Duguet, Aloïs; Carvalho, Margarida; Dragotto, Gabriele; Ngueveu, Sandra Ulrich

Computing approximate Nash equilibria for integer programming games

Optimization Letters

Provided in Cooperation with:
Springer Nature

Suggested Citation: Duguet, Aloïs; Carvalho, Margarida; Dragotto, Gabriele; Ngueveu, Sandra Ulrich (2025) : Computing approximate Nash equilibria for integer programming games, Optimization Letters, ISSN 1862-4480, Springer, Berlin, Heidelberg, Vol. 20, Iss. 1, pp. 231-255, https://doi.org/10.1007/s11590-025-02221-5

Optimization Letters (2026) 20:231–255
https://doi.org/10.1007/s11590-025-02221-5

ORIGINAL PAPER

Computing approximate Nash equilibria for integer programming games

Aloïs Duguet[1] · **Margarida Carvalho**[2] · **Gabriele Dragotto**[3] · **Sandra Ulrich Ngueveu**[4]

Received: 6 February 2024 / Accepted: 23 June 2025 / Published online: 31 July 2025

Abstract

We propose a framework to compute approximate Nash equilibria in integer programming games with nonlinear payoffs, *i.e.*, simultaneous and non-cooperative games where each player solves a parametrized mixed-integer nonlinear program. We prove that using absolute approximations of the players' objective functions and then computing its Nash equilibria is equivalent to computing approximate Nash equilibria where the approximation factor is doubled. In practice, we propose an algorithm to approximate the players' objective functions via piecewise linear approximations. The numerical experiments on a cybersecurity investment game combined with a detailed analysis of the results show the computational effectiveness of our approach.

Keywords Integer programming games · Algorithmic game theory · Integer programming · Piecewise linear approximations

✉ Aloïs Duguet
duguet@uni-trier.de

Margarida Carvalho
carvalho@iro.umontreal.ca

Gabriele Dragotto
gabrieledragotto@gmail.com

Sandra Ulrich Ngueveu
ngueveu@laas.fr

[1] Department of Mathematics, Trier University, Universitätsring 15, 54296 Trier, Germany

[2] CIRRELT and Département d'informatique et de recherche opérationnelle, Université de Montréal, Montreal, QC H3T1J4, Canada

[3] Department of Operations Research and Financial Engineering, Princeton University, Sherrerd Street, Princeton, New Jersey 08544, USA

[4] LAAS-CNRS, Université de Toulouse, CNRS, INP, 7 avenue du colonel Roche, 31400 Toulouse, France

1 Introduction

In this paper, we focus on the task of computing Nash equilibria for Integer Programming Games (IPGs) [1, 2], a broad class of *simultaneous* and *non-cooperative* games. In simple terms, an IPG is a game among a finite number of players, each of which decides by solving a mixed-integer optimization problem. Compared to more classic game representations, such as normal or extensive form games, IPGs can *implicitly* describe the space of strategies (*i.e.*, feasible points) of each player through a mixed-integer programming set. Thus, IPGs avoid the possibly expensive *explicit* enumeration of all the players' strategies; this is especially important when the number of strategies available to each player is large or even uncountable, for instance, in a combinatorial setting. We formally define IPGs in Definition 1.

Definition 1 (IPG) An IPG is a simultaneous and non-cooperative game with complete information among a finite set $M = \{1, 2..., m\}$ of players such that each player $p \in M$ solves the parametric optimization problem

$$\max_{x^p} \quad \Pi^p(x^p; x^{-p}) \tag{1a}$$

$$\text{s.t.} \quad x^p \in X^p := \{A^p x^p \leq b^p, x^p \in \mathbb{R}^{n_p^c} \times \mathbb{Z}^{n_p^i}\}, \tag{1b}$$

where $x^{-p} = (x^1, \ldots, x^{p-1}, x^{p+1}, \ldots, x^m)$ is the vector of strategies for all players except p, X^p and Π^p are the *strategy set* and the *payoff function* of player p, and A^p and b^p are a rational matrix and vector of appropriate dimensions, respectively.

When we say an IPG is a non-cooperative complete-information game, we mean that each player maximizes its payoff and has full information on the other players' optimization problems, namely, on the objective function and constraints of its opponents. Similarly to other classes of simultaneous games, the leading solution concept for IPGs is the so-called Nash equilibrium. Intuitively, a Nash equilibrium is a *stable* solution where no single player has an incentive to profitably defect from the solution. In recent years, several authors proposed a variety of algorithms to compute Nash equilibria in IPGs [3–8], even though this problem is at least PPAD-hard [9]. The majority of these algorithms assume the payoff functions to be linear or linear-quadratic, and often involve the solution of a so-called *best-response* program, *i.e.*, the solution of the optimization problem of player p (1a–1b) given a fixed set of other players' strategies x^{-p}. As motivated in the tutorial [2], solving the best-response program is pivotal for the correct identification and computation of an equilibrium strategy. However, from a computational perspective, the form of the payoff function Π^p intrinsically influences the difficulty of solving the best-response program. Whenever Π^p is not linear in x^p, the resulting best-response program is a Mixed-integer Nonlinear Program (MINLP), a well-known class of difficult non-convex optimization problems [10]. In general, there are different ways to handle nonlinear terms, *e.g.*, convex relaxations combined with branch-and-bound algorithms [11–13], or piecewise linear approximations of the nonlinear terms [14, 15].

However, to date, the computation of Nash equilibria in IPGs with nonlinear utilities (in each player's variables) is a rather unexplored topic, most likely because of the difficulty associated with, on the one hand, computing equilibria, and, on the other hand, handling nonlinearities in a computationally-efficient way.

Contributions. In this paper, we specifically focus on IPGs with nonlinear payoffs. We summarize our contributions as follows:

- We provide a general methodology to compute Nash equilibria for IPGs where players have nonlinear payoff functions in their variables. We prove that performing an approximation with pointwise guarantees on the payoff functions is equivalent to computing an approximate Nash equilibrium.
- We propose a piecewise-linear approximation scheme to enable the Sample Generation Method (SGM) from [4] to compute approximate Nash equilibria in IPGs with nonlinear payoffs. Specifically, we approximate the inherently nonlinear players' best-response programs with piecewise-linear approximations.
- Finally, we demonstrate the effectiveness of our framework via computational experiments on a cybersecurity investment game.

Outline. We organize the paper as follows. Section 2 reviews the current literature and provides some background definition. Section 3 presents the main theorem and the approach to compute approximate Nash equilibria. Section 4 introduces a game-theory model for cybersecurity investments, and the computational results. We conclude in Sect. 5.

2 Background

Approximations. Solving MINLPs can be a computationally-challenging task, due to the nonlinearities (and nonconvexities) involved in the formulations. A common approach to overcome these challenges is to approximate some of the functions (*e.g.*, the objective function or constraint terms) involved in the formulation, for instance, with an approximation satisfying an absolute error (Definition 2).

Definition 2 Given a function $f : \mathbb{X} \longrightarrow \mathbb{R}$, a function $\hat{f} : \mathbb{X} \longrightarrow \mathbb{R}$ approximates f with an absolute error $\delta > 0$ if and only if $|f(x) - \hat{f}(x)| \leq \delta$ for any $x \in \mathbb{X}$.

The approximation $\hat{f}$ in Definition 2 guarantees that, for any point in the domain $\mathbb{X}$ of f, the difference between $\hat{f}$ and f does not exceed δ, *i.e.*, the absolute approximation error. We also denote such $\hat{f}$ a δ-absolute approximation of f. In particular, if the function $\hat{f}$ is a piecewise linear (PWL) function, we say that $\hat{f}$ is a PWL absolute approximation of f. Figure 1 gives an example of such an approximation.

The family of PWL approximations is rather commonly used, and from a computational standpoint, there exist several methods to construct absolute PWL approximations [14]. Whenever the original function f has one (real) variable, we can construct a PWL absolute approximation by employing the smallest number

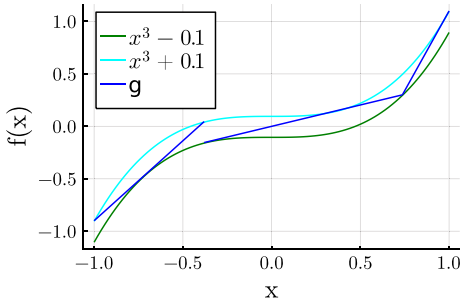


Fig. 1 Function g is a PWL 0.1-absolute approximation of x^3 on $[-1, 1]$ with 3 pieces

of pieces [16, 17]. This statement is also valid for sums of univariate functions of (potentially) different variables. However, for functions of two variables or more, the maximal length property of a piece used in [17] is not valid anymore and there exists no algorithm to compute a PWL absolute approximation with the minimal number of pieces [18–21]; in addition, the computation times required to build the approximation just in the two-variable case are way larger than in the univariate case.

