which have as l a pre-condition and del_l the set of actions where l is deleted as a post-condition. Each of the other meta-variables is computed with a similar set of constraints.

Enforcing plan validity amounts to ensuring that first, no mutually exclusive actions (e.g., actions that both add and delete a literal) are executed in the same time step, and second, that the state of a literal only changes at time t if there is an action executed at time t which either adds or deletes it. Formally, these constraints are:

$$\forall_{l,t}\ x^{pa}_{l,t} + x^{m}_{l,t} + x^{pd}_{l,t} \leq x^{add}_{l,t-1} + x^{pa}_{l,t-1} + x^{m}_{l,t-1} \tag{4}$$

$$\forall_{l,t}\ x^{pa}_{l,t} + x^{m}_{l,t} + x^{pd}_{l,t} \geq x^{add}_{l,t-1} \tag{5}$$

$$\forall_{l,t}\ x^{pa}_{l,t} + x^{m}_{l,t} + x^{pd}_{l,t} \geq x^{pa}_{l,t-1} \tag{6}$$

$$\forall_{l,t}\ x^{pa}_{l,t} + x^{m}_{l,t} + x^{pd}_{l,t} \geq x^{m}_{l,t-1}. \quad (7)$$

The final set of constraints establish the initial conditions and calculate which goal literals are satisfied by the plan:

$$\forall_l\ x^{add}_{l,|T|} + x^{pa}_{l,|T|} + x^{m}_{l,|T|} \geq s_l \quad (8)$$

$$\forall_l\ x^{add}_{l,|T|} \leq s_l \quad (9)$$

$$\forall_l\ x^{pa}_{l,|T|} \leq s_l \quad (10)$$

$$\forall_l\ x^{m}_{l,|T|} \leq s_l \quad (11)$$

$$\forall_l\ x^{add}_{l,0} = \begin{cases} 1 & \text{if } l \in I \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

To make the problem more concrete, consider the following example of an attack planning problem.

EXAMPLE 1. *The initial state includes the initial attacker capabilities, such as possession of a boot disk and port scanning utilities. Actions include both physical actions (breaking and entering and booting a machine from disk) as well as cyber actions, such as performing a port scan to find vulnerabilities.*[1] *Figure 1 shows an attack graph (with attack actions as nodes) for this scenario, with the actual attack plan highlighted in red.*

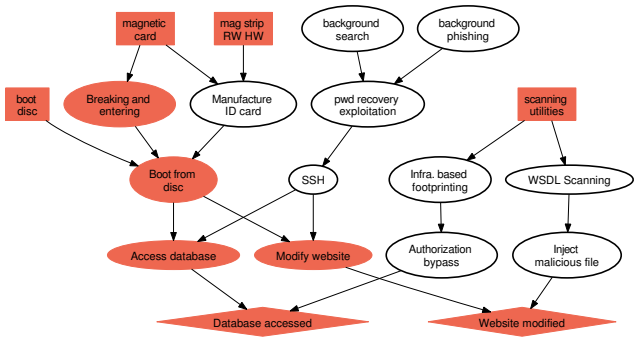

Figure 1: Example attack graph. Boxes correspond to initial attacker capabilities, ovals are attack actions, and diamonds are attacker goals. An optimal attack plan is highlighted in red.

4. DETERMINISTIC PLAN INTERDICTION

4.1 The Plan Interdiction Problem

Our ultimate goal is not merely to compute an optimal plan for the attacker, but rather to compute *an optimal defender interdiction strategy.* To this end, we model the interaction between the defender and attacker as a Stackelberg game in which the defender moves first, choosing to deploy a set of mitigations, and the attacker responds to these by constructing an optimal attack plan given the resulting environment. We formalize this game as a *deterministic plan interdiction problem (DPIP). DPIP* is described by a tuple $\{P, M, V_l^D, V_l^A, C_m^D, C_a^A\}$, where P is the planning problem for the attack in the absense of any mitigations, as described in Section 3, M is the set of mitigation strategies for the defender, V_l^D and V_l^A are the utilities of the defender and attacker respectively when the attacker achieves a goal literal $l \in G$, C_m^D is the cost of mitigation $m \in M$ to the defender, and C_a^A is the cost of action $a \in A$ to the attacker. A mitigation strategy $m \in M$ has two effects: it can protect against a subset of attack actions a (effectively removing them from A), and it can remove literals from the initial state I. Without loss of generality, we assume that m only has an effect on attacker actions (we can model removal of initial state literals by having actions with no preconditions, zero cost, and with $l \in I$ as the only add effect). For each mitigation $m \in M$, let $A_{m,a} = 1$ iff mitigation m removes action a. The defender's goal is to maximize his utility:

