Catcher-Evader Games*

Yuqian Li, Vincent Conitzer, Dmytro Korzhyk
Department of Computer Science, Duke University
{yuqian, conitzer}@cs.duke.edu, dima.korzhyk@gmail.com

Abstract
Algorithms for computing game-theoretic solutions have recently been applied to a number of security domains. However, many of the techniques developed for compact representations of security games do not extend to Bayesian security games, which allow us to model uncertainty about the attacker’s type. In this paper, we introduce a general framework of catcher-evader games that can capture Bayesian security games as well as other game families of interest. We show that computing Stackelberg strategies is NP-hard, but give an algorithm for computing a Nash equilibrium that performs well in experiments. We also prove that the Nash equilibria of these games satisfy the interchangeability property, so that equilibrium selection is not an issue.

1 Introduction
Algorithms for computing game-theoretic solutions have long been of interest to AI researchers. In recent years, applications of these techniques to security have drawn particular attention. These applications include airport security [Pita et al., 2009a], the assignment of Federal Air Marshals to flights [Tsai et al., 2009], scheduling Coast Guard patrols [An et al., 2012], scheduling patrols on transit systems [Yin et al., 2012], and the list goes on. Game-theoretic techniques are natural in these domains because they involve parties with competing interests (though the games are usually not zero-sum), and the use of mixed (randomized) strategies to avoid being predictable to one’s opponent is desirable.

These applications have typically used a Stackelberg model where one player (the defender) commits to a mixed strategy first and the other (the attacker) then optimally responds to this mixed strategy. Formally, the defender (player 1) chooses a mixed strategy \( \sigma_1 \in \arg \max_\sigma_1 \max_{\sigma_2} u_1(\sigma_1, \sigma_2) \), where \( \text{BR}^2(\sigma_1) \) is the set of best responses to \( \sigma_1 \) for player 2 (i.e., the responses that maximize player 2’s utility). This is in contrast to the more standard solution concept of Nash equilibrium, where both players play a mixed strategy in such a way that each plays a best response to the other—that is, a pair \((\sigma_1, \sigma_2)\) with \( \sigma_1 \in \text{BR}^1(\sigma_2) \) and \( \sigma_2 \in \text{BR}^2(\sigma_1) \). Arguably, the Stackelberg solution is well motivated in contexts where the attacker can learn the defender’s strategy over time by repeated observation, whereas if this is not the case perhaps the Nash solution is better motivated. It is known that under certain conditions in security games, Stackelberg strategies are also Nash equilibrium strategies [Korzhyk et al., 2011b].

Initial work in these domains modeled uncertainty over attacker preferences using the formalism of Bayesian games, assigning probabilities to different types of attackers. This included the original work at the airport at Los Angeles [Paruchuri et al., 2008]. However, subsequent research, which started to focus on compact representations of security games, mostly did not consider Bayesian games. In this paper, we introduce a more general framework that can capture such Bayesian security games, and study the computation of Stackelberg and Nash solutions in them (which in such games generally do not coincide). Our framework can also model certain types of test games in which a tester randomly chooses questions from a fixed database of questions [Li and Conitzer, 2013]. We show that computing a Stackelberg strategy is strongly NP-hard, but give an algorithm for computing Nash equilibria that combines and expands on earlier techniques in both security and test games. While we have been unable to show that our algorithm is guaranteed to require at most polynomially many iterations, it requires few iterations in experiments.

More benefits of our framework are listed below: (1) Our notation for Catcher-Evader games, once one becomes familiar with it, greatly simplifies analysis of those games, especially as it concerns utilities. For example, our notation expresses the utility delta of a target, which is often the crucial consideration in these uniquely optimal; this is why ties for the attacker are broken in favor of the defender.

*The full version of this paper is available at http://arxiv.org/abs/1602.01896. Dmytro contributed to this paper while he was a Ph.D. student at Duke University.

1 Generally, if the attacker is indifferent among multiple targets, the defender can slightly modify her strategy to make any one of

2 Note that these games are completely different from pursuit-evasion (or cops-and-robbers) games [Parsons, 1978; Borie et al., 2009]. Those games involve dynamically chasing another player on a graph. Our games, in contrast, occur in a single period, and concern the computation of an optimal random assignment.
cital quantity, directly as \( d \), rather than as a difference (e.g., \( u_2^i - u_1^j \)). (2) Our additional parameters \( a, b, c \) allow richer utility functions that security games did not capture previously. For example, targets may have different costs to defend even if the attacker does not attack them. Previous security game definitions always assumed no cost (or the same cost) if the attacker does not attack. (3) It lets us swap the roles of defenders and attackers. Therefore, we can also directly compute the attacker’s strategy as well as the defender’s strategy, an example of which is computing the tester’s strategy in test games. (4) Its connection between security games and test games brings enormous convenience for algorithm design. Previously, separate algorithms had to be designed for them, but now we can design a single algorithm for both. Moreover, we can potentially apply known algorithms for each of these game families to the other. For example, the aforementioned Nash equilibria algorithm combines techniques for security games (progressively increasing defender or attacker resources) and test games (using network flow to reallocate attacker or evader resources). (5) Besides security games and test games, it can also capture other interesting scenarios where resources must be assigned to different targets by two competing parties. For example, two companies, an incumbent and an entrant, might be allocating capital to different markets; the entrant may wish to evade the incumbent and build up market share, while the incumbent wants to catch the entrant to drive the latter out of business.

2 Notation

We model a Catcher-Evader game (CE game) as a game between one catcher and multiple evaders. Since we assume that the evaders do not care about each other’s actions, this is equivalent to a Bayesian game between a single-typed catcher and an evader with multiple types. Also, as we will show in section 3.3, the roles of catcher and evader can be swapped. Hence, our model also captures games between one evader and multiple catchers.

We represent a CE game by \((N, \Psi, r, \ell, a, b, c, d)\), where \( N = \{0,1,\ldots,n\} \) is the set of players and \( \Psi \) is the set of sites (e.g., the targets in a security game or the questions in a test game). We fix \( N = \{0\} \) to be the set of sites, for example, the expected number of resources that this type contributes with probability \( \ell_{i,\psi} \). We require \( \ell_{i,\psi} \leq 1 \). Also, as we will show in section 3.3, the roles of catcher and evader can be swapped. Hence, our model also captures games between one evader and multiple catchers.

