The Computer Journal Advance Access published September 20, 2013

doi:10.1093/comjnl/bxt094

Game-Theoretic Target Selection in Contagion-Based Domains

JASON TSAI*, THANH H. NGUYEN, NICHOLAS WELLER AND MILIND TAMBE

University of Southern California, Los Angeles, CA 90089, USA
**Corresponding author: jasontts@usc.edu*

Many strategic actions carry a 'contagious' component beyond the immediate locale of the effort itself. Viral marketing and peacekeeping operations have both been observed to have a spreading effect. In this work, we use counterinsurgency as our illustrative domain. Defined as the effort to block the spread of support for an insurgency, such operations lack the manpower to defend the entire population and must focus on the opinions of a subset of local leaders. As past researchers of security resource allocation have done, we propose using game theory to develop such policies and model the interconnected network of leaders as a graph. Unlike this past work in security games, actions in these domains possess a probabilistic, non-local impact. To address this new class of security games, we combine recent research in influence-blocking maximization with a double oracle approach and create novel heuristic oracles to generate mixed strategies for a real-world leadership network from Afghanistan, synthetic leadership networks and scale-free graphs. We find that leadership networks that exhibit highly interconnected clusters can be solved equally well by our heuristic methods, but our more sophisticated heuristics outperform simpler ones in less interconnected scale-free graphs.

Keywords: game theory; social networks; contagion

Received 16 November 2012; revised 27 July 2013
Handling editor: Juan Antonio Rodriguez-Aguilar

1. INTRODUCTION

Many adversarial domains exhibit 'contagious' actions for each player. For example, word-of-mouth advertising/viral marketing has been widely studied by marketers trying to understand why one product or video goes 'viral' while others go unnoticed [1].

Counterinsurgency (COIN) is the contest for the support of the local leaders in an armed conflict and can include a variety of operations such as providing security and giving medical supplies [2]. Just as in word-of-mouth advertising and peacekeeping operations, these efforts carry a social effect beyond the action taken that can cause advantageous ripples through the neighboring population [3]. Moreover, multiple intelligent parties attempt to leverage the same social network to spread their message, necessitating an adversary-aware approach to strategy generation.

We use a game-theoretic approach to the problem and develop algorithms to generate resource allocations strategies for such large-scale, real-world networks. We model the interaction as a graph with one player attempting to spread influence while the other player attempts to stop the probabilistic propagation of that influence by spreading their own influence. This 'blocking' problem models situations faced by governments/peacekeepers combatting the spread of terrorist radicalism and armed conflict with daily/weekly/monthy visits with local leaders to provide support and discuss grievances [4].

This follows work in security games from recent years [5–9]. While some works have also modeled interactions on a graph, we extend the approach into a new area where actions carry a 'contagion' effect. The problem is a type of influence blocking maximization (IBM) problems [10, 11], which are a competitive extension of the widely studied influence maximization problem [12, 13]. Past work in IBM has looked only at the best-response problems and has not produced algorithms to generate the game-theoretic equilibria necessary for this repeated-interaction domain.

The first major contribution of this work is opening up a new area of research that combines recent research in security games and in IBM. We provide the first techniques for generating game-theoretic equilibria. Drawing from recent work in security games, we propose using a double oracle algorithm where each oracle produces a single player's best response to the opponent's strategy and incrementally creates the payoff matrix being solved. This approach allows us to leverage advances in IBM research that has focused entirely on fast best-response calculations.

The second contribution of our work is in proving approximation quality bounds on the double oracle approach when one of the oracles is approximated and combining this with a greedy approximate oracle to produce a more efficient approximate algorithm. Our final major contribution is to introduce two heuristic oracles, LSMI and PAGERANK, that offer much greater efficiency to address scaling issues with the approximate technique. We conclude with an experimental exploration of a variety of combinations of oracles, testing runtime and quality on a real-world leadership network in Afghanistan, synthetic leadership networks and random scale-free graphs. We find that the performance of the basic PAGERANK oracle suffers minimal loss compared with LSMI in leadership networks that possess clusters of highly interconnected nodes, but performs far worse in sparsely interconnected scale-free graphs. Finally, an unintuitive blend of the two oracles offers the best combination of scalability and solution quality.

2. RELATED WORK

Recent works in game-theoretic security allocation have also dealt with domains that were modeled as graphs [5, 6, 14]; however, their actions were all deterministically defined and did not feature a probabilistic contagion component. This 'spreading' aspect of the problem is very closely related to influence maximization and inoculation problems. Influence maximization, in which a player attempts to optimize a selection of beginning 'seed' nodes from which to spread his influence in a known graph, saw its first treatment in computer science as a discrete maximization problem by Kempe *et al.* [15] who proposed a greedy approximation, followed up by numerous proposed speed-up techniques [12, 13, 16]. Although these are one-player games, we draw inspiration from their techniques to address efficiency issues in our work.

Standard inoculation games feature a defender that attempts to protect nodes in a graph and, usually, a random outbreak of a disease on a node in the graph. These games typically model nature as the adversary, which chooses an initial set of nodes with some predefined probability distribution that the defender is optimizing against [17–22]. Variations on this include distributed inoculation games where each node acts independently, in which results such as price of anarchy are generally considered [17, 20]. These games, however, do not include an optimizing adversary, amounting to only an attacker *or* defender best-response problem.

IBM problems, which we use to model our domain, have been explored with both independent cascade and linear threshold models of propagation [10, 11]. Both of these works only explored the defender's best-response problem instead of equilibrium strategy generation. Aside from IBM, a number of researchers have also explored mutual maximization models where all players seek to maximize their own influence [23, 24]. Finally, Hung *et al.* [25] and Howard [4] also address the COIN problem. However, Hung *et al.* [25] assume a static adversary and Howard [4] only solves for pure strategies. This forced predictability in a repeated-interaction situation is dangerous since a real adversary can directly ambush COIN teams. Additionally, it may be suboptimal since a real adversary has no such limitation.

3. PROBLEM DEFINITION

The COIN domain we focus on includes one party that attempts to subvert the population to their cause and another party that attempts to thwart the first party's efforts [3, 4, 25]. We assume that each side can carry out operations such as provide security or give medical supplies to sway the local leadership's opinion. Furthermore, local leaders will impact other leaders' opinions of the two parties. Specifically, one leader will convert other leaders to side with their affiliated party with some predetermined probability, giving each party's actions a 'spreading' effect. Since resources for COIN operations are very limited relative to the size of the task, each party is faced with a resource allocation task. Hung [3] models the leadership network of a single district in Afghanistan (based on real data) with 73 nodes and notes that recent organizational assignments show that a single battalion operates in 4–7 districts and divides into 3 and 4 platoons per 1 and 2 districts. This translates into 5–30 teams responsible for a network with 300–500 nodes.

We model the COIN domain as a two-player IBM problem, which allows us to draw from the extensive influence maximization literature. An IBM takes place on an undirected graph $G = (V, E)$. One player, the attacker, will attempt to maximize the number of nodes supporting his cause on the graph while the second player, the defender, will attempt to minimize the attacker's influence. Vertices represent local leaders that each player can attempt to sway to their cause, while edges represent the influence of one local leader on another. We note that these leaders do not report to one another and hence an undirected edge provides an apt representation of their influence relationship. Specifically, each edge, $e = (n, m)$, has an associated probability, P_e, which dictates the chance that leader n will influence leader m to side with n's chosen player. Since the graph is undirected, this is also the probability that m influences n to side with m's chosen player. Only uninfluenced nodes can be influenced.

In an IBM, the two players each choose a subset of nodes as their pure strategies (S_a, $S_d \subseteq V$), which we will also refer to as actions. Each action is composed of nodes (also referred to as 'sources') where the allowable number of nodes is referred to as the number of 'resources' a player has and is given for each

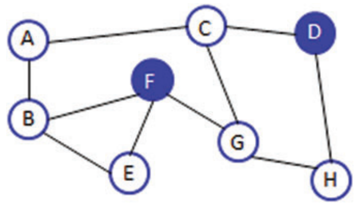

FIGURE 1. Example pure strategy for one player.

player ($|S_a| = r_a$, $|S_d| = r_d$). Figure 1 shows an example of a pure strategy for one player as the selection of the two nodes, D and F, filled in. The other player would similarly choose a set of nodes on the same graph from which to begin spreading his influence.

