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 $\sigma_1^* \in \arg\max_{\sigma_1} \max_{s_2 \in \mathrm{BR}^2(\sigma_1)} u_1(\sigma_1, s_2)$,[1] where $\mathrm{BR}^2(\sigma_1)$ is the set of best responses to σ_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 (σ_1, σ_2) with $\sigma_1 \in \mathrm{BR}^1(\sigma_2)$ and $\sigma_2 \in \mathrm{BR}^2(\sigma_1)$. 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[2] 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 cru-

*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 these uniquely optimal; this is why ties for the attacker are broken in favor of the defender.

[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.

cial quantity, directly as d, rather than as a difference (e.g., $u_i^c - u_i^u$). (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 catcher 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 Ψ is the set of *sites* (e.g., the targets in a security game or the questions in a test 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 ψ, 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 ψ if he appears, and therefore his marginal contribution of probability mass to ψ 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 ψ. 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 ψ. Then the utility is $\sum_{\psi \in \Psi} [(b_{0,\psi} + d_{0,\psi} x_{\Sigma,\psi}) x_{0,\psi} + a_{0,\psi} x_{\Sigma,\psi} + c_{0,\psi}]$ for the catcher and $\sum_{\psi \in \Psi} [(b_{i,\psi} + d_{i,\psi} x_{0,\psi}) x_{i,\psi} + a_{i,\psi} x_{0,\psi} + c_{i,\psi}]$ 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 $x_{-0,\psi} = x_{\Sigma,\psi}$ and $x_{-i,\psi} = x_{0,\psi}$ for $i \in N^+$. Then, we define $\mu_{i,\psi} = (b_{i,\psi} + d_{i,\psi} \cdot x_{-i,\psi})$ as the *per-resource utility* of player i on site ψ. That is, it is the increase in utility she experiences from putting one more resource there. So, player i's utility gained from site ψ 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 θ_i such that (1) for all ψ 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 ψ 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 θ_i is not necessarily unique, so for definiteness, let $\theta_0 = \max_{\psi \in \Psi : x_{0,\psi} < \ell_{0,\psi}} \mu_{0,\psi}$ and $\theta_i = \min_{\psi \in \Psi : x_{i,\psi} > 0} \mu_{i,\psi}$ for $i \in N^+$.

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

	Description
N	Set of players $\{0, 1, \ldots, n\}$
N^+	Evaders $\{1, 2, \ldots, n\}$ (0 is the catcher)
Ψ	Set of *sites* (e.g., targets in security games)
r_i	Resource of player i
$\ell_{i,\psi}$	Resource *limit* player i can put on site ψ
$a_{i,\psi}$	Alternating utility of player i on site ψ
$b_{i,\psi}$	Base utility of player i on site ψ
$c_{i,\psi}$	Constant utility of player i on site ψ
$d_{i,\psi}$	Utility change (*delta*) of player i on site ψ
$x_{i,\psi}$	Amount of resource i puts on ψ (*strategy*)
$x_{\Sigma,\psi}$	Sum of all evaders' resource on ψ
$x_{-i,\psi}$	Amount of resource i's opponent puts on ψ
$\mu_{i,\psi}$	Per-resource utility of i on ψ: $b_{i,\psi} + d_{i,\psi}x_{-i,\psi}$
$u_{i,\psi}$	Utility of i on ψ: $\mu_{i,\psi}x_{i,\psi} + a_{i,\psi}x_{-i,\psi} + c_{i,\psi}$
θ_i	Utility threshold of player i

Table 1: Symbols used for CE games.

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 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 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 undefended target t, it obtains utility $u_i^u(t)$ (and the defender obtains utility $u_d^u(t)$). If it attacks a defended (covered) target t, it obtains utility $u_i^c(t)$ (and the defender obtains utility $u_d^c(t)$). Both players obtain utility 0 from t if t is unattacked.