We represent a CE game by \((N, \Psi, r, \ell, a, b, c, d)\), where \( N = \{0,1,\ldots,n\} \) is the set of sites (e.g., the targets in a security game or the questions in a test game). We fix \( N = \{0\} \) to be the set of sites, for example, the expected number of resources that this type contributes with probability \( \ell_{i,\psi} \). We require \( \ell_{i,\psi} \leq 1 \). Also, as we will show in section 3.3, the roles of catcher and evader can be swapped. Hence, our model also captures games between one evader and multiple catchers.

Player \( i \) can assign a resource amount of at most \( \ell_{i,\psi} \) to site \( \psi \) in \( \Psi \). For example, we might set \( \ell_{i,\psi} = 1 \) to indicate that \( i \) can assign at most one resource to \( \psi \), or we might set \( \ell_{i,\psi} = 1/2 \) to indicate that, in a Bayesian game, a type \( i \) that appears with probability \( 1/2 \) can assign at most a single resource to \( \psi \) if it appears, and therefore his marginal contribution of probability mass to \( \psi \) is at most \( 1/2 \). Generally, \( r_i \leq \sum_{\psi \in \Psi} \ell_{i,\psi} \) so the player has to make a nontrivial decision about which site gets more of the resource amount and which one gets less.

Finally, the utility is encoded by \( a, b, c, d \) as follows. Let \( x \) be the strategy profile where \( x_{i,\psi} \) is the resource amount that player \( i \) puts on site \( \psi \). For convenience, we denote \( x_{\Sigma,\psi} = \sum_{i=1}^{n} x_{i,\psi} \) as the combined resource amount that all \( n \) evaders put on site \( \psi \). Then the utility is \( \sum_{\psi \in \Psi} \left[ (b_{0,\psi} + d_{0,\psi} x_{\Sigma,\psi}) x_{i,\psi} + a_{0,\psi} x_{\Sigma,\psi} + c_{0,\psi} \right] \) for the catcher and \( \sum_{\psi \in \Psi} \left[ (b_{i,\psi} + d_{i,\psi} x_{i,\psi}) x_{i,\psi} + a_{i,\psi} x_{0,\psi} + c_{i,\psi} \right] \) for evader \( i \). Here, \( b \) is the base utility for a player to put a resource at a site, and \( d \) is the utility change that results from putting a resource at that site when the opponent puts a resource there as well. Since \( c \) (constant utility) is not affected by any player’s strategy, we can ignore it (or let \( c = 0 \)) without affecting our analysis of both Stackelberg strategies and Nash equilibrium. Finally, \( a \) (for alternating utility) is the utility that a player receives when the opponent puts a resource at that site; the former player cannot affect this. Hence, for Nash equilibrium (but not for Stackelberg strategies), we can simply drop \( a \) (or let \( a = 0 \)). We require \( \sum_{\psi \in \Psi} x_{i,\psi} = r_i \) for feasibility, as well as \( d_{0,\psi} > 0 \) and \( d_{i,\psi} < 0 \) for \( i \in N^+ \) so that the catcher wants to catch the evader while the evader wants to evade.

For convenience, we define \( x_{i,\psi} = x_{\Sigma,\psi} \) and \( x_{i,\psi} = x_{0,\psi} \) for \( i \in N^+ \). Then, we define \( \mu_{i,\psi} = (b_{i,\psi} + d_{i,\psi} x_{i,\psi}) x_{i,\psi} + a_{i,\psi} x_{0,\psi} + c_{i,\psi} \) as the per-resource utility of player \( i \) on site \( \psi \). That is, it is the increase in utility she experiences from putting one more resource there. So, player \( i \)’s utility gained from site \( \psi \) can be written as \( u_{i,\psi}(x) = \mu_{i,\psi} x_{i,\psi} + a_{i,\psi} x_{i,\psi} + c_{i,\psi} \). In a best-response strategy, player \( i \) should have a utility threshold \( \theta_i \) such that (1) for all \( \psi \) with \( \mu_{i,\psi}(x) > \theta_i \), the player maximizes the resource amount it puts there \((x_{i,\psi} = \ell_{i,\psi}\)) and (2) for all \( \psi \) with \( \mu_{i,\psi}(x) < \theta_i \), the player puts no resource amount there \((x_{i,\psi} = 0\)).

Incidentally, note that if we do not require \( d_{0,\psi} > 0 \) and \( d_{i,\psi} < 0 \) for \( i \in N^+ \), then \( a, b, c, d \) can represent any utility function of the form \( \sum_{\psi \in \Psi} f(x_{i,\psi} x_{i,\psi}) \) where \( f \) is a quadratic polynomial without factors \( x_{i,\psi}^2 \) or \( x_{i,\psi}^2 \).

In Table 1, we summarize all symbols for reference.

3 Reducing Games to CE Games

In this section, we show how the framework of CE games let us capture several game families studied previously in the literature, namely security games and test games.

3.1 Security Games

A general definition of security games was given by [Kiekintveld et al., 2009]. That work considered only a single attacker resource; an attacker with multiple attacker resources was considered by [Korzhyk et al., 2011a]. More generally still, we can consider a Bayesian game in which there
is uncertainty about the type of the attacker. (Some of the earliest work in this line of research concerned Bayesian games [Paruchuri et al., 2008; Pita et al., 2009b], but the games were relatively small and so the techniques did not exploit the structure of security games.) We now define multi-resource Bayesian security games and show how to reduce intractability of security games.

There are a defender and an attacker. The latter has an unknown type $i \in \{1, \ldots, n\}$. An attacker of type $i$ occurs with probability $p_i$. There are $m$ targets $t_1, t_2, \ldots, t_m$. An attacker of type $i$ can attack $r_i$ distinct targets while the defender can defend $r_d$ distinct targets. A player’s utility is the sum of its utility over all targets. If an attacker of type $i$ attacks an undefined target $t$, it obtains utility $u_i^d(t)$ (and the defender obtains utility $u_i^c(t)$). If it attacks a defended (covered) target $t$, it obtains utility $u_i^c(t)$ (and the defender obtains utility $u_i^d(t)$). Both players obtain utility 0 from $t$ if $t$ is unattacked.