Each node in $S_a \cap S_d$ has a 50% chance of being influenced by each player, while all other nodes in S_a support the attacker and all other nodes in S_d support the defender. The influence then propagates via a synchronized independent cascade, where at time step t_0 only the initial nodes have been influenced and at t_1 each edge incident to nodes in $S_a \cup S_d$ is 'activated' with probability P_e. Uninfluenced nodes incident to activated edges become supporters of the influencing node's player. If a single uninfluenced node is incident to activated edges from both player's nodes, the node has a 50% chance of being influenced by each player. This process, which outlines a single stochastic outcome, is detailed in Algorithm 1, which outputs the total number of attacker-activated nodes for that particular sample. This process is polynomial with respect to the network size, since each edge will only be probabilistically activated a single time at most.

For a given pair of pure strategies, the attacker's payoff is equal to the *expected* number of nodes influenced to the attacker's side and the defender's payoff is the opposite of the attacker's payoff. We denote the function to calculate the expected number of attacker-influenced nodes as $\sigma(S_a, S_d)$. Each player chooses a mixed strategy, ρ_a for the attacker and ρ_d for the defender, over their pure strategies (subsets of nodes of size r_a or r_d) to maximize their expected payoff. At equilibrium, each player's mixed strategy will be a best response to the other player's. The defender's mixed strategy is a policy by which COIN teams can randomize their deployment each day/week/month, depending on the frequency of missions. The focus of the rest of this work will be to develop optimal, approximate and heuristic oracles that can be used in double oracle algorithms to generate strategies for these IBM problems.

4. DOUBLE ORACLE APPROACH

The most commonly used approach for a zero-sum game is a naïve Maximin strategy, shown in Algorithm 2. In Algorithm 2, P is the defender's expected payoff, C is the set of all column player actions iterated with c, Y is the set of all row player actions iterated with y and $u(y, c)$ is the utility for the row player when actions c and y are played. In our problem, the row player (defender) has a utility equivalent to the opposite of the column player's (attacker's), which is equivalent to the expectation of the propagation process, $\sigma(\cdot)$. That is, $u(y, c) = -\sigma(y, c)$. The primary constraint is Constraint 1, which restricts P to be no greater than the expected utility achieved by the row player in the worst outcome. This linear program, however, requires precalculating the payoffs for every pair of player actions to instantiate all constraints before it can efficiently solve for a Nash equilibrium. This naïve approach admits two faults.

Algorithm 1 Influence prop.: $S_a, S_d, G = (N, E)$.

```
1: $E^* = \emptyset$, $E^{active} = \emptyset$
2: $A \leftarrow \{s | s \in S_a \wedge s \notin S_d\}$, $D \leftarrow \{s | s \notin S_a \wedge s \in S_d\}$
3: **for** $\{s | s \in S_a \cap S_d\}$ **do**
4:     // randomly add $s$ to one of the player's sets
5:     RandomAdd($s, A, D$)
6: **end for**
7: $N^{new} = A \cup D$
8: **while** $N^{new} \neq \emptyset$ **do**
9:     **for** $\{(u, v) | u \in N^{new}, (u, v) \notin E^*\}$ **do**
10:        // activate the edge based on its probability
11:        $E^{active}$.add(RandomActivate($(u, v)$))
12:        $E^*$.add($(u, v)$)
13:    **end for**
14:    $N^{new} = \emptyset$
15:    **for** $\{s | s \notin A \cup D, \exists(u, s) \in E^{active}\}$ **do**
16:        $N^{new}$.add($s$)
17:        // Add $s$ to appropriate set
18:        AddToSet($s, A, D$)
19:    **end for**
20: **end while**
21: **return** $S_a$
```

Algorithm 2 Maximin Linear Program.

$$\text{MAXIMIZE} \quad P$$
$$\text{SUBJECT TO:}$$
$$\forall c \in C \quad P \leq \sum_{y \in Y} p_y \cdot u(y, c) \tag{1}$$
$$0 \leq p_y \leq 1, \forall y \in Y \tag{2}$$
$$\sum_{y \in Y} p_y = 1 \tag{3}$$

First, the payoff for a given pair of pure strategies in our problem is computationally intractable to calculate accurately.

As shown by Chen *et al.* [12, 20], calculating the analogous expectation in a basic influence maximization game exactly is $\#P$-Hard. Since influence maximization is a special case of IBM, it is trivial to show that calculating $\sigma(\cdot)$ exactly is also $\#P$-Hard. The standard method for estimating these expectations is a Monte Carlo approach that was adapted for the IBM problem by Budak *et al.* [10] and which we also adopt here. It involves simulating the propagation process thousands of times to reach an accurate estimate of the expected outcome. Although it runs in time polynomial in the size of the graph and is able to achieve arbitrarily accurate estimations, the thousands of simulation trials required for accurate results causes this method to be extremely slow in practice.

Secondly, the Maximin algorithm stores the entire payoff matrix in memory, which can be prohibitive for large graphs. For example, with 1000 nodes and 50 resources per player, each player has $\binom{1000}{50}$ actions. To overcome similar memory problems, double oracle algorithms have been proposed in the past [6, 14] and form the basis for our work.

Double oracle algorithms for zero-sum games use a Maximin linear program at the core, but the payoff matrix is grown incrementally by two oracles. This process is shown in Algorithm 3. We denote by **D** the set of defender actions generated so far, and **A** is the set of attacker actions generated so far. MaximinLP(**D**,**A**) solves for the equilibrium of the game that only has the pure strategies in **D** and **A** and returns ρ_{d} and ρ_{a}, which are the equilibrium defender and attacker mixed strategies over **D** and **A**. DefenderOracle($\cdot$) solves the defender's 'best-response problem'. That is, it generates a defender action that is a best response against ρ_{a} among *all* possible actions. This action is added to the set of available pure strategies for the defender **D**. A similar procedure then occurs for the attacker. Convergence occurs when neither best-response oracle generates a pure strategy that is superior to the given player's current mixed strategy against the fixed opponent mixed strategy. The number of attacker and defender actions in the payoff matrix varies depending on the speed of convergence, but is generally much smaller than the full matrix. It has been shown that, with two optimal best-response oracles, the double oracle algorithm converges to the Maximin equilibrium [26], although no guarantees are known regarding the time to convergence.

Algorithm 3 Double Oracle Algorithm.

```
1: Initialize D with random defender allocations.
2: Initialize A with random attacker allocations.
3: repeat
4:     $(\rho_d, \rho_a) = \text{MaximinLP}(\mathbf{D},\mathbf{A})$
5:     $\mathbf{D} = \mathbf{D} \cup \{\text{DefenderOracle}(\rho_a)\}$
6:     $\mathbf{A} = \mathbf{A} \cup \{\text{AttackerOracle}(\rho_d)\}$
7: until convergence
8: return $(\rho_d, \rho_a)$
```

TABLE 1. Example game's full payoff matrix.

	A	B	C
1	3, −3	−1, 1	2, −2
2	1, −1	2, −2	−2, 2

TABLE 2. Initial subgame.

	C
1	2, −2

4.1. Double oracle: example

To illustrate the double oracle algorithm in more detail, consider the game described by the payoff matrix featured in Table 1. As per standard game-theoretic notation, the row player's available actions are 1 and 2 and the column player's available actions include A, B and C. If the row player plays 1 and the column player plays A, then the row player receives a payoff of 3 and the column player a payoff of −3. Although this game could be solved by a single Maximin run, we will describe the solution procedure used by the double oracle algorithm to clarify the process. Initially, each player's actions are randomly seeded with a single action from the complete action space of the original game. Suppose the defender, the row player, is seeded with action '1' and the attacker, the column player, is seeded with action 'C' as shown in Table 2. Then $\mathbf{D} = \{1\}$ and $\mathbf{A} = \{C\}$. This subgame is trivially solved using a Maximin linear program that produces the optimal strategy for both players, which is to simply play their only available action 100% of the time ($\rho_{\mathrm{d}} = \rho_{\mathrm{a}} = \{1.0\}$).

