Solving Zero-Sum Security Games in Discretized Spatio-Temporal Domains

APPENDIX

LP Formulation for Constant Number of Resources (Fang et al. 2013)

For the sake of completeness, we describe the LP formulation in (Fang, Jiang, and Tambe 2013) as follows.

Let variable $p(i_1, j_1, i_2, j_2, ..., i_K, j_K; n)$ be the probability that resource k moves from position i_k at time n to position j_k at time $n+1$. Then, the following LP solves the problem (Fang, Jiang, and Tambe 2013).

$$
\begin{aligned}
& \min u \\
s.t \quad & f(j_1, ..., j_K; n) = \sum_{i_1,...,i_K=1}^{M} p(i_1, j_1, ..., i_K, j_K; n-1) \\
& f(i_1, ..., i_K; n) = \sum_{j_1,...,j_K=1}^{M} p(i_1, j_1, ..., i_K, j_K; n) \\
& p(i_1, j_1, ..., i_K, j_K; n) = 0, \text{if } \exists k\, s.t.\, |i_k - j_k| > \Delta \\
& \sum_{i_1,...,i_K=1}^{M} f(i_1, ..., i_K; n) = 1 \\
& u \geq (1 - \sum_{(i_1,...,i_K) \text{ protecting } t \text{ at } n} f(i_1, ..., i_K; n)) w_{rn} \\
& p(i_1, j_1, ..., i_K, j_K; n) \in [0, 1]
\end{aligned}
$$

where "$\sum_{(i_1,...,i_K) \text{ protecting } t \text{ at } n}$" means summing over all the profiles $(i_1, ..., i_K)$ in which there exists at least an i_k satisfying that target t is within the protection range of position i_k at n.

The size of the above LP formulation has order $O(NM^{2K})$, which is polynomial in M and N but exponential in K.

Proof of Lemma 1

Lemma Statement: Separation oracles for $\mathcal{P}_w$ and $\mathcal{P}_g$ reduce to each other in *poly(T,N)* time.

Proof. $\mathcal{P}_w \Rightarrow \mathcal{P}_g$: given a separation oracle $\mathcal{O}_w$ for $\mathcal{P}_w$, one can construct a separation oracle for $\mathcal{P}_g$ by simply checking the TN extra constraints $(1 - x_{tn})w_{tn} \leq u$.

$\mathcal{P}_g \Rightarrow \mathcal{P}_w$: let $u_0 = \max_{t,n} w_{tn}$. For any $x_0 \in \mathbb{R}^{TN}$, $x_0 \in \mathcal{P}_w$ if and only if $(x_0, u_0) \in \mathcal{P}_g$. Furthermore, if $x_0 \notin P_w$, any hyperplane $a^T x + bu = c$ separating (x_0, u_0) from $\mathcal{P}_g$ gives a hyperplane $a^T x = c - bu_0$ separating x_0 from $\mathcal{P}_w$. □

Proof of Lemma 2

Lemma Statement: the separation oracle problem for $\mathcal{P}_w$ reduces to the following LP (LP_w) in polynomial time:

$$
\begin{aligned}
& \max \sum_{t\in[T],n\in[N]} w_{tn} x_{tn} \\
s.t. \quad & x \in \mathcal{P}_w
\end{aligned}
$$

for arbitrary weight profile $\{w_{tn}\}$.

As a result, LP_g reduces to LP_w in polynomial time.

Proof. we're following an argument from (Grötschel, Lovász, and Schrijver 1984); proof is provided for completeness.

Assume we are given an oracle $\mathcal{O}$ that computes LP_w. Consider another polyhedron $\mathcal{P}_w^\circ = \{y : y^T x \leq 1, \forall x \in \mathcal{P}_w\}$. For any y, we can decide whether $y \in \mathcal{P}_w^\circ$ by solving an instance of LP_w: $\max y^T x$, s.t. $x \in \mathcal{P}_w$. This can be computed by oracle $\mathcal{O}$ and output an optimal solution x^*. If $y^T x^* \leq 1$, then $y \in \mathcal{P}_w^\circ$, otherwise $y^T x^* = 1$ is a hyperplane separating y from $\mathcal{P}_w^\circ$.