$$\max \sum_{l \in L} V_l^D s_l - \sum_{m \in M} D_m C_m^D, \quad (13)$$

where $D_m = 1$ iff mitigation m is chosen by the defender. The key complication is that, just as in general Stackelberg games, the defender's utility is a function of the attacker's best response, i.e., the optimal attack plan in response to the choice of defense mitigations. This raises the technical issue of tie breaking: if the attacker is indifferent between multiple plans, which would he choose? A common solution concept for Stackelberg security games is a Strong Stackelberg equilibrium, in which the attacker breaks ties in the defender's favor (we call this *optimistic tie breaking*). While counterintuitive, this solution concept is reasonable when the defender can commit to a randomized strategy, since he can resolve the attacker's indifference in his favor with an infinitesimal change in strategy. As we restrict the defender to choose mitigations deterministically, this justification becomes problematic, and it may be most reasonable to focus on the Weak Stackelberg equilibrium, that is, having the attacker break ties to minimize defender's utility (we call this *pessimistic tie breaking*). Below, we consider both variants and show empirically that in the context of deterministic commitment the difference between them is negligible in practice.

Before we delve into algorithmics, we proceed to answer the question of hardness of *DPIP*. First, we formulate this as a decision problem (*DPIP* decision problem, or *DPIPDP*).

DEFINITION 1 (DPIPDP). *Given a* DPIP*, is there a subset of mitigations $M^* \subset M$ that yields defender a utility of at least k?*

THEOREM 1. DPIPDP *is PSPACE-Complete under both optimistic and pessimistic tie breaking, even when all mitigation strategies remove only a single action.*

The proof can be found in the full version of this paper.

4.2 Integer Programming Formulation for Optimal Plan Interdiction

Despite the hardness result above, we now proceed to show how to compute optimal interdiction strategies in practice, and later experimentally demonstrate that our approaches are effective. For the moment, we ignore the issue of tie breaking, and will return to it in Section 4.5.

Our first step is to formulate the *DPIP* as a (very large; more on that later) integer program. The objective is, naturally, to maximize the defender's utility (Equation 13). The complication is that we must capture the attacker's best response to the defender's choice of mitigations. We do so by constructing a special set of constraints that amounts to ensuring that: a) the plan that the integer program chooses

[1]The actions in our example are taken from the CAPEC database (`http://capec.mitre.org`).

for the attacker is feasible (with mitigations imposing appropriate feasibility constraints), and b) this plan has a payoff to the attacker that is at least as high as any other feasible plan. The way we approach this is to consider (for now) the set of all possible attacker plans $\mathcal{P}$. Clearly, the optimal plan p^* is in $\mathcal{P}$. Moreover, we can use the set of constraints described in Section 3 to compute a feasible plan p, as well as its utility to the attacker. Thus, if we further ensure that the utility of p computed in the constraints is at least that of *any* feasible plan in $\mathcal{P}$, subject to the additional constraints imposed by the mitigations, we know that p is in fact the attacker's best response. To formalize this, let $U^A(p)$ be the (precomputed) attacker utility of a plan p, and let $\delta_p = 1$ if and only if plan p is interdicted (i.e., there is a deployed mitigation m that removes at least one action from p). Let Z be a large constant. The complete integer program for *DPIP*, which we call *DPIP_IP*, is given by

$$\max_{D_a, D_m, y_{a,t}, \delta_p} \quad \sum_{l \in L} V_l^D s_l - \sum_{m \in M} D_m C_m^D \tag{14}$$

s.t. :

$$\forall_a \quad D_a \leq \sum_m D_m A_{m,a} \tag{15}$$

$$\forall_{m,a} \; D_a \geq D_m A_{m,a} \tag{16}$$

$$\forall_{a,t} \; y_{a,t} \leq (1 - D_a) \tag{17}$$

$$\forall_{p,a} \; \delta_p \geq D_a \tag{18}$$

$$\forall_p \quad \delta_p \leq \sum_{a \in p} D_a \tag{19}$$

$$\forall_p \quad \sum_{l \in L} V_l^A s_l - \sum_{a,t} C_a^A y_{a,t} \geq U^A(p) - Z\delta_p \tag{20}$$

constraints on metavariables (Sec. 3)

constraints $4-7,\ 8-12$.