Next the algorithm consults two oracles for the next action to add to the subgame for each player. In this case, AttackerOracle produces 'B' as the optimal action for the attacker to take when the defender is playing a strategy of 100% '1'. The DefenderOracle, in contrast, produces '1' as the best response to the current adversary strategy of 100% 'C' and chooses to add action '1' which already exists in the subgame. The subgame is now composed of one action for the defender ($\mathbf{D} = \{1\}$) and two actions for the attacker ($\mathbf{A} = \{C, B\}$). A Maximin solver is again run to determine the optimal strategy for each player, producing a new pure-strategy equilibrium ($\rho_{\mathrm{d}} = \{1.0\}$, $\rho_{\mathrm{a}} = \{0.0, 1.0\}$).

Both oracles are consulted again, with the AttackerOracle again returning 'B' as the optimal action to the current defender strategy (play '1' 100%), but the DefenderOracle now returning action '2' as the best response to the current attacker strategy (play 'B' 100%). The subgame grows to the 2 × 2 matrix shown in Table 3 and the Maximin linear program is again run to solve it, producing new optimal strategies for each player ($\rho_{\mathrm{d}} = \{\frac{4}{7}, \frac{3}{7}\}$, $\rho_{\mathrm{a}} = \{\frac{4}{7}, \frac{3}{7}\}$). Another query to each oracle reveals that both players' best responses are already included

TABLE 3. At convergence.

	B	C
1	−1, 1	2, −2
2	2, −2	−2, 2

in the subgame. At this point, the algorithm has converged and the Maximin equilibrium strategies for both players in the full game have been determined.

The payoff matrix generated at convergence is shown in Table 3. Note that the attacker action 'A' was never added to the game. Not only does this limit the size of the payoff matrix stored in memory, but this also means that no payoffs associated with action 'A' need to be generated (recall that payoff generation in our problem is $\#P$-Hard).

4.2. Double oracle: approximation

Now we prove an approximate double oracle setup that admits a quality guarantee. We denote the defender and attacker's mixed strategies at convergence as ρ_d and ρ_a. Also, we denote the defender's expected utility given a pair of mixed strategies as $u_\mathrm{d}(\rho_\mathrm{d}, \rho_\mathrm{a})$. Assume that the *defender*'s oracle, D_AR, is an α-approximation of the optimal best-response oracle, D_BR, so that $D_\mathrm{AR}(\rho_\mathrm{a}) \geq \alpha \cdot D_\mathrm{BR}(\rho_\mathrm{a})$. The following theorem is a generalization of a similar result in Halvorson *et al.* [14].

Theorem 4.1. *Let $(\rho_\mathrm{d}, \rho_\mathrm{a})$ be the output of the double oracle algorithm using an approximate defender oracle and let $(\rho_\mathrm{d}^*, \rho_\mathrm{a}^*)$ be the optimal mixed strategies. Then $u_\mathrm{d}(\rho_\mathrm{d}, \rho_\mathrm{a}) \geq \alpha \cdot u_\mathrm{d}(\rho_\mathrm{d}^*, \rho_\mathrm{a}^*)$.*

Proof. Since we know D_AR is an α-approximation, $u_\mathrm{d}(\rho_\mathrm{d}, \rho_\mathrm{a}) \geq u_\mathrm{d}(D_\mathrm{AR}(\rho_\mathrm{a}), \rho_\mathrm{a}) \geq \alpha \cdot u_\mathrm{d}(D_\mathrm{BR}(\rho_\mathrm{a}), \rho_\mathrm{a})$. Since $(\rho_\mathrm{d}^*, \rho_\mathrm{a}^*)$ is a maximin solution, we know that $\forall \rho_\mathrm{d}', \rho_\mathrm{a}'$: $u_\mathrm{d}(\rho_\mathrm{d}^*, \rho_\mathrm{a}') \geq u_\mathrm{d}(\rho_\mathrm{d}^*, \rho_\mathrm{a}^*) \geq u_\mathrm{d}(\rho_\mathrm{d}', \rho_\mathrm{a}^*)$. Thus, $u_\mathrm{d}(D_\mathrm{BR}(\rho_\mathrm{a}), \rho_\mathrm{a}) \geq u_\mathrm{d}(\rho_\mathrm{d}^*, \rho_\mathrm{a}) \geq u_\mathrm{d}(\rho_\mathrm{d}^*, \rho_\mathrm{a}^*)$, implying $u_\mathrm{d}(\rho_\mathrm{d}, \rho_\mathrm{a}) \geq \alpha \cdot u_\mathrm{d}(\rho_\mathrm{d}^*, \rho_\mathrm{a}^*)$. □

5. ORACLES

A major advantage of double oracle algorithms is the ability to divide the problem into best-response components. This allows for easily creating variations of algorithms to meet runtime and quality needs by combining different oracles together. Here, we present four oracles that we can combine to create a suite of algorithms.

5.1. EXACT oracle

Solving for a best response in an IBM problem was shown to be NP-Hard by Budak *et al.* [10], but an optimal oracle may be useful when paired with an efficient second oracle, given the approximation result just shown. The first oracle we introduce is an optimal best-response oracle. Our oracle, which we call EXACT , determines the best response by iterating through the entire action set for a given player. For each action, the expected payoff against the opponent's strategy is calculated, which requires n calculations of $\sigma(\cdot)$, where n is the size of the support for the opponent's mixed strategy. In this oracle, $\sigma(\cdot)$ is evaluated via the Monte Carlo estimation method, the benchmark technique in influence maximization. This technique involves simulating the propagation process n times, where n is generally 10 000–20 000, and using the average propagation of the simulated trials as the estimate. The ϵ-error of the Monte Carlo estimation exists in the Maximin approach as well, but can be made arbitrarily small with sufficient simulations [15].

This oracle can be used for both the defender and the attacker to create an incremental, optimal algorithm that can potentially be superior to Maximin because of the incremental approach. However, the oracle will perform redundant calculations that can cause it to run slower than Maximin when the equilibrium strategies support size is very large.

5.2. APPROX oracle

Here we describe approximate oracles that draw from research in influence maximization, competitive influence maximization and IBM. Budak *et al.* [10] showed that the best-response problem for the blocker (the defender, in our setting) is submodular when both players share the same probability of influencing across a given edge. Thus, a greedy hill-climbing approach that provides the highest marginal gain in each round provides a $(1 - 1/e)$-approximation. This is outlined in Algorithm 4, where r_d is given for the problem instance, MCEst$(\cdot)$ is the Monte Carlo estimation of $\sigma(\cdot)$, ρ_a is the current attacker mixed strategy, Action() retrieves a pure strategy, S_a and Prob() retrieves a pure strategy's associated probability. The Lazy-Forward speed-up to the greedy algorithm introduced by Leskovec *et al.* [16] to tackle influence maximization problems is also implemented, but we do not show it in Algorithm 4 for clarity.

For the maximizer's best-response problem, we note that, given a fixed blocker strategy, the best-response problem of the maximizer in an IBM is exactly the best-response problem of the last player in a competitive influence maximization from Bharathi *et al.* [24], which they showed to be submodular. Thus, the attacker's best-response problem can also be approximated with a greedy algorithm with the same guarantees. These oracles are referred to as APPROX.

By combining an APPROX oracle for the defender and an EXACT oracle for the attacker, we can create an algorithm that generates a strategy for the defender more efficiently than the naive one and guarantees a reward within $(1 - 1/e)$ of the optimal strategy's reward by Theorem 4.1. An algorithm with two APPROX oracles no longer admits quality guarantees, but the iterative process still maintains the best-response reasoning crucial to adversarial domains.

Algorithm 4 APPROX -DefBR(ρ_a).

```
1: $S_d = \emptyset$
2: while $|S_d| < r_d$ do
3:     for $n \in (N - S_d)$ do
4:         U($n$) = $\sum_{i=1}^{\rho_a.\text{Size}()} \rho_a$.Prob($i$) ·
5:             MCEst($\rho_a$.Action($i$),$S_d \cup \{n\}$)
6:     end for
7:     $n^* = \arg\max_{n \in N}$ U($n$)
8:     $BR = BR \cup \{n^*\}$
9: end while
10: return $BR$
```

5.3. LSMI oracle

We introduce our main heuristic oracle, LSMI, which is also the name of the heuristic it is based on: Local Shortest-paths for Multiple Influencers (LSMI(·)). This oracle uses APPROX oracle's Algorithm 4. However, LSMI(·) is used to replace the MCEst(·) function and provides a fast, heuristic estimation of the marginal gain from adding a node to the best response. The heuristic is based on two assumptions: very low probability paths between two nodes are unlikely to have an impact and the highest probability path between two nodes estimates the relative strength of the influence. The probability associated with a path is defined as $P = \prod_e P_e$ over all edges e on the path. We then combine these heuristic influences from two players in a novel, efficient way.

