Near-Optimal Interdiction of Factored MDPs

Swetasudha Panda and Yevgeniy Vorobeychik
Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN
swetasudha.panda,yevgeniy.vorobeychik@vanderbilt.edu

Abstract

Stackelberg games have been widely used to model interactions between attackers and defenders in a broad array of security domains. One related approach involves plan interdiction, whereby a defender chooses a subset of actions to block (remove), and the attacker constructs an optimal plan in response. In previous work, this approach has been introduced in the context of Markov decision processes (MDPs). The key challenge, however, is that the state space of MDPs grows exponentially in the number of state variables. We propose a novel scalable MDP interdiction framework which makes use of factored representation of state, using a parity function basis for representing a value function over a Boolean space. We demonstrate that our approach is significantly more scalable than prior art, while resulting in near-optimal interdiction decisions.

1 INTRODUCTION

Stackelberg game approaches to security have received considerable attention in recent years, both in theoretical investigation and practical use [8, 11, 14]. A major challenge in such approaches is high-resolution modeling of adversarial evasion of defensive measures. Letchford and Vorobeychik proposed modeling such evasion in a Stackelberg framework of *plan interdiction* [12], where the Stackelberg leader eliminates a subset of attack actions, and the adversary computes an optimal plan in the restricted action space. This approach was developed both in the context of deterministic (PDDL-based) planning, and planning with Markov decision processes (MDPs). However, while interdiction of deterministic plans was quite scalable, the approach scaled very poorly in the context of MDPs. The central challenge with MDP interdiction is the exponential size of the state space in the number of state variables.

We aim to address the problem of MDP interdiction at scale by leveraging approximation techniques developed for factored MDPs. Scalability in factored MDPs has been achieved by two basic approaches: a) explotting structure in the MDP transition model and reward function [3, 2, 9], and b) value function approximation [1, 9, 10, 4, 16, 5]. Factored MDPs [6] represent the complex state space using state variables and the transition model using a dynamic Bayesian network. This representation allows an exponential reduction in the representation size of structured MDPs. Moreover, efficient approximate solution algorithms have been proposed that exploit structure in factored MDPs.

Starting with the approximation methods for factored MDPs, we develop a mixed-integer linear programming approach for factored MDP interdiction. In doing so, we face two challenges: 1) effective basis representation, and 2) a super-exponential set of constraints corresponding to alternative evasion plans for the attacker. To address the first challenge, we propose using a Fourier (parity function) basis over a Boolean hypercube to represent the value function over a binary factor space. While there always exists an exact Fourier basis for functions over a Boolean space, the representation is exponential in size. We address this challenge by developing iterative basis generation methods. Addressing the second challenge of an intractably large constraint space, we develop a novel constraint generation algorithm using a combination of linear programming factored MDP solvers and novel heuristics for attack plan generation. We demonstrate the effectiveness of the proposed approaches on realistic examples from the international planning competition (IPC). In particular, we show that our approach offers dramatically improved scalability without significantly compromising solution quality.

2 PRELIMINARIES

Our work builds on solution approaches for discounted infinite-horizon MDPs, and particularly for factored MDPs, which we now introduce.

MDPs and Factored MDPs Formally, a discounted infinite-horizon MDP is defined as a tuple $\mathcal{D} = (\mathbf{X}, A, R, P, \gamma)$ where $\mathbf{X}$ is a finite set of $|\mathbf{X}| = N$ states; A is a finite set of actions; R is a *reward function* $R : \mathbf{X} \times A \mapsto \mathbb{R}$, in which $R(\mathbf{x}, a)$ is the reward obtained by the agent in state **x** after taking action a; P is a *Markovian transition model* where $P(\mathbf{x}'|\mathbf{x}, a)$ is the probability of moving from state $\mathbf{x}$ to $\mathbf{x}'$, after taking action a; and $\gamma \in [0, 1)$ is the discount factor which exponentially discounts future rewards. It is well-known that such MDPs always admit an optimal *stationary deterministic policy*, which is a mapping $\pi : \mathbf{X} \mapsto A$, where $\pi(\mathbf{x})$ is the action the agent takes at state **x** [15]. Each policy can be associated with a *value function* $\mathcal{V}_\pi \in \mathbb{R}^N$, where $\mathcal{V}_\pi(\mathbf{x})$ is the discounted cumulative value obtained by starting at state $\mathbf{x}$ and following policy π. Formally,

$$\mathcal{V}_\pi(\mathbf{x}) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R(\mathbf{X}^t, \pi(\mathbf{X}^t)) \big| \mathbf{x}^{(0)} = \mathbf{x}\right],$$

where $\mathbf{X}^{(t)}$ is a random variable representing the state of the system after t steps.

The discounted reward MDP is a natural model for security, since it captures the fact that attackers prefer to achieve their goals (positive rewards) earlier, and incur costs (negative rewards) later. Additionally, it captures an element of deterrence: the attack which takes too many steps has far more opportunities to fail in practice. Thus, minor (low-reward) goals may be preferred over major (high-reward) goals if they can be achieved much more quickly.

Factored MDPs exploit problem structure to compactly represent MDPs. The set of states is described by a set of *random state variables* $\mathbf{X} = \{X_1, \ldots, X_n\}$. Let $\text{Dom}(X_i)$ be the domain of values for X_i. A state $\mathbf{x}$ defines a value $x_i \in \text{Dom}(X_i)$ for each variable X_i. *Throughout, we assume that all variables are Boolean.* The transition model for each action a is compactly represented as the product of local factors by using a DBN. Let X_i denote the variable X_i at the current time and X_i' the same variable at the next time step. For a given action a, each node X_i' is associated with a conditional probability distribution (CPD) $P_a(X_i'|\text{Parents}_a(X_i'))$. The transition probability is given by $P_a(\mathbf{x}'|\mathbf{x}) = \prod_i P_a(x_i'|\mathbf{x}[\text{Parents}_a(X_i')])$, where $\mathbf{x}[\text{Parents}_a(X_i')]$ is the value in $\mathbf{x}$ to the variables in $\text{Parents}_a(X_i')$. The complexity of this representation is linear in the number of state variables and exponential in the number of variables in the largest factor. The reward function is represented as the sum of a set of localized reward functions. Let $R_1^a, \ldots, R_r^a$ be a set of functions, where the scope of each R_i^a is restricted to the variable cluster $\mathbf{W}_i^a \subset \{X_1, \ldots, X_n\}$. The reward for taking action a at state $\mathbf{x}$ is then $R^a(\mathbf{x}) = \sum\limits_{i=1}^{r} R_i^a(\mathbf{W}_i^a) \in \mathbb{R}$.