Till now, we have constructed a separation oracle for $\mathcal{P}_w^\circ$ using $\mathcal{O}$. Therefore, we can maximize any linear function over $\mathcal{P}_w^\circ$ in polynomial time, again by ellipsoid method. Then, similar arguments as above implies that we can construct a separation oracle for $\mathcal{P}_w^{\circ\circ} = \{x : y^T x \leq 1, \forall y \in \mathcal{P}_w^\circ\} = \mathcal{P}_w$. This proves that the separation oracle problem for $\mathcal{P}_w$ reduces to LP_w.

Then we have, LP_g reduces by ellipsoid method to the separation oracle problem for $\mathcal{P}_g$, which again reduces to the separation oracle problem for $\mathcal{P}_w$ (by Lemma 1), which then reduces to LP_w. □

Proof of Lemma 3

Lemma Statement: If $K \leq M$, there exists an optimal vertex solution for LP_w corresponding to a defender pure strategy, say $\{v_k\}_{k\in[K]}$, satisfying that v_k is *strictly* under v_{k-1} for all k.

Proof. If $\Delta = 0$, i.e., each resource has to stay at the same position, this is trivial. We prove it for the case $\Delta \geq 1$.

Assume $\{v'_k\}_{k\in[K]}$ is any optimal pure patrol strategy with $v'_k = \{v_k^{(0)}, v_k^{(1)}, ..., v_k^{(N)}\}$. We first argue that there is a way to construct another K paths $\{v_k\}_{k\in[K]}$, such that v_k is *weakly* under v_{k-1} for all k.

We claim

$$v_k = \{\max_i k\{m_0^{(i)}\}, \max_i k\{m_1^{(i)}\}, ..., \max_i k\{m_N^{(i)}\}\}, \forall k \in [K]$$

are also another K feasible patrol paths. Here "$\max k$" is the k'th largest value among $\{m_n^{(k)}\}_{k\in[K]}$. Obviously, if this is true, these paths would satisfy optimality, and v_k is *weakly* under v_{k-1}.

Let m_n^k denote $\max k\{m_n^{(i)}\}$. With slight abuse of notation, we also say $m_n^k \in v_i$ if the path v_i passes through position m_n^k. We show the move from m_n^k to m_{n+1}^k is feasible for any $n \in [N]$ and $k \in [K]$. If $\exists i$ such that m_n^k and m_{n+1}^k are both in v'_i, then this is trivial.

Otherwise, let $m_n^k = m_n^{(i)} \in v'_i$. If $m_{n+1}^{(i)} > m_{n+1}^k$, since m_n^k is the k'th largest due to the construction, there are at least $k-1$ paths *weakly* above position m_n^k excluding v'_i, however there are at most $k-2$ paths *strictly* above position m_{n+1}^k excluding v'_i because v'_i is one of the those paths *strictly* above m_{n+1}^k ($m_{n+1}^{(i)} > m_{n+1}^k$). Therefore, there exists a v'_j satisfying that it is *weakly* above m_n^k but *weakly* under m_{n+1}^k. So we have $m_n^{(j)} \geq m_n^k$ but $m_{n+1}^{(j)} \leq m_{n+1}^k$. This yields that it is feasible to go from m_n^k to $m_n^k + (m_{n+1}^{(j)} - m_n^{(j)})$. Since $m_n^k + (m_{n+1}^{(j)} - m_n^{(j)}) \leq m_n^k + (m_{n+1}^k - m_n^k)) = m_{n+1}^k < m_{n+1}^{(i)}$ and the move m_n^k to $m_{n+1}^{(i)}$ is feasible, the move m_n^k to m_{n+1}^k is also feasible.

A similar argument applies to the case $m_{n+1}^{(i)} < m_{n+1}^k$. This shows that there exists an optimal set of K feasible patrol paths $v_1, ..., v_K$ such that v_{k+1} is *weakly* under v_k for any k.

Now, we show that there is a way to adjust $v_1, ..., v_K$ such that they do not overlap and meanwhile maintain optimality. We first adjust the time layer 0. Let m_k^0 in path v_k and m_{k+1}^0 in path v_{k+1} be the first overlap. We aim to adjust these paths to make $v_1, ..., v_{k+1}$ not overlap at time 0. If $v_k = v_{k+1}$, then we simply remove path v_{k+1} without any loss and start a new path v_{k+1} from an arbitrary uncovered position at time 0 (this position exists because $K \leq M$). Otherwise, let $n(\geq 1)$ be the first time layer at which v_k and v_{k+1} do not overlap, thus $m_k^n > m_{k+1}^n$. So either of the following adjustments would be feasible and maintain optimality.