The two heuristic assumptions have been applied successfully for one-player influence maximization in various forms, one of the most recent being Chen *et al*. [12]. As an application of the first assumption, when calculating the influence of a node, they only consider nodes reachable via a path with an associated probability of at least some θ. As an application of the second assumption, Chen *et al*. [12, 20] assume that each source will only affect nodes via the highest probability path. To improve the accuracy of this estimation, they disallow other sources from being on the path since the closer source's influence will supersede that of the further source along the same path. We use these ideas as well, but the approach of Chen *et al*. [12, 20] to the critical step of combining these influences efficiently relies on there being only one type of influence. In a two-player situation such as ours, there are two probabilities associated with each node, and the winning influencer depends not only on the probability but on the distance to sources as well. This ordering effect is a new issue that necessitates a novel approach to influence estimation.

L-Eval(·), described in Algorithm 6, is our new algorithm for determining the expected influence of the local neighborhood around a given node. LSMI(n, S_a, S_d) estimates the marginal gain of n by finding the difference between calling L-Eval(·) with and without n and replaces the MCEst(·) function in Algorithm 4. For the defender oracle, instead of a call of MCEst(S_a, $S_d \cup n$):

$$\begin{aligned} \text{LSMI}(S_a, S_d, n) &= \text{L-Eval}(V, S_a, S_d \cup \{n\}) \\ &\quad - \text{L-Eval}(V, S_a, S_d), \\ \text{s.t.} \quad V &= \text{GetVerticesWithin}\theta(n). \end{aligned}$$

GetVerticesWithinθ() is a modified Dijkstra's algorithm that measures path length by hop distance, tie-breaks with the associated probabilities of the paths, and stores all nodes' shortest hop distance and associated probability to the given node. It does not add a new node to the search queue if the probability on the path to the node falls below θ. This procedure is outlined in more detail in Algorithm 5. The overall structure remains identical to Dijkstra's algorithm, but distances are now measured with hop distance instead of summing the weights on edges and a cut-off is implemented when the probability on the path falls below θ. The probability on the path is calculated via probDistanceTo(), which simply calculates the product of probabilities on edges along the path from n to v. Since the algorithm exactly mirrors that of Dijkstra's algorithm, the runtime attributes are identical.

Algorithm 5 GetVerticesWithinθ(n).

```
1: for $v \in V$ do
2:     hopDistanceTo[$v$] := infinity
3:     probDistanceTo[$v$] := 0
4:     $Q$.enqueue($v$)
5: end for
6: hopDistanceTo[$n$] := 0
7: probDistanceTo[$n$] := 1
8:
9: while $Q \neq \emptyset$ do
10:     $u$ := vertex in $Q$ with smallest distance (by hopDistanceTo)
11:     remove $u$ from $Q$
12:     if hopDistanceTo[$u$] == infinity then
13:         break
14:     end if
15:     for each neighbor $v$ of $u$ do
16:         $t_{hop}$ := hopDistanceTo[$u$] + 1
17:         $t_{prob}$ := probDistanceTo[$u$] · $p_{(u,v)}$
18:         if $t_{hop} \leq$ hopDistanceTo[$v$] AND $t_{prob} > \theta$ then
19:             hopDistanceTo[$v$] := $t$
20:             if $t_{prob} >$ probDistanceTo[$v$] then
21:                 probDistanceTo[$v$] := $t_{prob}$
22:             end if
23:         end if
24:         /* Reorder $v$ in Queue, tie-break with probDistanceTo[]*/
25:         decrease-key($v$,$Q$)
26:     end for
27: end while
28: return all $v$ with hopDistanceTo[] less than infinity
```

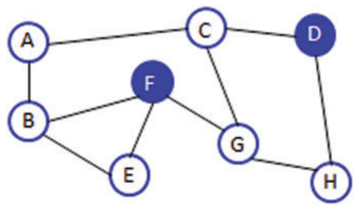

FIGURE 2. Example network.

In L-Eval(·), V is the set of n's local nodes and S_a/S_d are the attacker/defender source sets. Owing to the addition of n, we must recalculate the expected influence of each $v \in V$. First, we determine all the nearby nodes that impact a given v by calling GetVerticesWithin$\theta(v)$. Since only sources exert influence, we intersect this set with the set of all sources and compile them into a priority queue ordered from the lowest hop distance to the greatest. We represent P_a and P_d by, respectively, the probability is that the attacker and defender successfully influences the given node. From the nearest source, we aggregate the conditional probabilities in order. If the next nearest source is an attacker source, then P_a is increased by the probability that the new source succeeds, conditional on the failure of all closer defender and attacker sources. The probability that all closer sources failed is exactly $P_a + P_d$. Here P_d remains unchanged. If the next nearest source is a defender source, then a similar update is performed. The algorithm iterates through all impacted nodes and returns the total expected influence.

To illustrate the aggregation calculation, we reproduce the graph from Fig. 1 again here in Fig. 2. Consider node h and assume the adversary has chosen source f and the defender has chosen source d. Since influence travels along edges in an ordered fashion, the influence of f is only possible if d fails to influence h, since d is closer in terms of the hop distance. Thus, the probability that h is converted into an adversary node is

$$(1 - P_{(d,h)}) \cdot (P_{(f,g)} \cdot P_{(g,h)}). \tag{4}$$

In words, the probability is equal to the joint probability that d fails and the influence from f succeeds in influencing g and then h thereafter. Note that if the defender had a second source further away, it would be completely irrelevant, since only adversary-influenced nodes contribute to the payoff determination.

Although the estimated marginal gain of LSMI can be arbitrarily inaccurate, choosing the best action only requires that the *relative* marginal gain of different nodes be accurate. We show in the Experiments section that LSMI does a very good job of this in practice as evidenced by the high reward achieved by LSMI-based algorithms. The final algorithm for the LSMI best-response oracle is shown in Algorithm 7.

Algorithm 6 L-Eval(V, S_a, S_d).

```
1: $InfValue = 0$
2: for $v \in (V - S_a - S_d)$ do
3:     $\mathbf{N}$ = GetVerticesWithin$\theta(v) \cap (S_a \cup S_d)$
4:     /* Prioritize sources by lowest hop-distance to $v$*/
5:     $\mathbf{S}$ =makePriorityQueue($\mathbf{N}$)
6:     $p_a = 0, p_d = 0$
7:     while $\mathbf{S} \neq \emptyset$ do
8:         $s = \mathbf{S}$.poll()
9:         if $(s \in S_a)$ then
10:            $p_a = p_a + (1 - p_a - p_d)\cdot$ Prob$(s, v)$, $p_d = p_d$
11:        else /* $s$ must be in $S_d$ */
12:            $p_d = p_d + (1 - p_a - p_d)\cdot$ Prob$(s, v)$, $p_a = p_a$
13:        end if
14:    end while
15:    $InfValue = InfValue + p_a$
16: end for
17: return $InfValue$
```

Algorithm 7 LSMI-BR(ρ_a).

```
1: $S_d = \emptyset$
2: while $|S_d| < r_d$ do
3:     for $n \in (N - S_d)$ do
4:         U$(n) = \sum_{i=1}^{\rho_a.\text{Size()}} \rho_a$.Prob$(i)\ \cdot$
5:             LSMI($\rho_a$.Action$(i)$,$S_d \cup \{n\}$)
6:     end for
7:     $n^* = \arg\max_{n \in N}$ U$(n)$
8:     $BR = BR \cup \{n^*\}$
9: end while
10: return $BR$
```

5.4. PAGERANK oracle

PageRank is a popular algorithm to rank webpages [27], which we adapt here due to its frequent use in influence maximization as a benchmark heuristic. The underlying idea is to give each node a rating that captures the power it has for spreading influence that is based on its connectivity. For the purposes of describing PageRank, we will refer to directed edges $e_{u,v}$ and $e_{v,u}$ for every undirected edge between u and v. For each edge $e_{u,v}$, set a weight $w_{u,v} = P_e/P_v$ where $P_v = \sum_e P_e$ over all edges incident to v. The rating or 'rank' of a node u, $\tau_u = \sum_v w_{u,v} \cdot \tau_v$ for all non-source nodes v adjacent to u. The exclusion of source nodes is performed because u cannot spread its influence through a source node.