Linear Programming Methods for Solving MDPs A common method for computing an optimal policy of an MDP is by using the following linear program (LP):

$$\min \sum_{\mathbf{x}} \alpha(\mathbf{x}) V(\mathbf{x}) \tag{1a}$$

$$\text{s.t.: } \forall \mathbf{x}, a, V(\mathbf{x}) \geq R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x}, a) V(\mathbf{x}'). \tag{1b}$$

where the variables $V(\mathbf{x})$ represent the value function $\mathcal{V}(\mathbf{x})$, starting at state $\mathbf{x}$. The *state relevance weights* αs are such that $\alpha(\mathbf{x}) > 0$ and $\sum\limits_{\mathbf{x}} \alpha(x) = 1$. The optimal policy π^* can be computed as the greedy policy with respect to $\mathcal{V}^*$, $\pi^* = \arg\max_a [R(\mathbf{x}, a) + \gamma \sum\limits_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x}, a) V(\mathbf{x}')]$. The dual of this LP (dual LP) maximizes the total expected reward for all actions:

$$\max \sum_{\mathbf{x}} \sum_{a} \phi_a(\mathbf{x}) R(\mathbf{x}, a) \tag{2a}$$

$$\text{s.t.: } \forall \mathbf{x}, a, \quad \phi_a(\mathbf{x}) \geq 0 \tag{2b}$$

$$\forall \mathbf{x}, \sum_a \phi_a(\mathbf{x}) = \alpha(\mathbf{x}) + \gamma \sum_a \sum_{\mathbf{x}'} P(\mathbf{x}|\mathbf{x}', a) \phi_a(\mathbf{x}'). \tag{2c}$$

where $\phi_a(\mathbf{x})$ called the *visitation frequency* for state $\mathbf{x}$ and action a is the (discounted) expected number of times that state $\mathbf{x}$ will be visited and action a will be executed in this state. There is a one-to-one correspondence between policies in the MDP and feasible solutions to the dual LP.

In the case of factored MDPs, there is no guarantee that the structure extends to the value function [9] and linear value function approximation is a common approach. A factored (linear) value function $\mathcal{V}$ is a *linear function* over a set of *basis functions* $H = \{h_1, \ldots, h_k\}$, such that $\mathcal{V}(\mathbf{x}) = \sum_{j=1}^k w_j h_j(\mathbf{x})$ for some coefficients $\mathbf{w} = (w_1, \ldots, w_k)'$, where the scope of each h_i is restricted to some subset of variables $\mathbf{C}_i$. The approximate LP corresponding to (1) is given by Guestrin et al. [6]:

$$\min \sum_i \alpha_i w_i \tag{3a}$$

$$\text{s.t.: } \forall a, \max_{\mathbf{x}}\{R^a(\mathbf{x}) + \sum_i w_i[\gamma g_i^a(\mathbf{x}) - h_i(\mathbf{x})]\} \leq 0. \tag{3b}$$

where for basis h_i, $\alpha_i = \sum_{\mathbf{x}} \alpha(\mathbf{x}) h_i(\mathbf{x})$ is the factored equivalent of α and $g_i^a(\mathbf{x}) = \sum_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x}, a) h_i(\mathbf{x}')$ is the factored representation of expected future value. The non-linear constraint in LP (3) can be represented by a set of linear constraints using approaches similar to variable elimination in cost networks. The factored dual approximation LP [7] is defined on a set of variable clusters $\mathcal{B} \supseteq \mathcal{B}_{FMDP}$ where $\mathcal{B}_{FMDP} = \{\mathbf{W}_1^a, \ldots, \mathbf{W}_r^a : \forall a\} \cup \{\mathbf{C}_1, \ldots, \mathbf{C}_k\} \cup \{\Gamma_a(\mathbf{C}_1), \ldots, \Gamma_a(\mathbf{C}_k) : \forall a\}$, $\Gamma_a(\mathbf{C}) = \cup_{X_i \in \mathbf{C}} \text{PARENTS}_a(X_i) = \text{Scope}[g]$ is the set of parent state variables of variables in $\mathbf{C}$ (Scope$[h]$) in the DBN for action a. This factored dual LP is given by:

$$\max \sum_a \sum_{j=1}^{r} \sum_{\mathbf{w}_j^a \in \text{Dom}[\mathbf{W}_j^a]} \mu_a(\mathbf{w}_j^a) R_j^a(\mathbf{w}_j^a)$$

s.t.:

$$\forall i = 1, \ldots, k :$$

$$\sum_{\mathbf{c} \in \text{Dom}[\mathbf{C}_i]} \mu(\mathbf{c}) h_i(\mathbf{c}) = \sum_{\mathbf{c} \in \text{Dom}[\mathbf{C}_i]} \alpha(\mathbf{c}) h_i(\mathbf{c})$$

$$+ \gamma \sum_a \sum_{\mathbf{y} \in \text{Dom}[\mathbf{\Gamma}_a(\mathbf{C}_i')]} \mu_a(\mathbf{y}) g_i^a(\mathbf{y}) \tag{4a}$$

$$\forall \mathbf{B}_i, \mathbf{B}_j \in \mathcal{B}, \forall \mathbf{y} \in \text{Dom}[\mathbf{B}_i \cap \mathbf{B}_j], \forall a :$$

$$\sum_{\mathbf{b}_i \sim [\mathbf{y}]} \mu_a(\mathbf{b}_i) = \sum_{\mathbf{b}_j \sim [\mathbf{y}]} \mu_a(\mathbf{b}_j) \tag{4b}$$

