

Catcher-Evader Games

Paper 739

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 simulations. 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 [12], the assignment of Federal Air Marshals to flights [14], scheduling Coast Guard patrols [1], scheduling patrols on transit systems [15], 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 \( \sigma_1 \in \arg \max_{\sigma_1} \max_{\sigma_2 \in \text{BR}_2(\sigma_1)} u_1(\sigma_1, \sigma_2) \), 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 [8].

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 [11]. 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 testing games in which a tester randomly chooses questions from a fixed database of questions [9]. 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 testing 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 simulations.

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 quantity, directly as \( d \), rather than as a difference (e.g., \( u_i^e - u_i^r \)). (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

---

1Generally, if the attacker is indifferent among multiple targets, the defender can slightly modify her strategy to make any one of these uniquely optimal; this is why ties for the attacker are broken in favor of the defender.

2Note that these games are completely different from pursuit-evasion (or cops-and-robbers) games [10, 3]. 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.
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 testing games. (4) Its connection between security games and testing 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 catcher resources) and testing games (using network flow to reallocate attacker or evader resources). (5) Besides security games and testing 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 6.1, 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 testing game). We fix \( 0 \in N \) to be the catcher (e.g., the defender in a security game), and \( N^+ = \{1, 2, \ldots, n\} \) to be the set of evaders (e.g., the multiple types of attackers in a security game). Player \( i \in N \) has available a total resource amount of \( r_i \in \mathbb{R}^{\geq 0} \). For example, we might set \( r_i = 1 \) to indicate that \( i \) has only one resource, or we might set \( r_i = 1/2 \) to indicate that, in a Bayesian game, a type \( i \) that appears with probability 1/2 has only a single resource, and therefore the expected number of resources that this type contributes is 1/2. This resource amount can be split fractionally across the sites, for example, 1/3 could be assigned to one site and 2/3 to another. (This would typically correspond to assigning a single resource to the former site with probability 1/3.) Player \( i \) can assign a resource amount of at most \( \ell_{i, \psi} \in \mathbb{R} \) to site \( \psi \in \Psi \). For example, we might set \( \ell_{i, \psi} = 1 \) to indicate that \( i \) can assign at most a single 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 he 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 evaders put on site \( \psi \). Then the utility is \( \sum_{\psi \in \Psi} \left[ (b_0 \cdot x_{\Sigma, \psi} + d_0 \cdot x_{2, \psi}) x_{0, \psi} + a_0 \cdot x_{2, \psi} \right] + c_0 \) for the catcher and \( \sum_{\psi \in \Psi} \left[ (b_i \cdot x_{i, \psi} + d_i \cdot x_{\psi, \psi}) x_{i, \psi} + a_i \cdot x_{0, \psi} + c_i \right] \) 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 \( d_0 > 0 \) and \( d_i < 0 \) for \( i \in N^+ \) so that the catcher wants to catch the evader while the evader wants to evade.

For convenience, we define \( x_{-0, \psi} = x_{2, \psi} \) and \( x_{-i, \psi} = x_{0, \psi} \) for \( i \in N^+ \). Then, we define \( \mu_{i, \psi} = (b_i \cdot x_{i, \psi} + d_i \cdot x_{\psi, \psi}) \) 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 \( u_i(x) = \mu_{i, \psi} x_{i, \psi} + a_i x_{-i, \psi} + c_i \). 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 \( (x_{i, \psi} = 0) \). (There is no requirement for the case \( \mu_{i, \psi}(x) = \theta_i \).) The value of \( \theta_i \) is not necessarily unique, so for definiteness, let \( \theta_i = \max_{\psi \in \Psi \land x_{i, \psi} > 0} \mu_{i, \psi} \) and \( \theta_i = \min_{\psi \in \Psi \land x_{i, \psi} > 0} \mu_{i, \psi} \).

Incidentally, note that if we do not require \( d_0 > 0 \) and \( d_i < 0 \) 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.

<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>Per-resource 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_{\Sigma, \psi} )</td>
<td>Sum of all evaders’ resource on ( \psi )</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 )</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 1: Symbols used for CE games.
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 $u^i_0(t) = u^i_d(t)$, $u^i_u(t) = u^i_u(t)$ for convenience.

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 testing games.