For our oracles, since the defender's goal is to minimize the attacker's influence, the defender oracle will focus on nodes incident to attacker sources $N_a = \{n | n \in V \wedge \exists e_{n,m}, m \in S_a\}$. Specifically, ordering the nodes of N_a by decreasing rank value, the top r_d nodes will be chosen as the best response. In the attacker's oracle phase, the attacker will simply choose the nodes with the highest ranks. Although PAGERANK is very efficient, we expect its quality to be low, since the attacker oracle fails to account for the presence of a defender and the defender oracle only searches through nodes directly incident

TABLE 4. Algorithms evaluated.

Algo label	Def. oracle	Att. oracle	Nodes	R
DOEE	EXACT	EXACT	15	3
DOAE	APPROX	EXACT	20	3
DOAA	APPROX	APPROX	100	3
DOLE	LSMI	EXACT	20	3
DOLA	LSMI	APPROX	100–200	3
DOLL	LSMI	LSMI	450	20
DOLP	LSMI	PAGERANK	700	20
DOPE	PAGERANK	EXACT	40	3
DOPA	PAGERANK	APPROX	200–300	3
DOPL	PAGERANK	LSMI	1000+	20
DOPP	PAGERANK	PAGERANK	1000+	20

to the attacker's source nodes. We will refer to oracles based on this heuristic as PAGERANK.

6. EXPERIMENTS

In this section, we show experiments on both synthetic and real-world leadership and social networks. We evaluate the algorithms on scalability and solution quality. One advantage of double oracle algorithms is the ease with which the oracles can be changed to produce new variations of existing algorithms. This allows us to simulate various attacker/defender best-response strategies and test our heuristics' performance more thoroughly.

Ideally, we would report the performance of our mixed strategy against an optimal best response as a worst-case analysis. However, due to scalability issues with the EXACT best-response oracle, rewards for larger graphs can only be calculated against an approximate best response generated by the APPROX oracle. Unless otherwise stated, each datapoint is an average over 100 trials and the games created used contagion probability on edges of 0.3, 20 000 Monte Carlo simulations per estimation and an LSMI $\theta = 0.001$. All experiments were run on machines with CPLEX 12.2, 2.8 GHz CPU and 4 GB of RAM.

In addition to the optimal Maximin algorithm, we also test the set of double oracle algorithms listed in Table 4, where Nodes and R(esources) indicate the approximate problem complexity the algorithm can handle within 20 min based on experiments with scale-free graphs.

6.1. Leadership networks

In Hung [3], a leadership network was created based on real data of a district in Afghanistan with seven village areas, each with a few 'village leaders' with connections outside the village and a cluster of 'district leaders' shown in the middle.

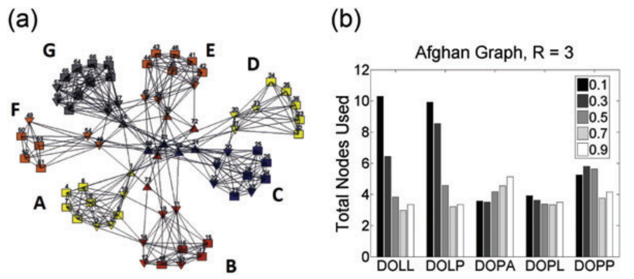

FIGURE 3. Afghanistan leadership network results. (**a**) Network from Hung (2010) and (**b**) nodes in defender strategy.

We recreate the same network, shown in Fig. 3a and run our algorithms on it. Although not shown, quality as measured against an APPROX attacker was very similar for all algorithms. Algorithms exceeding 20 min are not shown.

Closer examination of defender strategies reveals a difference in the oracles' approach. Since the PAGERANK defender oracle considers only attacker-adjacent nodes with the highest rank, most of its strategies focus on two high-degree district leaders (neither are maximal degree nodes) and on a regular member of the highest population Village G. In this graph structure, where sets of nodes are fully connected, this strategy works very well because the attacker's best response will often be the highest degree district leader and a node in Village G. This approach is more conservative than LSMI, which directly chooses the attacker's source nodes since the 50% chance of wiping out an attacker source provides slightly higher utility. The attacker oracles all select from the same set of four high-degree nodes. Aside from the highest degree district leader and Village G nodes, an additional high-degree village leader far from Village G is also used. This result suggests that not only connectivity, but also strategic spacing provided by our algorithms is a key point for the maximizer's target selection.

Experiments varying contagion probability, shown in Fig. 3b, show LSMI defender oracle algorithms randomizing over many more nodes at low contagion levels. This occurs because the attacker's initial set of nodes accounts for most of his expected utility, encouraging randomization over many nodes. PAGERANK ignores this since a given set of nodes is often adjacent to all sets of attacker-chosen nodes, while LSMI responds by matching the increased node use directly.

As noted previously, a battalion is responsible for 4–7 districts, so we create synthetic graphs with multiple copies of a village structure (70 nodes each) and link all district leaders together to create multi-district graphs. In our experiments, for every district, each player is given three resources. Figure 4 shows runtime and solution quality against an APPROX attacker best response. Since we create the graphs one district at a time, the graph sizes increase by 70 nodes at a time. The trend in rewards is once again that LSMI defender oracle algorithms very slightly outperform the others. All four algorithms scale to real-world problem sizes.

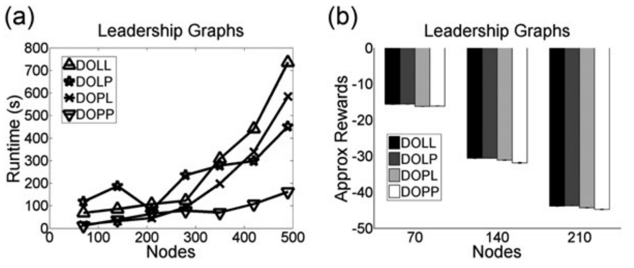

FIGURE 4. Synthetic leadership network results. (**a**) Runtime and (**b**) quality.

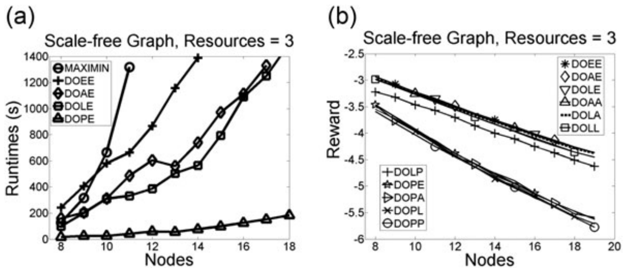

FIGURE 5. Scale-free, 8–20 nodes, 3 resources. (**a**) Runtime and (**b**) quality.

6.2. Random scale-free graphs

Scale-free graphs[1] have commonly been used as proxies for real-world social networks because the distribution of node degrees in many real-world networks have been observed to follow a power law [28]. We conduct experiments on randomly generated scale-free graphs of various sizes to illustrate both the runtime scalability and quality of each algorithm in graphs resembling social networks as opposed to leadership networks.

Figure 5 shows the results for small scale-free graphs of 8–20 nodes with 3 resources for each player. The runtime graph (Fig. 5a) shows only the algorithms that exceed 20 min for clarity. The remaining heuristic algorithms' results all hug the x-axis because they take minimal time for these graphs. As would be expected, Maximin scales the most poorly and is only able to handle graphs of up to 11 and 12 nodes. The approximate algorithm, DOAE improves upon DOEE and can handle up to 16 and 17 nodes, but swapping out the APPROX oracle for the very fast LSMI oracle does not improve runtime scalability very noticeably. This is because although the LSMI oracle is orders of magnitude faster than the APPROX oracle, the EXACT attacker oracle's runtime eclipses both of them, making the improvement irrelevant.