1. Push up v_k: set $m_k^{n'} \leftarrow m_k^{n'} + 1, \forall n' < n$; keep $m_k^{n'}$ the same if $m_k^{n'} = M$; if any path v_i satisfying $i < k$ overlaps with v_k at $m_k^{n'}$, also push v_i up at time n' by setting $m_i^{n'} \leftarrow m_i^{n'} + 1$.
2. Push down v_{k+1}: set $m_{k+1}^{n'} \leftarrow m_{k+1}^{n'} - 1, \forall n' < n$; keep $m_{k+1}^{n'}$ the same if $m_{k+1}^{n'} = 0$; if any path v_i satisfying $i > k+1$ overlaps with v_{k+1} at $m_{k+1}^{n'}$, also push v_i down at time n' by setting $m_i^{n'} \leftarrow m_i^{n'} - 1$.

Note that both of these two adjustments will keep the paths weakly separate. Since $v_1, ..., v_k$ do not overlap at time 0 due to our choice of k, if $m_k^0 = M-k+1$ which means there are no uncovered positions above v_k at time 0, then we take the second adjustment (if more than two paths overlap at m_k^0 in this case, push down all the paths, except v_k, by 1 grid as the second adjustment) and make the paths $v_1, ..., v_k, v_{k+1}$ not overlap at time 0. Otherwise, there is at least one uncovered position above m_k^0, then we take the first adjustment (always choose v_k in this case if more than two paths overlap, since we only care about v_k and v_{k+1} currently). If $m_k^0 + 1$ is uncovered originally, we have made the paths $v_1, ..., v_{k+1}$ not overlap at time 0. Otherwise $m_k^0 + 1 = m_{k-1}^0$ and we removed the overlap at m_k^0, but create a new overlap at m_{k-1}^0. Nevertheless, we know that there is an uncovered position above m_k^0 and we made the overlapping position (i.e., m_k^0+1 now) one step closer to this uncovered position. We then continue this adjustment until we get to that uncovered position and make the paths $v_1, ..., v_{k+1}$ not overlap at time 0.

By now, we went one step further and made $v_1, ..., v_{k+1}$ not overlap at time 0. We can continue this argument until finally making all the paths not overlap at time 0. We now consider time layer 1. Similarly, let m_k^1 and m_{k+1}^1 be the first overlap and $n(\geq 2)$ be the first time layer at which v_k and v_{k+1} do not overlap (let $n = N+1$ if v_k always overlaps v_{k+1} except at time 0). Since $m_k^0 > m_{k+1}^0$ and $m_k^n > m_{k+1}^n$ (if $n \neq N+1$), the following two adjustments are feasible and maintain optimality: set $m_k^{n'} \leftarrow m_k^{n'} + 1$ (keep $m_k^{n'}$ the same if $m_k^{n'} = M$) or $m_{k+1}^{n'} \leftarrow m_{k+1}^{n'} - 1$ (keep $m_{k+1}^{n'}$ the same if $m_{k+1}^{n'} = 0$), $\forall 1 \leq n' < n$. Then similarly to time 0, we can make $v_1, ..., v_{k+1}$ not overlap at time 1. Continuing this adjustment procedure until time N, we get K paths where for each pair of path, one is strictly under another one. □

Proof of Lemma 4

Lemma Statement: If N is a constant, Algorithm 1 runs in polynomial time and outputs an optimal vertex solution for LP_w, for any weight profile $\{w_{tn}\}$.

Proof. Lemma 3 guarantees there is always an optimal pure strategy in which paths do not cross or touch. Algorithm 1 computes such an "ordered" optimal pure strategy using dynamic programming. This algorithm is polynomial-time because N is a constant, therefore the number of states $OPT(v; k)$ is *poly(M,K)*. □

Proof of Theorem 2

Theorem Statement: If the protection ranges at different grid points do not overlap with each other, then an optimal vertex solution to LP_w can be found in polynomial time.