Now, we can reduce this to the following CE game $(N, \Psi, r', a', b', c') = (N, d')$ (see Table 2 for an example of utility reduction): $N = \{0, 1, 2, \ldots, n\}, \Psi = \{t_1, t_2, \ldots, t_m\}, r' = r_d, a' = p r_i (i \in N), r_0 = 1, \ell_0 = p_1 (\psi \in \Psi, i \in N^+), a'_i = u_i^d(\psi), a'_{H_i} = 0, b'_i = u_i^c(\psi), b'_{H_i} = 0, c'_i = u_i^c(\psi) - u_i^d(\psi), c'_{H_i} = u_i^c(\psi) - u_i^d(\psi) (i \in N^+)$.  

Note that in the original security game, $r$ consists of natural numbers and a pure strategy would put either 0 or 1 resources on each site. In the CE game, the strategy profile $x_i, \psi$ corresponds to the marginal probability that player $i$ puts a resource on $\psi$. Because resources can only be assigned to single targets, we can always use Birkhoff-von Neumann decomposition [Birkhoff, 1946] to generate a valid mixed strategy of the original security game with these marginals (see also [Korzhyk et al., 2010]).

### Table 1: Symbols used for CE games.

<table>
<thead>
<tr>
<th>N</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N^+$</td>
<td>Set of players ${0, 1, \ldots, n}$</td>
</tr>
<tr>
<td>$\Psi$</td>
<td>Set of sites (e.g., targets in security games)</td>
</tr>
<tr>
<td>$r_i$</td>
<td>Resource of player $i$</td>
</tr>
<tr>
<td>$\ell_i, \psi$</td>
<td>Resource limit player $i$ can put on site $\psi$</td>
</tr>
<tr>
<td>$a_{i, \psi}$</td>
<td>Alternating utility of player $i$ on site $\psi$</td>
</tr>
<tr>
<td>$b_{i, \psi}$</td>
<td>Base utility of player $i$ on site $\psi$</td>
</tr>
<tr>
<td>$c_{i, \psi}$</td>
<td>Constant utility of player $i$ on site $\psi$</td>
</tr>
<tr>
<td>$d_i, \psi$</td>
<td>Utility change (delta) of player $i$ on site $\psi$</td>
</tr>
<tr>
<td>$x_i, \psi$</td>
<td>Amount of resource $i$ puts on $\psi$ (strategy)</td>
</tr>
<tr>
<td>$x_{-i}, \psi$</td>
<td>Amount of resource $i$’s opponent puts on $\psi$</td>
</tr>
<tr>
<td>$\mu_i, \psi$</td>
<td>Per-resource utility of $i$ on $\psi$: $b_{i, \psi} + d_i, \psi x_{-i}, \psi$</td>
</tr>
<tr>
<td>$u_i, \psi$</td>
<td>Utility of $i$ on $\psi$: $\mu_i, \psi x_i, \psi + a_{i, \psi} x_{-i}, \psi + c_{i, \psi}$</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>Utility threshold of player $i$</td>
</tr>
</tbody>
</table>

### Table 2: Example of how a security game’s utility specification for a target $t$ is converted to a CE game’s utility specification for a site $\psi = t$. In this table, we let $u_i^d(t) = u_i^d(\psi), u_i^c(t) = u_i^c(\psi)$ for convenience.

<table>
<thead>
<tr>
<th>Player</th>
<th>Security Game $u_i^c(t)$</th>
<th>CE Game $u_i^c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Def $(i = 0)$</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Att 1 $(i = 1)$</td>
<td>-5</td>
<td>5</td>
</tr>
<tr>
<td>Att 2 $(i = 2)$</td>
<td>-9</td>
<td>10</td>
</tr>
</tbody>
</table>

### 3.2 Testing Games

Testing games were recently studied by [Li and Conitzer, 2013]. In that work, only test takers that do not fail any questions pass the test; therefore, it does not matter whether a test taker fails 1 question or 100. In contrast, we consider a variant—arguably more realistic—in which the losses and gains the players experience are additive across questions. We call this variant “scored tests”, which captures cases like the GRE, the TOEFL, and most college exams at school. It allows us to bypass the (co)NP-hardness results for computing the best test strategies from [Li and Conitzer, 2013]. On the other hand, the transformation to a zero-sum game described in that paper no longer works in this context.

Formally, a test game is a 2-player game between a tester and a test taker. The tester is uncertain about the test taker’s type $i \in \{1, 2, \ldots, n\}$, but she knows that a test taker of type $i$ occurs with probability $p_i$. The tester has a pool of questions $Q$, from which $t$ questions will be chosen to form a test $T \subseteq Q$. For a test taker of type $i$, a given subset $H_i \subseteq Q$ of questions is hard and he will not be able to solve them unless he memorizes their answers (or writes them on a cheat sheet). However, he can memorize at most $m_q$ questions, so if the tester randomizes over the choice of $T$, there is a good chance that most questions in $T$ have not been memorized. We denote the set of questions $i$ chooses to memorize as $M_i \subseteq Q$.  

So far, everything is identical to the games defined by [Li and Conitzer, 2013]. Now we introduce a question score $s_q$ for each $q \in Q$. If a test taker fails to solve $q$ in the test, $s_q$ is deducted from his score. Hence the test taker’s utility is $u_i(T, M_i) = -\sum_{q \in T \cap H_i \setminus M_i} s_q$.

We also introduce a weight $w_q$ for each question, representing how important the tester thinks it is to find out whether the test taker can solve $q$. This may or may not be equal to $s_q$. The tester’s utility is then $u_i(T, M_i) = \sum_{q \in T \cap H_i \setminus M_i} w_q$. Here, $v_i$ denotes the tester’s assessment of the importance of test taker type $i$. For example, it might be more (or less) important to figure out the true score of a bad test taker (with large $H_i$) than that of a good one. We reduce this game to the CE game $(N, \Psi, r, a, b, c = 0, d)$, where $N = \{0, 1, 2, \ldots, n\}, \Psi = Q, r_0 = t, r_i = p v_i m_i (i \in N^+), r_0 = 1, \ell_i = p v_i (i \in N^+), q \in Q = \Psi, a_0 = 0, b_0 = w_q \sum_{i \in H_i, q \in T \cap H_i} p v_i, d_0 = -w_q, a_i = -s_q$ for $q \in H_i, a_0 = 0$ for $q \not\in H_i, b_i = 0 (i \in N^+), d_{i, q} = s_q r_{i, q}$ for $q \in H_i, d_{i, q} = 0$ for $q \not\in H_i$.  

