Computing Nash equilibria for integer programming games

Margarida Carvalho [c,a]

Andrea Lodi [a,b]

João Pedro Pedroso [d]

[a] Canada Excellence Research Chair in Data Science for Real-Time Decision-Making

[b] Department of Mathematics and Industrial Engineering, Polytechnique Montréal (Québec) Canada, H3C 3A7

[c] DIRO, Université de Montréal, C.P. 6128, Succursale Centre-Ville, Montréal (Québec) Canada, H3C 3J7

[d] Faculdade de Ciências Universidade do Porto and INESC TEC, Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal.

carvalho@iro.umontreal.ca
andrea.lodi@polymtl.ca
jpp@fc.up.pt

Canada Excellence Research Chair in Data Science for Real-Time Decision-Making

Abstract: In this paper, we develop algorithmic approaches for a recently defined class of games, the integer programming games. Two general methods to approximate an equilibrium are presented and enhanced in order to improve their practical efficiency. Their performance is analysed through computational experiments in a knapsack game and a competitive lot-sizing game. To the best of our knowledge, this is the first time that equilibria computation methods for general integer programming games are build and computationally tested.

Keywords: Nash equilibria; Mixed integer programming; Algorithmic game theory.

Acknowledgments: Part of this work was performed while the first author was in the Faculty of Sciences University of Porto and INESC TEC. The first author thanks the support of Institute for data valorisation (IVADO), the Portuguese Foundation for Science and Technology (FCT) through a PhD grant number SFRH/BD/79201/2011,POPH/FSE.

Computing Nash equilibria for integer programming games

Margarida Carvalho * Andrea Lodi † João Pedro Pedroso ‡

submitted: October, 2018

Abstract

In this paper, we develop algorithmic approaches for a recently defined class of games, the integer programming games. Two general methods to approximate an equilibrium are presented and enhanced in order to improve their practical efficiency. Their performance is analysed through computational experiments in a knapsack game and a competitive lot-sizing game. To the best of our knowledge, this is the first time that equilibria computation methods for general integer programming games are build and computationally tested.

Keywords— Nash equilibria; Mixed integer programming; Algorithmic game theory.

1 Problem Statement and background

Following the seminal work of Köppe *et al.* [15], Carvalho *et al.* [3, 4] defined *integer programming games* (IPGs) that will be the subject of this paper. In order to formally define an IPG, let us fix the following notation. If C^i are sets for $i \in \mathcal{C}$, then $C = \prod_{i\in\mathcal{C}} C^i$ and the operator $(\cdot)^{-i}$ denotes $(\cdot)$ for all $j \in \mathcal{C} \setminus \{i\}$.

An IPG is defined as a game with a finite set of *players* $M = \{1, 2, \ldots, m\}$ such that each player $p \in M$ has the *set of strategies*

$$X^p = \{x^p : A^p x^p \leq b^p, \quad x^p_i \in \mathbb{N} \text{ for } i = 1, \ldots, B_p\},$$

where A^p is a $r_p \times n_p$ rational matrix (with $n_p \geq B_p$), b^p a rational column vector of dimension r_p, and a continuous payoff function $\Pi^p : X \to \mathbb{R}$ that can be evaluated in polynomial time.

An IPG is a *non-cooperative complete information game*, *i.e.*, players are self-interested and have full information of each other payoffs and strategies. We restrict our focus to the *simultaneous* case, *i.e.*, players play simultaneously. Each player p's goal is to select her *best response* against the opponents strategy $x^{-p} \in X^{-p}$, *i.e.* to solve

$$\underset{x^p \in X^P}{\text{maximize}} \quad \Pi^p(x^p, x^{-p}). \tag{1}$$

Given that players take decisions simultaneously, the expected game outcome is a *profile of strategies* $x \in X$ that solves the optimization problem (1) for all players, which is called *pure Nash equilibrium*. Since pure equilibria may fail to exist, a broader concept exists. Let Δ^p denote the space of Borel probability measures over X^p and $\Delta = \prod_{p\in M} \Delta^p$. Each player p's expected payoff for a profile of strategies $\sigma \in \Delta$ is

$$\Pi^p(\sigma) = \int_{X^p} \Pi^p(x^p, x^{-p}) d\sigma. \tag{2}$$

A *Nash equilibrium* (NE) [21] is a profile of strategies $\sigma \in \Delta$ such that

$$\Pi^p(\sigma) \geq \Pi^p(x^p, \sigma^{-p}), \qquad \forall p \in M \qquad \forall x^p \in X^p. \tag{3}$$

*DIRO, Université de Montréal, C.P. 6128, Succursale Centre-Ville, Montréal, QC H3C 3J7, Canada.

†Canada Excellence Research Chair, École Polytechnique de Montréal, C.P. 6128, Succursale Centre-Ville, Montréal, QC H3C 3A7, Canada.

‡Faculdade de Ciências Universidade do Porto and INESC TEC, Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal.

An ε-equilibrium ($\varepsilon \geq 0$) is a profile of strategies $\sigma \in \Delta$ such that

$$\Pi^p(\sigma) + \varepsilon \geq \Pi^p(x^p, \sigma^{-p}), \qquad \forall p \in M \qquad \forall x^p \in X^p. \tag{4}$$

For a player's p strategy $\sigma^p \in \Delta^p$, its *support* is defined as $\text{supp}(\sigma^p) = \{x^p \in X^p : \quad \sigma^p(x^p) > 0\}$, *i.e.*, the set of player p's strategies played with positive probability. When each player's support size is 1 for a given $\sigma \in \Delta$, then σ is called a *pure profile of strategies*, otherwise, we call it (strictly) mixed. For the sake of simplicity, whenever the context makes it clear, we use the term (strategy) profile to refer to a pure one.

Player p's payoff function is called *separable* if

$$\Pi^p(x) = \sum_{j_1=1}^{k_1} \cdots \sum_{j_m=1}^{k_m} a^p_{j_1 \ldots j_m} f^1_{j_1}(x^1) \ldots f^m_{j_m}(x^m), \tag{5}$$

where $a^p_{j_1 \ldots j_m} \in \mathbb{R}$ and the f^p_j are real-valued continuous functions. An IPG such that each player payoff function is separable and strategy set is nonempty and bounded is called *separable*.