Proof. We only need to focus on the pure strategies in which paths do not overlap with each other. Considering the orig-

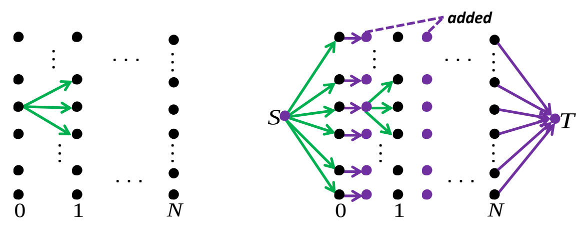


Figure 1: Left: Original Grid; Right: Constructed Network.

inal grid in Figure 1, we set the reward of each grid point as the sum of all the weights of targets within its protection range, then construct a weighted directed network as the right panel in Figure 1, in which weights indicate rewards obtained by passing through edges.

The network is constructed based on the original grid as follows: adding a source S and a sink T; $\forall n < N$, adding one more layer between time n and $n+1$ (blue points); all the feasible-move arrows become edges starting from each added layer to the next layer; one-to-one edges (purple in Figure 1) are added from each original layer to the corresponding added layer. In addition, all the edges have capacity 1 and only blue edges have non-zero weights (rewards), which equal the rewards of their starting points in the original grid.

Consider the problem of letting K units flow (possibly fractionally) from source S to sink T such that the sum of the rewards weighted weighted by the flow amount through the corresponding edge is maximized. The following facts hold for this problem (similar to the min-cost flow problem): 1.) its optimal solution can be computed in polynomial-time because it has an LP formulation ; 2.) it has an integer optimal solution because all capacities are integers; 3.) this integer optimal solution can be computed in polynomial-time.

It is straightforward to check that the above max-reward flow problem is equivalent to finding optimal non-overlapped pure strategy in original grid in the following sense: an integer flow in constructed network can be converted to an optimal non-overlapping pure strategy in the original grid (by contracting the blue edges) and vice versa, and the sums of the collected rewards in both cases are equal. □

Proof of Lemma 5

Lemma Statement: $OPT(PC_K) \geq \frac{K}{K_0}$ if $OPT(DC) = K_0$; and $OPT(DC) \leq \frac{K}{p}$ if $OPT(PC_K) = p$.

Proof. First Part: By sampling a combination of K paths from $[K_0]$ uniformly at random, we get a solution for PC covering each target with probability at least K/K_0.

Second Part: assume p is optimal for PC and $MS = \begin{bmatrix} u_{1,1} & ... & u_{1,K} \\ ... & ... & ... \\ u_{R,1} & ... & u_{R,K} \end{bmatrix}$ is the support of a mixed strategy achieving the optimality. Here each row r of MS corresponds to a pure strategy drawn with probability p_r and $\sum_{r=1}^{R} p_r = 1$. Assume Algorithm 2 outputs K_0 and paths $\{v_k\}_{k\in[K_0]}$. we show that $K_0 \leq \frac{K}{p}$.

We look at the mixed strategy MS from another perspective, namely, as a set of grid points assigned with probability weights, which we call *p-weights* from now on. For example, any position in path $u_{1,k}$ is a grid point assigned with p-weight p_1. Let $\mathcal{X}$ denote the set of all the grid points showing in MS and p_X denote the p-weight of grid point X. Each path $u_{r,k}$ has N grid points, so the sum of the p-weight over $\mathcal{X}$ is $\sum_{X\in\mathcal{X}} p_X = \sum_{r=1}^{R} p_r \times K \times N = KN$.

Now we count this sum of p-weights from another way. Let $\mathcal{X}_{kn}$ denote the set of all the grid points at time n that lie weakly above path v_k^n output by Algorithm 2. We show the following inequality: $\sum_{X\in\mathcal{X}_{kn}} p_X \geq pk, \forall k, n$. If this is true, by letting $k = K_0$ we have $KN = \sum_{X\in\mathcal{X}} p_X \geq \sum_{n=1}^{N}\sum_{X\in\mathcal{X}_{K_0 n}} p_X \geq pNK_0$ yielding $K_0 \leq \frac{K}{p}$.

We prove $\sum_{X\in\mathcal{X}_{kn}} p_X \geq pk$ by induction on k. When $k = 0$, this is trivial. Now, assume the inequality holds for any $k \leq r-1$, then we consider the case of $k = r$. We first sort all the positions of path v_k in a decreasing order (ties are broken randomly). Denote the index of the ordered time layers as n_j, i.e., $v_r^{n_j} \geq v_r^{n_{j+1}}$ for all j. We then prove $\sum_{X\in\mathcal{X}_{rn_j}} p_X \geq pr$ by induction on j.