---

3Section 6 of [Kiekintveld et al., 2009] also allowed resources to be assigned to schedules of multiple targets, which quickly leads to NP-hardness [Korzhyk et al., 2010].

4A constant $\sum_{q \in \Psi} s_q$ can be added to $u_i(T, M_i)$ to obtain the usual nonnegative test scores.
6 Computing Nash Equilibrium

The interchangeability established in the previous section provides good motivation for computing a Nash equilibrium in this domain. In this section, we provide an algorithm for doing so. The algorithm is significantly more involved than earlier algorithms, notably requiring a min-cost-flow subroutine. This is perhaps surprising as earlier algorithms—e.g., the one by [Korzhyk et al., 2011a] for computing a Nash equilibrium in non-Bayesian security games with multiple attacker resources—do not need to do so. However, in the next subsection, we show it is possible to reduce the problem of finding a minimum-cost fractional matching to our games, suggesting that this complexity is inherent in the problem. We have been unable to either give a polynomial upper bound on the number of iterations of our algorithm (each iteration takes...
Algorithm 1 Compute a Nash equilibrium of a given CE game \((N, \Psi, r, b, d)\). Let \((x_0, y_0)\) be the initial solution.

1. \(x_{i} \leftarrow 0\) \forall i \in N; \(y_{i} \leftarrow \Psi\) \triangleright Initialize
2. for i \in N do
3. \(x_{i} \leftarrow \min\{x_{i}, T_{i} - \sum_{j \in N} d_{ij} x_{j}\}\) \triangleright Update \(x_{i}\)
4. for all \(v \in V\) do
5. \(x_{v} \leftarrow \min\{x_{v}, \min\{T_{v}, \sum_{u \in \Psi} x_{u}\}\}\) \triangleright Update \(x_{v}\)
6. for all \(i \in I\) do
7. \(z_{i} \leftarrow \sum_{j \in N} x_{j}\) \triangleright Increase \(z_{i}\)
8. \(\text{report} (z_{i}, x_{i}, y_{i}, x_{v})\) \triangleright Increase \(x_{v}\)
9. end for
10. end for
11. end while

Algorithm 2 Given CE game \((N, \Psi, r, b, d)\), and an NE \(x_{0}\), we can find an optimal solution. Let \(\alpha\) be a Nash equilibrium.

1. \(\alpha_{0} \leftarrow \alpha\) \triangleright Initial solution
2. for \(i \in I\) do
3. \(\alpha_{i} \leftarrow \min\{\alpha_{i}, T_{i} - \sum_{j \in N} d_{ij} \alpha_{j}\}\) \triangleright Update \(\alpha_{i}\)
4. for all \(v \in V\) do
5. \(\alpha_{v} \leftarrow \min\{\alpha_{v}, \min\{T_{v}, \sum_{u \in \Psi} \alpha_{u}\}\}\) \triangleright Update \(\alpha_{v}\)
6. for all \(i \in I\) do
7. \(\Delta \leftarrow \min\{\Delta, (\alpha_{0} - \alpha_{0})/\gamma_{0}\}\) \triangleright Reduce \(\Delta\)
8. end for
9. end for
10. end for

Algorithm 3 We are given a CE game \((N, \Psi, r, b, d)\), and an NE \(x_{0}\) where no negative cycles exist among active edges \(A\) in the residual graph. This procedure either strictly increases some of the \(x_{i,0}\) (maintaining NE), or fails.

1. \(\gamma_{1} \leftarrow 0\) \triangleright Initialize
2. for \(i \in I\) do
3. \(\Delta \leftarrow \min\{\gamma_{i}, 2b_{i,0} \cdot d_{i,0} - \gamma_{i}\}\) \triangleright Update \(\Delta\)
4. for each \(\Delta \in [0, \Delta_{f}]\) do
5. \(\text{return} (\Delta, x_{0}, y_{0}, x_{v})\) \triangleright Update \(\Delta\)
6. end for
7. end for
8. end for

Algorithm 4 Given CE game \((N, \Psi, r, b, d)\) and an NE \(x_{0}\) resulting from a failed run of Algorithm 3, we reallocate evader resources among active edges \(A\) using max flow. This strictly decreases \(b_{i,0}\).

1. \(\Delta \leftarrow \min\{\gamma_{i}, 2b_{i,0} \cdot d_{i,0} - \gamma_{i}\}\) \triangleright Update \(\Delta\)
2. for each \(\Delta \in [0, \Delta_{f}]\) do
3. \(\text{return} (\Delta, x_{0}, y_{0}, x_{v})\) \triangleright Update \(\Delta\)
4. end for
5. end for
6. \(\text{return} (\Delta, x_{0}, y_{0}, x_{v})\) \triangleright Update \(\Delta\)
7. end for
8. end for

6.1 Reducing from Min-Cost Matching

We show that computing an NE in CE games (even with single-resource evaders, i.e., \(\forall i \in N, \ell_i = \ell_i \geq r_i\)) is as hard as computing minimum-cost fractional matching—a common type of flow problem—suggesting that we are unlikely to find a linear-time algorithm. Some of the ideas in the reduction, in particular having costs in the graphs corresponding to the logarithms of utility change rates \(d\), will also appear in the algorithm we present later.

Theorem 4. Computing a Nash equilibrium of a CE game is as hard as computing a minimum-cost fractional matching of a weighted bipartite graph. Specifically, if there is a Nash equilibrium finding algorithm that runs in \(O(T)\) time, where \(T\) is the input size of the CE game, then we can solve the matching problem in \(O(T^3)\) time, where \(T\) is the input size of the bipartite graph. So computing a CE is not possible in linear time unless there is a linear algorithm for matching.

Proof. We reduce the matching instance to a CE game whose NE can be straightforwardly translated back to an optimal solution to the matching instance. The reduction takes linear time, resulting in the bound in the theorem.