In the context of MINLP, PWL approximations are one of the tools to approximate the original problem with a Mixed-integer Linear Program (MILP). In practice, this means that we can obtain approximate or close to optimal solutions for the original MINLP by solving a computationally-easier MILP. Naturally, the number of pieces involved in the PWL approximation influences the quality of the approximation. On the one hand, a greater number of pieces in the PWL approximation results in a tighter approximation. On the other hand, a larger number of pieces may lead to increased computing times, mostly because more pieces require more variables and constraints in the resulting MILP [22].

Strategies and equilibria. We denote as X the set of combinations of all players' strategies, *i.e.*, the Cartesian product $X = \prod_{p=1}^{m} X^p$ of the sets X^p of all players p, and we denote any $x \in X$ as a *profile of strategies*. For each player p, we say $x^p \in X^p$ is a *pure strategy* for p. When players randomize over their pure strategies, they play a so-called *mixed strategy*, *i.e.*, a probability distribution σ^p over the set of pure strategies X^p; let Δ^p be the space of probability distributions over X^p for player p. Similarly to pure strategies, a *profile of mixed strategies* is a vector $\sigma = (\sigma^1, \ldots, \sigma^m)$ of mixed strategies such that $\sigma^p \in \Delta^p$. We call $\Pi^p(\sigma^p;\sigma^{-p})$ the expected payoff of player p associated with σ. We employ the *Nash equilibrium* [23] of Definition 3 as a solution concept.

Definition 3 Given an IPG instance and a scalar $\delta \in \mathbb{R}_+$, a profile of mixed strategies $\hat{\sigma}$ is a δ-Nash equilibrium if no player has incentive to deviate from it, i.e., if, for any player $p \in M$, it holds that

$$\Pi^p(\hat{\sigma}^i;\hat{\sigma}^{-i}) + \delta \geq \Pi^p(\bar{x}^p;\hat{\sigma}^{-p}), \quad \forall \bar{x}^p \in X^p. \tag{2}$$

Whenever $\delta = 0$, we call the Nash equilibrium *exact*; otherwise, if $\delta > 0$, we say that the Nash equilibrium is an *approximate* equilibrium.

Intuitively, (2) ensures that, for each player p, there exists no unilateral and profitable *deviation* $\bar{x}^p$ such that player p increases its payoff. From both a computational-complexity and practical perspective, computing exact Nash equilibria is a challenging task; for instance, even in normal-form two-player games [24], *i.e.*, a class of finite games contained in IPGs, computing exact equilibria is far from being trivial. On top of the difficulty of computing an exact equilibrium, even solving a best-response program for a player of an IPG accounts to solving an MINLP, a computationally-challenging problem itself [10]. One way to potentially reduce the computational difficulty of the task of determining an equilibrium is to consider approximate equilibria instead of exact ones. This can, at least from a practical perspective, allow us to approximate the players' best-response programs and compute an approximate equilibrium by solving a series of MILP problems.

3 Methodology

In this section, we present a methodology to compute approximate Nash equilibria via PWL approximations of the players' payoff functions. Specifically, we link approximate equilibria with approximations having absolute error guarantees. Then, we describe the existent IPG and PWL methods we employ, and, finally, we combine those results in an algorithm.

Assumptions. Without loss of generality, we represent the payoff of a player p as

$$\Pi^p(x^p;x^{-p}) = f^p(x^p) + g^p(x^p;x^{-p}),$$

where f^p is a function grouping all terms that depend only on the strategy of player p and g^p is a function grouping what we call the remaining terms. We focus on approximating f^p via PWL functions. We remark that we do not approximate g^p since it includes the other players' variables x^{-p} and, therefore, its PWL approximation would introduce x^{-p} in the constraints of p. In other words, approximating g^p would require the extension of the equilibrium concept to the one of *generalized Nash equilibrium*, i.e., the equilibrium of a game where the strategy set of each player depends on the strategies of the opponents; in practice, this would prevent us from building on top of the available algorithms for IPGs. We assume that f^p is *a sum of univariate nonlinear functions*. As explained in Sect. 2, this assumption corresponds to the state of the art in terms of approximation methods with PWL functions, although any progress in this domain is directly applicable to our methodology.

3.1 Approximate equilibria and payoff functions

We present a link between approximate equilibria and approximate payoff functions. Specifically, we prove that, given an IPG instance G, we can obtain an approximate

equilibrium by computing an equilibrium of a game $\hat{G}$ where we approximate each player's nonlinear payoff.

Proposition 1 *Let G be an IPG where the optimization problem of each player p is of the form (1a–1b). Assume that $\hat{G}$ is an IPG where each player p solves the optimization problem*

$$\max_{x^p} \{\hat{\Pi}^p(x^p;x^{-p}) := \hat{f}^p(x^p) + g^p(x^p;x^{-p}) : x^p \in X^p\},$$

where $\hat{f}^p$ is a δ-absolute approximation of f^p in X^p. Then, (i.) a Nash equilibrium $\hat{\sigma}$ of $\hat{G}$ is a 2δ-equilibrium of G, and (ii.) a δ_M-equilibrium of $\hat{G}$ is a $(2\delta + \delta_M)$-equilibrium of G.

Proof We first show that (ii.) holds. Consider an arbitrary player p in G. Remark that x^{-p} is a parameter in the optimization problem of player p, thus the function $\hat{\Pi}^p$ depends only on x^p. The function $\hat{\Pi}^p(x^p;x^{-p}) = \hat{f}^p(x^p) + g^p(x^p;x^{-p})$ is thus a δ-absolute approximation of $\Pi^p(x^p;x^{-p}) = f^p(x^p) + g^p(x^p;x^{-p})$ for any x^{-p} because $\hat{f}^p$ is a δ -absolute approximation of f^p. It follows that

$$\Pi^p(\hat{\sigma}^p;\hat{\sigma}^{-p}) \geq \hat{\Pi}^p(\hat{\sigma}^p;\hat{\sigma}^{-p}) - \delta, \tag{3}$$

$$\geq \hat{\Pi}^p(x^p;\hat{\sigma}^{-p}) - \delta - \delta_M \qquad \forall x^p \in X^p, \tag{4}$$

$$\geq \Pi^p(x^p;\hat{\sigma}^{-p}) - 2\delta - \delta_M \qquad \forall x^p \in X^p, \tag{5}$$

where we exploit the fact $\hat{\Pi}$ is a δ-absolute approximation in (3) and (5), and the fact $\hat{\sigma}$ is a δ_M-equilibrium in (4). The above shows that $\hat{\sigma}$ is a $(2\delta + \delta_M)$-equilibrium of G. Finally, (i.) holds as a special case of (ii.) by setting $\delta_M = 0$. □ □

Essentially, Proposition 1 claims that a Nash equilibrium of an approximate game $\hat{G}$ is an approximate equilibrium of the game G. As we explain in Sect. 3.3, we employ Proposition 1 to compute δ-equilibria of games where the payoffs are nonlinear. To that end, a method to approximate nonlinear functions up to a given absolute approximation error needs to be used. One such method is approximating by PWL function as discussed in the assumption paragraph of Sect. 3. This method has the additional advantage of leading to MILP instead of MINLP for which there are more efficient solvers available.

3.2 SGM and PWL approximations

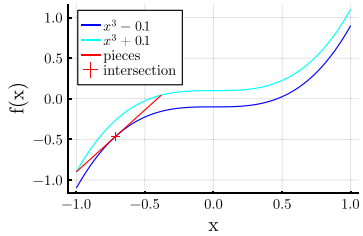
In this section, we briefly describe the SGM algorithm to compute a Nash equilibrium for IPGs [4] and how to build PWL approximations.

The sample generation method. SGM is an iterative method that fundamentally involves two steps. First, SGM computes an equilibrium σ of an *inner-approximated*

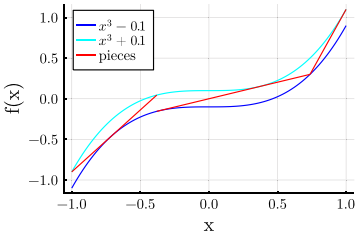
game, that is, it computes an equilibrium in a game where the strategy set of each player p is a subset (*i.e.*, an inner approximation) of X^p; in practice, we assume that the inner approximation of each player's strategy set is nonempty, *i.e.*, it contains at least one strategy. Second, given the equilibrium σ to the inner-approximated game, SGM computes the best response of each player p to σ^{-p}; in other words, it checks, for each player p, if there exist unilateral and profitable deviations from σ. On the one hand, if, for each player p and opponents' strategies σ^{-p}, the payoff of a best response $\bar{x}^p$ is less than or equal to the one of σ plus δ, then σ is a δ-equilibrium for the IPG. Equivalently, SGM verifies that the stopping criterion $\Pi^p(\sigma^p;\sigma^{-p}) + \delta \geq \Pi^p(\bar{x}^p;\sigma^{-p})$ holds for each player p. On the other hand, if there exists at least one profitable deviation $\bar{x}^p$, SGM enlarges the inner approximation of the first step by including $\bar{x}^p$ and iterates again. When solving the inner-approximated game, SGM solves a normal-form game with one of the several algorithms available in the literature. For ease of implementation, and due to the existence of powerful mixed-integer solvers, we employ the so-called *feasibility formulation* from Sandholm et al. [25].[1] It also has the advantage of allowing already computed equilibria to be used as warm start. Practically, solving the inner-approximate game accounts to solving a mixed-integer program that is infeasible if no equilibrium exists. Since this is a feasibility program, we also add an objective function that minimizes the support size, *i.e.*, the number of strategies played by each player with positive probability; we employ this objective because there is computational evidence that equilibria tend to have small supports [26].