When $j = 1$, this is the highest position in path v_r. So there exists a target pair, say (t, n_1), that is not protected by path $v_1, ..., v_{r-1}$, and $v_r^{n_1}$ is the lowest possible position that still covers (t, n_1), otherwise, v_r is not time-wise lowest because it can move $v_r^{n_1}$ to $v_r^{n_1} - 1$. As a result, any grid point in $\mathcal{X}_{r-1,n_1}$ does not protect the pair (t, n_1). So all the grid points that protect (t, n) are weakly above $v_r^{n_1}$, but are not in $\mathcal{X}_{r-1,n_1}$, i.e., they are in $\mathcal{X}_{r,n_1} \setminus \mathcal{X}_{r-1,n_1}$. The sum of the p-weights of these grid points is at least p, since (t, n_1) is covered by at least probability p. So we have $\sum_{X\in\mathcal{X}_{r,n_1}} p_X \geq \sum_{X\in\mathcal{X}_{r-1,n_1}} p_X + \sum_{X\in\mathcal{X}_{r,n_1}\setminus\mathcal{X}_{r-1,n_1}} p_X \geq (r-1)p + p = pr$ due to induction hypothesis.

Now assume the inequality holds for $j \leq i-1$, we consider the case $j = i$. Since v_r is time-wise lowest, the reason that v_r goes through $v_r^{n_i}$ can only be one of the following two: i.) there exists a pair (t, n_i) that is not covered by $\mathcal{X}_{r-1,n_i}$ and $v_r^{n_i}$ is the lowest possible position that still covers (t, n_i); ii.) the move from $v_r^{n_i} - 1$ to a position at a neighboring time layer, i.e., $v_r^{n_i+1}$ or $v_r^{n_i-1}$, is infeasible. In the first case, we have $\sum_{X\in\mathcal{X}_{r,n_i}} p_X \geq pr$ with a similar argument as $i = 1$. In the second case, w.l.o.g., say the move from $v_r^{n_i-1}$ to $v_r^{n_i}-1$ is not feasible. Then, $v_r^{n_i-1} - v_r^{n_i} = \Delta$, so all the paths going through points in $\mathcal{X}_{r,n_i-1}$ must arrive at a points weakly above $v_r^{n_i}$, i.e., points in $\mathcal{X}_{r,n_i}$. So we have $\sum_{X\in\mathcal{X}_{r,n_i}} p_X \geq \sum_{X\in\mathcal{X}_{r,n_i-1}} p_X \geq pr$, where the second "$\geq$" is due to the induction hypothesis on i since $n_i - 1$ ranks before n_i in the order. This finishes our proof. □

Proof of Theorem 3

Theorem Statement: $OPT(PC_K) = K/OPT(DC)$. Furthermore, the optimal solution of PC_K can be generated from that of DC efficiently.

Proof. Lemma 5 yields $OPT(PC_K)OPT(DC) \geq K$ (by first part) and $OPT(PC_K)OPT(DC) \leq K$ (by second part). So $OPT(PC_K) = K/OPT(DC)$.

Next, we prove that a solution to PC can be obtained from a solution to DC as follows. Given the optimal path set $[K_0]$ for DC, we can sample a combination of K paths (i.e., a pure strategy for the defender) from $[K_0]$ uniformly at random. This can be easily done in $poly(K_0)$ time. Any target is covered by a resource with probability $\mathcal{C}_{K_0-1}^{K-1} \times \frac{1}{\mathcal{C}_{K_0}^{K}} = \frac{K}{K_0}$ where $C_{K_0}^{K}$ means K_0 choose K. $\square$

Proof of Theorem 4

Theorem Statement: The K_0 output by Algorithm 2 is optimal.

Proof. Let T_k denote the set of all targets protected by resource k and $T_{-k} = \bar{T}_k$, i.e., the complement of T_k, denote all the targets not protected by k. It is easy to see that if K_0 is optimal (minimum) for protecting all targets, $k_0 - 1$ should be optimal for protecting $T_{-k}, \forall k \in [K_0]$.

We next show the key property of Algorithm 2: there exists an optimal solution for DC in which one resource follows precisely the same path as v_1 constructed in Algorithm 2. The theorem then follows by induction.