2Of course, network flow problems have an integrality property—but not if the input is fractional, as we allow here.

Let the matching instance be on a bipartite graph with vertices \(U = \{1, \ldots, n\}\) and \(V\). Each vertex \(v\) has a capacity \(k_v\) with \(\sum_{u \in U} k_u = \sum_{v \in V} k_v\). Each edge \((u, v)\) has a capacity \(\kappa(u, v)\) and a cost \(w(u, v)\). Our goal is to saturate all the vertices’ capacities at minimum cost. Equivalently, this is a flow problem where \(\sum_{u \in U} k_u\) flow must be pushed across the bipartite graph at minimum cost.

We construct a CE game \((N, \Psi, r, b, c, 0 = d)\) where \(N = \{0, 1, 2, \ldots, n\}\) (so \(N^+ = U\)), \(\Psi = V\), \(r_0 = 1\) and \(r_u = k_u\) for all \(u \in U\), \(b_0 = 0\) and \(b_u = \kappa(u, v)\) for all \(u \in U\) and \(v \in V\), \(b_{v,0} = 0\) for all \(v \in V\), \(d_0 = 1/k_v\) and \(d_u = w(u, v)\) for all \(u \in U\) and \(v \in V\). First, we note that the game has a feasible strategy for the evaders if and only if the matching problem has a feasible solution. This is because a feasible strategy \(x_{u,0}\) corresponds exactly to a feasible matching solution.

Second, \(x_{0,v} > 0\) must hold for all \(v\). Otherwise, because \(b_{u,0} = 0\) and \(d_u < 0\), all evaders will strictly prefer targets with \(x_{0,v} = 0\); but then the catcher would not be best-responding, because \(b_{u,0} = 0\) and \(d_u < 0\).

Finally, we show that the NE \(x\) must constitute an optimal solution to the matching problem. That is, if we let \(W = \sum_{u \in U, v \in V} x_{u,v} \cdot w(u, v)\), then \(W\) is the minimum cost in the matching problem. Suppose not; then, when interpreting \(x_{u,v}\) as a flow, in the residual graph of that flow, a negative cycle exists. Let that cycle be \(u_1 \rightarrow v_1 \rightarrow u_2 \rightarrow v_2 \rightarrow \ldots \rightarrow u_m \rightarrow v_m \rightarrow u_1\) with \(\sum_{1 \leq k \leq m} (w(v_k, u_{k+1}) - w(v_k, u_k) < 0, x_{u_k,v_k} < \kappa(u_k, v_k)\), and \(x_{u_k,v_k} > 0\) for all \(1 \leq k \leq m\) (letting \(u_{m+1} = u_1\)). Recall that \(\theta_{u}\) is
the *per-resource utility* threshold for evader $u$. So, $\mu_{u,k,v_k} = x_{0,v_k}d_{u,v_k} \leq \theta^{-1}_{u,k}$ and $\mu_{u,k+1,v_k} = x_{0,v_k}d_{u,k+1,v_k} \geq \theta^{-1}_{u,k+1}$. Equivalently, $x_{0,v_k}[d_{u,k+1,v_k} - \max(0, x_{0,v_k}d_{u,v_k})] \leq \theta^{-1}_{u,k}$ and $\theta^{-1}_{u,k} \geq \theta^{-1}_{u,k+1}$ because $d_{u,v_k} < 0$. It then follows that

$$\prod_{k=1}^{m} x_{0,v_k}[d_{u,k+1,v_k} - \max(0, x_{0,v_k}d_{u,v_k})] \leq \prod_{k=1}^{m} \theta^{-1}_{u,k} \leq \prod_{k=1}^{m} \max(0, x_{0,v_k}d_{u,v_k})$$

which implies $\prod_{k=1}^{m} x_{0,v_k}[d_{u,k+1,v_k} - \max(0, x_{0,v_k}d_{u,v_k})] \leq \prod_{k=1}^{m} \max(0, x_{0,v_k}d_{u,v_k})$ because $x_{0,v_k} > 0$ for all $v$ and thus $\theta^{-1}_{u,k} > 0$ for all $u \in U$. Taking the logarithm on both sides, we obtain $\sum_{1 \leq k \leq m}(w(u_k, v_k) - w(u_{k+1}, v_k)) \geq 0$, contradicting the negative cycle assumption $\sum_{1 \leq k \leq m}(w(u_k, v_k) - w(u_{k+1}, v_k)) < 0$. \hfill \square

### 6.2 Algorithm

We now present our algorithm. The algorithm works by initializing the catcher’s resource amount to 0 and gradually increasing it to $\nu^*$, maintaining the equilibrium throughout. The same high-level approach was used by an earlier paper [Korzhyk et al., 2011a] for the case of a single attacker type (evader) with multiple resources, obtaining an efficient algorithm there. However, the case with multiple evaders is significantly more involved. The (polynomial-time) algorithm given in [Korzhyk et al., 2011a] did not require anything like a network-flow subroutine, whereas the reduction in section 6.1 suggests that this is necessary when we have multiple evaders. Our algorithm also incorporates ideas used in the contest of test games [Li and Conitzer, 2013], specifically the binary search and max-flow techniques used there.

We first introduce some notation. Let $B_i = \{\psi \mid \mu_{i,\psi} = \theta_i\}$ be the boundary sites of player $i$. Let $B_i^+ = \{\psi \in B_i \land x_i,\psi > 0\}$ be evader $i$’s (in $N^+$)’s positive boundary sites, whose resource amount can be reduced. Let $B_0 = \{\psi \in B_0 \land x_0,\psi < \ell_0,\psi\}$ be the catcher’s open boundary sites, to which more resources could be assigned. Let the *active edges* be $A = \{(i,\psi) \mid i \in N^+, \psi \in \Psi, \psi \in B_i\}$. The main algorithm is Algorithm 1. After initialization, the algorithm repeatedly loops through Algorithms 2, 3, and 4, which together provably (eventually) increase the catcher’s (allocated) resource amount while maintaining equilibrium. Algorithm 2 ensures that a “no negative cycle” property holds by solving a min-cost flow problem (since the residual flow of Algorithm 2 ensures that a “no negative cycle” property holds). Taking the logarithm on both sides, we obtain $\sum_{1 \leq k \leq m}(w(u_k, v_k) - w(u_{k+1}, v_k)) \geq 0$, contradicting the negative cycle assumption $\sum_{1 \leq k \leq m}(w(u_k, v_k) - w(u_{k+1}, v_k)) < 0$. \hfill \square

**Lemma 2.** Algorithm 2 maintains Nash equilibrium without changing $\mu_{i,\psi}$ or $\theta_i$ for any $i \in N^+$ and $\psi \in \Psi$. As a result, the active edges $A$ are also unchanged.