PWL approximations. We approximate the best-response programs in SGM with PWL δ-absolute approximations of univariate functions. More precisely, when we need to approximate a univariate function f with a PWL function $\hat{f}$, we can employ either the algorithm by Codsi et al. [27, 27*f* with the least number of pieces if f is convex or concave [27, Lemma 4], whereas the latter always approximates f with the least number of pieces in the general case, but is slower in practice. As for the algorithm in Codsi et al. [27, Algorithm 5], it builds a PWL function $\hat{f}$ with the least number of pieces such that it is a δ-absolute approximation of f. At each iteration, the algorithm constructs a new piece (segment) of $\hat{f}$ from a specific tangent of the upper-estimator function $f + \delta$ (or lower-estimator $f - \delta$). In Fig. 2a, we illustrate the rationale behind this algorithm. Specifically, the red piece intersects the upper estimation $x^3 + 0.1$ at its endpoints, and intersects the lower estimation $x^3 - 0.1$ on one point, making it a tangent of the lower estimation. Also, Fig. 2b features the PWL function created by Codsi et al. [27, Algorithm 4] for the approximation of function $f(x) = x^3$ with absolute error 0.1 and domain $[-1, 1]$.

[1] Although Sandholm et al. [25] concentrates on 2-player games, the formulation can be generalized for m-player games in a straightforward manner.



(a) Creation of a piece of the PWL approximation by Codsi et al. [27, Algorithm 5].



(b) Complete PWL approximation with three pieces by Codsi et al. [27, Algorithm 4].

Fig. 2 Illustration of PWL approximation

3.3 Computing approximate equilibria

In this section, we employ the algorithmic ingredients introduced to present two algorithmic procedures for computing approximate equilibria. Specifically, we propose two algorithmic procedures to compute a δ_f-equilibrium of an IPG G based on SGM and δ_f-absolute approximations of the players' payoffs.

Direct approximation method. In a nutshell, the *direct approximation method* performs a single-round approximation of the players' payoffs, before resorting to SGM to compute an equilibrium. In particular, SGM will employ the approximated players' payoff functions in the best-response computations. Algorithm 1 shows a pseudocode of the direct approximation method. The procedure `ApproximateIPG`(G, δ) Line 1 computes a PWL δ-absolute approximation of each of the functions f^p, creating an approximate game $\hat{G}$ with the same set of players and equivalent strategies, and the payoffs given by the computed δ-approximations $\hat{f}^p$ plus g^p. We chose $\delta = \mu\frac{\delta_f}{2}$ with parameter $\mu \in]0, 1[$. We then compute a δ_M-equilibrium $\hat{\sigma}$ of $\hat{G}$ with procedure `SGM`$(\hat{G}, \delta_M)$ Line 2. We chose $\delta_M = (1-\mu)\delta_f$ in this case. According to Proposition 1, $\hat{\sigma}$ is a $2\delta + \delta_M = 2\mu\frac{\delta_f}{2} + (1-\mu)\delta_f = \delta_f$-equilibrium of G. The parameter μ determines which proportion of the tolerance δ_f is given to the two operations needing a tolerance: the PWL approximation and the SGM. It thus operates a tradeoff that should be such that, from a computational perspective, the approximation is computationally easy to compute and SGM terminates (Fig. 2).

Algorithm 1 Direct approximation method

Input: A game G, tolerance δ_f and parameter $\mu \in (0, 1)$
Output: A δ_f-equilibrium $\hat{\sigma}$

1: $\hat{G} =$ `ApproximateIPG`$(G, \mu\frac{\delta_f}{2})$ ▷ create the approximate game $\hat{G}$
2: $\hat{\sigma} =$ `SGM`$(\hat{G}, (1-\mu)\delta_f)$ ▷ find a $(1-\mu)\delta_f$-equilibrium of $\hat{G}$
3: **return** $\hat{\sigma}$

2-level approximation method. The time spent inside the direct approximation method is mostly taken by the computation of best responses and the resolution of the normal-form game. More precisely, while the best-response computations have a non-negligible cost due to the integer variables, the time for the resolution of each normal-form game is expected to increase substantially as the iteration number increases. Indeed, there is a binary variable added at each iteration. Thus, a decrease in the number of iterations of the SGM or a decrease in the size of the best responses and normal-form game is expected to lead to a shorter computation time.

With this idea in mind, we introduce the 2-*level approximation method*. Its functioning is in two steps. First, we solve the IPG with the direct approximation method and an approximation error δ_0 much bigger than the wanted approximation error δ_f, for example 1000 times bigger. It gives an approximate NE σ_0. Second, we use the strategies of σ_0 as a warm start for another call to the direct approximation method so that we obtain a δ_f-NE $\hat{\sigma}$ to game G.

This algorithm is expected to be faster than the direct approximation method for two reasons. On the one hand, the optimization problems solved in the first call to the direct approximation method are of smaller sizes due to the bigger approximation error δ_0. On the other hand, the number of iterations of the SGM inside the second call of the direct approximation method are expected to be much smaller than in the first call due to the warm start.

The parameter values chosen for the two calls to the direct approximation method are shown in Algorithm 2. The second call to the SGM uses a warm start to the normal-form game: the strategies of each player are initialized with the strategies inside $\hat{\sigma}_0$. According to Proposition 1 case (ii), $\hat{\sigma}$ is a $(1-\mu)\delta_f + 2\mu\frac{\delta_f}{2} = \delta_f$-equilibrium of G.

Algorithm 2 2-level approximation method

Input: A game G, starting tolerance δ_0, final tolerance δ_f and parameter $\mu \in (0,1)$
Output: A δ_f-equilibrium $\hat{\sigma}$

1: $\hat{G}_0 = \texttt{ApproximateIPG}(G, \delta_0)$ ▷ create the approximate game $\hat{G}_0$
2: $\hat{\sigma}_0 = \texttt{SGM}(\hat{G}_0, (1-\mu)\delta_f)$ ▷ find a $(1-\mu)\delta_f$-equilibrium of $\hat{G}_0$
3: $\hat{G} = \texttt{ApproximateIPG}(G, \mu\frac{\delta_f}{2})$
4: $\hat{\sigma} = \texttt{SGM}(\hat{G}, (1-\mu)\delta_f, \text{warm start} = \hat{\sigma}_0)$
5: **return** $\hat{\sigma}$

4 Experiments and computational results

In this section, we test the algorithms introduced in the previous section. In the first place, we describe an IPG motivated by an existent cybersecurity investment game. Afterward, we detail the experimental setup and we analyze the computational results.

Table 1 Nonlinear cybersecurity costs tested

Cybersecurity costs	Best-response class	Best-response solver	SGM label
$h^p_{\log}(s^p) = -\alpha^p \log(1 - s^p)$	Mixed-integer exponential cone program	MOSEK [30]	SGM-ExpCone
$h^p_{\text{root}}(s^p) = \alpha^p \left(\frac{1}{\sqrt{1-s^p}} - 1 \right)$	MIQCQP	Gurobi [31]	SGM-MIQCQP
$h^p_{\text{nonconvex}}(s^p) = \alpha^p \left(\frac{1}{\sqrt{1-s^p}} + \frac{2}{1+\exp(-20s^p)} \right) - 2$	Mixed-integer nonlinear program	SCIP [32]	SGM-SCIP

In practice, the approximation procedures introduced are efficient and manage to compute approximate equilibria within relatively modest computation times.

4.1 The cybersecurity investment game

The Cybersecurity Investment Game (CIG) is a non-cooperative game where a set of players (retailers) sell a homogeneous product in an online marketplace. When selling the product, the transaction can be subject to cyberattacks, which damage the player's reputation and diminish its payoff. This game extends the model of [28] by including integer decisions and letting players have nonlinear payoff functions, while still respecting the practical considerations proposed in [28]. We refer to [28] for a complete description of the motivations behind this model.