3.1 Security Games

A general definition of security games was given by [5]. That work considered only a single attacker resource; an attacker with multiple attacker resources was considered by [7]. 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 [11, 13], 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 them to CE games. Note that in our definition, a resource is assigned to a single target.3

There are a defender and an attacker. The latter has 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 $d_i$ distinct targets. A player’s utility is the sum of its utility over all targets. If an attacker of type $i$ attacks an undefended target $t$, it obtains utility $u^i_d(t)$ (and the defender obtains utility $u^i_u(t)$). 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 $(N, \Psi, r', a', b', c', d')$ (see Table 2 for an example of utility reduction): $N = \{0, 1, 2, \ldots, n\}, \Psi = \{t_1, t_2, \ldots, t_m\}, r'_i = r_i, a'_i = a_i, b'_i = b_i, c'_i = c_i, d'_i = d_i$, $\ell_{a'\psi} = 1$, $u^i_d(\psi) = u^i_d(\psi)$, $u^i_u(\psi) = u^i_u(\psi)$, $u^i_c(\psi) = 0$, $d'_i = u^i_d(\psi) - u^i_u(\psi)$ ($i \in N^+$).

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 [2] to generate a valid mixed strategy of the original security game with these marginals (see also [6]).

3.2 Testing Games

Testing games were recently studied by [9]. 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 course exams at school. It allows us to bypass the (co)NP-hardness results for computing the best testing strategies from [9]. On the other hand, the transformation to a zero-sum game described in that paper no longer works in this context.

Formally, a testing 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 $Q_i$ from which $t$ questions will be chosen to form a test $T \subseteq Q$ (|$T$| = $t$). 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 memorized $M_i \subseteq Q$ (|$M_i$| = $m_q$).

So far, everything is identical to the games defined by [9]. Now we introduce a question score $s_q$ for each question $q \in Q$. 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 $u_i(T, M_i) = -\sum_{q \in T \cap H_i \setminus M_i} s_q$.4 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 $u_i(T, M_i) = v_i \sum_{q \in T \cap H_i \setminus M_i} w_q$. 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 to the CE game $(N, \Psi, r, a, b, c = 0, d) = (0, 1, 2, \ldots, n), \Psi = Q, r_0 = t, r_i = p_i \cdot m_i$ ($i \in N^+$), $\ell_{a'\psi} = 1$, $b'_i = b_i$, $c'_i = c_i$, $d'_i = d_i$, $u^i_d(\psi) = u^i_d(\psi)$, $u^i_u(\psi) = u^i_u(\psi)$, $u^i_c(\psi) = 0$, $\ell_{b'\psi} = u^i_d(\psi) - u^i_u(\psi)$ ($i \in N^+$).

Note that 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 [2] to generate a valid mixed strategy of the original security game with these marginals (see also [6]).

3.3 Swapping Roles

The reduction from testing games has one issue: the utilities change at rates $d_0 > 0, d_i > 0$ ($i \in N^+$) but CE games require $d_0 < 0, d_i < 0$ ($i \in N^+$). In a sense, the tester is an evader who wants to evade by asking questions that are not memorized by the test taker; but as we have defined them, in CE games, player 0 is a catcher.

We handle this by redefining player 0’s resources to their opposites. That is, we focus on which questions she does not memorize.4 A constant $\sum_{q \in T} s_q$ can be added to $u_i(T, M_i)$ to obtain the usual nonnegative test scores.
test. Hence, the modified $x'_0,q$ will be the marginal probability that she does not test $q$ (i.e., $q \notin T$).

In general, we can swap roles between catchers and evaders (i.e., negate $d$) by rewriting CE game $(N, \Psi, r, a, b, c, d)$ as CE game $(N, \Psi, r', a', b', c', d')$: $r'_0 = -r_0 + \sum_{\psi \in \Psi} T_{0,\psi}, \ r'_i = r_i$ $(i \in N^+), \ell'_i,\psi = \ell_i,\psi$ $(i \in N), \ a'_i,\psi = a_0,\psi + d_0,\psi \ell_0,\psi, c'_i,\psi = c_0,\psi + b_0,\psi \ell_0,\psi, b'_i,\psi = -b_0,\psi \ell_0,\psi, d'_i,\psi = -a_0,\psi \ell_0,\psi, c'_i,\psi = c_0,\psi + a_0,\psi \ell_0,\psi, b'_i,\psi = b_0,\psi \ell_0,\psi.

Hence, the utilities are exactly the same as in the original game. As previously mentioned, $c$ does not affect our game-theoretic analysis. However, it is essential for establishing these equations so we can swap roles. Of course, after the transformation, we can freely drop $c'$. Table 3 shows an example of a testing game and role swapping.

Table 3: Example of a testing game and role swapping.

<table>
<thead>
<tr>
<th>$q$</th>
<th>$x_{0,q}$</th>
<th>$b_{1,q}$</th>
<th>$c_{1,q}$</th>
<th>$d_{1,q}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>test</td>
<td>1</td>
<td>-0.5</td>
<td>-0.5</td>
<td>0</td>
</tr>
<tr>
<td>don't test</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>test</td>
<td>0</td>
<td>-0.5</td>
<td>-0.5</td>
<td>0</td>
</tr>
<tr>
<td>don't test</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

4 Complexity of Stackelberg Strategies

Theorem 1. If there is only one evader who can put all resources on any single site ($\forall \psi \in \Psi, \ell_1,\psi \geq r_1$), then catcher Stackelberg strategies can be computed in polynomial time.