$$\forall \mathbf{B} \in \mathcal{B}, \forall \mathbf{b} \in \text{Dom}[\mathbf{B}], \forall a, \mu_a(\mathbf{b}) \geq 0 \tag{4c}$$

$$\mu(\mathbf{b}) = \sum_{a'} \mu_{a'}(\mathbf{b}), \tag{4d}$$

$$\sum_{\mathbf{b}' \in \text{Dom}[\mathbf{B}]} \mu(\mathbf{b}^{'}) = \frac{1}{1-\gamma} \tag{4e}$$

where $\mu_a(\mathbf{b}) = \sum_{\mathbf{x} \sim [\mathbf{b}]} \phi_a(\mathbf{x}), \forall \mathbf{b} \in \text{Dom}[\mathbf{B}]$ is the *marginal visitation frequency* for a subset of state variables $\mathbf{B} \subset \mathbf{X}$ ($\mathbf{b} \in \text{Dom}[\mathbf{B}]$ represents enumeration of the variables in $\mathbf{B}$ and $\mathbf{x} \sim [\mathbf{b}]$ are the assignments of $\mathbf{x}$ that are consistent with $\mathbf{b}$), and $\mu(\mathbf{b}) = \sum_a \mu_a(\mathbf{b})$. The constraints ensure that these μ_a variables are consistent across variable subsets. The factored dual approximation is guranteed to be equivalent to the dual LP-based approximation [7] if the factored MDP cluster set $\mathcal{B}$ forms a junction tree. Triangulation $\mathbf{Tr}(\mathcal{B}_{FMDP})$ constructs a junction tree by adding cluster sets to $\mathcal{B}$ if needed. Approximate triangulation $\hat{\mathbf{Tr}}(\mathcal{B})$ returns some cluster set $\mathcal{B}'$ such that $\mathcal{B} \subseteq \mathcal{B}'$. The constant basis function h_0—i.e., with scope as the empty set $\{\emptyset\}$—is always included in H for feasibility of the above factored LPs.

Computing Attack Policies Policies in factored MDPs can be compactly represented assuming the default action model [10]. Different actions often have very similar transition dynamics, only differing in their effect on a small subset of variables. In factored MDPs that follow a *default transition model* for each action a, $Effects[a] \subset \mathbf{X}'$ are the variables in the next state whose local probability model is different from the model for the default action d, i.e., $P_a(X_i'|\text{Parents}_a(X_i')) \neq P_d(X_i'|\text{Parents}_d(X_i'))$ [10]. Similarly, in the *default reward model*, there is a set of reward functions for the default action d. The extra reward of any action a has scope restricted to $\mathbf{W}_i^a$. With the above assumptions, the greedy policy relative to a factored value function can be represented as a decision list [6].

However, this default action model is often not applicable in many real world examples. In such cases, we solve the approximate factored dual LP and monitor the values of the μ_a variables as a proxy to determine if a certain action appears in the computed policy. More precisely, if ϕ_a is a feasible solution to the exact dual LP, then in a state $\mathbf{x}$, $\phi_a(\mathbf{x}) > 0$ if $a = \pi(\mathbf{x})$ and $\phi_a(\mathbf{x}) = 0$ for all other actions. In the approximate solution with a subset of basis functions, all ϕ_a variables may not be represented by the set of μ_a variables. However, we can approximately determine the set of actions in a policy by removing those actions a from the set of allowed actions, for which $\mu_a(\mathbf{b}) = 0 \; \forall \mathbf{b} \in \text{Dom}[\mathbf{B}], \forall \mathbf{B} \in \mathcal{B}$.

3 MDP INTERDICTION

3.1 PROBLEM DEFINITION

We model MDP interdiction as a Stackelberg (two-stage, one-shot) game with two players: defender and attacker. The defender, who is the Stackelberg leader, commits to a set of mitigations, and the attacker, who is the follower, computes an MDP policy which optimally responds to (e.g., evades) these mitigations.

Formally, the *MDP interdiction problem (MDPI)* is defined by a tuple $\{\mathcal{M}, C_m, R^D, R^A, \mathbf{X}, A, P, \gamma\}$, $\mathcal{M}$ is the set of mitigation strategies available to the defender, R^D and R^A are the reward functions for the defender and the attacker respectively, C_m is the cost of a mitigation $m \in \mathcal{M}$ to the defender, and $\mathbf{X}, A, P, \gamma$ are the state space, action space, transition function, and discount factor of an infinite-horizon discounted MDPs which the attacker is solving in response to mitigations deployed by the defender. The semantics of a mitigation $m \in \mathcal{M}$ is that it removes (protects against) a subset of attack actions from the original attacker action space

A.[1] For a given set of mitigations $M \subseteq \mathcal{M}$ deployed by the defender, we can define an attacker's resulting MDP, $\tau(M) = [\mathbf{X}, A(M), R^A, P, \gamma]$ over the restricted action space $A(M)$ which includes only the actions which are not removed by any mitigation $m \in M$. In the MDPI Stackelberg game, the defender first chooses $M \subseteq \mathcal{M}$, and the attacker subsequently chooses a policy π in the resulting restricted MDP $\tau(M)$. Since the attacker is effectively facing a decision problem, it will suffice to restrict attention to optimal attacker policies which are deterministic and stationary. Let $\Pi^*(M)$ be the set of optimal deterministic stationary policies of $\tau(M)$. Define $\mathcal{V}^A(\mathbf{x}, \pi)$ to be the attacker's value function for a policy π starting at state $\mathbf{x}$ in MDP $\tau(M)$, and let $\mathcal{V}^D(\mathbf{x}, \pi)$ be the defender's value function (i.e., using the defender's reward function R^D). Let $\mathbf{x}_0$ be the initial state of the MDP. We seek a *strong Stackelberg equilibrium (SSE)* of MDPI, in which the defender solves

$$\max_{M \subseteq \mathcal{M}} \mathcal{V}^D(\mathbf{x}_0, \pi^*(M)) - \sum_{m \in M} C_m,$$

where $\pi^*(M) \in \arg\max_{\pi \in \Pi^*(M)} \mathcal{V}^A(\mathbf{x}_0, \pi)$, and the attacker breaks ties in the defender's favor.