In [4] the following useful results were proven:

Theorem 1 (Carvalho *et al.* [4]) *Every IPG such that X^p is nonempty and bounded has a Nash equilibrium.*

Theorem 2 (Carvalho *et al.* [4]) *For any Nash equilibrium σ of a separable IPG, there is a Nash equilibrium τ such that each player p mixes among at most $k_p + 1$ pure strategies and $\Pi^p(\sigma) = \Pi^p(\tau)$.*

Theorem 1 ensures that under a mild condition on the players' sets of strategies, an IPG has a NE. Furthermore, if an IPG is separable, any NEcan be described through a finite number of the players' strategies. These results will guarantee the correctness of our algorithmic approach.

2 Contributions

Our main contribution is in the development of a flexible framework to compute an equilibrium for IPGs. Based on the theorems of the previous section, we are able to show that *(i)* our framework is guaranteed to compute an NE for IPGs in which all the player's decision variables are finite and bounded, and *(ii)* it is guaranteed to compute an ε-equilibrium for IPG satisfying some mild conditions that are expected to be satisfied by real-world games. Nevertheless, our framework is capable of processing any IPG, although, it might fail to stop, *e.g.*, if the input game has no equilibria.

Our framework requires game theory and mathematical optimization algorithms. Different algorithms in these fields are implemented in different solvers. Our framework provides the user with the flexibility of selecting the solvers to be used. In particular, the user can select solvers that might be more efficient for the specific IPG in hand. However, for the game theory solver, we strongly advice the use of Porter-Nudelman-Shoham method due to its practical efficiency, simple implementation and easy integration of heuristics. The later characteristic will be explored and it also represents another degree of freedom in our framework. In summary, the flexibility of our method should be understood as a way of making it more efficient for the game at hand. To conclude the paper, we evaluate our algorithms through computational experiments that allow us to understand their performance. Given that this is the first general algorithm for IPGs, there is no other method in the literature to which our experiments can be compared.

Our paper is structured as follows. Section 3 reviews the literature in algorithmic game theory for the computation of Nash equilibria. Section 4 formalizes our framework and develops two methods to compute ε-equilibria for IPGs (*approximated* NE), providing specialized functions to speed up the methods. In Section 5, we introduce two relevant IPGs, the knapsack game and the competitive lot-sizing game, and validate our methods through computational experiments on these games. Finally, we conclude and discuss further research directions in Section 6.

3 Related literature

There are important real-world games (*e.g.*, electricity markets [23], production planning [20], heath-care [5]), where each player's payoff maximization subject to her set of feasible strategies is described by a mixed integer programming formulation as required in the definition of IPGs. This motivates the importance of understanding the equilibria of IPGs. Moreover, IPGs contain the well-known class of finite games (see [4]) (games with a finite number of strategies and arbitrary payoff functions) and quasi-concave games (strategies sets are convex and payoffs are quasi-concave). The existent tools and standard approaches for finite games and quasi-concave games are not directly applicable to IPGs. Additionally, the literature on IPGs focuses in the particular structure of specific games.

Kostreva [16] describes the first theoretical approach to compute pure equilibria to IPGs, where integer variables are required to be binary. The binary requirement in a binary variable x is relaxed by adding in the payoff a penalty $Mx(1-x)$. Then, the Karush-Kuhn-Tucker (KKT) [14, 17] conditions are applied to each player optimization problem and merged into a system of equations for which the set of solutions contains the set of pure equilibria. To find the solutions for that system of equations, the author recommends the use of a path following in a homotopy [27] or Gröbner basis [8]. Additionally, it must be verified which of the system's solutions are equilibria[1], which results in long computational times. Gabriel *et al.* [10] proposed an optimization model for which the optimal solution is a pure Nash equilibrium of a game that approximates an IPG with concave payoffs. In that paper, integer requirements are relaxed, the players' concave optimization problems are transformed in constrained problems through the KKT conditions and the complementary conditions are relaxed (not required to be satisfied) in order to satisfy the integrality requirements. On the few experimental results presented, this approach leads to the computation of a pure Nash equilibrium for the original game. However, there is neither a theoretical nor computational evidence showing the applicability of these ideas to the general case. Köppe *et al.* [15] were the pioneers to investigate IPGs with all the players' decision variables restricted to be integer. The payoff functions in [15] are differences of piecewise-linear concave functions. In order to compute NE, the authors use generating functions of integer points inside of polytopes. Finally, the application of Köppe *et al.*'s results rely on computational implementations that are still in the preliminary stage, although theoretically, the approach can be proven to run in polynomial time (under restrictive conditions, like number of players fixed and sum of the number of players' decision variables fixed, to name few). Hemmecke *et al.* [13] considered IPGs with an additional feature: a player p's set of feasible strategies depends on the opponents' strategies. The authors study (generalized) pure equilibria assuming that the player's decision variables are all integer and payoffs are monotonously decreasing in each variable.

Lee and Baldick [18] study the computation of mixed equilibria for an IPG in the context of the electric power market. There, the player's set of strategies is approximated through a discretization of it, resulting in a finite game to which there are general algorithms to compute NE. Nevertheless, there is a trade-off between having a good discretized approximation and an efficient computation of NE: the more strategies are contained in the discretization, the longer the time to compute a NE will be.

None of the approaches above solve general IPGs, failing to either consider mixed NE or players' strategies with continuous and integer decision variables.

4 Algorithmic approach

In [4] it was proven that even deciding the existence of a NE for a two-player IPG with bilinear payoffs is complete for the second level of the polynomial hierarchy, which is a class of problems believed to be hard to solve. As we show next, even when an IPG is guaranteed to have an equilibrium it is unlikely that it can be computed in polynomial time.

Any finite game is an IPG [4]. Furthermore, Chen *et al.* [7]'s result stated that solving a finite game even with only two players is PPAD-complete. Informally, for a PPAD-complete problem it is known that a

[1]The KKT conditions applied to non-concave maximization problems are only necessary.

solution exists; however the proof of solution existence is not constructive and it is believed to be "hard" to compute it. The result in [7] together with Theorem 1 and the fact that finite games are separable (see [4]) leads to:

Lemma 3 *The problem of computing an equilibrium to a separable IPG is PPAD-hard.*

In what follows, we focus in separable IPGs since their set of NE can be characterized by finitely-supported equilibria (Theorem 2).