We prove this by contradiction. W.l.o.g. (Lemma 3), let $\{u_1, u_2, ..., u_{K_0'}\}$ be a set of optimal paths satisfying that u_i is strictly above u_{i+1}, where $u_i = (u_i^0, ..., u_i^N)$. We show that v_1 constructed in Algorithm 2 is able to cover all the targets covered by u_1, and therefore $\{v_1, u_2, ..., u_{K_0'}\}$ is a new optimal solution, in which one resource follows v_1.

Assume, for the sake of proof by contradiction, that this is false. Then, there exist targets covered by u_1 but not by v_1. The uncovered targets must be under the protection range of v_1 because v_1 does not leave any pair (t, n) above its protection range uncovered by construction. So there exists i such that $u_1^i < v_1^i$. Note that u_1 does not leave any pair (t, n) above its protection range uncovered, either, because $\{u_1, u_2, ..., u_{K_0'}\}$ is a cover and u_1 is the highest path. However, the conclusion $u_1^i < v_1^i$ means v_1 is not the time-wise lowest path that does not leave any pair (t, n) above its protection range uncovered, which contradicts the construction in Algorithm 2. $\square$

Proof of Lemma 6

Lemma Statement LP_w is NP-hard, if $A_n \leq 2(\Delta - 1)$ for any n.

Proof. We prove this by reducing from vertex cover.

Construct the following special case: set $\Delta = 3$, set patrol radius $r = \frac{2}{3}d_\Delta$, i.e., the protection range is $(m - \frac{2}{3}d_\Delta, m + \frac{2}{3}d_\Delta)$ at position m. Here d_Δ is the distance between two neightboring spatial points. Therefore, the position $m + \frac{1}{2}d_\Delta$ can be covered by a resource either at m or at $m + 1$, but the position m can only be covered by a resource at m.

First, the following vertex cover problem (VC) is known to be NP-hard: given integer K and any graph $G = (V, E)$, finding a subset $V_0 \subseteq V$ that maximizes the number of edges it covers subject to $|V_0| = K$. We reduce VC to LP_w.

Given K and any graph instance $G = (V, E)$, we construct an LP_w instance as follows. Create $|V|$ targets, each one corresponding to a node in V. The moving paths for these $|V|$ targets are constructed as path 1,2,3, etc. in the left of Figure 2. Generally, two neighboring paths differ by a horizontal translation of 4 time layers. These paths are constructed in such way that each pair of targets cross each other exactly once and all the crossings have the same local geometry as shown in the right of Figure 2. Any pair of targets does not occupy the same discretized position at any time. We set the weight of each target to be 1 at any time layer.

Now, for any edge in E, we add a "*tiny*" target with small enough weight. Specifically, $\forall e = (u, v) \in E$, let n_e denote the discretized time when path u and v are closest, and u_{n_e} and v_{n_e} denote the position of path u and v at time n_e.[1] We put a tiny target with weight ϵ at the time n_e on the point in the middle between u_{n_e} and v_{n_e} (see the position of ϵ in Figure 2). Note that a resource can capture this ϵ weight at either u_{n_e} or v_{n_e}. One may think of this as a tiny target e that has weight ϵ only at this specific time and has weight 0 otherwise. We set ϵ small enough to satisfy $\epsilon < \frac{1}{|E|}$, so that sum of all the tiny weights is still less than 1, i.e., $|E|\epsilon < 1$. This completes the construction of the LP_w instance.

Our first observation is that any optimal vertex solution to LP_w must first optimally capture the weights from the target set V, because $|E|\epsilon < 1$, meaning that the loss of failing to cover any target corresponding to a node in V cannot be recovered even by covering all the ϵ tiny targets.

Now we show the *only* way to collect weights optimally from those $|V|$ targets is to pick any K targets and follow their paths precisely. The reason is that K resources can capture at most K targets at any time layer because of limited protection radius and non-overlapping paths. So capturing a sum of weights of KN is the best one can do and following any K targets achieves this. The reason that these are the only optimal strategies is that the only time when a resource can switch to another path is when the current path is about to intersect that path, however the switch at that time is infeasible because it needs an acceleration of $2\Delta - 1$, greater than the limit $2(\Delta - 1)$.