3.2 GENERAL APPROACH

Letchford and Vorobeychik proposed a general approach for MDP interdiction [12] based on a mixed-integer linear programming (MILP). If we define variables D_m which are 1 iff the defender chooses a mitigation m, the defender's objective becomes $\max \sum_{\mathbf{x}} \sum_{a} \phi_a(\mathbf{x}) R^D(\mathbf{x}, a) - \sum_{m \in M} D_m C_m$, where $\phi_a(\mathbf{x})$ are the dual variables of the MDP linear program as before. The attacker's objective is then given by $\max \sum_{\mathbf{x}} \sum_{a} \phi_a(\mathbf{x}) R^A(\mathbf{x}, a)$. To account for the attacker's best response, a set of constraints was introduced for the defender so that a) the optimal attacker policy chosen by the MILP is feasible given the set of defender mitigations, and b) the attacker's utility corresponding to this computed policy is better than that of any other feasible policy. Since the general approach relies on the exact representation of the state space, it fails to scale. Below we introduce our general framework which leverages the factored representation of MDPs, enabling scalability to practical problem instances.

3.3 A MILP FORMULATION FOR FACTORED MDP INTERDICTION

We first exhibit the defender and attacker objectives using a factored representation of states. Given a set of basis functions H and the variable cluster set $\mathcal{B}_{FMDP}$ the defender's utility is given by

$$\sum_a \sum_{j=1}^{r} \sum_{\mathbf{w}_j^a \in \text{Dom}[\mathbf{W}_j^a]} \mu_a(\mathbf{w}_j^a) R_j^{Da}(\mathbf{w}_j^a) - \sum_{m \in M} D_m C_m,$$

where the expected sum of rewards is represented by the μ_a variables, the factored version of the visitation frequencies with scope restricted to that of the local reward functions. The first term is to minimize the attacker's value of the initial state (we set $R^D = -R^A$ in our experiments) and the second term represents mitigation costs. The attacker's objective, in turn, is

$$\sum_a \sum_{j=1}^{r} \sum_{\mathbf{w}_j^a \in \text{Dom}[\mathbf{W}_j^a]} \mu_a(\mathbf{w}_j^a) R_j^{Aa}(\mathbf{w}_j^a).$$

For each mitigation $m \in M$, let $A_{m,a} = 1$ iff m removes action a. To ensure that the computed policy is feasible, we add the constraints 4a-4e in the approximate factored dual LP. Let $\delta_\pi = 1$ if and only if the policy π is interdicted, i.e., there is a deployed mitigation m that removes at least one action from π. We denote the following MILP formulation for *MDPI* by *MDPI_MILP*:

$$\max \sum_a \sum_{j=1}^{r} \sum_{\mathbf{w}_j^a \in \text{Dom}[\mathbf{W}_j^a]} \mu_a(\mathbf{w}_j^a) R_j^{Da}(\mathbf{w}_j^a) - \sum_m D_m C_m$$

s.t.:

$$\forall a, m, D_m A_{m,a} \leq D_a \leq \sum_{m'} D_{m'} A_{m',a} \tag{5a}$$

$$\forall a, \forall \mathbf{B} \in \mathcal{B}, \forall \mathbf{b} \in \text{Dom}[\mathbf{B}],$$
$$\mu_a(\mathbf{b}) \leq Z(1 - D_a) \tag{5b}$$

$$\forall \pi, a \in \pi, D_a \leq \delta_\pi \leq \sum_{a' \in \pi} D_{a'} \tag{5c}$$

$$\forall \pi, \sum_a \sum_{j=1}^{r} \sum_{\mathbf{w}_j^a \in \text{Dom}[\mathbf{W}_j^a]} \mu_a(\mathbf{w}_j^a) R_j^{Aa}(\mathbf{w}_j^a)$$
$$\geq \mathcal{V}^A(\mathbf{x}_0, \pi) - Z\delta_\pi \tag{5d}$$

constraints $4a - 4e$

where Z is a large number and $\mathcal{B} = \mathbf{Tr}(\mathcal{B}_{FMDP})$. (We use $\hat{\mathbf{Tr}}(\mathcal{B}) = \mathcal{B}$ so that $\mathcal{B} = \mathcal{B}_{FMDP}$). The constraints 5a compute a variable D_a such that $D_a = 1$ iff there is a mitigation m that interdicts action a. Constraint 5b ensures that if an action is interdicted, the corresponding visitation frequencies are 0. Constraints 5c compute

[1] This is quite general; for example, we can model mitigations which modify the initial state by including actions with no preconditions and effects which represent initial state, and allow interdiction of these actions.

δ_π. Constraints 5d represents the condition that the policy generated for the attacker is its best response to the defender's choce of mitigations.

If H, the set of basis functions considered is general enough to include the full value function space, the solution to this MILP yields the optimal interdiction decision for the defender. The key challenge, however, is (a) what basis function space we should consider, and (b) given that capturing arbitrary value functions in the basis space is likely intractable, how can we best approximate a value function basis in this space. Finally, the set of constraints captures all possible attack policies, thereby rendering the MILP too large to be tractable even with a compact set of bases. We address these challenges next, starting with the issue of iteratively generating constraints to avoid complete enumeration of the policy space (Section 4), and proceeding to address the basis selection problem thereafter (Section 5).

4 CONSTRAINT GENERATION FOR FACTORED MDP INTERDICTION

The MDP interdiction algorithm requires the addition of policies and the corresponding utilities as constraints (captured by Constraint 5d). To compute the attacker's best response, we solve the approximate primal LP 3 for a given basis set H (we deal with the basis selection problem in Section 5). We can then compute the attacker's policy as discussed in Section 2.

4.1 CONSTRAINT GENERATION WITH BASIS FUNCTION SELECTION

We define the *master* problem *MDPI_MASTER($\hat{\mathcal{P}}$)* as a relaxed version of the MILP with the constraints 5a-5d corresponding to a subset $\hat{\mathcal{P}}$ of all possible policies. For now, suppose that we have a method for selecting a subset of "important" basis functions (Section 5).