Now, we can reduce this to the following CE game $(N, \Psi, r', a', b', c' = 0, d')$ (see Table 2 for an example of utility reduction): $N = \{0, 1, 2, \ldots, n\}, \Psi = \{t_1, t_2, \ldots, t_m\}, r'_0 = r_d, r'_i = p_i r_i \ (i \in N^+), \ell'_{0,\psi} = 1, \ell'_{i,\psi} = p_i \ (\psi \in \Psi, i \in N^+), a'_{0,\psi} = u_d^u(\psi), b'_{0,\psi} = 0, d'_{0,\psi} = u_d^c(\psi) - u_d^u(\psi), a'_{i,\psi} = 0, b'_{i,\psi} = u_i^u(\psi), d'_{i,\psi} = u_i^c(\psi) - u_i^u(\psi) \ (i \in N^+)$.

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 ψ. 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]).

[3] Section 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].

Player	Security Game		CE Game			
	$u_i^c(t)$	$u_i^u(t)$	$a_{i,t}$	$b_{i,t}$	$c_{i,t}$	$d_{i,t}$
Def ($i = 0$)	1	-10	-10	0	0	11
Att 1 ($i = 1$)	-5	5	0	5	0	-10
Att 2 ($i = 2$)	-9	10	0	10	0	-19

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 $u_0^c(t) = u_d^c(t), u_0^u(t) = u_d^u(t)$ for convenience.

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 course 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 $T \subseteq Q$ ($|T| = t$). For a test taker of type i, a given subset $H_i \subseteq Q$ of questions are 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_θ 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 as $M_i \subseteq Q(|M_i| = m_i)$

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 $u_i(T, M_i) = -\sum_{q \in T \cap H_i \setminus M_i} s_q$.[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 $u_i^t(T, M_i) = v_i \sum_{q \in T \cap H_i \setminus M_i} w_q$. 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 game to the CE game $(N, \Psi, r, a, b, c = 0, d)$ where $N = \{0, 1, 2, \ldots, n\}, \Psi = Q, r_0 = t, r_i = p_i v_i m_i \ (i \in N^+), \ell_{0,q} = 1, \ell_{i,q} = p_i v_i \ (i \in N^+, q \in Q = \Psi), a_{0,q} = 0, b_{0,q} = w_q \sum_{i: q \in H_i} p_i v_i, d_{0,q} = -w_q, a_{i,q} = -s_q$ for $q \in H_i, a_{i,q} = 0$ for $q \notin H_i, b_{i,q} = 0 \ (i \in N^+), d_{i,q} = s_q / r_{i,q}$ for $q \in H_i, d_{i,q} = 0$ for $q \notin H_i$.

[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.

(a) An example of test game players' utility on a question q

test taker's utility, tester's utility	don't test q	test q
don't memorize q	0, 0	-5, 4
memorize q	0, 0	0, 0

(b) Swapping roles for the above example test game

	Player	$a_{i,q}$	$b_{i,q}$	$c_{i,q}$	$d_{i,q}$
test q:	Tester ($i = 0$)	0	4	0	-4
$x_{0,q} = 1$	Test taker ($i = 1$)	-5	0	0	5
test q:	Tester ($i = 0$)	-4	-4	4	4
$x_{0,q} = 0$	Test taker ($i = 1$)	5	5	-5	-5

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

Similar to security games, the resulting strategy profile $x_{i,q}$ denotes the marginal probability that a player puts q on the test / memorizes q; again, the Birkhoff-von Neumann theorem allows us to obtain a strategy with these marginals.