In Fig. 5b, we show the reward obtained by the defender when using the strategies generated against an EXACT attacker's best

[1]The networks are built using the Barabási–Albert network-building algorithm and will be referred to as scale-free networks.

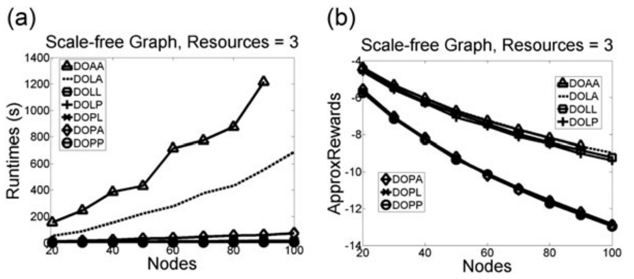

FIGURE 6. Scale-free, 20–100 nodes, 3 resources. (**a**) Runtime and (**b**) quality.

response as described earlier. The key point is that the majority of rewards are indistinguishable from the optimal algorithms. The DOLL algorithm begins to diverge slightly when the graph nears 100 nodes, but the major exceptions are the algorithms featuring PAGERANK defender oracles. Interestingly, DOLP, which uses LSMI for the defender and PAGERANK for the attacker still generates high rewards.

Figure 6 shows runtime and quality for larger scale-free graphs of 20–100 nodes with 3 resources for each player. As can be seen, the algorithms featuring the APPROX oracle (DOAA, DOLA) begin to exceed our 20-min cut-off near 100 nodes, while the remaining heuristic algorithms continue to hug the x-axis because even these games are completed in minimal time. As discussed previously, due to the inefficiency of the EXACT oracle, we use an APPROX best response to calculate a more conservative reward value. Figure 6b again shows algorithms with PAGERANK defender oracles performing noticeably more poorly than the other algorithms. DOLP is again very close to the top performers. Note that while this may be due to the APPROX best response being used instead of an EXACT best response, it is very unlikely than an attacker could perform any better given the hardness of the best-response problem.

7. STRATEGY ANALYSIS

In addition, three types of variations were explored on scale-free networks in more depth. First, we varied the size of the graph and kept all other parameters constant. Second, we varied the average contagion probability in the graphs at three separate graph sizes. Finally, we varied the standard deviation of the contagion probability in the graphs and again tested these at three separate graph sizes. All experiments featured a randomly generated scale-free graph, 10 resources per player (S_d, S_a = 10) and contagion probabilities on edges that were drawn from a normal distribution. Scale-free graphs were chosen due to their widespread use as proxies for general social networks and were generated according to the principle of 'preferential attachment' as introduced by Barabasi and Albert [29]. Our particular implementation adds edges between existing vertices

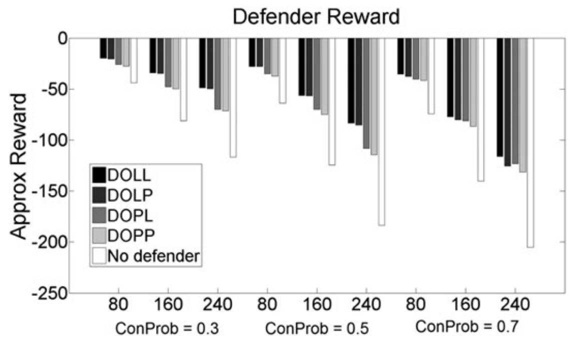

FIGURE 7. Preliminary test, $r = 10$, avg. $= 0.3$, s.d. $= 0.1$.

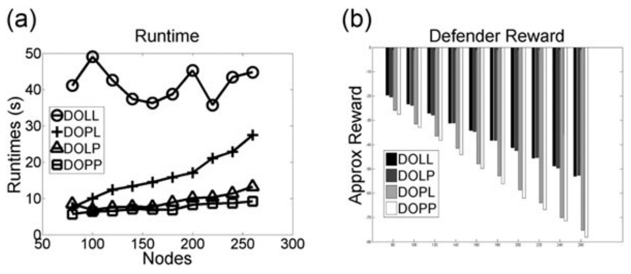

FIGURE 8. Scale-up results, $r = 10$, avg. $= 0.3$, s.d. $= 0.1$. (**a**) Runtime and (**b**) quality.

and newly added vertices with a probability of $P = (\deg(v) + 1)/(|E| + |V|)$.[2] 100 trials were run for every data point shown.

Figure 7 shows a preliminary test that was conducted to provide a benchmark for the quality results. It shows the reward for the defender when each of the four algorithms is used as well as when no defender is present as well for graphs of size 80, 160 and 240 and with the average contagion probability set to 0.3, 0.5 and 0.7. Again, the reward reported is the reward achievable by an adversary that best responds to our algorithm's generated defender strategy by calculating the approximate best response via the algorithm proposed by Budak *et al.* [10]. As mentioned, the graph sizes tested were limited to 260 nodes because for larger graphs even calculating the approximate best response outlined above begins to take longer than 20 min as well.

As can be seen, all of the algorithms provide at least a 30–40% improvement in reward obtained as opposed to having no defender present across all of the cases tested. Since this was intended as a preliminary justification for the algorithms, we will provide more in-depth analysis of the solution quality of the algorithms in the following subsections.

7.1. Graph size scale-up

The first set of experiments explored the impact of scaling up the size of the graph alone. Specifically, the more efficient four algorithms (all combinations of the LSMI and PAGERANK oracles) were run on randomly generated scale-free graphs with 80–260 nodes in increments of 20, with 10 resources and contagion probabilities drawn from a normal distribution $\mathcal{N}(0.3, 0.1)$. Graph sizes were limited to 260 nodes because the adversary best-response technique used to determine the defender's reward became too cumbersome for larger graphs.

Figure 8a shows the impact on runtime as the graph size is scaled up. As can be seen, the solution technique that features two LSMI oracles (DOLL) requires the longest run time at 40–50 s for all of the game sizes tested. Interestingly, there did not appear to be a consistent increase in runtime as was observed in the other three algorithms (each of which had at least one PAGERANK oracle). This is due to the fact that the runtime depends on the size of the problem but also on the ability of the oracles to find new, higher-quality pure strategies to add to the subgame being solved. DOLL features two highly adaptive LSMI oracles and, as evidenced, tends to generate many more actions for the smaller graph sizes. Thus, although the graphs get larger, fewer iterations are used, causing minimal runtime increase as the graph size is increased.

The other three algorithms were much faster across the board, all requiring less than 30 s with a consistent trend as the graph size increases. DOPL requires more time than DOLP because of the fact that the defender PAGERANK oracle explicitly adapts to the attacker's strategy (only uses nodes adjacent to attacker nodes), while the attacker PAGERANK oracle does not. Previous work explored scaling to larger graphs with more resources, but since this is not the focus of our work, we refer the interested reader to Tsai *et al.* [30].

Figure 8b shows the impact on solution quality as the graph size is scaled up. Unsurprisingly, as the size of the graph increases, it becomes increasingly difficult for the defender to block the adversary's influence spread and the defender receives a correspondingly lower reward. Again, we also observe a large difference between algorithms that use an LSMI oracle for the defender as opposed to a PAGERANK oracle for the defender, with the latter providing much lower rewards. This is expected, due to the higher sophistication of the LSMI defender oracle as was noted earlier.

Figure 9a shows the final number of actions in the defender's action set as the size of the graph is increased. The action set is defined as the number of actions available to the defender in the CoreLP phase of the double oracle algorithm and is exactly the number of new best responses that have been found by the defender oracle. In the worst case, this would include all possible actions in the game, but as can be seen is generally far smaller, making the problem much more tractable. The attacker's action set size was always extremely similar if not identical to the defender's action set size.

[2] http://jung.sourceforge.net/doc/api/edu/uci/ics/jung/algorithms/generators/random/BarabasiAlbertGenerator.html.