Next, in Section 4.1, we will analyze the standard idea in mathematical programming of relaxing the integrality requirements and we argue that this seems not to provide useful information. Thus, the problem must be tackled from another perspective. The algorithm designed in Section 4.2 will approximate an IPG iteratively. This framework incorporates an algorithm for the computation of an NE for finite games and a mathematical programming solver. Given the practical effectiveness of Porter-Nudelman-Shoham method (PNS) [25] for finite games, we review it and explore its flexibility. The basic algorithm obtained from our framework is modified in Section 4.2.2, in an attempt to improve its performance.

4.1 Game relaxations

A typical procedure to solve optimization problems consists in relaxing constraints that are hard to handle and to use the information associated with the solution of the relaxed problem to guide the search for the optimum. Thus, in this context, such ideas seem a natural direction to investigate. Call *relaxed integer programming game* (RIPG) the game resulting from an IPG when the integrality constraints are removed. In the following examples, we compare the NE between IPG and the associated RIPG.

Example 1 (RIPG has more equilibria than IPG) *Consider an instance with two players, in which player A solves*

$$\max_{x^A} 5x_1^A x_1^B + 23x_2^A x_2^B \text{ subject to } 1 \leq x_1^A + 3x_2^A \leq 2 \text{ and } x^A \in \{0,1\}^2$$

and player B solves

$$\max_{x^B} 5x_1^A x_1^B + 23x_2^A x_2^B \text{ subject to } 1 \leq x_1^B + 3x_2^B \leq 2 \text{ and } x^B \in \{0,1\}^2.$$

There is only one feasible strategy for each player in the IPG. Thus, it is easy to see that IPG has a unique equilibrium: $(x^A, x^B) = ((1,0),(1,0))$. *This equilibrium also holds for RIPG. However, RIPG possesses at least one more equilibrium:* $(x^A, x^B) = ((0, \frac{2}{3}), (0, \frac{2}{3}))$.

Example 2 (RIPG with less equilibria than IPG) *Consider the duopoly game such that player A solves*

$$\max_{x^A} 12x_1^A x_1^B + 5x_2^A x_2^B \text{ subject to } 2x_1^A + 2x_2^A \leq 3 \text{ and } x^A \in \{0,1\}^2,$$

and player B solves

$$\max_{x^B} 12x_1^A x_1^B + 5x_2^A x_2^B + 100x_1^B \text{ subject to } 2x_1^B + x_2^B \leq 1 \text{ and } x^B \in \{0,1\}^2.$$

There are at least 2 equilibria: $(x^A, x^B) = ((0,0),(0,0))$ *and* $(x^A, x^B) = ((0,1),(0,1))$*; however, none is an equilibrium of the associated RIPG. In fact, in the RIPG, it is always a dominant strategy for player B to select* $x^B = (\frac{1}{2}, 0)$*, and the unique equilibrium is* $(x^A, x^B) = ((1,0),(\frac{1}{2},0))$. *In conclusion, the game has at least 2 equilibria while the associated relaxation has 1.*

These examples show that no bounds on the number of NE and, thus, on the players' payoffs in an NE can be extracted from the relaxation of an IPG. Moreover, there are no general methods to compute mixed equilibria of RIPGs, implying that we would be restricted to pure equilibria (which may fail to exist).

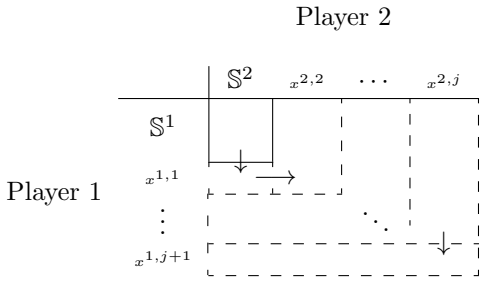

Figure 1: SGM: Sampled Generation Method for $m = 2$. The notation $x^{p,k}$ represents the player p's strategy added at iteration k. A vertical (horizontal) arrow represents player 1 (player 2) incentive to unilaterally deviate from the sampled game computed equilibrium to a new strategy of her.

4.2 Algorithm formalization

Our goal is to determine an NE. Thus, from the Nash equilibria definition, we aim to find $(\sigma^1, \ldots, \sigma^m)$ such that

$$\sigma^p \in \Delta^p \qquad \forall p \in M \tag{6a}$$

$$\Pi^p(\sigma^p, \sigma^{-p}) \geq \Pi^p(x^p, \sigma^{-p}) \qquad \forall p \in M, \qquad \forall x^p \in X^p, \tag{6b}$$

that is, determine a mixed profile of strategies such that no player has incentive to unilaterally deviate from it. The number of pure strategies in each X^p is likely to be infinite or, in case the variables are all required to be integer and bounded, to be exponential. Thus, in general, the equilibria inequalities (6b) are unsuitable to be written for each pure strategy in X^p. We call *sampled game* of an IPG to the finite game that results from restricting the players to a finite subset of feasible strategies of X.

We then follow the motivating idea of column generation [11] and cutting plane [12] approaches: not all variables and constraints in problem (6) may be needed to find a solution. In this context, the natural idea to find a solution to the constrained programming problem (6) is through generation of strategies: start by solving the constrained problem for a finite subset of feasible strategies $\mathbb{S} = \mathbb{S}^1 \times \mathbb{S}^2 \times \ldots \mathbb{S}^m$ (this is, compute an equilibrium for the sampled game); while there is a strategy for player p that gives her an incentive to deviate from the computed equilibrium, add the "destabilizing" strategy to $\mathbb{S}^p$. We call this scheme *sampled generation method* (SGM). Figure 1 illustrates the increase in the number of players' strategies as SGM progresses. Intuitively, we expect that SGM will enumerate the most "relevant" strategies and/or "saturate" the space X after a sufficient number of iterations and thus, approximate to an equilibrium. Hopefully, we would not need to enumerate all feasible strategies in order to compute an equilibrium. In an IPG, there might exist players' decision variables which are continuous. Under this case, SGM can only guarantee the computation of an ε-equilibrium. In this way, $\varepsilon > 0$ becomes an input of SGM and the stopping criterion becomes the following: if there is no player able to unilaterally increase her payoff at the equilibrium σ of the current sampled game more than ε, return σ. Before providing the SGM's proof of correctness, in an attempt to clarify the method, we apply it to an instance of IPGs.