3.3 Swapping Roles

The reduction from test 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* 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, Ψ, r, a, b, c, d) as CE game $(N, \Psi, r', a', b', c', d')$: $r'_0 = -r_0 + \sum_{\psi \in \Psi} \ell_{0,\psi}$, $r'_i = r_i$ $(i \in N^+)$, $\ell'_{i,\psi} = \ell_{i,\psi}$ $(i \in N)$, $a'_{0,\psi} = a_{0,\psi} + d_{0,\psi}\ell_{0,\psi}$, $c'_{0,\psi} = c_{0,\psi} + b_{0,\psi}\ell_{0,\psi}$, $b'_{0,\psi} = -b_{0,\psi}$, $d'_{0,\psi} = -d_{0,\psi}$, $a'_{i,\psi} = -a_{i,\psi}$, $d'_{i,\psi} = -d_{i,\psi}$, $b'_{i,\psi} = b_{i,\psi} + d_{i,\psi}\ell_{0,\psi}$, $c'_{i,\psi} = c_{i,\psi} + a_{i,\psi}\ell_{0,\psi}$

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 test game and how we swap roles in it.

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 [Korzhyk *et al.*, 2011a]. 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 [Paruchuri *et al.*, 2008] 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 test games under certain conditions [Korzhyk *et al.*, 2011b; 2011a; Li and Conitzer, 2013]. 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 ψ, either $x_{0,\psi}$ is the same for all NE or $x_{\Sigma,\psi}$ 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 $x''_{i,\psi} = x'_{i,\psi}$ $(i \in N^+)$.*

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 [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, Ψ, r, b, d) (a, c are ignored as discussed earlier).

$x_{i,\psi} \leftarrow 0$ $(\forall i \in N, \psi \in \Psi)$ ▷ Initialize
for $i \in N^+$ **do**
▷ Construct an initial NE with 0 resources for the catcher, by simply assigning evader i's resources to the sites with maximum $b_{i,\psi}$. (Recall that $\mu_{i,\psi} = b_{i,\psi} + d_{i,\psi} x_{-i,\psi}$ and $\theta_i = \max_{\psi : x_{i,\psi} < \ell_{i,\psi}} \mu_{i,\psi}$, and currently $x_{-i,\psi} = x_{0,\psi} = 0$.)
  for $\psi \in \Psi$ where $\mu_{i,\psi} = \theta_i$ **do**
    $x_{i,\psi} \leftarrow \min(\ell_{i,\psi}, r_i - \sum_{\psi \in \Psi} x_{i,\psi})$
  end for
end for
while $\sum_{\psi \in \Psi} x_{0,\psi} < r_0$ **do** ▷ Iteratively increase $x_{0,\psi}$
  Run Algorithm 2 (min-cost-flow)
  Run Algorithm 3 to weakly increase $x_{0,\psi}$ for all ψ
  Run Algorithm 4 (max-flow) if Algorithm 3 failed
end while

Algorithm 2 Given CE game (N, Ψ, r, b, d) and an NE x, reallocate the evaders' resources $x_{i,\psi}$ 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.

▷ Construct graph (V, E) with cost w and capacity κ
▷ σ and τ are the source and sink of our flow problem
$V \leftarrow \{\sigma\} \cup N^+ \cup \Psi \cup \{\tau\}$
$E \leftarrow \{\sigma\} \times N^+ \cup A \cup \Psi \times \{\tau\}$
Initialize $w \leftarrow 0, \kappa \leftarrow 0$
$w(i,\psi) \leftarrow \log(-d_{i,\psi})$ $(\forall (i,\psi) \in A)$
$\kappa(i,\psi) \leftarrow \ell_{i,\psi}$ $(\forall (i,\psi) \in A)$
$\kappa(\sigma, i) = \sum_{\psi : \mu_{i,\psi} = \theta_i} x_{i,\psi}$ $(\forall i \in N^+)$
$\kappa(\psi, \tau) = \sum_{i : \mu_{i,\psi} = \theta_i} x_{i,\psi}$ $(\forall \psi \in \Psi)$
▷ Reallocate according to the min-cost flow
$f \leftarrow$ min-cost σ-τ flow in the graph constructed above
$x_{i,\psi} \leftarrow f(i,\psi)$ $(\forall (i,\psi) \in A)$
return the reallocated x

Algorithm 3 We are given a CE game (N, Ψ, 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 $x_{0,\psi}$ (maintaining NE), or fails.