(Note that we use p here both as a plan index and as a set of actions that make up this plan). Constraints 15 and 16 compute a variable D_a which is 1 iff there is a mitigation that interdicts action $a \in A$. Constraint 17 ensures that only actions which are *not* interdicted are used in the attack plan. Constraints 18 and 19 compute δ_p. Finally, Constraints 20 ensure that the plan computed for the attacker is his best response to the defender's mitigations.

4.3 Scaling Up with Constraint Generation

The main problem with *DPIP_IP* is that the number of feasible plans and, consequently, the number of constraints, is exponential in the number of actions. To manage this problem in practice we develop several constraint generation (Bender's decomposition) approaches [2].

First, consider a relaxed version of *DPIP_IP* with Constraints 15-20 corresponding to a *subset*, $\hat{\mathcal{P}}$, of all possible plans. We call this *master* problem *DPIP_MASTER*($\hat{\mathcal{P}}$). Now, suppose that we solve the master problem, obtaining a set of mitigations $\hat{M} \subset M$ and that *DPIP_MASTER*($\hat{\mathcal{P}}$) identifies $\hat{p} \in \hat{\mathcal{P}}$ with a utility of $\hat{U}$ as the attacker's best response from the plans restricted to $\hat{\mathcal{P}}$. There are now two possibilities: either $\hat{p}$ is the best response of the attacker to $\hat{M}$, or the true attacker best response is not in $\hat{\mathcal{P}}$. To see which it is, we need to compute the actual best response of the attacker; fortunately, that is just the standard planning problem, and we can use the integer program from Section 3 upon removing the actions $a \in A$ that are blocked by mitigations $\hat{M}$. The plan that we thereby compute will either have a utility to the attacker that is exactly $\hat{U}$, confirming that $\hat{M}$ is the optimal solution (since $\hat{p}$ is a true best response), or strictly higher. In the latter case, we add the newly computed plan to the master program, and repeat. This procedure is presented in Algorithm 1.

```
P̂ = ∅;
Û = 0;
U = ∞;
while Û < U do
    (M̂, D_a, Û) = DPIP_MASTER(P̂);
    A_M̂ = ∅;
    for a ∈ A do
        if D_a = 0 then
            A_M̂ = A_M̂ ∪ a;
        end
    end
    (p, U) = optimalPlan(A_M̂);
    if U > Û then
        P̂ = P̂ ∪ p;
    end
end
```

Algorithm 1: Constraint generation algorithm.

To see that Algorithm 1 converges to the optimal interdiction strategy in finite time, consider what happens in any given iteration: either a new plan is added to the master program, or we prove that we have already computed an optimal solution. Since the total number of plans is finite—even if extremely large—the number of iterations must be finite. Therefore, the algorithm is sound (since we are generating best responses in each iteration) and complete.

One more subtle but practically consequential point (as we shall see in the experiments). While the initial incarnation of the master program is missing the Constraints 15-20 altogether, the set of constraints is still non-trivial, as it ensures that the attacker's response to mitigations is *feasible* (and most favorable for the defender). Therefore, the initial set of mitigations will in general be non-empty, and will already interdict some of the attack actions. In other words, even this initial master program can be viewed as making non-negligible progress towards the goal.

EXAMPLE 2. *We applied our method to the problem in Example 1, allowing the defender to interdict each attack action at a cost. The solution for a somewhat arbitrary assignment of parameters is shown in Figure 2.*

4.4 Optimistic Constraint Generation

Generating constraints for the master problem using an integer program from Section 3 clearly "works", in the sense that we will typically need to generate a relatively small set of plans (as we show in the experiments below). However, observe that to make progress in each iteration of Algorithm 1 we need only to generate a plan with a higher utility than any in $\hat{\mathcal{P}}$, and not necessarily an optimal plan. Doing so may, of course, result in more iterations, but if we can sufficiently speed up each iteration, on balance we could have a win.

There are two natural candidates for such optimistic constraint generation. The first is to still use the planning *IP*

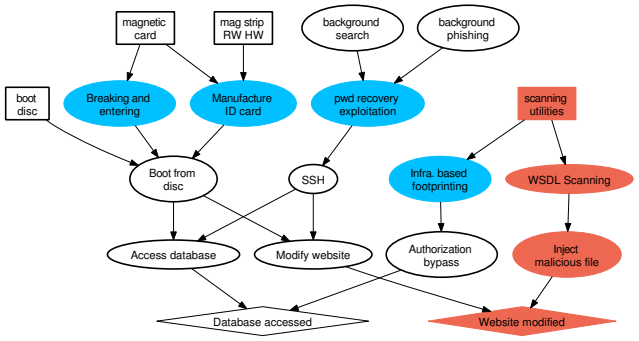

Figure 2: Example interdiction plan: actions that are blocked are colored in blue, and the final attack plan (circumvention) is highlighted in red.

from Section 3, but to cut off the solver after a fixed time limit. The second is to leverage the extensive heuristic planning work in AI, and use an off-the-shelf state-of-the-art heuristic planner, such as SGPLAN [6].