The constraint generation procedure (Algorithm 1) works as follows. In any iteration, $\hat{\mathcal{P}}$ contains a small set of attack policies generated thus far. We solve the master problem with $\hat{\mathcal{P}}$ to obtain a set of mitigations $\hat{M} \subseteq \mathcal{M}$, along with a policy $\hat{\pi} \in \hat{\mathcal{P}}$ with a utility of $\hat{V} = \mathcal{V}^A(\mathbf{x}_0, \hat{\pi})$ which is the attacker's best decision from the feasible subset of policies in $\hat{\mathcal{P}}$. Now there are two possibilities: either $\hat{\pi}$ is the actual best response of the attacker, in the presence of the deployed mitigations, or the actual attacker best response is not in $\hat{\mathcal{P}}$. To confirm, we can compute the best response for the attacker by solving a factored MDP (LP 3), removing actions which are blocked by the mitigations $\hat{M}$. At this point, we also improve our basis function set, as described in Section 5 (GENERATEBASIS($A_{\hat{M}}, H$)). The resulting solution will either have a utility of $\hat{V}$ to the attacker, confirming $\hat{M}$ as the optimal set of mitigations, or will be a strict improvement on $\hat{V}$, in which case we add the resulting policy (computed as described in Section 2), and its utility, to the master program, and repeat.

Algorithm 1 Constraint Generation with Basis Selection

```
function CONSTRAINTGENERATION($\hat{\mathcal{P}}, H$)
    $V = \infty$
    $\hat{V} = 0$
    while $\hat{V} < V$ do
        $(\hat{M}, D_a, \hat{V})$ =MDPI_MASTER($\hat{\mathcal{P}}, H$)
        $A_{\hat{M}} = \emptyset$
        for $a \in A$ do
            if $D_a = 0$ then
                $A_{\hat{M}} = A_{\hat{M}} \cup a$
        $(\pi, V, \hat{H})$ = GENERATEBASIS($A_{\hat{M}}, H$)
        if $V > \hat{V}$ then
            $\hat{\mathcal{P}} = \hat{\mathcal{P}} \cup \pi$
            $H = \hat{H}$
```

This constraint generation procedure suffers from two important bottlenecks: the number of iterations can be large, and each iteration can be computationally costly. Next we improve upon the baseline procedure above by alleviating both of these issues.

4.1.1 Reducing the Number of Iterations of Constraint Generation

A natural way to begin the constraint generation process is with an empty subset of attack policies $\hat{\mathcal{P}}$. However, this results in a large number of iterations of the constraint generation procedure building up enough attack policies to prevent trivial mitigation solutions, which mitigate no actions, or a single action sufficient to make all policies in $\hat{\mathcal{P}}$ infeasible. To address this, we start by selecting a subset of (possibly all) attacker actions $\hat{a} \in A$. For each $\hat{a}$, we solve the attacker's best response with the single action $\hat{a}$, using the approximate primal LP 3. The corresponding attack policies have only a single action. We then define the initial subset $\hat{\mathcal{P}}$ using these sets of attacker policies and the corresponding utilities, allowing us to warm start the constraint iteration procedure and saving a considerable amount of computation time in the process.

4.1.2 Fast Constraint Generation

While warm starting considerably reduces the number of iterations of the constraint generation procedure, each iteration still involves a costly set of computational operations even to evaluate whether new policies need to be added. However, we observe that to make progress in constraint generation, we only need to find *some* policy which yields a better utility for the attacker than the optimal policy computed in the master program.

A major part of the overhead is the size of the basis set. To speed up computation, we propose to attempt generating an improved policy (AttackerPolicy(A, H)) first using only a small subset of the basis function (e.g., H_1 with each basis using a single state variable, as discussed in Section 5). If the attacker's utility computed in the subproblem is greater than the best response computed by the master program, we add this policy to the set of constraints. Otherwise, we fall back on the full combined basis selection and factored MDP solution approach. This approach may need more iterations to converge, but each iteration will be much faster. Algorithm 2 is a formalization of these ideas.

Algorithm 2 Fast Constraint Generation

```
function ConstraintGeneration($\hat{\mathcal{P}}, H_1$)
    $V = \infty$
    $\hat{V} = 0$
    while $\hat{V} < V$ do
        ($\hat{M}, D_a, \hat{V}$)=MDPI_MASTER($\hat{\mathcal{P}}, H_1$)
        $A_{\hat{M}} = \emptyset$
        for $a \in A$ do
            if $D_a = 0$ then
                $A_{\hat{M}} = A_{\hat{M}} \cup a$
        $(\pi, V)$ = AttackerPolicy($A_{\hat{M}}, H_1$)
        if $V > \hat{V}$ then
            $\hat{\mathcal{P}} = \hat{\mathcal{P}} \cup \pi$
        else
            $(\pi, V, \hat{H})$ = GenerateBasis($A_{\hat{M}}, H_1$)
```

5 BASIS GENERATION

In this section we address the issue of selecting a basis function space for linear value function approximation and, subsequently, the incremental generation of the "important" set of basis functions H.

5.1 FOURIER BASIS FUNCTIONS ON BOOLEAN FEATURE SPACE

We start by making use of the assumption that all variables are Boolean. In this case, the Fourier (parity) basis for Boolean function is a natural basis choice: every function $f : \{0, 1\}^n \to \mathbb{R}$ can be uniquely represented as $f(\mathbf{x}) = \sum_{S \subseteq \{1,\ldots,n\}} \hat{f}(S) h_S(\mathbf{x})$ [13], where h_S is a parity function over the subset S of the variables:

$$h_S(\mathbf{x}) = \prod_{i \in S} (-1)^{x_i} = \begin{cases} +1, & \text{if } \sum_{i \in S} x_i \bmod 2 = 0. \\ -1, & \text{if } \sum_{i \in S} x_i \bmod 2 = 1. \end{cases} \tag{6}$$

While the full Fourier representation of the value function is therefore linear, and exact, it has 2^n bases. Consequently, it is crucial to intelligently select a small subset which yields a sufficiently good approximation of the value function for the purposes of computing an approximately optimal set of mitigations. We do this by an iterative basis function selection process described below.