Example 3 (Computing an equilibrium with SGM) *Consider an IPG with two players. Player i wishes to maximize the payoff function $\max_{x^i>0} -(x^i)^2 + x^i x^{-i}$. The best reaction is given by $x^i(x^{-i}) = \frac{1}{2}x^{-i}$, for $i = 1, 2$. The only equilibrium is $(x^1, x^2) = (0, 0)$. Initialize SGM with the sampled game $\mathbb{S}^i = \{10\}$ for $i = 1, 2$, then in each iteration each player reduces by half the value of her variable, see Figure 2. Thus, SGM converges to the equilibrium $(0, 0)$. If in the input of SGM, $\varepsilon = 10^{-6}$ then, after 14 iterations, SGM would return an ε-equilibrium of the game.*

Our goal is to guarantee that SGM computes an ε-equilibrium in finite time. To this end, some conditions on the IPGs are necessary. If a player p's set of feasible strategies is unbounded, the game may fail to have equilibria, and even if it has equilibria, SGM may not converge. Furthermore, as pointed out by Stein *et al.* [26] for a specific separable game, it seems that there must be some bound on the speed variation of

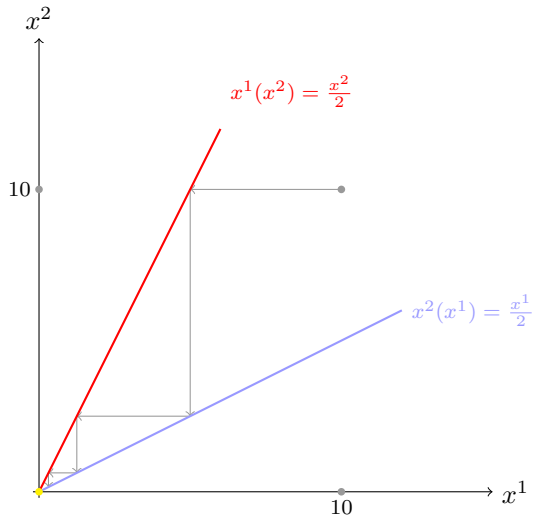

Figure 2: Players' best reaction functions.

the payoffs in order to guarantee that an algorithm computes an equilibrium in finite time. The Lipschitz condition ensures this bound.

Theorem 4 *If X is nonempty and bounded, then in a finite number of steps, SGM computes*

1. *an equilibrium if all players' decision variables are integer;*
2. *an ε-equilibrium if each player p's payoff function is Lipschitz continuous in X^p.*

Proof. Since X is bounded, by Corollary 2, there is a finitely supported NE.

SGM stops once an equilibrium of the sampled game coincides with an equilibrium (case 1) or ε-equilibrium (case 2) of the IPG. Suppose that the method does not stop. This means that in every iteration at least a new strategy is added to the current $\mathbb{S}$.

Case 1: Given that X is bounded and players' variables are integer, each player has a finite number of strategies. Thus, after a finite number of iterations, the sampled game will coincide with IPG, *i.e.*, $\mathbb{S} = X$. This means that an NE of the sampled game is an NE of the IPG.

Case 2: Each player p payoff function is Lipschitz continuous in X^p, which means that there is a positive real number L^p such that

$$|\Pi^p(x^p, \sigma^{-p}) - \Pi^p(\hat{x}^p, \sigma^{-p})| \leq L^p \parallel x^p - \hat{x}^{-p} \parallel \quad \forall x^p, \hat{x}^p \in X^p,$$

where $\parallel \cdot \parallel$ is the Euclidean norm.
The set $\mathbb{S}$ strictly increases from one iteration of SGM to the next one. Thus, after a sufficient number of iterations, for each player p, given $x^p \in X^p$ there is $\hat{x}^p \in \mathbb{S}^p$ such that $\parallel x^p - \hat{x}^p \parallel \leq \frac{\varepsilon}{L^p}$. Let σ be an NE of the sampled game. Then,

$$\begin{aligned}
\Pi^p(x^p, \sigma^{-p}) - \Pi^p(\sigma) &= \Pi^p(x^p, \sigma^{-p}) - \Pi^p(\hat{x}^p, \sigma^{-p}) + \Pi^p(\hat{x}^p, \sigma^{-p}) - \Pi^p(\sigma) \\
&\leq \Pi^p(x^p, \sigma^{-p}) - \Pi^p(\hat{x}^p, \sigma^{-p}) \\
&\leq |\Pi^p(x^p, \sigma^{-p}) - \Pi^p(\hat{x}^p, \sigma^{-p})| \\
&\leq L^p \parallel x^p - \hat{x}^{-p} \parallel \leq L^p \frac{\varepsilon}{L^p} \leq \varepsilon.
\end{aligned}$$

The first step follows from the fact that σ is an NE of the sampled game and thus $\Pi^p(\hat{x}^p, \sigma^{-p}) \leq \Pi^p(\sigma)$. The next inequality holds because we are just applying the absolute value. The third step follows from the fact the player p's payoff is Lipschitz continuous in X^p. In this way, σ is an ε-equilibrium of the IPG. □

A payoff function which is linear in that player's variables is Lipschitz continuous; a quadratic payoff function when restricted to a bounded set satisfies the Lipschitz condition as will be the case of the competitive lot-sizing game described in Section 5.1.2. Therefore, this seems to be a reasonable condition in practice.

4.2.1 Computation of NE for finite games

A relevant fact about computing equilibria for a sampled game with the set of strategies $\mathbb{S} \subseteq X$ is that $\mathbb{S}$ is finite and, consequently, enables the use of general algorithms to compute mixed equilibria. Given the good results achieved by PNS [25] for the computation of a NE in normal-form games, this is the method that our framework will apply to solve the sampled games (additional advantages for adopting PNS will be given in the end of this section). PNS solves the constrained program (6) associated with a sampled game (*i.e.*, $X = \mathbb{S}$) through the resolution of simpler subproblems (note that in constraints (6b) the expected profits $\Pi^p(\sigma^p, \sigma^{-p})$ are highly non-linear due to the multiplication of the probability variables). To this end, PNS bases its search for an equilibrium σ by guessing its support and using the fact that in an equilibrium $\sigma \in \Delta$, each player must be indifferent among her strategies in the support at which her payoff is maximized. Thus, an equilibrium σ of a sampled game $\mathbb{S}$ satisfies