The CIG is modeled as an IPG, where each player p solves a parametric box-constrained MINLP, maximizing their payoff $\Pi^p(x^p, x^{-p}) = f^p(x^p) + g^p(x^p, x^{-p})$. The function $g^p(x^p, x^{-p})$ is linear in x^p, while $f^p(x^p)$ is a nonlinear function given by

$$f^p(x^p) = \sum_{j \in J} \left(d^p_{j,lin} x^p_j + d_{j,quad} (x^p_j)^2 \right) - h^p(x^p),$$

where J is the set of markets and $h^p(x^p)$ represents the cybersecurity cost. More specifically, $h^p(x^p) = h^p(s^p)$ is a univariate function, with $s^p \in x^p$ in the domain $[0, 1]$ denoting the cybersecurity level investment. The complexity of this game, particularly the computation of a best-response, arises from the presence of integer variables – one for each market $j \in J$ – along with the quadratic and nonlinear terms in the payoff function, present within $f^p(x^p)$. In Appendix A.1, we detail the decision variables, terms in the payoffs, and constraints of the IPG model for CIG.

In this paper, we aim to solve CIG to empirically validate the value and performance of the algorithms we introduced in Sect. 3. To this end, we test different nonlinear functions h^p satisfying the properties described in [28] and summarized in Appendix A.2. Table 1 summarizes the three cybersecurity costs considered in our study as well as the corresponding best-response optimization class, where MIQCQP is Mixed-integer Quadratically Constrained Quadratic Program. As those cybersecurity costs make the payoffs lipschitz continuous, the

SGM converges in a finite number of iterations to the wanted tolerance, and so our algorithms stop in a finite time. Given the availability of solvers tailored to the specified best-response classes, we compare our methodology to the baseline, SGM, which utilizes the solver specified in the table. The selection of these solvers is mainly guided by the state of the art of advances in the respective type of best-response programs. The exception is SCIP which is a state-of-the-art solver for MINLPs among noncommercial solvers.[2]

4.2 Description of experiments

The presence of integer variables in CIG means that the methodology of [28] cannot be used as a baseline. Instead, we employ SGM – as referenced in the last column of Table 1 – and compare it with the algorithms introduced in Sect. 3.3.

Instances. We generate two groups of instances:

Group 1: 10 instances for each triplet of size parameters (m, n, h^p), with the number of players $m \in \{2, 3, 4, 5, 6, 7\}$, the number of markets $n \in \{2, 6, 10\}$ and the cybersecurity cost $h^p = h^p_{\text{root}}, h^p_{\log}$, or $h^p_{\text{nonconvex}}$.

Group 2: 2 instances for each triplet of size parameters (m, n, h^p), with the number of players $m \in \{8, 10, 12, 15\}$, the number of markets $n \in \{2, 8, 15\}$ and the cybersecurity cost $h^p = h^p_{\text{root}}, h^p_{\log}$, or $h^p_{\text{nonconvex}}$.

Group 1 is designed to analyze the performance of our algorithms in detail when compared to the baseline. In contrast, Group 2 focuses on testing the scalability of our methodologies on larger instances. This difference in purpose explains the variation in the number of instances between the two groups. The generation of instance parameters required for the IPG formulation of the CIG is detailed in Appendix 4.2.

In total, we generate 612 instances. In the analysis of the computational results, we often refer to a specific subset of instances with a short name. Specifically, we abbreviate *logarithmic* in *log*, *inverse square root* in *root* and *nonconvex* in *nonconvex* for the cybersecurity costs. In addition, a cybersecurity cost followed by numbers refer to instances with the corresponding cybersecurity cost and number of players. For example, root8-15 refer to instances with the inverse square root function and 8 to 15 players.

Experimental setup.

For all instances, we aim to compute a δ_f-equilibrium for $\delta_f \in \{10^{-2}, 10^{-3}, 10^{-4}\}$. To this end, we apply SGM, direct-approximation method and 2-level approximation method. The best-response solver used by SGM is as described in Table 1. When we employ our methods, the direct-approximation procedure and the 2-level approximation method, the best-response solver is Gurobi because the models are MILPs.

We use an MILP model to compute the solution of the restricted normal-form game, and we solve it via Gurobi. As opposed to Sandholm et al. [25], we avoid using explicit Big-M formulations, and we employ so-called indicator constraints [33]; in practice, this will let the solver decide the best strategy to reformulate the

[2] We do not have access to the commercial state-of-the-art nonconvex solvers according to the Mittelmann's benchmarks [29].

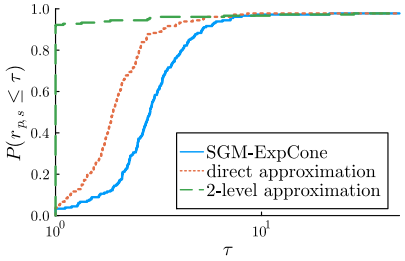
logical conditions. The parameter μ of Algorithms 1 and 2 is set to 0.5 and the initial approximation level δ_0 of Algorithm 2 to 0.05. The value of δ_0 is chosen as a balance between relatively small approximation error and relatively small best-response models; a tolerance of $\delta_0 = 0.05$ implies that a δ-Nash equilibrium computed with it would be a δ-Nash equilibrium with $\delta = 0.1$, which is 1000 times the minimum value used for δ_f in the experiments. Our goal with this value is to show that using a really large value δ_0 compared to δ_f can still be sufficient to find a δ_f-Nash equilibrium faster than with the direct approximation method. We set a time limit of 900 seconds for all methods. The instances are solved using an Intel Core i5-1335U as CPU with 32 GB of RAM. The solvers used are Gurobi 11.0.3, MOSEK 10.2.8 and SCIP 8.1.0. They run on 1 thread for reproducibility. The best-response models solved with Gurobi are modeled with the gurobipy library, while the best-response models solved with MOSEK are modeled with the pyomo library version 6.4. The main part of the code concerning the PWL approximations is in Julia 1.6, while SGM is coded in Python 3.8. Thus there is a little delay each time SGM is called to load the Python environment and libraries. This Python loading time is removed from the computation time because it could have been removed by an implementation in a single language, and it adds uncertainty to computation times.

Appendix B complements the experimental settings described here.

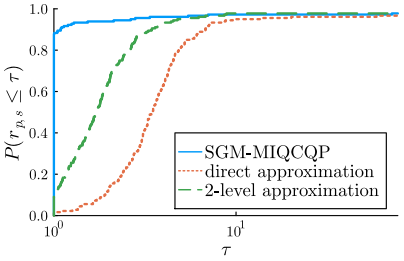
4.3 Computational analysis

Methods Comparison We start by comparing the baseline with our two methods. Figure 3 shows the performance profiles of the three methods for Group 1 instances with the tightest tolerance, $\delta_f = 10^{-4}$. According to [34], we call performance ratio of a given instance with method s and instance i among a set of methods S the number greater or equal to 1 computed as $t_{i,s} / \min_{s' \in S} t_{i,s'}$, where $t_{i,s}$ is the computation time of method s for instance i. A performance profile shows the proportion of instances solved by the corresponding method with a performance ratio of no more than τ. When a performance profile lies strictly above another, it indicates that the method represented by the higher curve performs better on average than the one represented by the lower curve, for all values of τ. In other words, the higher method consistently achieves better results in the set of experiments. In the performance profiles presented, the x-axis, which represents the ratio τ, is on a logarithmic scale.

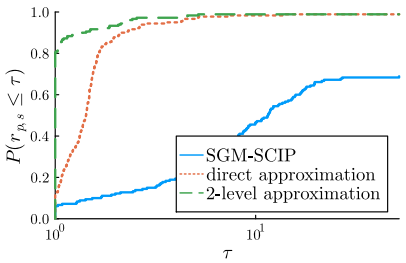
The results reveal a strong dependence of the methods' performance on the choice of the cybersecurity cost. For the logarithmic and nonconvex cybersecurity costs (see Fig. 3a, and c), the 2-level approximation method consistently outperforms the other methods, followed by the direct approximation method and then the baseline (SGM). Notably, for the nonconvex cost, the baseline is significantly outperformed by both proposed methods. In contrast, the results for the inverse square root cost (see Fig. 3b) show that SGM achieves the best performance, followed by the 2-level approximation method. For instance, considering $\tau = 1$, which corresponds to the percentage of instances solved in the minimum time, SGM solves 88% of the instances fastest, whereas the 2-level approximation method does so for only 9% of the instances.



(a) Results for the logarithmic function.



(b) Results for the inverse square root function.



(c) Results for the nonconvex function.

Fig. 3 Performance profiles for Group 1 instances with $\delta_f = 10^{-4}$

For Group 2 instances, the performance trends of the methods remain consistent with those observed in Group 1, although fewer instances are solved within the time limit. Refer to Fig. 6 in Appendix C for the corresponding performance profiles.