▷ All the graph computations below are based on the residual graph of the min-cost flow in Algorithm 2.
$\psi^* \leftarrow$ None
for $\psi \in \hat{B}_0$ **do**
  Compute single-source shortest paths from ψ
  Let $dist(v)$ be the shortest distance from ψ to v
  $\Psi_{reachable} \leftarrow \{\psi' \mid dist(\psi') < \infty\}$
  $\Psi_{unincreasable} \leftarrow \Psi \setminus \hat{B}_0$
  if $\Psi_{reachable} \cap \Psi_{unincreasable} = \emptyset$ **then**
    $\psi^* \leftarrow \psi$ ▷ $dist$ is shortest path from ψ^*
    break ▷ Increase $x_{0,\psi}$ starting from ψ^*
  end if
end for
if $\psi^* =$ None **then**
  return failure
end if
▷ Increase $x_{0,\psi}$ at rate γ_ψ
$\gamma_\psi \leftarrow e^{-dist(\psi)}$ $(\forall \psi \in \Psi)$
▷ Decrease threshold θ_i at rate γ_i
$\gamma_i \leftarrow e^{-dist(i)}$ $(\forall i \in N^+)$
$\Delta = (r_0 - \sum_{\psi \in \Psi} x_{0,\psi}) / \sum_{\psi \in \Psi} \gamma_\psi$
for $\psi \in \Psi$ **do** ▷ Reduce max feasible increase Δ
  if $\gamma_\psi > 0$ **then**
    $\Delta \leftarrow \min(\Delta, (\ell_{0,\psi} - x_{0,\psi})/\gamma_\psi)$
  end if
  ▷ Consider how much we can increase before an inactive edge should become active
  for $i \in N^+$ where $\mu_{i,\psi} \neq \theta_i$ **do**
    $\Delta' \leftarrow (\mu_{i,\psi} - \theta_i)/(\gamma_\psi \times (-d_{i,\psi}) - \gamma_i)$
    if $\Delta' > 0$ **then**
      $\Delta \leftarrow \min(\Delta, \Delta')$
    end if
  end for
end for
$x_{0,\psi} \leftarrow x_{0,\psi} + \Delta \cdot \gamma_\psi$ $(\forall \psi \in \Psi)$

Algorithm 4 Given CE game (N, Ψ, r, b, d) and an NE x resulting from a failed run of Algorithm 3, we reallocate evader resources among active edges A using max flow. This strictly decreases θ_0.

$\bar{\Delta} \leftarrow \min_{\psi \in \Psi : \theta_0 > \mu_{0,\psi}} \theta_0 - \mu_{0,\psi}$
for each value of Δ in a binary search for the max $\Delta \in [0, \bar{\Delta}]$ such that G (below) has a max flow saturating all edges from source σ **do**
  $G \leftarrow$ min-cost flow's residual graph in Algorithm 2
  Remove the edges connected to σ or τ in G
  for $\psi \in \Psi$ **do**
    if $\psi \in \hat{B}_0$ **then**
      Add edge (σ, ψ) to G
      Capacity $\kappa(\sigma, \psi) \leftarrow \Delta / d_{0,\psi}$
    else
      Add edge (ψ, τ) to G
      if $\mu_{0,\psi} \geq \theta_0$ **then**
        Capacity $\kappa(\psi, \tau) \leftarrow \infty$
      else
        $\kappa(\psi, \tau) \leftarrow (\theta_0 - \Delta - \mu_{0,\psi}) / d_{0,\psi}$
      end if
    end if
  end for
  ▷ Determine whether the max flow on G saturates all edges from σ to see how to proceed with the binary search
  Run max flow on G
end for
$f \leftarrow$ max σ-τ flow of G for max feasible Δ
▷ If $f(\psi, i) > 0$, then $f(i, \psi) = -f(\psi, i)$
$x_{i,\psi} \leftarrow x_{i,\psi} + f(i,\psi)$ $(\forall (i,\psi) \in A)$

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 experiments 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[5] matchings—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 $T(I)$ time, where I is the input size of the CE game, then we can solve the matching problem in $T(O(I'))$ time, where I' is the input size of the bipartite graph. So computing a NE 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.