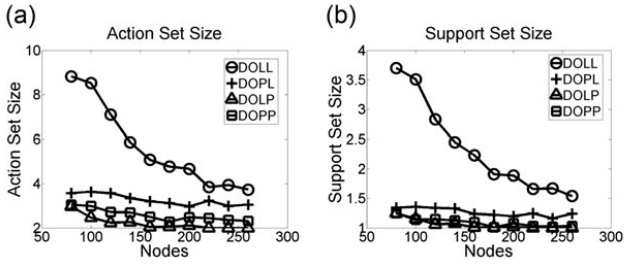

FIGURE 9. Scale-up results, $r = 10$, avg. $= 0.3$, s.d. $= 0.1$. (**a**) Action set size and (**b**) support set size.

Figure 9b shows a similar metric and features the number of actions in the support set of the final defender strategy. The support set is the set of actions that have non-zero probability in the final mixed strategy. Again, the final attacker support set size was always extremely similar if not identical to the defender's.

As can be seen, both the action set and the support set sizes are much larger with the DOLL algorithm than for any of the other algorithms. This is due to the sophistication of the LSMI oracles as opposed to the PAGERANK oracle. The PAGERANK oracles converge extremely quickly to a small set of actions and often do not generate new actions in response to new adversary strategies. This is especially true for the PAGERANK attacker oracle, since the defender oracle actually chooses nodes directly adjacent to the attacker. Thus, even when only one PAGERANK oracle is used, the algorithm overall converges quickly. The DOLL algorithm iterates many more times than algorithms featuring the PAGERANK oracle, leading to the previous runtime result with DOLL being far slower than the other algorithms.

Furthermore, the trends seen in both Fig. 9a and b show the size of the final action set and support set decreasing as the graph size is increased. This is due to the fact that as the graph grows larger, very few actions are useful for the defender to use to defend against the spread of the attacker's influence. For the attacker, randomization becomes less essential for the same reason. Thus, both players converge to a very small set of actions for the final mixed strategy.

7.2. Contagion probability: average

To explore the impact of changing the contagion probabilities on the four algorithms, we tested three different contagion probability averages for three separate graph sizes. Specifically, we ran all four algorithms with the contagion probabilities drawn from normal distributions $\mathcal{N}(0.3, 0.1)$, $\mathcal{N}(0.5, 0.1)$ and $\mathcal{N}(0.7, 0.1)$. The graph sizes tested were 80, 160 and 240 node random scale-free graphs with 10 resources allowed per player. We measured the same four metrics as in the previous section: runtime, solution quality, action set size and support set size.

Figure 10a shows the results pertaining to runtime. The x-axis is divided into three sets of three bars each. Each set represents one setting for the contagion probability average (0.3, 0.5, 0.7), while each bar represents the runtime result for one algorithm. At averages of 0.5 and 0.7, consistent trends can be seen, with larger graphs taking longer and higher probabilities leading to longer runtimes for algorithms with LSMI oracles. This is because LSMI oracles speed up heuristic estimation by calculating only high probability influences, but when contagion probabilities are higher, this leads to many more nodes that must be processed by the algorithm.

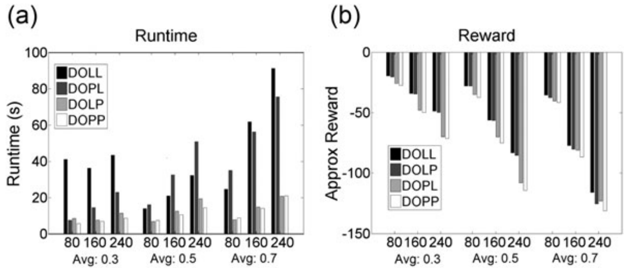

FIGURE 10. Contagion probability average results, s.d. $= 0.1$. (**a**) Runtime and (**b**) quality.

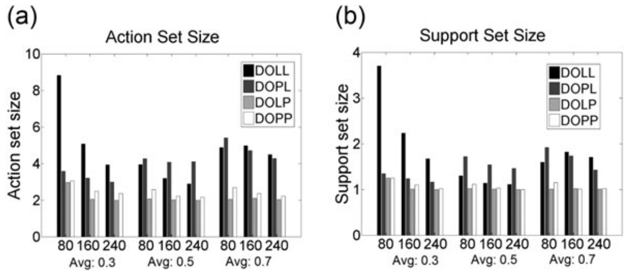

FIGURE 11. Contagion probability average results, s.d. $= 0.1$. (**a**) Action set size and (**b**) support set size.

For the case of 0.3, however, the trend is not consistent for the DOLL algorithm. Experiments suggest that, with low contagion probabilities, two LSMI oracles continually find new best responses to each other's strategies. This occurs because, at low contagion probabilities, different parts of the graph interact minimally and the attacker is able to move to 'new' nodes and entirely avoid the defender, resulting in a cat-and-mouse game that requires many more iterations to converge than when a PAGERANK oracle is used.

Figure 10b shows the reward for the defender using the same approximate best-response technique described previously. Unsurprisingly, larger graphs lead to a lower reward for the defender because it is harder to defend. Higher contagion probabilities also result in lower defender rewards for the same reason.

As we noted in the scale-up experiments, larger graphs lead to fewer actions in the action set as well as the final support set, as shown in Fig. 11a and b. As mentioned, at the lowest contagion probability tested (0.3), the action and support set sizes are very

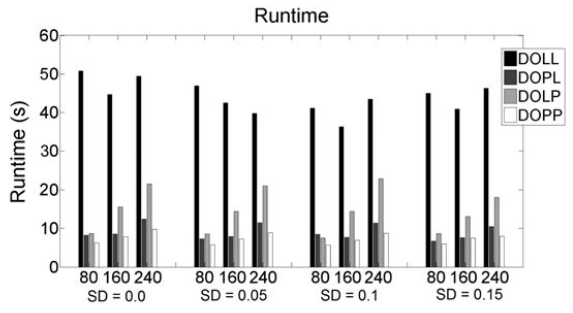

FIGURE 12. Contagion probability s.d. results, avg. = 0.3.

large for DOLL, causing very high runtimes due to the many iterations required to generate the observed action sets.

7.3. Contagion probability: standard deviation

Next we tested variations of the standard deviation of the normal distribution that the contagion probabilities on edges are drawn from. Specifically, we ran all four algorithms with the contagion probabilities drawn from normal distributions $\mathcal{N}(0.3, 0.0)$, $\mathcal{N}(0.3, 0.05)$, $\mathcal{N}(0.3, 0.1)$ and $\mathcal{N}(0.3, 0.15)$. These results, however, did not show statistically significant differences in the results when the standard deviation was changed under the particular parameter settings we tested. We only show the runtime results in Fig. 12 to support this claim, but the quality, action set size and support set size results all looked similarly homogeneous across the different standard deviations tested.

8. CONCLUSION

With increasingly informative data about interpersonal connections, principled methods can finally be applied to inform strategic interactions in social networks. Our work combines recent research in IBM, operations research and game-theoretic resource allocation to provide the first set of solution techniques for a novel class of security games with contagious actions. Experiments on real-world leadership and scale-free graphs reveal that a simple PAGERANK oracle can provide high quality solutions for graphs with clusters of highly interconnected nodes, whereas more sophisticated techniques can be very beneficial in sparsely connected graphs. The methods used herein are a first step into a new area of research in game-theoretic security with wide-ranging applications.

9. FUTURE DIRECTIONS

This type of maximization/mitigation scenario can be used to model a number of other domains that we hope to apply them to. For example, anti-vaccination groups have become a serious issue for health organizations to address [31]. By modeling the interaction as an adversarial information diffusion problem, the techniques here can help health organizations mitigate the impact of anti-vaccination propaganda. In political campaigns, candidates often attempt to disseminate negative information about their opponents to sway votes against them. Again, we can model this scenario with one party attempting to maximize the spread of this information while another party attempts to block the spread by disseminating its own news (e.g. their own negative propaganda, positive spin on the negative news, bigger news). In addition to the open theoretical questions for the existing model and algorithms such as runtime and quality guarantees, these new domains introduce novel challenges as we improve the fidelity of our models to fit these problems.

ACKNOWLEDGEMENTS

Any opinions, findings, conclusions or recommendations herein are those of the authors and do not reflect views of the United States Department of Homeland Security, or the University of Southern California, or CREATE.

FUNDING

This work was supported by the United States Department of Homeland Security through the National Center for Risk and Economic Analysis of Terrorism Events (CREATE) [2010-ST-061-RE0001].