Further Insights The results from the previous experiments highlight the potential of the 2-level approximation method. In the following, we examine its strengths.

As summarized in Sect. 3.2, SGM, used both as the baseline and within our methodologies, is an iterative method. Therefore, we analyze not only the solution times, but also the number of SGM iterations made by the methods. Table 2 presents, for each method, the percentage of instances solved, the geometric mean time, and the average number of iterations in the SGM component. The geometric mean is computed as $\sqrt[n]{\Pi_{i\in 1,\ldots,n} t_i}$ with t_i the computation time of instance i. We use it instead of the arithmetic mean to diminish the effect of the outliers on the mean time. The geometric mean time and iteration count are computed only among solved instances. For the 2-level approximation method, the *iter.* count represents the total iterations in both SGM calls, while *first iter.* refers to iterations in the first SGM call (recall Algorithm 2).

The percentage of instances solved and average solving times confirm our conclusion: the 2-level approximation method is more effective for logarithmic and nonconvex cybersecurity costs and, even for the inverse square root function, solves more instances within the time limit with comparable average solving time to the baseline. The 2-level approximation method requires slightly more iterations than the other methods, mostly

Table 2 Overview of computational results with $\delta_f = 10^{-4}$

	SGM			Direct approximation			2-level approximation			
	% solved	time (s)	iter	% solved	time (s)	iter	% solved	time (s)	iter	first iter
log-Group 1	98.0	4.49	20.51	98.0	3.13	20.78	98.0	1.68	22.57	20.84
log-Group 2	50.0	28.37	44.57	54.0	10.96	42.66	54.0	7.62	44.29	42.59
root-Group 1	98.0	1.08	20.51	97.0	3.46	20.65	98.0	1.77	22.86	20.93
root-Group 2	46.0	6.93	40.41	42.0	16.12	40.96	50.0	8.94	44.66	42.29
nonconvex-Group 1	69.0	16.73	18.54	99.0	4.03	18.77	99.0	3.11	20.34	18.68
nonconvex-Group 2	25.0	32.22	29.83	46.0	16.37	30.38	54.0	10.37	31.19	29.79

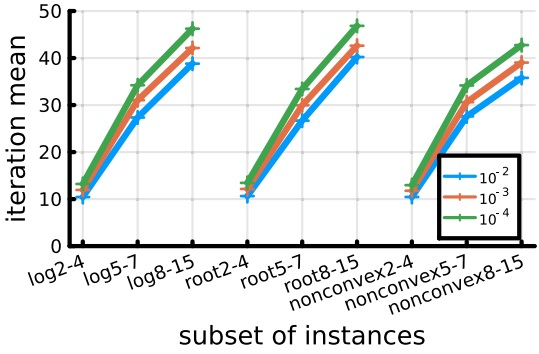


Fig. 4 Geometric mean of iterations in the first call to the SGM for the 2-level approximation method for instances solved for all three tolerances

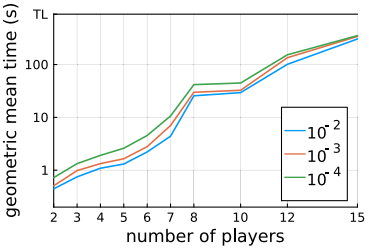
during the first call to the SGM (the maximum number of iterations in the second call is 6). This indicates that the number of iterations in the first call is strongly influenced by the tolerance value δ_f. Figure 4 confirms this, showing the geometric mean of iterations for the first call of the SGM in the 2-level approximation method. For a fixed tolerance, the iteration counts of SGM and of the first iter. of the 2-level approximation method are similar, suggesting that the latter's superior performance stems from faster iterations. Since the only difference lies in the approximated game via PWL approximations, the speedup is attributed to this approach. The second call to SGM by the 2-level approximation benefits from the warm start, making it efficient. This suggests that a K-level approximation method, which iteratively refines the game approximation with decreasing tolerance, and calls to SGM also with decreasing tolerance, could potentially achieve even greater efficiency.

Scalability Assessment We conclude the computational study with a discussion of the scalability limits of the 2-level approximation method.

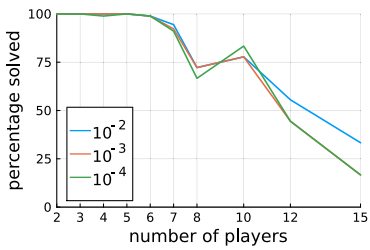
Figure 5 shows the geometric mean times and percentage of instances solved by the 2-level approximation method for different tolerance values, grouped by the number of players and markets. A computation time of 900 s was assigned to runs that reached the time limit to prevent skewing the mean time when a method with a specific tolerance value solves an instance just under the time limit. The y-axis in Fig. 5a, c and e is logarithmic, with TL denoting the 900-second time limit.

The figures show an increase in average computation time and a decrease in the percentage of solved instances as the tolerance value tightens. The geometric mean computation times for instances solved without reaching the time limit are 1.32 s for $\delta_f = 10^{-2}$, 1.67 s for $\delta_f = 10^{-3}$, and 2.48 s for $\delta_f = 10^{-4}$. Thus, instances are solved, on average, 1.88 times faster with $\delta_f = 10^{-2}$ than with $\delta_f = 10^{-4}$, and 1.48 times faster with $\delta_f = 10^{-3}$ than with $\delta_f = 10^{-4}$.

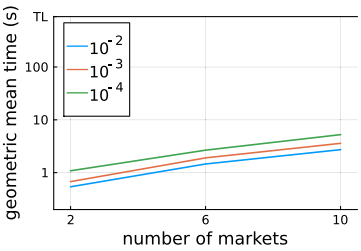
While tolerance values influence the performance of our methodology, Figure 5 shows that the number of players and markets has an even greater impact. Specifically, there is a noticeable drop in the number of solved instances when there are at least 8 markets and more than 8 players. In Appendix C, we present similar figures with curves separated by cybersecurity costs, leading to comparable conclusions about the scalability of our method.



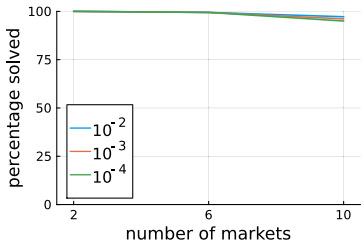
(a) Geometric mean computation time depending on the number of players.



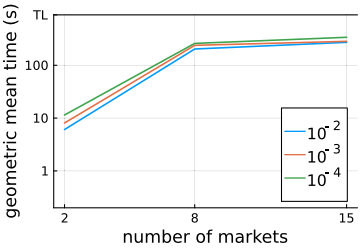
(b) Percentage of solved instances depending on the number of players.



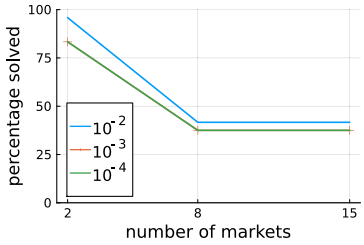
(c) Geometric mean computation time depending on the number of markets for Group 1 instances.



(d) Percentage of solved instances depending on the number of markets for Group 1 instances.



(e) Geometric mean computation time depending on the number of markets for Group 2 instances.



(f) Percentage of solved instances depending on the number of markets for Group 2 instances.

Fig. 5 Scalability of the 2-level approximation method

5 Conclusion

In this work, we proposed a framework to compute approximate Nash equilibria in IPGs with nonlinear payoff functions. We proved that approximating the payoff functions is equivalent to computing approximate equilibria. From a methodological standpoint, we introduced two procedures based on PWL approximations to compute approximate Nash equilibria. Finally, we proved the efficiency of our methods by proposing and solving a cybersecurity investment game, establishing

a computational benchmark. Our best method, the 2-level approximation, seems appropriate when the type of optimization problems encountered in the best responses is not efficiently solved by state-of-the-art solvers compared to MILP solvers. Additionally, the analysis of the number of iterations in the SGM calls within the 2-level approximation method suggests a potential avenue for improvement through the use of multiple levels of progressively tighter game approximations, where each level (i.e., SGM call) benefits from warm-starting based on the previous one.

We believe there are many directions to extend this work. Among those, we propose four ideas. The first idea is the extension of Proposition 1 to different types of pointwise approximation errors, *e.g.*, relative errors [17]. Second, we believe the nonlinear functions can be approximated by PWL approximations refined only on some well-chosen intervals or in an iterative fashion so that those approximations have less pieces, see for example [35]. Third, in addition to SGM, our methods can employ different algorithms to compute equilibria, *e.g.*, the Cut-and-Play algorithm from [3]. Fourth, a numerical study on the impact of the approximation factor–specifically, the compounding of errors induced by Proposition 1–would be valuable. While these errors could affect the anticipated equilibria, it is important to note that this is not a limitation of our methodology per se, but rather a reflection of the tolerance for approximation errors that the decision maker is willing to accept. By quantifying this relationship, future work could provide guidelines for selecting appropriate approximation values based on the desired balance between computational efficiency and solution accuracy in real-world applications.