$$\Pi^p(\sigma) = \Pi^p(\hat{x}^p, \sigma^{-p}) \quad \forall p \in M, \quad \forall \hat{x}^p \in \text{supp}(\sigma^p) \tag{8a}$$

$$\Pi^p(\sigma) \geq \Pi^p(x^p, \sigma^{-p}) \quad \forall p \in M, \quad \forall x^p \in \mathbb{S}^p \tag{8b}$$

$$\sum_{x^p \in \text{supp}(\sigma^p)} \sigma^p(x^p) = 1 \qquad \forall p \in M \tag{8c}$$

$$\sigma^p(x^p) \geq 0 \qquad \forall p \in M, \quad \forall x^p \in \text{supp}(\sigma^p) \tag{8d}$$

$$\sigma^p(x^p) = 0 \qquad \forall p \in M, \quad \forall x^p \in \mathbb{S}^p - \text{supp}(\sigma^p), \tag{8e}$$

with $\text{supp}(\sigma^p) \subseteq \mathbb{S}^p$. Problem (8) is called *Feasibility Problem*. When the payoff functions are quadratic on x, the constraints in Problem (8) become linear, and thus, it can be solved in polynomial time.

The computation of an NE to the sampled game $\mathbb{S}$ reduces to *(i)* finding an NE support and *(ii)* solving the associated Feasibility Problem. Therefore, support sets in $\mathbb{S}$ are enumerated and the corresponding Feasibility Problems are solved, until an NE is found (*i.e.*, a Feasibility Problem is proven to be feasible). PNS implements this enumeration with an additional step that decreases the number of Feasibility Problems to be solved, in other words, reduces the number of candidates to be the support of an equilibrium. A strategy $x^p \in X^p$ is *conditionally dominated* given a subset of strategies $R^{-p} \subseteq X^{-p}$ for the remaining players, if the following condition holds

$$\exists \hat{x}^p \in X^p \;\; \forall x^{-p} \in R^{-p} : \quad \Pi^p(x^p, x^{-p}) < \Pi^p(\hat{x}^p, x^{-p}). \tag{9}$$

PNS prunes the support enumeration search by making use of conditionally dominated strategies, since such strategies will never be selected with positive probability in an equilibrium. In addition, we consider in the support enumeration the property given by Theorem 2: each player p has a support size of at most $k_p + 1$; recall that to determine $k_p + 1$, one just needs write player p's payoff as in the form (5).

We conclude SGM description by highlighting an additional advantage of PNS, besides being in practice the fastest algorithm. The authors' implementation of PNS [25] searches the equilibria by following a specific order for the enumeration of the supports. In specific, for two players' games, $|M| = 2$, the algorithm starts by enumerating supports, first, by increasing order of their total size and, second, by increasing order of their balance (absolute difference in the players' support size). The idea is that in the case of two players, each equilibrium is likely to have supports with the same size and small. When $|M| > 2$, PNS exchanges the importance of these two criteria. We expect SGM to start converging to an equilibrium as it progresses. Therefore, it may be advantageous to use the past computed equilibria to guide the support enumeration. Including rules for support enumeration in PNS is straightforward; these rules can be problem specific. On the other hand, doing so for other state-of-the-art algorithms is not as easy. For instance, the well-known Lemke-Howson algorithm [19] implies to start the search for equilibria in an artificial equilibrium or in an equilibrium of the game (allowing to compute a new one). Thus, since at iteration k of SGM, none of the equilibria computed for the sampled games in iterations 1 to $k-1$ is an NE of the current sampled game, there

is no direct way of using past information to start or guide the Lemke-Howson algorithm. Moreover, this algorithm's search is performed by enumerating vertices of polytopes built according to the game strategies. Therefore, since in each iteration of SGM a new strategy is added to the sampled game, these polytopes may change deeply.

4.2.2 Modified SGM

Finally, through the tools described, we can slightly change the scheme of SGM presented in Section 4.2 in an attempt to speed up its running time. Its new version will be a depth-first search: while in SGM the size of the sampled game strictly increases from one iteration to the next one, in its depth-first search version it will be possible to backtrack to previous sampled games, with the aim of decreasing the size of the sampled game. In each iteration of the improved SGM, we search for an equilibrium which has in the support the last strategy added to the sampled game; in case such equilibrium does not exist, the method backtracks, and computes a new equilibrium to the previous sampled game. While in each iteration of SGM all supports can be considered, in the *modified* SGM (m-SGM) we limit the search to the ones with the new added strategy. Therefore, this modified SGM attempts to keep the size of the sampled game small and decreases the number of supports enumerated.

Next, we concentrate in proving under which conditions the m-SGM computes an ε-equilibrium in finite time and provide its detailed description.

Theorem 5 *Let $\mathbb{S} = \mathbb{S}^1 \times \mathbb{S}^2 \times \ldots \times \mathbb{S}^m$ represent a sampled game associated with some IPG. If the normal-form game that results from $\mathbb{S}$ has a unique equilibrium σ, then one of the following implications holds:*

1. *σ is an equilibrium of the IPG;*
2. *given any player p with incentive to deviate from σ^p to $x^p \in X^p$, the normal-form game associated with $\mathbb{S}' = \mathbb{S}^1 \times \ldots \mathbb{S}^{p-1} \times \mathbb{S}^p \cup \{x^p\} \times \mathbb{S}^{p+1} \times \ldots \times \mathbb{S}^m$ has x^p in the support of all its equilibria.*

Proof. Suppose σ is not an equilibrium of the IPG. Then, by the definition of equilibrium, there is a player, say player p, with incentive to unilaterally deviate to some $x^p \in X^p \setminus \mathbb{S}^p$. By contradiction, assume that there is an equilibrium τ in $\mathbb{S}'$ such that x^p is played with zero probability (it is not in the support of τ). First, τ is different from σ because now $\mathbb{S}'$ contains x^p. Second, τ is an equilibrium for the game restricted to $\mathbb{S}$, contradicting the fact that σ was its unique equilibrium. □

In this way, if in an iteration of SGM the sampled game has an unique NE, in the subsequent iteration, we can prune the support enumeration search of PNS by forcing the new strategy added to be in the support of the NE to be computed. Note that it might occur that in the consecutive sampled games there is more than one NE and thus, an equilibrium with the new added strategy in the support may fail to exist (Theorem 5 does not apply). Therefore, backtracking is introduced so that a previously processed sampled game can be revisited and its support enumeration continued in order to find a new NE and to follow a promising direction in the search. The algorithm m-SGM is described in Algorithm 1. The algorithms called by it are described in Table 1 and can be defined independently. We will propose an implementation of them in Section 5.2.