**Proof.** Algorithm 2 only reallocates $x_i,\psi$ among $A$, hence the evaders necessarily continue to best respond. Both the original flow and the min-cost flow are required to saturate all edges $(\psi, \tau)$. Hence $x_i,\psi$ is unchanged for all $\psi \in \Psi$ and the catcher necessarily continues to best respond. Each evader $i$’s $\mu_{i,\psi}$ is clearly unchanged as $x_0,\psi$ is untouched by Algorithm 2. By the definition of $\theta_i = \min_{\psi \in B_i} x_i,\psi > 0$. So no matter how we reallocate, some $\psi \in B_i$ must remain positive. Hence, $\theta_i$ is unchanged because the $\mu_{i,\psi}$ are unchanged.

**Lemma 3.** In Algorithm 3, the evaders’ thresholds $\theta_i (i \in N^+) decrease at rate $\gamma_i$. That is, the algorithm decreases $\theta_i$ by $\Delta \cdot \gamma_i$.

We omit the proof of Lemma 3 to save space.

**Lemma 4.** After Algorithm 2, if Algorithm 3 successfully increases $x_0,\psi$, it maintains Nash equilibrium.

**Proof.** The catcher’s strategy remains a best response because Algorithm 3 does not change any evader’s strategy and the catcher only increases $x_0,\psi$ for which $\mu_{0,\psi} = \theta_0$. Thus we only have to check whether each evader’s strategy remains a best response.

By Lemma 3 and the notation $\mu^0, \theta^0, A^0$ defined in its proof, evaders are best-responding if and only if:

$$\forall i \in N^+, \psi \in \Psi \mid x_i,\psi > 0 \quad \mu_{i,\psi} = \mu^0_{i,\psi} + \Delta \cdot \gamma_i \cdot d_i,\psi \geq \theta^0_i - \Delta \cdot \gamma_i$$

$$\forall i \in N^+, \psi \in \Psi \mid x_i,\psi < \ell_i,\psi \quad \mu_{i,\psi} = \mu^0_{i,\psi} + \Delta \cdot \gamma_i \cdot d_i,\psi \geq \theta^0_i - \Delta \cdot \gamma_i$$

For $(i,\psi) \notin A^0$, line 25 of Algorithm 3 maintains the conditions above. Now consider $(i,\psi) \in A^0$. There, we have $\mu^0_{i,\psi} = \theta^0_i$, so we only need to check

$$\forall i \in N^+, \psi \in \Psi \mid x_i,\psi > 0 \Delta \cdot \gamma_i \cdot d_i,\psi \geq \Delta \cdot \gamma_i$$

$$\forall i \in N^+, \psi \in \Psi \mid x_i,\psi < \ell_i,\psi \Delta \cdot \gamma_i \cdot d_i,\psi \geq \Delta \cdot \gamma_i$$

If $x_i,\psi > 0$, a backward edge $(\psi, i)$ with weight $-w(i,\psi)$ exists in the residual graph. Hence $\text{dist}(i) \leq \text{dist}(\psi) - w(i,\psi)$, which implies $-e^{-\text{dist}(\psi) - w(i,\psi)} \geq -e^{-\text{dist}(i)}$. That is, $\gamma_i d_i,\psi \geq -\gamma_i$, and therefore $\Delta \cdot \gamma_i d_i,\psi \geq -\gamma_i$, because $\Delta \geq 0$.

If $x_i,\psi < \ell_i,\psi$, a forward edge $(i,\psi)$ with weight $w(i,\psi)$ exists in the residual graph. Hence $\text{dist}(\psi) \leq \text{dist}(i) + w(i,\psi)$, which implies $-e^{-\text{dist}(\psi) - w(i,\psi)} \leq -e^{-\text{dist}(i)}$. 

334
That is, \( \gamma_{\psi} d_{i, \psi} \leq -\gamma_i \), and therefore \( \Delta \cdot \gamma_{\psi} \cdot d_{i, \psi} \leq -\Delta \cdot \gamma_i \) because \( \Delta \geq 0 \), completing the proof.

**Lemma 5.** If Algorithm 3 fails, then Algorithm 4 strictly decreases \( \theta_0 \) while maintaining Nash equilibrium.

**Proof.** If Algorithm 3 fails, then for each \( \psi \in \hat{B}_0 \), there must be another site \( \psi' \in \Psi_{\text{reachable}} \cap \Psi_{\text{unincreasable}} \). That is, for each site \( \psi \in \hat{B}_0 \) to which the catcher could increase resource assignment, there is a path in the residual graph that goes from \( \psi \) to some site \( \psi' \in \Psi_{\text{unincreasable}} \) to which the catcher cannot increase resource assignment.

The site \( \psi' \) is, in contrast, a good site for evaders: if they put more resources there, the catcher cannot penalize them (because the catcher cannot increase its resources there). Formally, there are two cases for \( \psi' \): 1) \( x_{\psi, \psi'} = \ell_{\psi, \psi'} \); 2) \( x_{\psi, \psi'} < \theta_0 \). In the former case, evaders can increase their resource assignment there as much as possible because the catcher has hit the limit of what it can assign there. In the latter case, evaders can increase until \( \mu_{0, \psi'} \) meets \( \theta_0 \).

Therefore, for each \( \psi \in \hat{B}_0 \), evaders can move a positive amount of resource from that site \( \psi \) to some \( \psi' \notin \hat{B}_0 \) using the corresponding residual graph path. The evaders continue to best-respond because the residual graph only includes active edges \( A \). The catcher’s best-response condition is maintained by decreasing \( \mu_{0, \psi} \) by the same positive amount \( \Delta \) (the number found by the binary search in Algorithm 4) for each \( \psi \in \hat{B}_0 \), with evaders increasing all edges leaving \( \psi \), and not letting \( \mu_{0, \psi} \) (\( \psi \in \hat{B}_0 \)) decrease below \( \mu_{0, \psi'} \) (\( \psi \notin \hat{B}_0 \)) (line 14 of Algorithm 4). Because \( \mu_{0, \psi} \) has strictly decreased for each \( \psi \in \hat{B}_0 \) and has not become lower than any \( \mu_{0, \psi'} \) (\( \psi' \notin \hat{B}_0 \)), \( \theta_0 \) must have strictly decreased (by \( \Delta \)).