Let the matching instance be on a bipartite graph with vertices $U = \{1, \ldots, n\}$ and V. Each vertex v has a capacity κ_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 $(N, \Psi, r, a, b, c = 0, d)$ where $N = \{0, 1, 2, \ldots, n\}$ (so $N^+ = U$), $\Psi = V$, $r_0 = 1$ and $r_u = \kappa_u$ for all $u \in U$, $\ell_{0,v} = 1$ and $\ell_{u,v} = \kappa(u, v)$ for all $u \in U$ and $v \in V$, $b_{i,v} = 0$ for all $i \in N, v \in V$, $d_{0,v} = 1/\kappa_v$ and $d_{u,v} = -e^{w(u,v)}$ 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, $x_{0,v} > 0$ must hold for all v. Otherwise, because $b_{u,v} = 0$ and $d_{u,v} < 0$, all evaders will strictly prefer targets with $x_{0,v} = 0$; 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 $u_1 \to v_1 \to u_2 \to v_2 \to \ldots \to u_m \to v_m \to u_1$ 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 θ_u is

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

the *per-resource utility* threshold for evader u. So, $\mu_{u_k,v_k} = x_{0,v_k} d_{u_k,v_k} \leq \theta_{u_k}$ and $\mu_{u_{k+1},v_k} = x_{0,v_k} d_{u_{k+1},v_k} \geq \theta_{u_{k+1}}$. Equivalently,
$x_{0,v_k}|d_{u_{k+1},v_k}| \leq |\theta_{u_{k+1}}|$ and $|\theta_{u_k}| \leq x_{0,v_k}|d_{u_k,v_k}|$ because $d_{u,v} < 0$. It then follows that

$$\prod_{k=1}^{m} x_{0,v_k}|d_{u_{k+1},v_k}| \cdot |\theta_{u_k}| \leq \prod_{k=1}^{m} x_{0,v_k}|d_{u_k,v_k}| \cdot |\theta_{u_{k+1}}|$$

which implies $\prod_{k=1}^{m}|d_{u_{k+1},v_k}| \leq \prod_{k=1}^{m}|d_{u_k,v_k}|$ because $x_{0,v} > 0$ for all v and thus $|\theta_u| > 0$ 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 [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 context 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 $B_i^+ = \{\psi \in B_i \wedge x_{i,\psi} > 0\}$ be evader i ($i \in N^+$)'s *positive* boundary sites, whose resource amount can be reduced. Let $\hat{B}_0 = \{\psi \in B_0 \wedge x_{0,\psi} < \ell_{0,\psi}\}$ be the catcher's *open* boundary sites, to which more resources could be assigned. Let the *active edges* be $A = \{(i, \psi) \mid i \in N^+, \psi \in \Psi, \psi \in B_i\}$.

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 evaders' best responses and the min-cost flow's "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 catcher's resource amount—that is, for each ψ 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 catcher's resource amount, or change the open boundary sites $\hat{B}_0$ (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 catcher's utility threshold θ_0. Such rerouting must also maintain the catcher's best-response condition. For this we use max-flow and binary search: first, we binary search on Δ , the decrease in θ_0; then, we calculate each edge's rerouting capacity using Δ, and see whether a max-flow can saturate all capacities, thereby maintaining the best-response condition.