As a result, maximizing the collected weights is equivalent to choosing a target set $[K] \subseteq V$ to follow, such that the number of covered tiny targets is maximized. Note that a tiny target $e = (u, v)$ can be covered if and only if one of its end points corresponds to a path in $[K]$. So, an optimal vertex solution (i.e., optimal pure strategy) to LP_w gives an optimal solution to the original VC problem instance. Since $G = (V, E)$ is arbitrarily chosen, LP_w is NP-hard. $\square$

Column Generation

Column generation has been used to solve large linear programs as well as large scale games. Specifically in our problem, we start from an empty support set of defender pure

[1]In a slight abuse of notation, here we use u, v to denote both nodes in G and paths corresponding to these nodes.

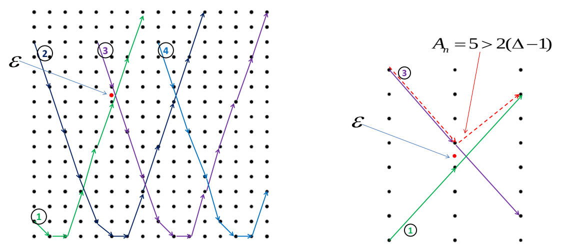


Figure 2: Left: enumerated target paths; example position to put an ϵ tiny targets; Right: local intersection geometry, tiny target position and an infeasible acceleration.

strategies, use LP_w as an oracle to iteratively compute the defender's best response for any attacker mixed strategy and add it to the support set. In each iteration, we first solve for an optimal mixed strategy of the attacker under the assumption that the defender is only allowed to select a pure strategy from a restricted set Γ. Then, we use our algorithms for the weight collection problem to compute an optimal deterministic defender strategy with respect to the attacker's mixed strategy. The optimal pure defender strategy is added to Γ and the whole algorithm ends when the strategy to be added is already in the set. Although the size of the Γ may be exponential in worst case, this method is empirically efficient and can converge or get to near-optimal solutions by enumerating only a small fraction of all pure strategies (Jain et al. 2011; Halvorson, Conitzer, and Parr 2009).

Experiment Results with Small Scale Randomly Generated Instances

We test the algorithms proposed in this paper with small scale randomly generated settings. The results are similar to that with practical settings in the ferry domain. Figure 3(a) shows the performance of the baseline strategy (LP) as the number of resources (K) increases. The x-axis shows the number of resources. LP is ensured to obtain the optimal solution (AttEU Ratio equals to 1); however, the runtime increases exponentially when K increases. When $K >= 4$, LP runs out of memory and fails to return a solution. So LP is only applicable when K is small.

Figure 3(b) shows that DP always achieves the optimal solution. When the number of time steps (M) is small enough ($N <= 2$), DP runs much faster than LP. However, the runtime of DP increases exponentially as N increases and it can be even slower than the baseline algorithm. So DP is suitable for cases with small N.

Figure 3(c) shows that NonOverlap achieves the optimal solution when the protection radius is small enough to make the protection range non-overlapped ($r <= 0.083$). As the protection radius increases, the AttEU Ratio increases, indicating a decreasing solution quality. However, when the protection radius is close to $d_\Delta/2$ (e.g., $r = 0.1$ and $r = 0.13$), the AttEU Ratio of NonOverlap is close to 1, meaning the solution quality is close to optimal. Given that NonOverlap outperforms the baseline in runtime significantly, we conclude that NonOverlap should be chosen when the protection range does not overlap and is a good approximation of optimal solution when protection radius is close to $d_\Delta/2$.

Figure 3(d) shows the performance of Hom as the utility range increases. Utility range is defined as the difference between the maximum and minimum utility of the targets. When utility range equals zero, all targets are homogeneous. From the figure, we know Hom obtains an optimal solution when utility range is zero. As the utility range increases, the solution quality of Hom degrades compared to the baseline. When utility range is small, the solution quality is close to optimal, which means Hom provides a good approximation of the optimal solution. Furthermore, its runtime is orders of magnitude shorter than the baseline.

The heuristic algorithm OrderGreedy achieves an optimal or near-optimal solution in most of the small scale cases. OrderGreedy also outperforms LP and DP significantly in runtime, which indicates it to be a good heuristic algorithm in many different settings.