If we use optimistic constraint generation, Algorithm 1 is no longer sound, as we may generate a suboptimal plan which would make a current result appear optimal, even though it is not. To ensure soundness, we must therefore check the final solution obtained using a planner which does guarantee optimality, such as the planning *IP* from Section 3. However, since this needs only to be done relatively rarely (often only at the very end), most iterations will still run much faster than they would otherwise.

4.5 Tie breaking

The approach above breaks ties somewhat arbitrarily, since only a single best response is ever generated. Below we show how to handle pessimistic tie breaking and, thus, compute a Weak Stackelberg equilibrium; we can deal with optimistic tie breaking analogously.

Our first step is to ensure that the defender's utility is at most that of any non-interdicted attack plan *currently generated*. To that end we introduce binary variables α_p, and add the following set of constraints to set $\alpha_p = 1$ for all attacker-optimal non-interdicted plans:

$$\forall_p \quad \sum_{l \in L} V_l^A s_l - \sum_{a,t} C_a^A y_{a,t} \geq U^A(p) - Z\delta_p - \alpha_p + \epsilon, \quad (21)$$

where $\epsilon \in (0,1)$ is a sufficiently small number such that the requirement can only be satisfied by an attacker-optimal plan p if $\alpha_p = 1$. Then, for each plan p with $\alpha_p = 1$, the following constraint will be effective, placing the desired limit on the defender's utility:

$$\forall_p \quad \sum_{l \in L} V_l^D s_l \leq U^D(p) + Z(1 - \alpha_p), \quad (22)$$

with $U^D(p)$ the (precomputed) defender utility of a plan p. Since Constraints 20 ensure that the goals reached using an attacker-optimal plan are represented by the variables s_l, Constraints 22, in combination with Constraints 21 will ensure that the worst plan is chosen for the defender of those that are optimal for the attacker *from the set of plans that have been generated thus far*.

The problem with the approach above is that since we only generate and add a single optimal plan for the attacker at any given time, rather than all optimal attacker plans, we may overestimate the worst-case utility for the defender. To handle this, once Algorithm 1 finds no additional constraints to add, we solve another integer program to check if the best response is, indeed, defender pessimal. To do this we need to modify the *IP* from Section 3 in two ways. First, we change the objective to minimize the defender's value (maximize his losses) for goals achieved by the attacker:

$$\min \quad \sum_l V_l^D s_l$$

Second, we add a constraint that forces the attacker utility to equal the best response that we have computed in solving *DPIP_IP*, which we denote by $\bar{U}$:

$$\sum_l V_l^A s_l - \sum_a C_a^A \sum_t y_{a,t} = \bar{U}.$$

If the resulting objective value matches $\sum_{l \in L} V_l^D s_l$ computed by Constraints 22, we are done. If not, we have just identified a plan that can be added to the master program, at which point we restart Algorithm 1.

4.6 Heuristic Plan Interdiction

While we are the first to consider the problem of optimal plan interdiction at this level of generality, there are a number of informal ways in which attacker models, such as attack graphs or attack plans, are used to guide the design of mitigations in practice. In most cases, these actually require the specification of the entire attack graph, making it both suboptimal and intractable for even a modest number of state literals (for example, one would attempt to interdict a shortest path to a goal).

$A_{cur} = A$;
while *true* **do**
  $(p, U, G(p))$ =optimalPlan(A_{cur});
  $U'_{best} = U$;
  $\tilde{A}_{best} = \emptyset$;
  for $g \in G(p)$ **do**
    $\tilde{A}$ =chooseCheapest(p, g);
    $(p', U', G(p'))$ =optimalPlan($A_{cur} \setminus \tilde{A}$);
    if $U' > U'_{best}$ **then**
      $U'_{best} = U'$;
      $\tilde{A}_{best} = \tilde{A}$;
    end
  end
  if $U'_{best} - U > \sum_{a \in \tilde{A}_{best}} C_a^D$ **then**
    $A_{cur} = A_{cur} \setminus \tilde{A}_{best}$;
  else
    break;
  end
end

Algorithm 2: Heuristic plan interdiction.