Lemma 2. *Algorithm 2 maintains Nash equilibrium without changing $\mu_{i,\psi}$ or θ_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 (ψ, τ). Hence $x_{\Sigma,\psi}$ 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 $\theta_i = \min_{\psi: x_{i,\psi} > 0} \mu_{i,\psi}$, the set of positive boundary sites B_i^+ must be non-empty, which means $\sum_{\psi \in B_i} x_{i,\psi} > 0$. So no matter how we reallocate, some $\psi \in B_i$ must remain positive. Hence, θ_i is unchanged because the $\mu_{i,\psi}$ are unchanged. □

Lemma 3. *In Algorithm 3, the evaders' thresholds θ_i ($i \in N^+$) decrease at rate γ_i. That is, the algorithm decreases θ_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 μ^0, θ^0, A^0 defined in its proof, evaders are best-responding if and only if:

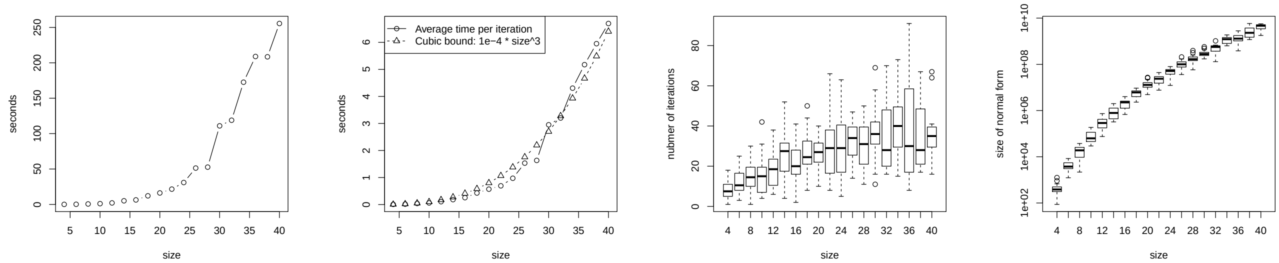
$$(\forall i \in N^+, \psi \in \Psi : x_{i,\psi} > 0)$$
$$\mu_{i,\psi} = \mu_{i,\psi}^0 + \Delta \cdot \gamma_\psi \cdot d_{i,\psi} \geq \theta_i = \theta_i^0 - \Delta \cdot \gamma_i$$
$$(\forall i \in N^+, \psi \in \Psi : x_{i,\psi} < \ell_{i,\psi})$$
$$\mu_{i,\psi} = \mu_{i,\psi}^0 + \Delta \cdot \gamma_\psi \cdot d_{i,\psi} \leq \theta_i = \theta_i^0 - \Delta \cdot \gamma_i$$

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 $\mu_{i,\psi}^0 = \theta_i^0$, so we only need to check

$$(\forall i \in N^+, \psi \in \Psi : x_{i,\psi} > 0) \;\; \Delta \cdot \gamma_\psi \cdot d_{i,\psi} \geq -\Delta \cdot \gamma_i$$
$$(\forall i \in N^+, \psi \in \Psi : x_{i,\psi} < \ell_{i,\psi}) \;\; \Delta \cdot \gamma_\psi \cdot d_{i,\psi} \leq -\Delta \cdot \gamma_i$$

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

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



(a) Average total running time (b) Average time per iteration (c) Number of iterations (d) Size of normal form

Figure 1: Performance of Algorithm 1, and the size of the normal form, for randomly generated CE games.

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. $\square$

Lemma 5. *If Algorithm 3 fails, then Algorithm 4 strictly decreases θ_0 while maintaining Nash equilibrium.*

Proof. If Algorithm 3 fails, then for each $\psi \in \hat{B}_0$, there must be another site $\psi' \in \Psi_{reachable} \cap \Psi_{unincreasable}$. 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 ψ to some site $\psi' \in \Psi_{unincreasable}$ to which the catcher cannot increase resource assignment.

That site ψ' 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 ψ': 1) $x_{0,\psi'} = \ell_{0,\psi'}$; 2) $x_{0,\psi'} < \theta_0$. 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 θ_0.