The cybersecurity investment game as an integer programming game

The model

In CIG, the set of players M represents the retailers selling a homogeneous (or similar) product in a finite set of markets $J = \{1, ..., n\}$. Each retailer $p \in M$ decides the quantity Q_j^p of product sold in each market $j \in J$ such that Q_j^p does not exceed an upper bound $\bar{Q}_j^p$. On the one hand, as the transaction takes place online, external attacker can potentially perform a cyberattack on the transaction; if the attack is successful, the retailer is deemed responsible for it, and incurs in financial and reputation damages. On the other hand, each retailer also decides a level of cybersecurity $s^p \in [0, \bar{s}^p]$ to protect its transactions, with $\bar{s}^p < 1$. The higher the level of cybersecurity s^p, the least likely cyberattacks will be successful on the transactions of player p. Specifically, if $s^p = 1$, then no cyberattack on the transactions of player p will succeed. Furthermore, when player p performs a transaction with a market j, it incurs in a fixed one-time setup cost, activated through a binary variable b_j^p, *i.e.*, $b_j^p = 1$ if and only if player p sells some products in market j. All considered, the optimization problem of each player p is the following box-constrained MINLP:

$$\max \quad \sum_{j\in J} \hat{\rho}_j(Q,s)Q_j^p - c^p_{\text{prod}} \sum_{j\in J} Q_j^p - \sum_{j\in J} c^p_{j,\text{setup}} b_j^p \tag{6a}$$

$$- \sum_{j\in J} \left(c^p_{j,\text{quad}} (Q_j^p)^2 + c^p_{j,\text{lin}} Q_j^p \right) - h^p(s^p) - \beta(s) D^p \tag{6b}$$

$$\text{s.t.} \quad Q^p = \sum_{i\in M} \sum_{j\in J} Q_j^i, \tag{6c}$$

$$0 \le Q_j^p \le b_j^p \bar{Q}_j^p, \quad b_j^p \in \{0,1\} \qquad \forall j \in J, \tag{6d}$$

$$0 \le s^p \le \bar{s}^p. \tag{6e}$$

The objective function of (6a) contains the following terms:

1. **Profits.** The term $\sum_{j\in J} \hat{\rho}_j(Q,s)Q_j^p$ represents the profits of player p. The variable Q^p is the aggregation of the variables Q_j^p for all players p and markets j. The function $\hat{\rho}_j(Q,s)$ is the unitary selling price fixed by market j given by the inverse demand function $(q_j + r_j\bar{s}) - m_j \sum_{i\in M} Q_j^i$, where $\bar{s} = \frac{1}{m}\sum_{p=1}^{m} s^p$ is the players' average cybersecurity level, and q_j, m_j and r_j are parameters. This price decreases with the total quantity of product sold, and increases with the average cybersecurity level of the players.
2. **Production costs.** The term $c^p_{\text{prod}} \sum_{j\in J} Q_j^p$ represents the production costs of player p, where $c^p_{\text{prod}} \in \mathbb{R}_+$ is a parameter.
3. **One-time setup costs.** The term $\sum_{j\in J} c^p_{j,\text{setup}} b_j^p$ represents the one-time setup transaction costs player p pays for entering each market j, with $c^p_{j,\text{setup}} \in \mathbb{R}_+$ being a parameter.
4. **Variable transaction costs.** The term $\sum_{j\in J} \left(c^p_{j,\text{quad}} (Q_j^p)^2 + c^p_{j,\text{lin}} Q_j^p \right)$ represents the variable transaction costs of player p, which is a sum of linear and quadratic terms in Q_j^p. The parameters $c^p_{j,\text{quad}}$ and $c^p_{j,\text{lin}}$ are nonnegative real values.
5. **Cybersecurity costs.** The term $h^p(s^p)$ is a function representing the cybersecurity cost of player p, bounded by the maximum cybersecurity budget B^p. We will precise this term later in the section.
6. **Cybersecurity damage.** The term $\beta(s)D^p$ represents the expected cost player p pays in case of a successful cyberattack. The function $\beta(s)$ is the probability of a successful attack when the security levels are $s = (s^1, ..., s^m)$ and D^p is the estimated cost of a successful cybersecurity attack on player p. The function $\beta(s)$ is given by $(1 - s^p)(1 - \bar{s})$.

The difficulty of (6a–6e) comes from the integer variables b_j^p, the quadratic terms of the players' payoffs and the nonlinear cybersecurity cost $h^p(s^p)$.

Cybersecurity costs

In contrast to [28], we upper bound the cybersecurity level s^p to $\bar{s}^p$. Nagurney et al. [28] employ a nonlinear constraint $h^p(s^p) \leq B^p$, with $h^p : [0,1] \mapsto \mathbb{R}^+$ representing the cost of reaching the cybersecurity level s^p. Nagurney et al. [28] also assumes h^p is an increasing function, and $h^p(0) = 0$, $h^p(1) = \infty$, because, no matter the amount spent on cybersecurity, in reality, a cyberattack is always a possibility. Under these conditions, $h^p(s^p) \leq B^p \Leftrightarrow s^p \leq \bar{s}^p$ with $\bar{s}^p$ uniquely defined by $h^p(\bar{s}^p) = B^p$; therefore, the nonlinear constraint $h^p(s^p) \leq B^p$ can be equivalently replaced by the linear constraint $s^p \leq \bar{s}^p$, with $\bar{s}^p$ strictly less than 1 because of the bounds on h^p. Due to these reasons, in the experiments we use three different functions for the cybersecurity cost: a logarithmic function, an inverse square root function and a nonconvex function. It allows to experiment with best responses of different subclasses of MINLP.

Logarithmic function. The logarithmic function we consider is $h^p_{\log}(s^p) = -\alpha^p log(1 - s^p)$, which is convex on [0, 1[. We can model this function with the linear term $\alpha^p t_{NL}$ in the objective function and the exponential cone constraint $(1 - s^p, 1, t_{NL}) \in K_{exp}$, where K_{exp} is the convex subset of $\mathbb{R}^3$ defined as $\{(x_1, x_2, x_3) : x_1 \geq x_2 \exp(x_3/x_2), x_1, x_2 \geq 0\}$ [30]. The best responses (6a–6e) with this formulation of $h^p_{\log}$ are the optimal solution to exponential cone problems.

Inverse square root function. The inverse square root function $h^p_{\text{root}}(s^p)$ we consider takes the form of $\alpha^p\left(1/\sqrt{1 - s^p} - 1\right)$ with a positive parameter α^p. It is convex on [0, 1[. We can reformulate $h^p_{\text{root}}(s^p)$ to obtain an MIQCQP, with two auxiliary non-negative continuous variables s_{NL}, t_{NL}, two quadratic constraints $s^2_{NL} \leq 1 - s^p$ and $s_{NL} t_{NL} \geq 1$, and a linear term in the objective function $-\alpha^p(t_{NL} - 1)$. Indeed, the term in the objective function forces t_{NL} to be finite because it is a maximization problem. Combined with the constraint $s_{NL} t_{NL} \geq 1$, it forces $s_{NL} > 0$ and $t_{NL} \geq 1/s_{NL}$. In addition, with the constraint $s^2_{NL} \leq 1 - s^p$, we also have $t_{NL} \geq 1/\sqrt{1 - s^p}$. Finally, the objective term $-\alpha^p(t_{NL} - 1)$ represents the inverse square root term $\alpha^p/\sqrt{1 - s^p} - 1$ because the maximization forces t_{NL} to be as small as possible. With this formulation of h^p_{root}, the best responses (6a–6e) are the optimal solutions of an MIQCQP.

Nonconvex function. The nonconvex function we consider is

$$h^p_{\text{nonconvex}}(s^p) = \alpha^p\left(\left(1/\sqrt{1 - s^p} + 2/(1 + exp(-20 s^p))\right)\right) - 2.$$

The function $h^p_{\text{nonconvex}}$ is nonconvex and increasing on [0, 1[. With this function, the best responses (6a–6e) can be obtained by solving a nonconvex MINLP.

For the logarithmic and inverse square root functions, we reformulate the best-response model from a convex MINLP into an exponential cone program and an MIQCQP, respectively, allowing us to leverage solvers tailored to these specific problem classes. This reformulation is expected to reduce computation times.