5.2 ITERATIVE BASIS FUNCTION SELECTION

The attacker solves the approximate LP (3) to compute the best response to the imposed mitigations. Observe that the basis functions correspond to variables in this LP. Column generation can be used to generate only those variables which have the potential to improve the objective function. Thus, basis functions can be iteratively generated while computing the attacker's policy. However, since the variables $\mathbf{w}$ corresponding to the basis functions are unconstrained, the concept of reduced cost is not well-defined. In this case, we compute the magnitude of the constraint violation in the dual LP instead.

Recall that the non-linear constraint in the LP (3) $\max_{\mathbf{x}}\{R^a(\mathbf{x}) + \sum\limits_i w_i[\gamma g_i^a(\mathbf{x}) - h_i(\mathbf{x})]\} \leq 0$ is represented as a set of linear constraints using variable elimination [6]. Instead of enumerating the entire state space, one variable is eliminated at a time. There is one set of factored LP constraints for each action a. Let X_j be the variable being eliminated. If X_j appears in any set $\mathbf{C} \cup \Gamma_a(\mathbf{C})$, ($\mathbf{C}$ = Scope$[h]$), and/or $\mathbf{W}^a$ (the scope of any local reward function) these set of state variables are "relevant" while eliminating X_j. Denote this set of relevant variables by $\mathbf{Z}_j^a$. Only these variables are enumerated while maximizing over X_j. For each enumeration, the linear constraint is of the form $u^{max} \geq u^R + \sum_i w_i u^{\gamma g_i - h_i}$, where u^{max} is the variable introduced after elimination, u^R is the relevant factored reward term and $u^{\gamma g_i - h_i}$ represents $\gamma g_i - h_i$ for a relevant h_i. After all state variables are eliminated, the remaining elimination-introduced variables have empty scope and the final maximization constraint is added. The number of constraints in this LP grows exponentially in the induced width of the *cost network*, the undirected graph defined over the variables $X_1, \ldots, X_n$, with an edge between X_l and X_m if they appear together in $\mathbf{Z}_j^a$. Given this construction, we describe our basis function selection approach as follows.

We begin with a subset of basis functions H^0 and solve the above factored LP. It is necessary to include $h_0 = \emptyset$ in H^0 to ensure feasibility of the LP. Next, we need to determine whether a new basis function will improve the current LP objective. We consider the dual LP

$$\begin{aligned} \max_{\lambda_k \geq 0} \quad & \sum_k u_k^R \lambda_k \\ \text{s.t.:} \quad & \sum_k u_k^{\gamma g_i - h_i} \lambda_k = \alpha_i, \forall i, \end{aligned} \tag{7}$$

where λ_k is the dual variable corresponding to a factored linear constraint k in the primal LP, and u_k^R and $u_k^{\gamma g_i - h_i}$ are the reward function and basis function terms respectively in constraint k. If a new basis h_l is added, it generates a new column in the primal LP, and thus, a new constraint in the dual LP. If the new constraint is not satisfied given the current λ, the objective can be improved by adding this basis. More precisely, if the new constraint is violated given the current λ, the amount of violation $\beta = |\sum_k u_k^{\gamma g_l - h_l} \lambda_k - \alpha_l|$ can be used to decide whether to include the new basis. We compute the magnitude of constraint violation for a possible new basis and choose the basis which maximizes this violation. We add this basis to the primal LP and repeat. Finally, we return the updated set of basis functions. The corresponding LP objective is the attacker's utility, given a set of actions A. We outline this procedure formally in Algorithm 3.

Algorithm 3 Iterative Basis Function Selection

```
function GENERATEBASIS($A, H$)
    $\lambda, V'$ =ATTACKERPOLICY($A, H$)
    for $s \in \{1, \ldots, s_{max}\}$ do
        while $H_s \neq \emptyset$ do
            $\beta = 0$
            for $h_l \in H_s$ and $h_l \notin H$ do
                if $|\sum_k u_k^{\gamma g_l - h_l} \lambda_k - \alpha_l| > \beta$ then
                    $\beta = |\sum_k u_k^{\gamma g_l - h_l} \lambda_k - \alpha_l|$
                    $\hat{h}_l = h_l$
            $H = H \cup \hat{h}_l$
            $(\lambda, V)$ =ATTACKERPOLICY($A, H$)
            if $|V - V'| < \theta$ then return $V', H$
            $H_s = H_s \setminus \hat{h}_l$
            $V' = V$
```

In Algorithm 3, ATTACKERPOLICY(A, H) solves the LP (3) and H_s is the set of parity basis functions over s state variables. To maintain smaller cost networks, we consider all bases of a particular size before moving to the next size until $s = s_{max}$, for some $s_{max} \leq n$. Within a particular size, we consider those variable clusters that are also connected in the underlying DBN of the factored MDP (i.e., one variable is the parent node of the other variable). We observe that many dual variables λ_k will be 0 so that we can restrict all computations to the set of active constraints $\{k, \lambda_k > 0\}$. Finally, using the parity basis functions allows two simplifications. First, we consider the g^a variable corresponding to a basis h: $g^a(\mathbf{y}) = \sum_{\mathbf{c}\in\mathbf{C}} \prod_{i|X_i \in \mathbf{C}} P_a(\mathbf{c}[X_i]|\mathbf{y})h(c)$, for each assignment $\mathbf{y} \in \Gamma_a(\mathbf{C})$, where $\mathbf{C}$ is the scope of h and the sum is over Dom$[\mathbf{C}]$, the enumeration of variables in $\mathbf{C}$. In our case, using the parity basis, this sum of products can be reorganized as a product of sums: $g^a(\mathbf{y}) = \prod_{i|X_i \in \mathbf{C}} P(x_i = 0|\mathbf{y}, a) - P(x_i = 1|\mathbf{y}, a)$. These terms can be precomputed for each state variable allowing efficient computation. Second, we consider $\alpha = \sum_{\mathbf{x}} \alpha(\mathbf{x})h(\mathbf{x}) = \sum_{\mathbf{c}\in\mathbf{C}} \alpha(\mathbf{c})h(\mathbf{c})$, where $\alpha(\mathbf{c})$ is the marginal of α over Dom$[\mathbf{C}]$. In the case of parity basis functions, $\alpha_l = 0, \forall l \neq 0$ and $\beta = |\sum_k u_k^{\gamma g_l - h_l} \lambda_k|$.