The proof of Theorem 1 (in the full version of this paper) uses a by now fairly standard linear program technique.

In contrast, it has been shown that computing Stackelberg strategies in a multi-resource security game (even with only a single type, i.e., non-Bayesian) is (weakly) NP-hard [7]. Hence, by our reduction of such security games to CE games, even if the CE game has only one evader ($n = 1$), it is (weakly) NP-hard to compute Stackelberg strategies if we allow $\ell_1,\psi < r_1$ (so the evader/attacker will put resources on multiple sites).

Next, we show that even if $\ell_{i,\psi} \geq r_i$ for all $i \geq 1$, it is strongly NP-hard to compute Stackelberg strategies if we allow $n > 1$. This corresponds to the case of a Bayesian security game in which each attacker has only a single resource. Note that the initial LAX airport paper [11] assumed a Bayesian security game with a single attacker resource. To our best knowledge, no hardness result has been given for computing Stackelberg strategies of such games. Also, unlike the known weak NP-hardness result for multiple resources, this rules out pseudopolynomial-time algorithms. The proof is in the full version of this paper to save space.

Theorem 2. Computing Stackelberg strategies in Bayesian security games is strongly NP-hard even if each attacker type has only a single resource. Consequently, computing Stackelberg strategies in a Catcher-Evader game is strongly NP-hard (if $n > 1$), even if $\ell_{i,\psi} \geq r_i$ for all $i \in N^+$. (This result is tight in the sense that this problem is also in NP.)

5 Interchangeability of NE

We now move on to studying Nash equilibria. In general, a downside of the Nash equilibrium concept is that Nash equilibria can fail interchangeability; if one player plays according to one Nash equilibrium and the other according to another, the result may not be a Nash equilibrium. However, it has been shown that interchangeability of Nash equilibria is guaranteed in security games and testing games under certain conditions [8, 7, 9]. We now show that this also holds for CE games. The key lemma and theorem are shown below. Their proofs are in the full version to save space.

Lemma 1. For each site $\psi$, either $x_{0,\psi}$ is the same for all NE or $x_{1,\psi}$ is the same for all NE.

Theorem 3. The Nash equilibria (NE) of a Catcher-Evader game are interchangeable. That is, if $x$ and $x'$ are two Nash equilibrium strategy profiles, then so is $x''$ where $x''_{0,\psi} = x_{0,\psi}$ and $x''_{1,\psi} = x'_{1,\psi}$ ($i \in N^+$).

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 [7] 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 polynomial time), or any class of instances that results in superpolynomially many iterations. We only give an exponential upper bound. However, as we will show, in simulations few iterations suffice.

6.1 Reducing from Min-Cost Matching

We show that computing an NE in CE games (even with single-resource evaders, i.e., $\forall i \in N^+, \ell_{i,\psi} \geq r_i$) is as hard as computing minimum-cost fractional matchings—a common type of flow problem—suggesting that we are unlikely to find a very efficient (e.g., 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 $T(I)$ time, where

$\text{(if } n > 1\text{), even if } \ell_{i,\psi} \geq r_i \text{ for all } i \in N^+. \text{ (This result is tight in the sense that this problem is also in NP.)}$

$\text{Of course, network flow problems have an integrality property—but not if the input is fractional, as we allow here.}$
Algorithm 1: Compute a Nash equilibrium of a given CE game $(N, \Psi, r, b, d)$, and then solve the matching problem where the vertices' capacities at minimum cost. Equivalently, this is the bipartite graph at minimum cost. The reduction takes linear time, resulting in the bound in the theorem.

Algorithm 2: Given CE game $(N, \Psi, r, b, d)$ and an NE $x$, reallocate the evaders' resources $x_v$ across active edges $A$ using min-cost flow. This procedure ensures that no negative cycle exists among active edges in the residual graph of $x$, NE is maintained as we only reallocate across $A$.

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

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.

Let the matching instance be on a bipartite graph with vertices $U = \{1, \ldots, n\}$ and $V$. Each vertex $v$ has a capacity $\kappa_v$, with $\sum_{u \in U} \kappa_u = \sum_{v \in V} \kappa_v$. 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 $\sum_{u \in U} \kappa_u$ flow must be pushed across the bipartite graph at minimum cost.

We construct a CE game $(N, \Psi, r, b, c, 0, 0)$ where $N = \{0, 1, 2, \ldots, n\}$ (so $N^+ = U$), $\Psi = V$, $r_0 = 1$ and $r_u = \kappa_u$ for all $u \in U$, $r_v = 1$ and $r_u, u = \kappa(u, v)$ for all $u \in U$ and $v \in V$, $b_i = 0$ for all $i \in N, v \in V$, $b_0, v = 1/\kappa_v$ and $d_u, u = -w(u, v)$ 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 $x_{u,v}$ corresponds exactly to a feasible matching solution.