Let us explain in more detail our method for which Figure 3 can be illustrative. Fundamentally, whenever m-SGM moves *forward*, Step 3, a new strategy $x(k+1)$ is added to the sampled game k that is expected to be in the support of the equilibrium of that game (due to Theorem 5). For the sampled game k, if the algorithm fails to compute an equilibrium with $x(k)$ in the support and $\mathbb{S}_{dev_{k+1}}$ not in the supports, see "if" part of Step 4, the algorithm *backtracks*: it *revisits* the sampled game $k-1$ with $\mathbb{S}_{dev_k}$ added, so that no equilibrium is recomputed. It is crucial for the correctness of the m-SGM that it starts from a sampled game of the IPG with an unique equilibrium; to this end, the initialization determines one feasible solution for each player. See example 5 in the Appendix A to clarify the application of m-SGM.

Next, m-SGM correctness will be proven.

Lemma 6 *In the m-SGM, the sampled game 0 is never revisited.*

Algorithm 1: Modified SGM (m-SGM).

Input: An IPG instance and $\varepsilon > 0$.

Output: ε-equilibrium, last sampled game and number of the last sampled game.

Step 1 **Initialization:**

$\mathbb{S} = \prod_{p=1}^{m} \mathbb{S}^p \leftarrow Initialization(IPG)$

$k \leftarrow 0$

set $\mathbb{S}_{dev_k}$, $\mathbb{S}_{dev_{k+1}}$ and $\mathbb{S}_{dev_{k+2}}$ to be $\prod_{p=1}^{m} \emptyset$

$\sigma_k \leftarrow (1, \ldots, 1)$ is Nash equilibrium of the current sampled game $\mathbb{S}$

$list \leftarrow PlayerOrder(\mathbb{S}_{dev_0}, \ldots, \mathbb{S}_{dev_k})$

Step 2 **Termination:**

while *list non empty* **do**

  $p \leftarrow list.pop()$

  $x(k+1) \leftarrow DeviationReaction(p, \sigma_k^{-p}, \Pi^p(\sigma_k), \varepsilon, IPG)$

  if $\Pi^p(\sigma_k) + \varepsilon < \Pi^p(x(k+1), \sigma_k^{-p})$ **then**

    go to Step 3

return σ_k, $\mathbb{S}$, k

Step 3 **Generation of next sampled game:**

$k \leftarrow k + 1$

$\mathbb{S}^p_{dev_k} \leftarrow \mathbb{S}^p_{dev_k} \cup \{x(k)\}$

$\mathbb{S}^p \leftarrow \mathbb{S}^p \cup \{x(k)\}$

$\mathbb{S}_{dev_{k+2}} \leftarrow \prod_{p=1}^{m} \emptyset$

Step 4 **Solve sampled game k:**

$Sizes_{ord} \leftarrow SortSizes(\sigma_0, \ldots, \sigma_{k-1})$

$Strategies_{ord} \leftarrow SortStrategies(\mathbb{S}, \sigma_0, \ldots, \sigma_{k-1})$

$\sigma_k \leftarrow \text{PNS}_{adaptation}(\mathbb{S}, x(k), \mathbb{S}_{dev_{k+1}}, Sizes_{ord}, Strategies_{ord})$

if $PNS_{adaptation}(\mathbb{S}, x(k), \mathbb{S}_{dev_{k+1}}, Sizes_{ord}, Strategies_{ord})$ *fails to find equilibrium* **then**

  $\mathbb{S} \leftarrow \mathbb{S} \backslash \mathbb{S}_{dev_{k+1}}$

  remove from memory σ_{k-1} and $\mathbb{S}_{k+2}$

  $k \leftarrow k - 1$

  go to Step 4

else

  $list \leftarrow PlayerOrder(\mathbb{S}_{dev_0}, \ldots, \mathbb{S}_{dev_k})$

  go to Step 2

ALGORITHMS	DESCRIPTION
$Initialization(IPG)$	Returns sampled game of the IPG with one feasible strategy for each player.
$PlayerOrder(\mathbb{S}_{dev_0}, \ldots, \mathbb{S}_{dev_k})$	Returns a list of the players order that takes into account the algorithm history.
$DeviationReaction(p, \sigma_k^{-p}, \Pi^p(\sigma_k), \varepsilon, IPG)$	If there is $x^p \in X^p$ such that $\Pi^p(x^p, \sigma_k^{-p}) > \Pi^p(\sigma_k) + \varepsilon$, returns x^p; otherwise, returns any player p's feasible strategy.
$SortSizes(\sigma_0, \ldots, \sigma_{k-1})$	Returns an order for the support sizes enumeration that takes into account the algorithm history.
$SortStrategies(\mathbb{S}, \sigma_0, \ldots, \sigma_{k-1})$	Returns a order for the players' strategies in $\mathbb{S}$ that takes into account the algorithm history.
$\text{PNS}_{adaptation}(\mathbb{S}, x(k), \mathbb{S}_{dev_{k+1}}, Sizes_{ord}, Strategies_{ord})$	Applies PNS in order to return a Nash equilibrium σ of the sampled game $\mathbb{S}$ of the IPG such that $x(k) \in supp(\sigma)$ and $\mathbb{S}_{dev_{k+1}} \cap supp(\sigma) = \emptyset$; makes the support enumeration according with $Sizes_{ord}$ and $Strategies_{ord}$.

Table 1: Specialized algorithms.