Table 3 Parameters of the generated instances

Parameter	Uniform distribution's domain
q_j	$\{100, 101, ..., 200\}$
m_j	$\{0.5, 0.51, ..., 2\}$
r_j	$\{0.1, 0.11, ..., 0.5\}$
c^p_{prod}	$\{1, 2, ..., 10\}$
$c^p_{j,\text{setup}}$	$\{500, 501, ..., 2000\}$
$c^p_{j,\text{lin}}$	$\{1, 1.01, ..., 4\}$
$c^p_{j,\text{quad}}$	$\{0.25, 0.26, ..., 1\}$
α^p	$\{1, 2, ..., 10\}$
D^p	$\{50, 51, ..., 100\}$
$\bar{Q}^p_j$	$\{50, 51, ..., 200\}$
B^p	$\{0.5, 1, ..., 5\}$

Instance generation

Given the triplet (m, n, h^p) as the instances Group 1 or 2 (see Sect. 4.2), we generate randomly all parameters that describe the games of our CIG instances. They are chosen among a uniform distribution in regularly sampled intervals close to the values used in the instances of Nagurney et al. [28], as described in Table 3.

Implementation details

We describe here some technical details on the experimental part, such as specific parameter values of solvers.

The formulation of the PWL functions approximating the convex functions (inverse square root and logarithmic functions) are decided by the Gurobi function *addGenConstrPWL*. It is not the case for the nonconvex function because *addGenConstrPWL* can only represent continuous PWL functions and LinA may compute a discontinuous PWL function in the nonconvex case (in the convex case only continuous PWL functions can be produced). Function *addGenConstrPWL* is used because it is tailored to handle a PWL relationship $y = f(x)$. The formulation used for the PWL approximation of the nonconvex function is the disaggregated logarithmic convex combination model [22].

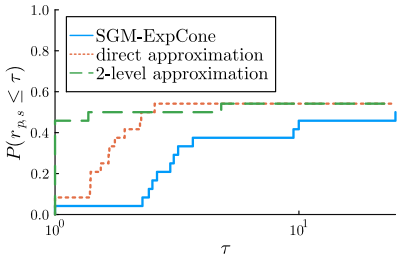
In the SGM, there are two places with tolerances that lead to inexactness of the equilibrium: the best-response computation and when checking the stopping criterion of the SGM, *i.e.* the definition of approximate equilibrium. To ensure that a call to the function `SGM`(G,δ) produces a δ-equilibrium, we set the relative gap of best-response solvers and normal-form game solver to 0 and split δ into two parts $\delta = \delta_{solver} + \delta_{def}$. More precisely, $\delta_{solver} = \frac{\delta}{5}$ is used in best responses and normal-form game computations, while $\delta_{def} = \frac{4\delta}{5}$ is used to check if a candidate strategy

satisfies the definition of δ_{def}-equilibrium. Thus, the stopping criterion of SGM used is, for a given vector of mixed-strategies σ,

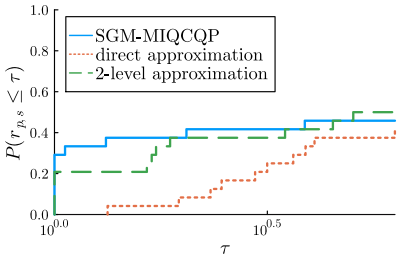
$$\max_{x^p \in \hat{X}^p} \hat{\Pi}^p(x^p;\sigma^{-p}) - \hat{\Pi}^p(\sigma^p;\sigma^{-p}) < \delta_f - \delta_{def}, \qquad \forall p \in M,$$

where $\hat{\Pi}^p$ and $\hat{X}^p$ are the approximated payoff and strategy set of player p. Also, for numerical reasons, a value of 10^{-9} is affected to the parameters *IntFeasTol* and *FeasibilityTol* of Gurobi, *mioTolAbsRelaxInt* and *mioTolFeas* of MOSEK and *numerics/feastol* of SCIP. It sets the feasibility tolerance and the integer feasibility tolerance to a common value for all best-response solvers.

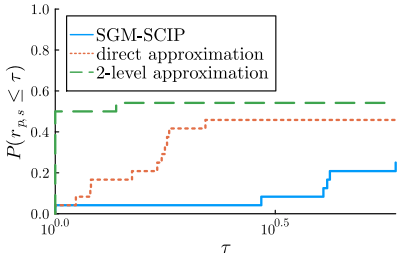
To approximate the cybersecurity costs with PWL functions, we use the Julia package `LinA` [27], an open-source implementation of Algorithm 5 of [27] available at https://github.com/LICO-labs/LinA.jl. As the inverse square root function and the logarithmic function are convex, this algorithm produces a PWL approximation with the minimum number of pieces satisfying the absolute approximation error. Regarding the nonconvex function, the algorithm produces a PWL approximation with at most one more piece than the minimum according to Lemma 4 of [27] and the fact that the nonconvex function is convex and then concave on [0, 1]. Algorithm 4 of [27] would produce a PWL approximation with the minimum number of pieces but it would also take a higher computation time. Thus it was not used to approximate the nonconvex function because we estimated it was



(a) Results for the logarithmic function.

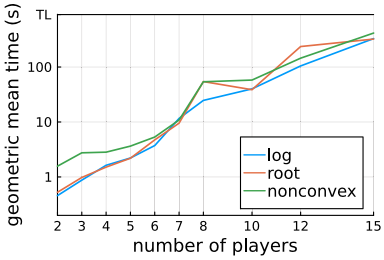


(b) Results for the inverse square root function.

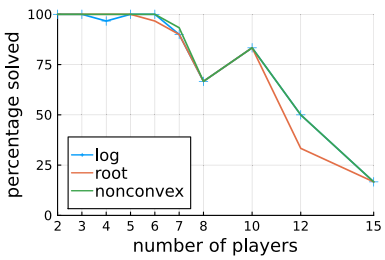


(c) Results for the nonconvex function.

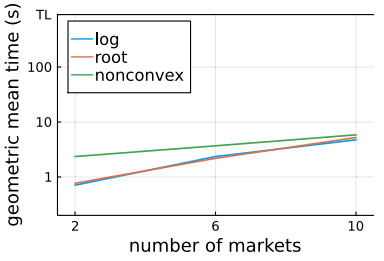
Fig. 6 Performance profiles for Group 2 with $\delta_f = 10^{-4}$



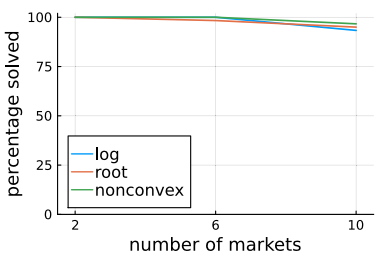
(a) Geometric mean computation time depending on the number of players.



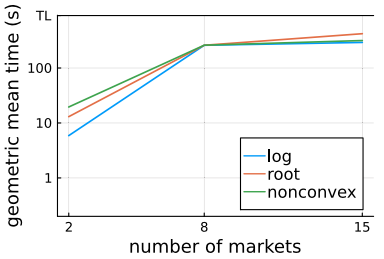
(b) Percentage of solved instances depending on the number of players.



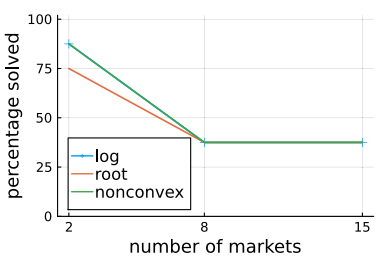
(c) Geometric mean computation time depending on the number of markets for Group 1 instances.



(d) Percentage of solved instances depending on the number of markets for Group 1 instances.



(e) Geometric mean computation time depending on the number of markets for Group 2 instances.



(f) Percentage of solved instances depending on the number of markets for Group 2 instances.

Fig. 7 Influence of cybersecurity costs in the peformance of the 2-level approximation method when $\delta_f = 10^{-4}$

not worth to spend more time computing this approximation to get at best one piece less.

Additional computational results

Methods Comparison. Figure 6 provides the performance profiles for instances of Group 2.

Scalability Assessment.

Figure 7 displays results of the 2-level approximation method with tolerance value $\delta_f = 10^{-4}$. It plots the geometric mean computation time and the percentage of instances solved depending on the number of players or markets. In Fig. 7a, c and e a solve reaching the time limit is counted as 900s, the y-axis is in logarithmic scale and TL represents the time limit of 900s.

Figure 7a demonstrates that computation time increases rapidly with the number of players. Regarding scalability with the number of markets, Figure 7c and e indicate a significant increase in computation time between 2 and 8 markets for instances with at least 8 players, which may represent a threshold between manageable and non-manageable instances. The percentage of instances solved within the time limit, shown in Fig. 7b, d and f, supports this threshold. It remains near 100% for fewer than 8 players but drops significantly for 8 or more players. For 2 markets and more than 8 players, the percentage of solved instances hovers around 80%. Notably, the only instance with 15 players that is solved for all cybersecurity costs involves just 2 markets. Thus, the 2-level approximation method consistently solves instances with 7 or fewer players or instances with more players but only 2 markets. Additionally, the increase in percentage of solved instances between 8 and 10 players in Fig. 7b is not statistically significant, as each data point for 8 or more players is based on only 6 instances.