In this section, we propose a greedy heuristic, shown as Algorithm 2, which relies only on the ability to generate plans in response to specific mitigations. Our heuristic can be viewed as a kind of formalization of the way attack models are actually used to obtain mitigation strategies: that is, by iteratively interdicting attacker's goals. Since different goals have different value to the defender, but may also carry

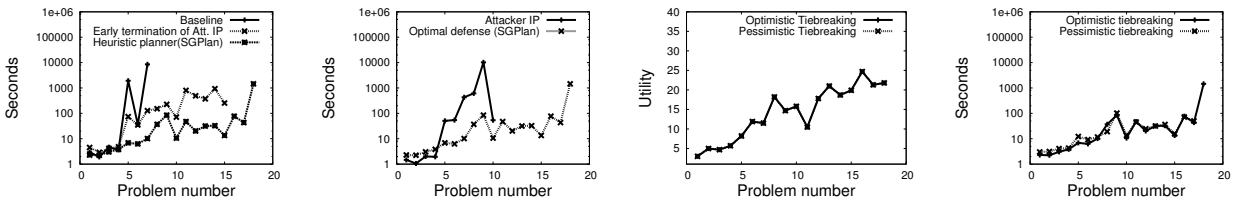

Figure 3: (a) Runtime comparison for several variants of constraint generation. (b) DPIP runtime vs. optimal planning runtime. (c) Runtime and (d) utility comparison of tie breaking approaches.

varying interdiction costs, in each iteration we choose a goal to interdict to maximize the defender's marginal utility. For simplicity, we assume that each mitigation blocks exactly one attack action, so that $C_m^D = C_a^D$ for action a interdicted by mitigation m. In Algorithm 2, the function *chooseCheapest*(p, g) chooses the cheapest set of actions from plan p such that goal g is no longer satisfied, and $G(p)$ represents the set of goals satisfied by a plan p.

Notice that since Algorithm 2 still requires us to compute optimal attacker plans, it is not a given that it will be faster than *DPIP_IP*. Below, we compare *DPIP_IP* with the greedy heuristic, demonstrating that in some cases it can be quite effective.

5. EXPERIMENTS WITH DETERMINISTIC PLAN INTERDICTION

For the experimental evaluation, we used the *pathways* planning domain from the 2006 international planning competition (IPC). To formulate these as interdiction problems, we let the set of mitigations correspond to the set of plan actions (thus, each mitigation m blocks exactly one attack action). We let the defender losses equal attacker gains, that is, $V_l^D = -V_l^A$ for all $l \in L$; note that the game remains non-zero-sum, since the costs are accounted for differently by the defender and the attacker. Interdiction costs are fixed at 1 for interdicting all actions $a \in A$, with the exception of special actions generated to capture logical expressions or combinations of goals, for which the interdiction costs are set to infinity. The problems that we ran our experiments on ranged in size from 46 possible attacker actions and one potential goal (problem number 1) to 490 attacker actions and 27 potential goals (problem number 20), with a general trend of a larger index having a larger number of possible attacker actions and goals. All computational experiments were performed on a 64 bit Linux 2.6.18-164.el5 computer with 96 GB of RAM and two quad-core hyperthreaded Intel Xeon 2.93 GHz processors, unless otherwise specified. We did not make use of any parallel or multi-threading capabilities, restricting a solver to a single thread, when relevant. Integer programs were solved using CPLEX version 12.4.

In the first set of experiments we compare the performance of our proposed algorithms. (Note that comparing to alternatives such as DOBSS [11] is a non-starter, since there are vastly more feasible plans than could be stored in memory.) The results are shown in Figure 3 (a). The "Baseline" solution in the figure uses Algorithm 1 as is, that is, with constraints generated by solving the integer program in Section 3. As we can see, even with constraint generation, this approach scales rather poorly: indeed, on most problems it is entirely infeasible. If we terminate the planning *IP* early, the performance is now far more reasonable, and most IPC problems can be solved. Finally, using SGPLAN to generate constraints yields another improvement (up to a factor of 100 on some problem instances); indeed, the running time of *DPIP* is now 10-100 seconds on all but one problem.