**Lemma 6.** After Algorithm 4, either a new site \( \psi \) that previously had \( \mu_{0, \psi} < \theta_0 \) now has \( \mu_{0, \psi} = \theta_0 \), or the next run of Algorithm 3 will be successful.

**Proof.** Suppose that the next run of Algorithm 3 fails. Then for each \( \psi \in \hat{B}_0 \), a path exists in the residual graph (after the run of Algorithm 4) from \( \psi \) to some \( \psi' \notin \hat{B}_0 \), as argued in the proof of Lemma 5. Suppose, for the sake of contradiction, that none of the edges entering \( \tau \) were saturated during the run of Algorithm 4 (for the value of \( \Delta \) resulting from the binary search), i.e., \( (\forall \psi' \notin \hat{B}_0) \cdot f(\psi, \tau) < \kappa(\psi, \tau) \). Then, in Algorithm 4, we could have increased \( \Delta \) further, resulting in a contradiction. Therefore, there exists at least one \( \psi' \notin \hat{B}_0 \) for which the run of Algorithm 4 made it the case that \( f(\psi', \tau) = \kappa(\psi', \tau) \). For that \( \psi' \), the total amount of evader resource \( x_{\Sigma, \psi'} \) was increased by \( \kappa(\psi', \tau) = (\theta_0^0 - \Delta - \mu_{0, \psi'})/d_{0, \psi'} \) (where superscript 0 denotes the value prior to the run of Algorithm 4). It follows that \( \mu_{0, \psi'} = \mu_{0, \psi} + \theta_0^0 - \Delta - \mu_{0, \psi} = \theta_0^0 - \theta_0 = \theta_0^0 \), while \( \mu_{0, \psi} < \theta_0 \).

**Lemma 7.** Each site \( \psi \) enters \( \hat{B}_0 \) at most once; hence, \( \hat{B}_0 \) changes at most \( 2|\Psi| \) times.

**Proof.** Only Algorithm 4 can change \( \mu_{0, \psi} \) or \( \theta_0 \). That algorithm ensures that \( \mu_{0, \psi} \) decreases at the same rate for all \( \psi \in \hat{B}_0 \), and stops decreasing if a new \( \psi' \mu_{0, \psi'} < \theta_0 \) enters \( \hat{B}_0 \). So a site \( \psi \) can only leave \( \hat{B}_0 \) by being saturated \( (x_{\psi, \psi'} = \ell_{\psi, \psi'}) \). Since we never decrease \( x_{\psi', \psi'} \) during the algorithm, saturated sites \( \psi \) can never enter \( \hat{B}_0 \) again.

**Lemma 8.** Algorithm 3 runs successfully at most \( 2|\Psi| \cdot 3^n \) times.

**Proof.** By Lemma 7, we only have to argue that there are at most \( 3^n \) successful runs before \( \hat{B}_0 \) changes. Now we assume that \( \hat{B}_0 \) remains unchanged and check how many runs there can be.

We classify an edge \( (i, \psi) \) where \( i \in N^+ \), \( \psi \in \Psi \) into 3 cases: either (1) superior \( \mu_{i, \psi} > \theta_i \) (above threshold), or (2) inferior \( \mu_{i, \psi} < \theta_i \) (below threshold), or (3) active \( \mu_{i, \psi} = \theta_i \) (on threshold).

Let \( \phi \) be a vector of length \( n|\Psi| \) where each element \( \phi_e \in \{1, 2, 3\} \) denotes edge \( e \)’s case number. We will show that \( \phi \) changes after each successful run, and it will not repeat if \( \hat{B}_0 \) remains unchanged. Hence the lemma holds.

We first show that for a fixed \( \phi \), Algorithm 4 always returns the same \( x \) (assuming that \( \hat{B}_0 \) remains unchanged).

Recall that in Algorithm 4, we proved that if the final flow saturates any edge \( (\psi', \tau) \) that enters sink \( \tau \), then \( \psi' \) will newly enter \( \hat{B}_0 \). Hence if \( \hat{B}_0 \) remains unchanged, we can ignore the capacity of those edges entering \( \tau \). Also, with \( \hat{B}_0 \) fixed, the set of edges leaving source \( \sigma \) and their capacities are fixed. Therefore, the resulting value of Algorithm 4 is solely...
determined by the edges between \( N^+ \) and \( \Psi \), which is fixed by \( \phi \).

We then conclude that if there were two Algorithm 3 runs that have the same resulting \( \phi \), then between those two runs, there must be no Algorithm 4 run that has positive \( \Delta \) which strictly decreases \( \theta_0 \). Otherwise, we would have two Algorithm 4 runs (after those two Algorithm 3 runs) with the same \( \phi \), where the latter run has strictly smaller \( \theta_0 \) (note that \( \theta_0 \) never increases), contradicting that the returning \( x \) of Algorithm 4 is completely determined by \( \phi \).

Therefore, if there were two Algorithm 3 runs that result in the same \( \phi \), all Algorithm 4 runs between those two runs must do nothing (\( \Delta = 0 \)). Hence, \( x_{\Sigma, \Psi} \) is unchanged between those two runs for all \( \psi \), because only Algorithm 4 can change \( x_{\Sigma, \Psi} \).

Now consider graph \( G_1 \) which extends graph \( G \) in the Algorithm 2 by assuming that all edges are active (\( A = N^+ \cup \Psi \)). That is, for each edge leaving source \( \sigma \), its capacity is \( \kappa(\sigma, i) = r_i \); for each edge entering sink \( \tau \), its capacity is \( \kappa(\psi, \tau) = x_{\Sigma, \Psi} \); the weight and capacity of edge \((i, \psi)\) is \( w(i, \psi) = \log(-d_{i, \psi}) \), \( \kappa(i, \psi) = \ell(i, \psi) \) for all \( i \in N^+ \), \( \psi \in \Psi \).