6 GREEDY INTERDICTION

In this section, we propose a greedy heuristic for factored MDP interdiction which requires the generation of attacker policies in response to specific mitigations. Specifically, we start with a mitigation strategy by randomly choosing an action to block. The attacker then computes a policy with utility V using the restricted set of actions. Next, we evaluate actions in the available set of actions A_{av}, at random, choosing an action to block if it decreases the sum of the attacker utility and total mitigation cost. Here we assume that each mitigation blocks exactly one action. The algorithm proceeds until no action can be found to be blocked so as to improve the defender's utility. This greedy algorithm is outlined as Algorithm 4.

We speed up greedy interdiction similar to fast constraint generation in Section 4.1.2 by computing policies with a small subset of basis functions (e.g., H_1). At the very end, we make further additions to the set of basis functions to compute the attacker's policy in response to the greedily computed mitigation strategy to check whether the attacker can indeed improve on the approximate best response computed over the restricted space.

7 EXPERIMENTS

We evaluate our MDP interdiction algorithms on several instances of three problem domains from the international planning competition (IPC 2014): a) sysadmin b) academic advising and c) wildfire. While these have little direct connection to security, they provide the most meaningful evaluation of our approaches in terms of effectiveness and scalability: prior security-related domains which consider multi-stage attacks use toy examples which would not provide a meaningful evaluation.

Algorithm 4 Greedy Factored MDP Interdiction

```
$A_{av} = A$
$A_m = \emptyset$
$A_n = A_{av}$
$\hat{V} = \infty$
while $A_n \neq \emptyset$ do
    $a =$ ChooseRandom($A_{av}$)
    $V =$ AttackerPolicy($A_{av} \setminus a, H$)
    if $V^A < \hat{V}$ then
        $A_m = A_m \cup a$
        $\hat{V} = V$
        $A_{av} = A_{av} \setminus a$
        $A_n = A_{av}$
        if $A_{av} = \emptyset$ then
            break
    else
        $A_n = A_n \setminus a$
return $V =$ GenerateBasis($A \setminus A_m, H$)
```

For all experiments, each defender mitigation $m \in \mathcal{M}$ blocks exactly one action a. We also let $R^A(\mathbf{x}, a) = -R^D(\mathbf{x}, a) - C_a$, where C_a is the cost of action a, which we set to 0 for the default (no-op) action and to 0.5 for all other actions.. We set the cost of imposing a mitigation $C_m = 1$ for all m. We use the discount factor of $\gamma = 0.9$. The experiments are run on a 2.4GHz hyperthreaded 8-core Ubuntu Linux machine with 16 GB RAM, with CPLEX version 12.51 used to solve MILP instances.

7.1 COMPARISON WITH EXACT MDP INTERDICTION

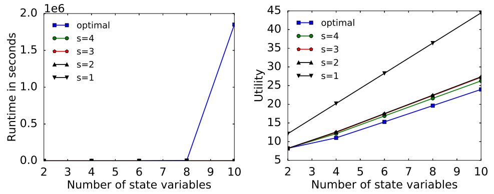

Figure 1: Comparison of exact and approximate MDP interdiction in terms of runtime (left) and attacker utility (right; lower is better for the defender).

First, we compare the performance of the constraint generation with basis selection algorithm to the state-of-the-art optimal solution in MDP interdiction proposed by Letchford and Vorobeychik [12]. We consider the sysadmin domain with $n = 2 - 10$ state variables (2^n states) and 10 actions. We evaluate our approach with $s = 1, 2, 3$ and 4, where s is the maximum number of state variables in the scope of any basis.

As expected, the runtime of the exact MDPI is dominated by our approach for sufficiently many state variables (Figure 1(a)); more significantly, the exact approach runs out of memory for larger problem sizes.

From Figure 1(b) we can see that while the utility of approximate interdiction improves significantly as s increases from 1 to 2, it already becomes close to optimal when $s = 2$, with little added value from increasing it further. The results are similar for other IPC domains. Consequently, our experiments below use $s = 2$.

7.2 SCALABILITY

We evaluate the constraint generation approach on larger problem sizes on the sysadmin domain (up to 60 state variables and 60 actions). Even with constraint generation with only a subset of basis functions, our baseline algorithm (marked as "slow bilevel") scales poorly for $n > 30$. On the other hand, the use of fast constraint generation (Algorithm 2, marked as "fast bilevel"), significantly improves scalability (Figure 2 left). Indeed, the baseline (slow bilevel) becomes intractable for $n \geq 50$, whereas we can successfully solve these with the "fast" approach. Since we compute the utility of the final attacker policy using basis generation, the solution accuracy is not compromised (Figure 2 right).

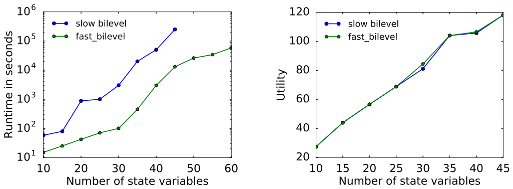

Figure 2: Comparison between baseline (slow) and fast interdiction on the sysadmin domain in terms of runtime (left) and utility (right).

In the second set of scalability experiments, we evaluate our approaches on 10 problem instances of the academic advising domain. The problem size increases with problem number from 10 to 30 courses (20 to 60 state variables and 10 to 30 actions). For each problem size, there are two instances, corresponding to different program requirements and course prerequisites. The first (odd numbered) problem instance is somewhat simpler (fewer prerequisites per course). The second (even numbered) instance is more complicated, with a larger number of prerequisites per course (larger number of connections in the underlying DBN). Problem 10 has the largest problem size with 30 courses, 11 program requirements, 3 prerequisites for most courses and 4 prerequisites for 8 courses. As demonstrated in Figure 3, we observe a similar trend as before: the fast constraint generation approach significantly outperforms baseline without compromising much solution quality. The baseline is intractable for problems 7 to 10 ($n \geq 50$).