The second set of experiments compares the solution time of *DPIP* to the time it takes to run the attack planning *IP* on the same problem instance. The result, shown in Figure 3 (b), is quite surprising: *DPIP* with constraint generation appears to be many orders of magnitude *easier* to solve, even though it uses the same structure as the attack planning *IP*! The reason is that even in the absence of generated plans, the master *IP* still ensures that some *feasible* plan is chosen by the attacker. The result is that the mitigations chosen in the initial iteration already significantly restrict the planning problem, making it much easier to solve.

Our third set of experiments compares optimistic and pessimistic tie breaking. As one would expect, there is no discernible difference in terms of defender utility (Figure 3 (c)). What may be slightly more surprising is that there is essentially no difference in running time either (Figure 3 (d)).

Our fourth set of experiments compares the greedy heuristic for *DPIP* to our optimal plan interdiction algorithm (Figure 4). The results are mixed. On the one hand, the heuristic is remarkably good at approximating the optimal interdiction strategy for the defender (Figure 4, left). Moreover, for the smaller (lower numbered) problems, it is an order of magnitude faster than Algorithm 1. On the other hand, for many of the larger problems (except the largest), the optimal interdiction algorithm is actually much faster than the heuristic (Figure 4, right). Finally, while the runtime comparison between the greedy heuristic and optimal *DPIP* is ambiguous, greedy wins hands down in terms of memory utilization: the memory footprint of greedy is about 10% that for Algorithm 1 on the larger problems.

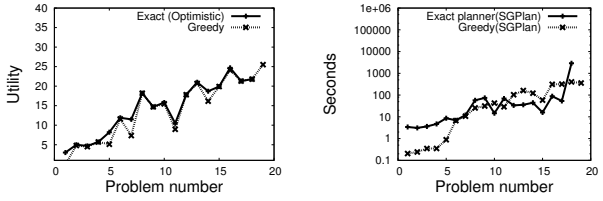

Figure 4: Comparison of the greedy heuristic (Algorithm 2) to optimal in terms of defender utility (left) and runtime (right).

6. DEALING WITH UNCERTAINTY

A crucial limiting feature of the approach we presented thus far is that it assumes determinism in every facet of the plan interdiction problem. In this section we incrementally relax this assumption.

6.1 Uncertainty about Attacker's Capabilities, Costs, and Goals

It is, in general, a severe simplification to treat attacker's capabilities, goals, and action costs as known to the defender. We relax this assumption in a relatively standard way by using a Bayesian Stackelberg game model, and focus here on uncertainty in attacker's capabilities (uncertainty about goals and costs can be handled similarly). Let Θ be the set of attacker types, and let I_θ be the set of capabilities (i.e., initial state) of type $\theta \in \Theta$. Additionally, let p_θ be the probability that the attacker has type θ, and index the variables s, V^A, V^D, and δ by the attacker's type. The *DPIP* objective function becomes

$$\sum_\theta \sum_{l \in L} p_\theta V^D_{\theta,l} s_{\theta,l} - \sum_{m \in M} D_m C^D_m,$$

while Constraints 17-20 now have to hold for each attacker type θ. Finally, in the constraint generation algorithm we solve a separate planning problem for each type, and add all the corresponding plans to the master program.

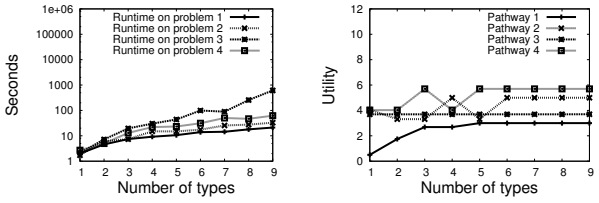

Figure 5: Runtime (left) and defender utility (right) of Bayesian plan interdiction as a function of the number of attacker types.

In Figure 5 (left) we show the runtime of the Bayesian plan interdiction framework with the appropriate modifications to the master problem and constraint generation. For this figure, we first generated 9 attacker types corresponding to initial capabilities, and then incrementally abstracted these to obtain smaller numbers of types. The good news is that on the three of four problems the runtime trend is subexponential. Another piece of good news is that it appears that type abstraction is rather effective, and the utility of the defender reaches near optimal with the number of abstracted types less than half of the number of actual types (indeed, grouping all types into one seems to already yield a near-optimal solution on three of the four problems). The bad news is that the bottleneck of this approach seems to be memory (we used a machine with 8GB of memory for this set of experiments): we were unable to run larger problem instances due to memory limitations.