REFERENCES

[1] Trusov, M., Bucklin, R.E. and Pauwels, K. (2009) Effects of word-of-mouth versus traditional marketing: findings from an internet social networking site. *J. Mark.*, **73**, 90–102.

[2] U.S. Dept. of the Army and U.S. Marine Corps (2007) The U.S. Army/Marine Corps Counterinsurgency Field Manual 3-24. University of Chicago Press.

[3] Hung, B.W.K. (2010) Optimization-based selection of influential agents in a rural Afghan social network. Masters Thesis, MIT Sloan School of Management.

[4] Howard, N.J. (2011) Finding optimal strategies for influencing social networks in two player games. Masters Thesis, MIT Sloan School of Management.

[5] Basilico, N. and Gatti, N. (2011) Automated Abstractions for Patrolling Security Games. *Proc. 25th AAAI Conf. Artificial Intelligence (AAAI)*, Menlo Park, CA, USA, August 7–11. AAAI Press.

[6] Jain, M., Korzhyk, D., Vanek, O., Conitzer, V., Pechoucek, M. and Tambe, M. (2011) A Double Oracle Algorithm for Zero-Sum Security Games on Graphs. *Proc. 10th Int. Conf. Autonomous Agents and Multiagent Systems (AAMAS) Volume 1*, Richland, SC, USA, pp. 327–334. International Foundation for Autonomous Agents and Multiagent Systems.

[7] Letchford, J. and Vorobeychik, Y. (2011) Computing Randomized Security Strategies in Networked Domains. *Proc. Workshop on Applied Adversarial Reasoning and Risk Modeling (AARM) at AAAI*, San Francisco, CA, USA. International Foundation for Autonomous Agents and Multiagent Systems.

[8] Paruchuri, P., Pearce, J.P., Marecki, J., Tambe, M., Ordóñez, F. and Kraus, S. (2008) Playing Games with Security: An Efficient Exact Algorithm for Bayesian Stackelberg Games. *Proc. 7th Int. Joint Conf. Autonomous Agents and Multiagent Systems (AAMAS)*, Richland, SC, USA, May, pp. 895–902. International Foundation for Autonomous Agents and Multiagent Systems.

[9] Conitzer, V. and Sandholm, T. (2006) Computing the Optimal Strategy to Commit to. *Proc. 7th ACM Conf. Electronic Commerce (EC)*, New York, NY, USA, pp. 82–90. ACM.

[10] Budak, C., Agrawal, D. and Abbadi, A.E. (2011) Limiting the Spread of Misinformation in Social Networks. *Proc. 20th Int. Conf. World Wide Web (WWW)*, New York, NY, USA, pp. 665–674. ACM.

[11] He, X., Song, G., Chen, W. and Jiang, Q. (2012) Influence Blocking Maximization in Social Networks Under the Competitive Linear Threshold Model. *Proc. 12th SIAM Int. Conf. Data Mining (SDM)*, Anaheim, CA, USA, April, pp. 463–474. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[12] Chen, W., Wang, C. and Wang, Y. (2010) Scalable Influence Maximization for Prevalent Viral Marketing in Large-Scale Social Networks. *Proc. 16th ACM SIGKDD Int. Conf. Knowledge Discovery and Data mining (KDD)*, New York, NY, USA, pp. 1029–1038. ACM.

[13] Kimura, M., Saito, K., Nakano, R. and Motoda, H. (2010) Extracting influential nodes on a social network for information diffusion. *Data Min. Knowl. Discov.*, **20**, 70–97.

[14] Halvorson, E., Conitzer, V. and Parr, R. (2009) Multi-Step Multi-Sensor Hider-Seeker Games. *Proc. 21st Int. Joint Conf. Artificial Intelligence (IJCAI)*, San Francisco, CA, USA, pp. 159–166. Morgan Kaufmann Publishers.

[15] Kempe, D., Kleinberg, J.M. and Tardos, É. (2003) Maximizing the Spread of Influence Through a Social Network. *Proc. 9th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD)*, New York, NY, USA, pp. 137–146. ACM.

[16] Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J.M. and Glance, N.S. (2007) Cost-Effective Outbreak Detection in Networks. *Proc. 13th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD)*, New York, NY, USA, pp. 420–429. ACM.

[17] Aspnes, J., Chang, K. and Yampolskiy, A. (2005) Inoculation Strategies for Victims of Viruses and the Sum-of-Squares Partition Problem. *Proc. 16th Annual ACM-SIAM Symp. Discrete Algorithms, SODA'05*, Philadelphia, PA, USA, pp. 43–52. Society for Industrial and Applied Mathematics.

[18] Aspnes, J., Rustagi, N. and Saia, J. (2007) Worm Versus Alert: Who Wins in a Battle for Control of a Large-Scale Network? *Proc. 11th Int. Conf. Principles of Distributed Systems, OPODIS'07*, Berlin, Heidelberg, December 17–20, pp. 443–456. Springer.

[19] Habiba, H., Yu, Y., Berger-Wolf, T.Y. and Saia, J. (2010) Finding Spread Blockers in Dynamic Networks. *Proc. 2nd Int. Conf. Advances in Social Network Mining and Analysis, SNAKDD'08*, Berlin, Heidelberg, pp. 55–76. Springer.

[20] Chen, P.-A., David, M. and Kempe, D. (2010) Better Vaccination Strategies for Better People. *Proc. 11th ACM Conf. Electronic Commerce (EC)*, New York, NY, USA, pp. 179–188. ACM.

[21] Kuhlman, C.J., Kumar, V.S.A., Marathe, M.V., Ravi, S.S. and Rosenkrantz, D.J. (2010) Finding Critical Nodes for Inhibiting Diffusion of Complex Contagions in Social Networks. *Proc. 2010 European Conf. Machine Learning and Knowledge Discovery in Databases: Part II, ECML PKDD'10*, Berlin, Heidelberg, pp. 111–127. Springer.

[22] Kumar, V.S.A., Rajaraman, R., Sun, Z. and Sundaram, R. (2010) Existence Theorems and Approximation Algorithms for Generalized Network Security Games. *Proc. 2010 IEEE 30th Int. Conf. Distributed Computing Systems, ICDCS'10*, Washington, DC, USA, pp. 348–357. IEEE Computer Society.

[23] Borodin, A., Filmus, Y. and Oren, J. (2010) Threshold Models for Competitive Influence in Social Networks. *Proc. 6th Int. Workshop on Internet and Network Economics (WINE)*, Berlin, Heidelberg, pp. 539–550. Springer.

[24] Bharathi, S., Kempe, D. and Salek, M. (2007) Competitive Influence Maximization in Social Networks. *Proc. 3rd Int. Workshop on Internet and Network Economics (WINE)*, San Diego, CA, USA, December 12–14, pp. 306–311. Springer.

[25] Hung, B.W.K., Kolitz, S.E. and Ozdaglar, A.E. (2011) Optimization-Based Influencing of Village Social Networks in a Counterinsurgency. *Proc. 4th Int. Conf. Social computing, Behavioral-cultural modeling and Prediction (SBP)*, Berlin, Heidelberg, pp. 10–17. Springer.

[26] McMahan, H.B., Gordon, G.J. and Blum, A. (2003) Planning in the Presence of Cost Functions Controlled by an Adversary. *Proc. 20th Int. Conf. Machine Learning (ICML)*, Menlo Park, CA, USA, pp. 536–543. AAAI Press.

[27] Brin, S. and Page, L. (1998) The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, **30**, 107–117.

[28] Clauset, A., Shalizi, C.R. and Newman, M.E.J. (2009) Power-law distributions in empirical data. *SIAM Rev.*, **51**, 661–703.

[29] Barabási, A.-L. and Albert, R. (1999) Emergence of scaling in random networks. *Science*, **286**, 509–512.

[30] Tsai, J., Nguyen, T.H. and Tambe, M. (2012) Security Games for Controlling Contagion. *Proc. 26th AAAI Conf. Artificial Intelligence (AAAI)*, Menlo Park, CA, USA, July 22–26. AAAI Press.

[31] Shetty, P. (2010) Experts concerned about vaccination backlash. *Lancet*, **375**, 970–971.