Then we consider the original set of active edges \( A \) and make \( G_2 \) by revising the following edges in \( G_1 \): for each edge \( e = (i, \psi) \) that is not active, set its weight \( w(i, \psi) = \infty \) if \( e \) is inferior, and \( w(i, \psi) = -\infty \) if \( e \) is superior.

Clearly, running min-cost flow on \( G_2 \) would result in the same \( x \) as running Algorithm 2 because we fix non-active edges’ flow by setting their weights to \( \infty \) or \(-\infty \). Moreover, for the same flow, the shortest path in the residual graph of \( G_2 \) is exactly the same as the residual graph of \( G \). Hence when we talk about flow or distance, they could refer to either \( G \) or \( G_2 \). But when we talk about the total cost of the flow, we are referring to \( G_1 \), as \( G_2 \) has infinity cost edges and \( G \) only has active edges. That is, the cost of a flow \( x \) is \( \sum_{e \in N^+ \cup \psi \in \Psi} x_{e, \psi} \log(-d_{e, \psi}) \).

Each time that Algorithm 3 runs successfully but the Algorithm 1 continues (\( r_0 \) is not used up), some constraint of line 23 in Algorithm 3 must be tight, which means that a new edge \((i, \psi)\) must be entering the active edge set \( A \) (otherwise we will either continue increasing \( x_{0, \psi} \) or change \( B_0 \)). For that newly active edge, if \( x_{i, \psi} = 0 \), then \( \text{dist}(\psi) > \text{dist}(i) + w(i, \psi) \) must be true (recall that \( \text{dist} \) is the shortest distance from \( \psi^* \) in the residual graph) because: \( \mu_{i, \psi} \)'s decrease rate \( \gamma_{\psi}(-d_{i, \psi}) \) must be slower than \( \theta_i \)'s decrease rate \( \gamma_i \). Similarly, if \( x_{i, \psi} = \ell_i, \psi \), then \( \text{dist}(i) > \text{dist}(\psi) - w(i, \psi) \) must be true. Hence by adding \( (i, \psi) \), either \( \text{dist} \) has to decrease or a negative cycle exists which leads to a decrease in flow cost (w.r.t. \( G_1 \)).

Also note that \( \text{dist} \) and flow cost are completely determined by \( \phi \) if \( x_{\Sigma, \Psi} \) is fixed for all \( \psi \in \Psi \). Hence \( \phi \) cannot repeat since each \( \phi \) change has either to either decrease flow cost, or maintain the flow cost and decrease \( \text{dist} \). This completes our proof.

With this, we obtain an exponential upper bound on the algorithm’s runtime. Because the algorithm only terminates when the number of catcher resources has reached \( r_0 \), and we have shown that the algorithm maintains equilibrium throughout, this also establishes the algorithm’s correctness.

**Theorem 5.** Algorithm 1 computes a Nash equilibrium of the given CE game in \( 2|\Psi| + 4|\Psi|^3 |\Psi|^{|\Psi|} \) iterations.

**Proof.** Because of Lemmas 2, 4, and 5, Nash equilibrium is always maintained. So we only need to prove that the algorithm stops after at most \( 2|\Psi| + 4|\Psi|^3 |\Psi|^{|\Psi|} \) iterations. We have at most \( 2|\Psi|^3 |\Psi|^{|\Psi|} \) iterations where Algorithm 3 runs successfully, by Lemma 8. Each failed iteration must either be followed by a successful iteration, or \( B_0 \) is changed (by Lemma 6). Lemma 7 ensures that \( B_0 \) can be changed at most \( 2|\Psi| \) times. So overall there can be at most \( 2 \times 2|\Psi| + 4|\Psi|^3 |\Psi|^{|\Psi|} \) iterations. \( \square \)

6.3 Experiments

Although Theorem 5 only gives an exponential bound on the number of iterations, the number of iterations in Algorithm 1 grows much more slowly—about linearly—in our experiments, as shown in Figure 1(c). In our experiments, parameters \( r, b, d \) are generated uniformly at random from \( \{1, \ldots, 10\} \) or \( \{-10, \ldots, -1\} \). Each instance of size \( n \) has \( n \) evaders and \( n \) sites; for each \( n \) we solved 20 instances.

The running time per iteration is provably polynomial and it grows about cubically as Figure 1(b) shows. That is consistent with how the network flow subroutine (which is used in each of our iterations) typically scales.

An alternative approach to solving for these Nash equilibria would be to construct the normal form of the game and use a standard NE-finding algorithm. This approach, however, is doomed regardless of the precise choice of algorithm, because the size of the normal form blows up exponentially, as shown in Figure 1(d).

Note that we implemented our algorithm completely in Python (including the min-cost network flow subroutine). Hence there is room to further improve the performance by using C/C++, or some optimized network flow libraries.

7 Future Research

The obvious direction for future research is whether our algorithm in fact provably runs in polynomial time—and, if not, whether there is another algorithm that does. The algorithm’s success in experiments gives us hope that the answer to at least one of these two questions is positive, but we have not been able to decisively answer them. There are several indications that the question is inherently difficult to answer. The earlier algorithm for multiple attacker resources in the non-Bayesian case and the proof of the polynomial bound on its runtime [Korzhyk et al., 2011a] were already quite involved, and we showed that the Bayesian case requires us to deal with additional challenging issues (Subsection “Reducing from Min-Cost Matching”). Still, we believe that the importance of solving Bayesian security games would justify the deviation of further effort to resolving this question, as well as to extending these techniques to related problems.

---

6The residual graph might be different for the same \( \phi \), depending on what the min-cost flow is; but \( x \) is the additional max-flow applied to the min-cost flow, so what really determines \( x \) is \( \phi \).
Acknowledgments

We thank Ronald Parr for his contributions to our early discussions about Bayesian security games. We are also thankful for support from ARO under grants W911NF-12-1-0550 and W911NF-11-1-0332, NSF under awards IIS-0953756, IIS-1527434, CCF-1101659, and CCF-1337215, and a Guggenheim Fellowship. Part of this research was done while Conitzer was visiting the Simons Institute for the Theory of Computing.

References