6.2 Uncertain Attack Execution with Retry

Aside from Bayesian plan interdiction we had just considered, there is another type of uncertainty that we can handle within the basic *DPIP* framework: uncertainty about execution of attack actions, when the attacker can retry an action that fails an arbitrary number of times. The reason is that if the action cost is C^A_a, and the probability that the action succeeds is P_a, we can reformulate the problem by assigning a new action cost $\bar{C}^A_a = C^A_a / P_a$, and then solve the corresponding *DPIP*.

6.3 MDP Interdiction Problem

While certain special cases of uncertainty can be handled within the basic *DPIP* framework we introduced, in general we must move beyond it. In this section we explore one such generalization, modeling the attacker's problem as a *Terminating Markov Control Problem (TMCP)*. A TMCP is quite similar to a MDP, but in every state σ there is a "cash out" action that yields a reward based on which goals are satisfied and moves the MDP into an absorbing state, and, in addition, every action in each state has a non-zero probability of transitioning into the absorbing state. The convenience of using such termination probabilities (which can represent the probability of the attacker being caught) is that we can equivalently formulate the problem as an MDP with a discount factor that depends on current and next state, $\gamma_{\sigma,\sigma'}$. Let $V^A_{a,\sigma}$ be the attacker's value (reward) in state σ if he takes action a, and let I and G be the initial state, and the state in which all of the attacker's goals are satisfied, respectively. Finally, define $T^{a,\sigma}_{\sigma'}$ to be the probability of moving to state σ' from σ if the attacker takes action a. We formulate this as an interdiction problem in precisely the same way as we had done for *DPIP*, endowing the defender with the set of mitigation strategies and associated costs, and assigning the defender a utility function $V^D_{a,\sigma}$ for each state σ and attack action a performed in that state.

As the first step we leverage a linear programming formulation for computing the optimal policy of a TMCP [7]:

$$\max_{y_{a,\sigma}} \sum_{a,\sigma} V^A_{a,\sigma} y_{a,\sigma} \tag{23}$$

s.t. :

$$\forall_{\sigma' \neq I} \quad \sum_{\sigma,a} (\delta_{\sigma,\sigma'} - T^{a,\sigma}_{\sigma'}) y_{a,\sigma} = 0 \tag{24}$$

$$\sum_{a,\sigma} (\gamma_{\sigma,I} - T^{a,\sigma}_I) y_{a,\sigma} = 1 \tag{25}$$

$$\sum_{a,\sigma} (\gamma_{\sigma,G} - T^{a,\sigma}_G) y_{a,\sigma} = -1, \tag{26}$$

where $y_{a,\sigma} = 1$ iff the attacker chooses action a in state σ. The full MDP interdiction problem (*MDPIP*) can then be formulated as the following integer program:

$$\max \sum_{a,\sigma} V^D_{a,\sigma} y_{a,\sigma} - \sum_m C^D_m D_m$$

s.t. :

$$\forall_p \sum_{a,\sigma} V^A_{a,\sigma} y_{a,\sigma} \geq U^A(p) - \delta_p Z$$

$$\text{constraints } 15-19,\ 24-26.$$

We can then use Algorithm 1 as is, with constraints generated using standard methods for solving MDPs.

To study the difference between the runtime and memory characteristics of *MDPIP* and *DPIP*, we use the planning problem in Example 1, varying the number of actions available to the attacker between 13 and 16 (*MDPIP* runs out of memory for larger problems, as well as for all problems

from IPC 2006). To introduce uncertainty, we added attack step execution uncertainty with retry; as we noted in Section 6.2, such uncertainty can also be captured in the *DPIP* framework. Figure 6 shows what we would have expected:

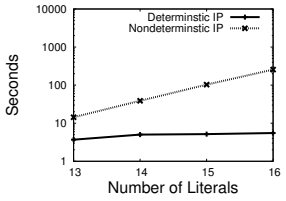

Figure 6: Runtime comparison between *MDPIP* and *DPIP*.

MDPIP, while much more general, is an order of magnitude (and growing) slower than *DPIP*. Additionally, the memory footprint of *MDPIP* is an even greater issue: for 16 literals, solving *MDPIP* takes about a factor of 20 more memory.