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

Finally, we show that the NE $x$ must constitute an optimal solution to the matching problem. That is, if we let $W = \sum_{u \in U, v \in V} x_{u,v} w(u, v)$ 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 $u_1 \rightarrow v_1 \rightarrow u_2 \rightarrow \ldots \rightarrow u_m \rightarrow v_m \rightarrow u_1$ with $\sum_{1 \leq k \leq m} (w(u_k, v_k) - w(u_{k+1}, v_k)) < 0$, $x_{u_k, v_k} = \kappa(u_k, v_k)$, and $x_{u_{k+1}, v_k} > 0$ for all $1 \leq k \leq m$ (letting $u_{m+1} = u_1$). Recall that $\theta_v$ is the per-resource utility threshold for evader $v$. So, $\mu_{u_k, v_k} = x_{u_k, v_k} - \theta_{u_k}$ and $\mu_{u_{k+1}, v_k} = x_{u_{k+1}, v_k} - \theta_{u_{k+1}}$.

Equivalently, $x_{u_k, v_k} - \theta_{u_k} \leq |\theta_{u_k}|$ and $|\theta_{u_k}| \leq x_{u_k, v_k} - \theta_{u_{k+1}}$ because $d_{u_v, u} < 0$. It then follows that

$$\sum_{k=1}^{m} x_{u_k, v_k} |d_{u_{k+1}, v_k}| \cdot |\theta_{u_k}| \leq \sum_{k=1}^{m} x_{u_k, v_k} |d_{u_{k+1}, v_k}| \cdot |\theta_{u_{k+1}}|$$

which implies $\sum_{1 \leq k \leq m} |d_{u_{k+1}, v_k}| \leq \sum_{1 \leq k \leq m} |d_{u_{k+1}, v_k}|$, because $x_{u_k, v_k} > 0$ for all $v$ and thus $|\theta_v| > 0$ 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 $\sum_{1 \leq k \leq m} (w(u_k, v_k) - w(u_{k+1}, v_k)) < 0$.

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 $r_0$, maintaining the equilibrium throughout. The same high-level approach was used by an earlier paper [7]
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 [7] 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 context of testing games [9], 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 \( B_i^+ = \{ \psi \in B_i \wedge x_{i,\psi} > 0 \} \) be the evader \( i \)’s positive boundary sites, whose resource amount can be reduced. Let \( B_0 = \{ \psi \in B_0 \wedge x_{0,\psi} < \ell_{0,\psi} \} \) be the catcher’s open boundary sites, to which more resources could be assigned. Let the active edges be \( A = \{ (i, \psi) \mid i \in \mathbb{N}^+, \psi \in \Psi, \psi \in B_i \} \).

The main algorithm is Algorithm 1. After initializing, 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 a min-cost flow cannot have a negative cycle). Here, the relationship between the evader’s best responses and the min-cost flows “no negative cycle” property is similar to the reduction from min-cost matching that we gave earlier. Given that no negative cycle remains, Algorithm 3 then attempts to increase the catchers resource amount—that is, for each \( \psi \) it attempts to increase \( x_{0,\psi} \)—without breaking the evaders best-response conditions. However, Algorithm 3 can still fail to increase the catcher’s resource amount even without negative cycles. If so, we call Algorithm 4, which will either allow the next run of Algorithm 3 to strictly increase the catchers resource amount, or change the open boundary sites \( B_0 \) (which provably cannot happen too often). Specifically, if Algorithm 3 failed to increase the catcher’s resource amount, we have to reroute evaders resources among their best-response sites, in a way that strictly decreases the catchers utility threshold \( \theta_0 \). Such rerouting must also maintain the catchers best-response condition. For this we use max-flow and binary search: first, we binary search on \( \Delta \), the decrease in \( \theta_0 \); then, we calculate each edges rerouting capacity using \( \Delta \), and see whether a max-flow can saturate all capacities, thereby maintaining the best-response condition.

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

The proof the algorithm’s correctness and runtime bound is in the full version of the paper to save space.

### 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 \( r, b, \) and \( d \) 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++, and/or some optimized network flow libraries.

### 7 Future Research

The obvious direction for future research is resolving 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 simulations 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 algorithms for multiple attacker resources in the non-Bayesian case and the proof of the polynomial bound on its runtime [7] were already quite involved, and we showed that the Bayesian case requires us to deal with additional challenging issues (Subsection “Reducing from MinCost Matching”). Still, we believe that the importance of solving Bayesian security games would justify the devotion of further effort to resolving this question, as well as to extending these techniques to related problems.
References