Therefore, for each $\psi \in \hat{B}_0$, evaders can move a positive amount of resource from that site ψ to some $\psi' \notin B_0$ 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 Δ (the number found by the binary search in Algorithm 4) for each $\psi \in \hat{B}_0$ (saturating all edges leaving σ), 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 $\mu_{0,\psi'}$ ($\psi' \notin \hat{B}_0$), θ_0 must have strictly decreased (by Δ). $\square$

Lemma 6. *After Algorithm 4, either a new site ψ 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 ψ 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 τ were saturated during the run of Algorithm 4 (for the value of Δ resulting from the binary search), i.e., $(\forall \psi' \notin \hat{B}_0), f(\psi, \tau) < \kappa(\psi, \tau)$. Then, in Algorithm 4, we could have increased Δ 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 ψ', the total amount of evader resource $x_{\Sigma,\psi'}$ was increased by $\kappa(\psi', \tau) = (\theta_0^0 - \Delta - \mu_{0,\psi'}^0)/d_{0,\psi'}$ (where superscript 0 denotes the value prior to the run of Algorithm 4). It follows that $\mu_{0,\psi} = \mu_{0,\psi}^0 + \theta_0^0 - \Delta - \mu_{0,\psi}^0 = \theta_0^0 - \Delta = \theta_0$, while $\mu_{0,\psi}^0 < \theta_0^0$. $\square$

Lemma 7. *Each site ψ 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 θ_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 ψ' $\mu_{0,\psi'} < \theta_0$ enters $\hat{B}_0$. So a site ψ can only leave $\hat{B}_0$ by being saturated ($x_{0,\psi} = \ell_{0,\psi}$). Since we never decrease $x_{0,\psi}$ during the algorithm, saturated sites ψ can never enter $\hat{B}_0$ again. $\square$

Lemma 8. *Algorithm 3 runs successfully at most $2|\Psi| \cdot 3^{n|\Psi|}$ times.*

Proof. By Lemma 7, we only have to argue that there are at most $3^{n|\Psi|}$ 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, ψ) 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 ϕ 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 ϕ 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 ϕ, 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 (ψ', τ) that enters sink τ, then ψ' will newly enter $\hat{B}_0$. Hence if $\hat{B}_0$ remains unchanged, we can ignore the capacity of those edges entering τ. Also, with $\hat{B}_0$ fixed, the set of edges leaving source σ and their capacities are fixed. Therefore, the resulting x of Algorithm 4 is solely determined by the edges between N^+ and Ψ, which is fixed by ϕ.[6]

We then conclude that if there were two Algorithm 3 runs that have the same resulting ϕ, then between those two runs, there must be no Algorithm 4 run that has positive Δ which strictly decreases θ_0. Otherwise, we would have two Algorithm 4 runs (after those two Algorithm 3 runs) with the same ϕ, where the latter run has strictly smaller θ_0 (note that θ_0 never increases), contradicting that the returning x of Algorithm 4 is completely determined by ϕ.

Therefore, if there were two Algorithm 3 runs that result in the same ϕ, 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 ψ, 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 ($A = N^+ \times \Psi$). That is, for each edge leaving source σ, its capacity is $\kappa(\sigma, i) = r_i$; for each edge entering sink τ, its capacity is $\kappa(\psi, \tau) = x_{\Sigma,\psi}$; the weight and capacity of edge (i, ψ) 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 ∞ 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 $\sum_{i \in N^+, \psi \in \Psi^+} x_{i,\psi} \log(-d_{i,\psi})$

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, ψ) must be entering the active edge set A (otherwise we will either continue increasing $x_{0,\psi}$ or change $\hat{B}_0$). For that newly active edge, if $x_{i,\psi} = 0$, then $dist(\psi) > dist(i) + w(i, \psi)$ must be true (recall that $dist$ is the shortest distance from ψ^* in the residual graph) because: $\mu_{i,\psi}$'s decrease rate $\gamma_\psi(-d_{i,\psi})$ must be slower than θ_i's decrease rate γ_i. Similarly, if $x_{i,\psi} = \ell_{i,\psi}$, then $dist(i) > dist(\psi) - w(i, \psi)$ must be true. Hence by adding (i, ψ), either $dist$ has to decrease or a negative cycle exists which leads to a decrease in flow cost (w.r.t. G_1).

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

[6] The residual graph might be different for the same ϕ, 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 ϕ.