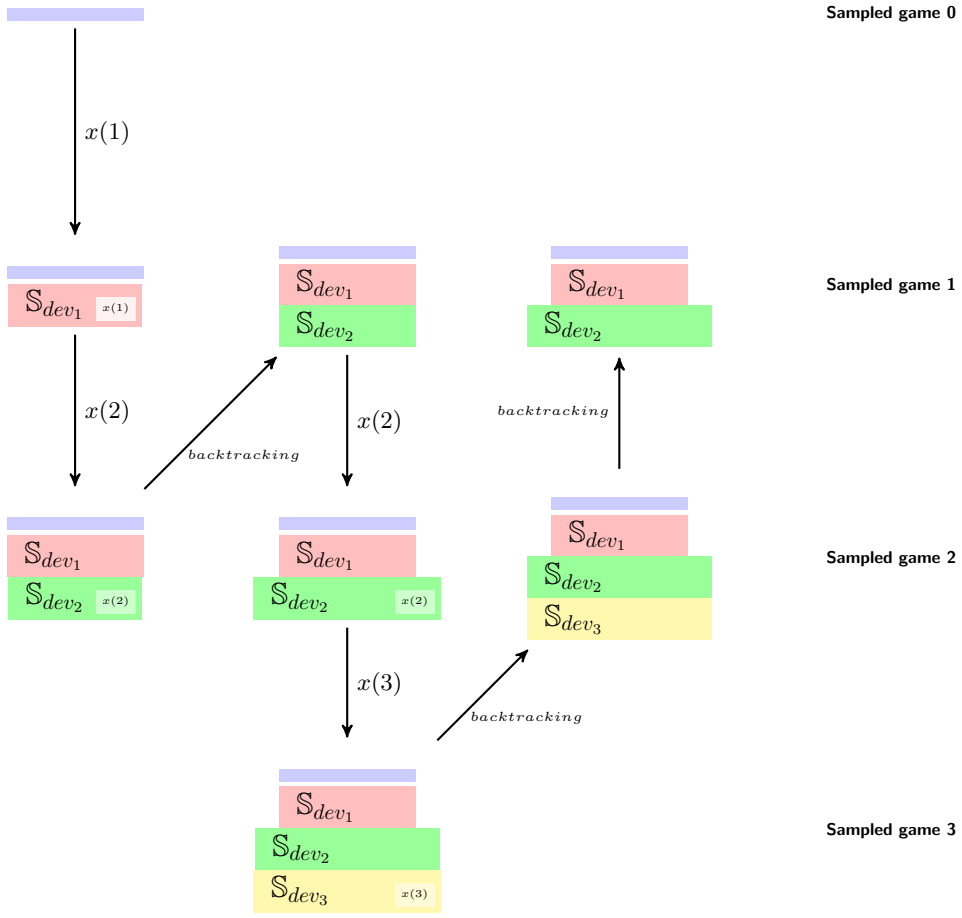

Figure 3: Illustration of the sampled games generated by modified SGM during its execution.

Proof. If the sampled game 0 is revisited, it would be because the algorithm backtracks. Suppose that at some sampled game $k > 0$, the algorithm consecutively backtracks up to the sampled game 0. Consider the first sampled game $j < k$ that is revisited in this consecutive bactracking such that the last time that it was built by the algorithm it had an unique equilibrium where $x(j)$ was in the support and its successor, sampled game $j + 1$, had multiple equilibria. By Theorem 5, when the algorithm moves forward from this sampled game j to $j + 1$, all its equilibria have $x(j + 1)$ in their support. Therefore, at this point, the m-SGM successfully computes an equilibrium and moves forward. The successor, sampled game $j + 2$, by construction, has at least one equilibrium and all its equilibria must have $x(j+1)$ or $x(j+2)$ in the supports. Thus, either the algorithm *(case 1)* backtracks to the sampled game $j+1$ or *(case 2)* proceeds to the sampled game $j + 3$. In case 1, the algorithm successfully computes an equilibrium with $x(j + 1)$ in the support and without $x(j + 2)$ in the support, since the backtracking proves that there is no equilibrium with $x(j + 2)$ in the support and, by construction, the sampled game $j + 1$ has multiple equilibria. Under case 2, the same reasoning holds: the algorithm will backtrack to the sampled game $j + 2$ or move forward to the sampled game $j+3$. In this way, because of the multiple equilibria in the successors of sampled game j, the algorithm will never be able to return to the sampled game j and thus, to the sampled game 0. □

Observe that when a sampled game $k - 1$ is revisited, the algorithm only removes the strategies $\mathbb{S}_{dev_{k+1}}$ from the current sampled game k "if" part of Step 4. This means that in comparison with the last time that the algorithm builds the sampled game $k - 1$, it has the additional strategies $\mathbb{S}_{dev_k}$. Therefore, there was a strictly increase in the size of the sampled game $k - 1$.

Lemma 7 *There is a strict increase in the size of the sampled game k when the m-SGM revisits it.*

Corollary 8 *If X is nonempty and bounded, then in a finite number of steps, m-SGM computes*

1. *an equilibrium if all players' decision variables are integer;*
2. *an ε-equilibrium if each player p's payoff function is Lipschitz continuous in X^p.*

Proof. The **while** Step 2 ensures that when the algorithm stops, it returns an equilibrium (case 1) or ε-equilibrium (case 2). Since by Lemma 6 the algorithm does not revisit sampled game 0, it does not run into an error. Moreover, if the algorithm is moving forward to a sampled game k, there is a strict increase in the size from the sampled game $k-1$ to k. If the algorithm is revisiting a sampled game k, by Lemma 7, there is also a strict increase of it in comparison with the previous sampled game k. Therefore, applying the reasoning of Theorem 4 proof, the m-SGM will compute an equilibrium (case 1) or ε-equilibrium (case 2) in a finite number of steps. □

The m-SGM is initialized with a sampled game that contains one strategy for each player and thus, ensures that the equilibrium of it is unique. However, note that in our proof of the algorithm correctness any initialization with a sampled game with an unique equilibrium is valid. Furthermore, the m-SGM might be easily adapted in order to be initialized with a sampled game containing more than one NE. In the adaptation, backtracking to the sampled game 0 can occur and thus, the PNS support enumeration must be total, this is, all NE of the sampled game 0 must be feasible. The fundamental reasoning is similar to the one of the proof of Lemma 6: if there is backtracking up to the initial sampled game 0, it is because it must contain an NE not previously computed, otherwise the successors would have successfully computed one.

5 Computational investigation

Section 5.1 presents the two (separable) simultaneous IPGs, a knapsack game and a competitive lot-sizing game, in which m-SGM and SGM will be tested. In Section 5.2, our implementations of the specific components in Table 1 are described, which have practical influence in the algorithms' performance. Our algorithms are validated in Section 5.3 by computational results on instances of the two presented IPGs.

5.1 Case studies

Next, the two games in which we test our algorithms are described: the knapsack game, the simplest purely integer programming game that one could devise, and the competitive lot-sizing game, whose practical applicability is discussed.