7. CONCLUSION AND FUTURE WORK

We presented a Stackelberg game model of security in which the defender chooses optimal mitigations that reduce the capability of the attacker to achieve his goals. We offered an integer programming formulation of this problem, and presented several variants of constraint generation, leveraging a state-of-the-art AI planning tool to dramatically increase scalability. Additionally, we presented a heuristic alternative, and showed that it uses far less memory than the optimal plan interdiction algorithm, yields nearly optimal solutions, and runs considerably faster than the optimal algorithm on some, though not all, test problems. We extended the classical planning framework to incorporate uncertainty about attacker's capabilities, costs, goals, as well as execution uncertainty, and showed that these extensions retain the basic structure of the deterministic plan interdiction problem and, therefore, its scalability. More generally, we provided an integer programming formulation for computing optimal interdiction strategies for MDPs, but generality here comes at a substantial cost to scalability.

Throughout, we tackled problems in which the defender can only use *deterministic* mitigation strategies, somewhat in contrast with much recent work that allows the defender to randomize, achieving, in general much higher utility. As mixed strategy commitment in our context presents a number of both conceptual and technical challenges, we leave it for future work.

8. REFERENCES

[1] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *ACM Conference on Computer and Communications Security*, pages 217–224, 2002.

[2] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[3] Stefano Bistarelli, Marco Dall'Anglio, and Pamela Peretti. Strategic games on defense trees. In *Fourth International Conference on Formal Aspects of Security and Trust*, pages 1–15, 2006.

[4] Mark Boddy, Johnathan Gohde, Tom Haigh, and Steven Harp. Course of action generation for cyber security using classical planning. In *International Conference on Automated Planning and Scheduling*, pages 12–21, 2005.

[5] Gerald G. Brown, W. Matthew Carlyle, Robert C. Harney, Eric M. Skroch, and R. Kevin Wood. Interdicting a nuclear-weapons project. *Operations Research*, 57(4):866–877, 2009.

[6] Yixin Chen, Benjamin W. Wah, and Chih wei Hsu. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research*, 26:323–369, 2006.

[7] Jerzy Filar and Koos Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.

[8] P.M. Ghare, D.C. Montgomery, and W.C. Turner. Optimal interdiction policy for a flow network. *Naval Research Logistics Quarterly*, 18(1):37–45, 1971.

[9] A.W. McMasters and T.M. Mustin. Optimal interdiction of a supply network. *Naval Research Logistics Quarterly*, 17(3):261–268, 1970.

[10] Jorge Lucangeli Obes, Carlos Sarraute, and Gerardo Richarte. Attack planning in the real world. In *Second Workshop on Intelligent Security*, 2010.

[11] Praveen Paruchuri, Jonathan P. Pearce, Janusz Marecki, Milind Tambe, Fernando Ordóñez, and Sarit Kraus. Playing games with security: An efficient exact algorithm for Bayesian Stackelberg games. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems*, pages 895–902, 2008.

[12] Cynthia A. Phillips. The network inhibition problem. In *ACM Symposium on Theory of Computing*, pages 776–785, 1993.

[13] James Pita, Milind Tambe, Chris Kiekintveld, Shane Cullen, and Erin Steigerwald. Guards - game theoretic security allocation on a national scale. In *Tenth International Conferenceon Autonomous Agents and Multiagent Systems*, pages 37–44, 2011.

[14] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. Dynamic security risk management using bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing*, 9:61–74, 2012.

[15] Ronald W. Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *IEEE Symposium on Security and Privacy*, pages 156–165, 2000.

[16] J. Salmeron, K. Wood, and R. Baldrick. Worst-case interdiction analysis of large-scale electric power grids. *IEEE Transactions on Power Systems*, 24(1):96–104, 2009.

[17] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *IEEE Symposium on Security and Privacy*, pages 273–284, 2002.

[18] Laura P. Swiler, Cynthia Phillips, David Ellis, and Stefan Chakerian. Computer-attack graph generation tool. In *DARPA Information Survivability Conference and Exposition II*, 2001.

[19] Menkes van den Briel, Romeo Sanchez, Minh B. Do, and Subbarao Kambhampati. Effective approaches for partial satisfaction (over-subscription) planning. In *Nineteenth National Conference on Artifical Intelligence*, pages 562–569, 2004.

[20] Thomas Vossen, Michael Ball, and Robert H. Smith. On the use of integer programming models in ai planning. In *Sixteenth International Joint Conference on Artificial Intelligence*, pages 304–309, 1999.

[21] Dan Zerkle and Karl Levitt. NetKuang — A multi-host configuration vulnerability checker. In *USENIX Unix Security Symposium*, 1996.

[22] Saman A. Zonouz, Himanshu Khurana, William H. Sanders, and Timothy M. Yardley. RRE: A game-theoretic intrusion response and recovery engine. In *International Conference on Dependable Systems and Networks*, pages 439–448, 2009.