Acknowledgements The authors are thankful for the support of the ETI program of Toulouse INP through the project POLYTOPT, and Institut de valorisation des données (IVADO) and Fonds de recherche du Québec (FRQ) through the FRQ-IVADO Research Chair and NSERC grant 2019-04557. Gabriele Dragotto is thankful for the support of the *Data X* program from the *Center for Statistics and Machine Learning* at Princeton University. Most of the work of the first author was performed while he was working in LAAS-CNRS, Université de Toulouse.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability The code, instances and the results of the experiments are available at https://github.com/LICO-labs/SGM-and-PWL.

References

1. Köppe, M., Ryan, C.T., Queyranne, M.: Rational Generating Functions and Integer Programming Games. Oper. Res. **59**(6), 1445–1460 (2011). https://doi.org/10.1287/opre.1110.0964
2. Carvalho, M., Dragotto, G., Lodi, A., Sankaranarayanan, S.: Integer programming games: a gentle computational overview. INFORMS Tutorials in Operations Research, 31–51 (2023). Chap. 2. https://doi.org/10.1287/educ.2023.0260
3. Carvalho, M., Dragotto, G., Lodi, A., Sankaranarayanan, S.: The cut-and-play algorithm: computing nash equilibria via outer approximations. Oper. Res. (2025). https://doi.org/10.1287/opre.2023.0327
4. Carvalho, M., Lodi, A., Pedroso, J.P.: Computing equilibria for integer programming games. Eur. J. Oper. Res. (2022). https://doi.org/10.1016/j.ejor.2022.03.048
5. Crönert, T., Minner, S.: Equilibrium Identification and Selection in Finite Games. Oper. Res. 0(0), (2022) https://doi.org/10.1287/opre.2022.2413
6. Dragotto, G., Scatamacchia, R.: The zero regrets algorithm: optimizing over pure Nash equilibria via integer programming. INFORMS J. Comput. **35**(5), 1143–1160 (2023). https://doi.org/10.1287/ijoc.2022.0282
7. Sagratella, S.: Computing All Solutions of Nash Equilibrium Problems with Discrete Strategy Sets. SIAM J. Optim. **26**(4), 2190–2218 (2016)
8. Schwarze, S., Stein, O.: A branch-and-prune algorithm for discrete Nash equilibrium problems. Comput. Optim. Appl. **86**(2), 491–519 (2023). https://doi.org/10.1007/s10589-023-00500-4
9. Carvalho, M., Lodi, A., Pedroso, J.P.: Existence of Nash Equilibria on Integer Programming Games. In: Vaz, A.I.F., Almeida, J.P., Oliveira, J.F., Pinto, A.A. (eds.) Operational Research, pp. 11–23. Springer, Cham (2018)
10. Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., Mahajan, A.: Mixed-integer nonlinear optimization. Acta. Numer. **22**, 1–131 (2013). https://doi.org/10.1017/S0962492913000032
11. Gounaris, C.E., Floudas, C.A.: Tight convex underestimators for C2-continuous problems: I. univariate functions. J. Global Optim. **42**, 51–67 (2008). https://doi.org/10.1007/s10898-008-9287-9
12. Gounaris, C.E., Floudas, C.A.: Tight convex underestimators for C2-continuous problems: II. multivariate functions. J. Global Optim. **42**, 69–89 (2008). https://doi.org/10.1007/s10898-008-9288-8
13. Liberti, L.S.: Reformulation and Convex Relaxation Techniques for Global Optimization. PhD thesis, Imperial College London (2004). https://doi.org/10.1007/s10288-004-0038-6
14. Geißler, B., Martin, A., Morsi, A., Schewe, L.: Using Piecewise Linear Functions for Solving MINLPs. In: Lee, J., Leyffer, S. (eds.) Mixed Integer Nonlinear Programming, pp. 287–314. Springer, New York, NY (2012)
15. Zhang, H., Wang, S.: Linearly constrained global optimization via piecewise-linear approximation. J. Comput. Appl. Math. **214**(1), 111–120 (2008). https://doi.org/10.1016/j.cam.2007.02.006
16. Rebennack, S., Krasko, V.: Piecewise Linear Function Fitting via Mixed-Integer Linear Programming. INFORMS J. Comput. **32**(2), 507–530 (2020). https://doi.org/10.1287/ijoc.2019.0890
17. Ngueveu, S.U.: Piecewise linear bounding of univariate nonlinear functions and resulting mixed integer linear programming-based solution methods. Eur. J. Oper. Res. **275**(3), 1058–1071 (2019). https://doi.org/10.1016/j.ejor.2018.11.021
18. Rebennack, S., Kallrath, J.: Continuous Piecewise Linear Delta-Approximations for Bivariate and Multivariate Functions. J. Optim. Theory Appl. **167**, 102–117 (2015). https://doi.org/10.1007/s10957-014-0688-2
19. Kazda, K., Li, X.: Nonconvex multivariate piecewise-linear fitting using the difference-of-convex representation. Computers & Chemical Engineering **150**, 107310 (2021). https://doi.org/10.1016/j.compchemeng.2021.107310
20. Kazda, K., Li, X.: A linear programming approach to difference-of-convex piecewise linear approximation. Eur. J. Oper. Res. (2023)
21. Duguet, A., Ngueveu, S.U.: Piecewise Linearization of Bivariate Nonlinear Functions: Minimizing the Number of Pieces Under a Bounded Approximation Error. In: Ljubić, I., Barahona, F., Dey, S.S., Mahjoub, A.R. (eds.) Combinatorial Optimization, pp. 117–129. Springer, Cham (2022)
22. Vielma, J.P., Ahmed, S., Nemhauser, G.: Mixed-Integer Models for Nonseparable Piecewise-Linear Optimization: Unifying Framework and Extensions. Oper. Res. **58**(2), 303–315 (2010). https://doi.org/10.1287/opre.1090.0721
23. Nash, J.: Non-cooperative Games. Ann. Math. **54**(2), 286–295 (1951)

24. Chen, X., Deng, X.: Settling the complexity of two-player Nash equilibrium. In: Foundations of Computer Science, 2006. FOCS '06. 47th Annual IEEE Symposium On, pp. 261–272 (2006). https://doi.org/10.1109/FOCS.2006.69
25. Sandholm, T., Gilpin, A., Conitzer, V.: Mixed-integer programming methods for finding nash equilibria. In: AAAI, pp. 495–501 (2005)
26. Porter, R., Nudelman, E., Shoham, Y.: Simple search methods for finding a Nash equilibrium. Games and Economic Behavior **63**(2), 642–662 (2008)
27. Codsi, J., Ngueveu, S.U. Gendron, B.: LinA: a faster approach to piecewise linear approximations using corridors and its application to mixed-integer optimization. Math. Program. Comput. **17**, 265–306 (2025). https://doi.org/10.1007/s12532-024-00274-8
28. Nagurney, A., Daniele, P., Shukla, S.: A supply chain network game theory model of cybersecurity investments with nonlinear budget constraints. Annals of Operations Research (2017). https://doi.org/10.1007/s10479-016-2209-1
29. Mittelmann, H.: Benchmarks for Optimization Software. https://plato.asu.edu/bench.html. Accessed: 2023-12-21
30. MOSEK: MOSEK Optimization Reference Manual. (2023). https://docs.mosek.com/10.0/toolbox/index.html
31. Gurobi: Gurobi Optimizer Reference Manual (2023). https://www.gurobi.com
32. Gamrath, G., Anderson, D., Bestuzheva, K., Chen, W.-K., Eifler, L., Gasse, M., Gemander, P., Gleixner, A., Gottwald, L., Halbig, K., Hendel, G., Hojny, C., Koch, T., Le Bodic, P., Maher, S.J., Matter, F., Miltenberger, M., Mühmer, E., Müller, B., Pfetsch, M.E., Schlösser, F., Serrano, F., Shinano, Y., Tawfik, C., Vigerske, S., Wegscheider, F., Weninger, D., Witzig, J.: The SCIP Optimization Suite 7.0. Technical report (March 2020). http://www.optimization-online.org/DB_HTML/2020/03/7705.html
33. Bonami, P., Lodi, A., Tramontani, A., Wiese, S.: On mathematical programming with indicator constraints. Math. Program. **151**, 191–223 (2015)
34. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Mathematical Programming (2002). https://doi.org/10.1007/s101070100263
35. Contardo, C., Ngueveu, S.U.: On the approximation of separable non-convex optimization programs to an arbitrary numerical precision. Technical report, Concordia University, Montréal, Canada, Université de Toulouse, CNRS, INP, LAAS, Toulouse, France (2021). https://hal.science/hal-03336022/document