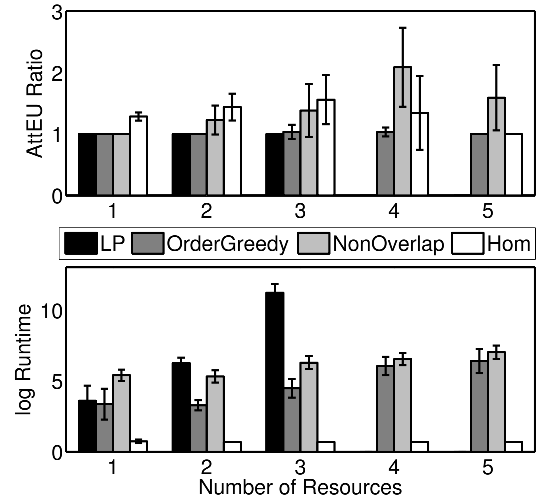
References

Fang, F.; Jiang, A. X.; and Tambe, M. 2013. Optimal patrol strategy for protecting moving targets with multiple mobile resources. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

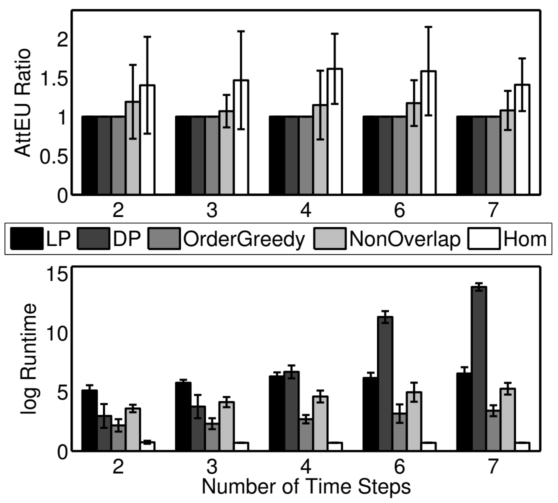
Grötschel, M.; Lovász, L.; and Schrijver, A. 1984. "the ellipsoid method and its consequences in combinatorial optimization". *Combinatorica* 4(4):291–295.

Halvorson, E.; Conitzer, V.; and Parr, R. 2009. Multi-step multi-sensor hider-seeker games. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, 159–166.

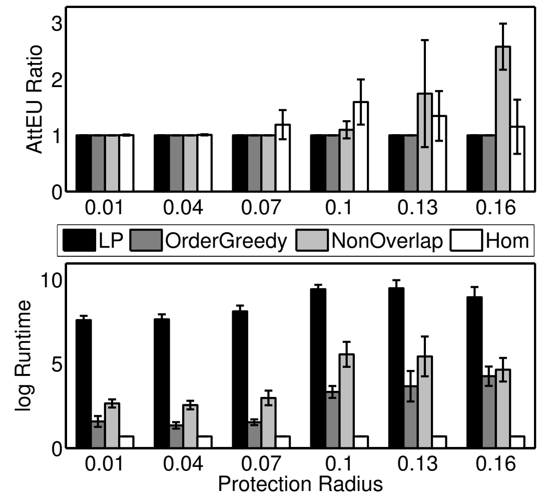
Jain, M.; Korzhyk, D.; Vaněk, O.; Conitzer, V.; Pěchouček, M.; and Tambe, M. 2011. A double oracle algorithm for zero-sum security games on graphs. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '11, 327–334.



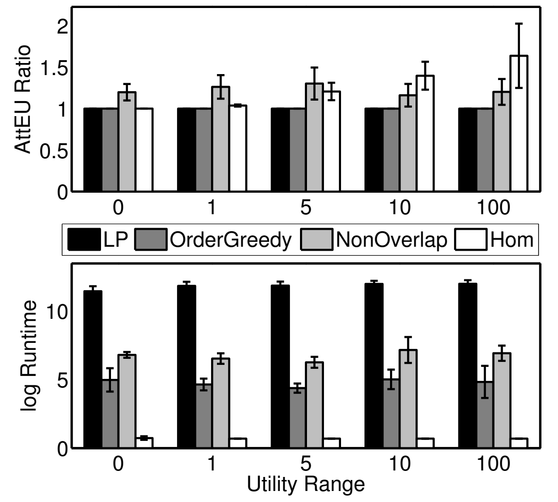
(a) Increase K



(b) Increase N



(c) Increase Re



(d) Increase Range

Figure 3: Experimental Results For Randomized Settings