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* $2|\Psi| + 4|\Psi|3^{n|\Psi|}$ *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 $2|\Psi| + 4|\Psi|3^{n|\Psi|}$ iterations. We have at most $2|\Psi|3^{n|\Psi|}$ iterations where Algorithm 3 runs successfully, by Lemma 8. Each failed iteration must either be followed by a successful iteration, or $\hat{B}_0$ is changed (by Lemma 6). Lemma 7 ensures that $\hat{B}_0$ can be changed at most $2|\Psi|$ times. So overall there can be at most $2 \times 2|\Psi|3^{n|\Psi|} + 2|\Psi|$ 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, 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 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 devotion of further effort to resolving this question, as well as to extending these techniques to related problems.

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

[An *et al.*, 2012] Bo An, Eric Shieh, Milind Tambe, Rong Yang, Craig Baldwin, Joseph DiRenzo, Ben Maule, and Garrett Meyer. PROTECT - A deployed game theoretic system for strategic security allocation for the United States Coast Guard. *AI Magazine*, 33(4):96–110, 2012.

[Birkhoff, 1946] Garrett Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucumn Rev, Ser. A, no. 5*, pages 147–151, 1946.

[Borie *et al.*, 2009] Richard B Borie, Craig A Tovey, and Sven Koenig. Algorithms and complexity results for pursuit-evasion problems. In *IJCAI*, volume 9, pages 59–66, 2009.

[Kiekintveld *et al.*, 2009] Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Fernando Ordóñez, and Milind Tambe. Computing optimal randomized resource allocations for massive security games. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 689–696, Budapest, Hungary, 2009.

[Korzhyk *et al.*, 2010] Dmytro Korzhyk, Vincent Conitzer, and Ronald Parr. Complexity of computing optimal Stackelberg strategies in security resource allocation games. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 805–810, Atlanta, GA, USA, 2010.

[Korzhyk *et al.*, 2011a] Dmytro Korzhyk, Vincent Conitzer, and Ronald Parr. Security games with multiple attacker resources. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pages 273–279, Barcelona, Catalonia, Spain, 2011.

[Korzhyk *et al.*, 2011b] Dmytro Korzhyk, Zhengyu Yin, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. Stackelberg vs. Nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. *Journal of Artificial Intelligence Research*, 41(2):297–327, 2011.

[Li and Conitzer, 2013] Yuqian Li and Vincent Conitzer. Game-theoretic question selection for tests. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, pages 254–262, Beijing, China, 2013.

[Parsons, 1978] Torrence D Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, pages 426–441. Springer, 1978.

[Paruchuri *et al.*, 2008] Praveen Paruchuri, Jonathan P. Pearce, Janusz Marecki, Milind Tambe, Fernando Ordóñez, and Sarit Kraus. Playing games for security: An efficient exact algorithm for solving Bayesian Stackelberg games. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 895–902, Estoril, Portugal, 2008.

[Pita *et al.*, 2009a] James Pita, Manish Jain, Fernando Ordóñez, Christopher Portway, Milind Tambe, and Craig Western. Using game theory for Los Angeles airport security. *AI Magazine*, 30(1):43–57, 2009.

[Pita *et al.*, 2009b] James Pita, Manish Jain, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Using game theory for Los Angeles airport security. *AI Magazine*, 30(1):43–57, 2009.

[Tsai *et al.*, 2009] Jason Tsai, Shyamsunder Rathi, Christopher Kiekintveld, Fernando Ordóñez, and Milind Tambe. IRIS - A tool for strategic security allocation in transportation networks. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 37–44, Budapest, Hungary, 2009.

[Yin *et al.*, 2012] Zhengyu Yin, Albert Xin Jiang, Milind Tambe, Christopher Kiekintveld, Kevin Leyton-Brown, Tuomas Sandholm, and John P. Sullivan. TRUSTS: Scheduling randomized patrols for fare inspection in transit systems using game theory. *AI Magazine*, 33(4):59–72, 2012.