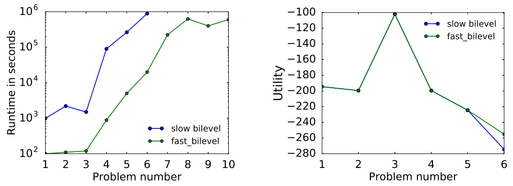

Figure 3: Comparison between baseline (slow) and fast interdiction on the academic advising domain in terms of runtime (left) and utility (right).

In the third set of experiments, we evaluate on 6 problem instances of the wildfire domain. The grid size increases with problem number from $m = 3$ to 5 ($n = 2 \times m^2 =$ 18 to 50 state variables, and 36 to 100 actions). For each grid size, there are two instances, corresponding to different neighbourhood configurations and targets (cells on the grid that need to be protected). The first (odd numbered) problem instance has fewer targets than the second (even numbered) instance. The results are shown in Figure 4. The baseline is again intractable on problems 5 and 6 ($n = 50$) which can be solved by fast bilevel.

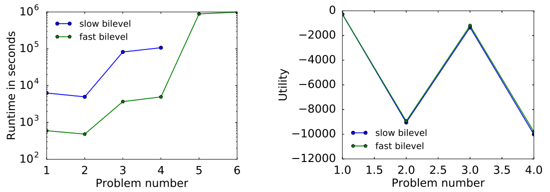

Figure 4: Comparison between baseline (slow) and fast interdiction on the wildfire domain in terms of runtime (left) and utility (right).

7.3 EFFECTIVENESS OF GREEDY INTERDICTION

Finally, we compare the greedy interdiction algorithm to fast constraint generation. As shown in Figures 5-7, the greedy algorithm is faster for larger problem sizes, saving up to an order of magnitude of computation time, without significantly compromising solution quality.

8 CONCLUSION

We presented a MILP approach for factored MDP interdiction, using a parity basis for linear value function approximation over binary state variables. We offered an iterative basis generation approach to select the most effective set of basis functions, and presented several variations of constraint generation, combined with basis selection, to solve the MILP. We evaluated our approaches on several realistic problem instances and demonstrated significantly increased scalability while achieving near-optimal solutions. Finally, we proposed a greedy algorithm for MDP interdiction and showed that it can further improve scalability.

In this paper, we only model deterministic mitigation strategies. Related research on Stackelberg games for security often considers randomized defensive resource allocation, which in our case would translate to randomized mitigations that can yield considerably higher utility to the defender. Within our framework, such an extension is quite non-trivial, and remains an important question for future research.

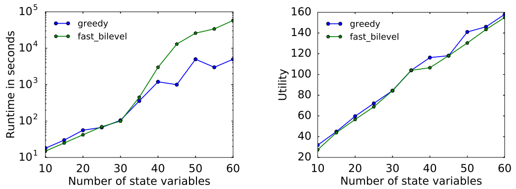

Figure 5: Comparison between fast interdiction and greedy in terms of runtime (left) and utility (right) on the sysadmin domain.

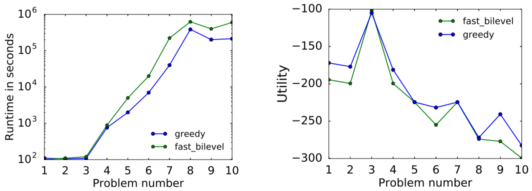

Figure 6: Comparison between fast interdiction and greedy in terms of runtime (left) and utility (right) on the academic advising domain.

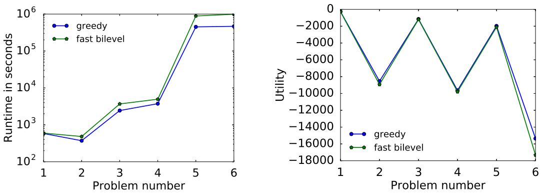

Figure 7: Comparison between fast interdiction and greedy in terms of runtime (left) and utility (right) on the wildfire domain.

Acknowledgements This research was partially supported by NSF (CNS-1640624, IIS-1526860, IIS-1649972, CNS-1238959), ONR (N00014-15-1-2621), ARO (W911NF-16-1-0069), NIH (UH2 CA203708-01, R01HG006844), and AFRL (FA 8750-14-2-0180).

References

[1] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE, 1995.

[2] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11(1):94, 1999.

[3] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial intelligence*, 121(1):49–107, 2000.

[4] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 13:227–303, 2000.

[5] Carlos Guestrin, Daphne Koller, and Ronald Parr. Max-norm projections for factored mdps. In *IJCAI*, volume 1, pages 673–682, 2001.

[6] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

[7] Carlos Ernesto Guestrin. *Planning under uncertainty in complex structured environments*. PhD thesis, Stanford University, 2003.

[8] Manish Jain, James Pita, Milind Tambe, Fernando Ordónez, Praveen Paruchuri, and Sarit Kraus. Bayesian stackelberg games and their application for security at los angeles international airport. *ACM SIGecom Exchanges*, 7(2):10, 2008.

[9] Daphne Koller and Ronald Parr. Computing factored value functions for policies in structured mdps. In *IJCAI*, volume 99, pages 1332–1339, 1999.

[10] Daphne Koller and Ronald Parr. Policy iteration for factored mdps. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 326–334. Morgan Kaufmann Publishers Inc., 2000.

[11] Dmytro Korzhyk, Zhengyu Yin, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. Stackelberg vs. nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. *J. Artif. Intell. Res.(JAIR)*, 41:297–327, 2011.

[12] Joshua Letchford and Yevgeniy Vorobeychik. Optimal interdiction of attack plans. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 199–206. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[13] Ryan O'Donnell. Some topics in analysis of boolean functions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 569–578. ACM, 2008.

[14] Praveen Paruchuri, Jonathan P Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, and Sarit Kraus. Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 895–902. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[15] Martin L Puterman. Markov decision processes: Discrete stochastic dynamic programming. 1994.

[16] Robert St-Aubin, Jesse Hoey, and Craig Boutilier. Apricodd: Approximate policy construction using decision diagrams. In *NIPS*, pages 1089–1095, 2000.