5.1.1 Knapsack game.

One of the most simple and natural IPGs would be one with each player's payoff function being linear in her variables. This is our main motivation to analyze the knapsack game. Under this setting, each player p aims to solve

$$\max_{x^p \in \{0,1\}^n} \sum_{i=1}^{n} v_i^p x_i^p + \sum_{k=1, k\neq p}^{m} \sum_{i=1}^{n} c_{k,i}^p x_i^p x_i^k \tag{10a}$$

$$\text{s. t.} \quad \sum_{i=1}^{n} w_i^p x_i^p \leq W^p. \tag{10b}$$

The parameters of this game are integer (but are not required to be non-negative). This model can describe situations where m entities aim to decide in which of n projects to invest such that each entity budget constraint (10b) is satisfied and the associated payoffs are maximized (10a). The second summation in the payoff (10a) can describe a benefit, $c_{k,i}^p > 0$, or a penalization, $c_{k,i}^p < 0$, when both player p and player k invest in project i; note also that since all variables are binary $(x_i^p)^2 = x_i^p$; player p's payoff function is linear in x^p (note that in our algorithms, when we verify if a player has incentive to deviate from her current strategy, the variables x^{-p} are fixed).

In the knapsack game, each player p's set of strategies X^p is bounded, since she has at most 2^n feasible strategies. Therefore, by Corollary 2, it suffices to study finitely supported equilibria.

Since payoffs are linear, through the proof of Corollary 2, we deduce that the bound on the equilibria supports for each player is $n+1$. We can sightly improve this bound using basic polyhedral theory (see Nemhauser and Wolsey [22]). First, note that a player p's optimization problem is linear in her variables, implying her set of pure optimal strategies to a fixed profile of strategies $\sigma^{-p} \in \Delta^{-p}$ to be in a facet of $\text{conv}(X^p)$. Second, the part in the payoffs of player p's opponents that depends on player p's strategy, only takes into account the expected value of x^p. The expected value of x^p is a convex combination of player p's pure strategies. Thus, putting together these two observations, when player p selects an optimal mixed strategy σ^p to σ^{-p}, the expected value of x^p is in a facet of $\text{conv}(X^p)$. A facet of $\text{conv}(X^p)$ has dimension $n-1$, therefore, by Carathéodory's theorem [1], any point of this facet can be written as a convex combination of n strategies of X^p. Thus,

Lemma 9 *Given an equilibrium σ of the knapsack game, there is an equilibrium τ such that $|supp(\tau^p)| \leq n$ and $\Pi^p(\sigma) = \Pi^p(\tau)$, for each $p = 1, \ldots, m$.*

5.1.2 Competitive lot-sizing game.

The competitive lot-sizing game [6] is a Cournot competition played through T periods by a set of firms (players) that produce the same good. Each firm has to plan its production as in the lot-sizing problems (see [24]) but, instead of satisfying a known demand in each period of the time horizon, the demand depends on the total quantity of the produced good that exists in the market. Each firm p has to decide how much will be produced in each time period t (production variable x_t^p) and how much will be placed in the market (variable q_t^p). There are set-up and variable (linear) production costs, upper limit on production quantities, and a producer can build inventory (variable h_t^p) by producing in advance. In this way, we obtain the following model for each firm p:

$$\max_{y^p, x^p, q^p, h^p} \quad \sum_{t=1}^{T}(a_t - b_t \sum_{j=1}^{m} q_t^j) q_t^p - \sum_{t=1}^{T} F_t^p y_t^p - \sum_{t=1}^{T} C_t^p x_t^p - \sum_{t=1}^{T} H_t^p h_t^p \tag{11a}$$

$$\text{subject to} \quad x_t^p + h_{t-1}^p = h_t^p + q_t^p \qquad \text{for } t = 1, \ldots, T \tag{11b}$$

$$0 \leq x_t^p \leq M_t^p y_t^p \qquad \text{for } t = 1, \ldots, T \tag{11c}$$

$$h_0^p = h_T^p = 0 \tag{11d}$$

$$y_t^p \in \{0, 1\} \qquad \text{for } t = 1, \ldots, T \tag{11e}$$

where F_t^p is the set-up cost, C_t^p is the variable cost, H_t^p is the inventory cost and M_t^p is the production capacity for period t; $a_t - b_t \sum_{j=1}^{m} q_t^j$ is the unit market price. The payoff function (11a) is firm p's total profit; constraints (11b) model product conservation between periods; constraints (11c) ensure that the quantities produced are non-negative and whenever there is production ($x_t^p > 0$), the binary variable y_t^p is set to 1 implying the payment of the setup cost F_t^p.

Each firm p's payoff function (11a) is quadratic in q^p due to the term $\sum_{t=1}^{T} -b_t (q_t^p)^2$. Next, we show that it satisfies the Lipschitz condition which guarantees that our algorithms compute an ε-equilibrium in finite time. Noting that player p does not have incentive to select $q_t^p > \frac{a_t}{b_t}$ (since it would result in null market price), we get

$$\begin{aligned}
|\sum_{t=1}^{T} b_t (q_t^p)^2 - \sum_{t=1}^{T} b_t (\hat{q}_t^p)^2| &= |\sum_{t=1}^{T} b_t \left((q_t^p)^2 - (\hat{q}_t^p)^2 \right)| \\
&= |\sum_{t=1}^{T} b_t \left((q_t^p) + (\hat{q}_t^p) \right) \left((q_t^p) - (\hat{q}_t^p) \right)| \\
&\leq \sqrt{\sum_{t=1}^{T} b_t^2 \left((q_t^p) + (\hat{q}_t^p) \right)^2} \sqrt{\sum_{t=1}^{T} \left((q_t^p) - (\hat{q}_t^p) \right)^2}
\end{aligned}$$

$$\leq \sqrt{\sum_{t=1}^{T} b_t^2 \left(\frac{2a_t}{b_t}\right)^2} \cdot \| q^p - \hat{q}^p \|$$
$$\leq \sqrt{\sum_{t=1}^{T} 4a_t^2} \cdot \| q^p - \hat{q}^p \| .$$

In the third step, we used Cauchy–Schwarz inequality. In the forth inequality, we use the upper bound $\frac{a_t}{b_t}$ on the quantities placed in the market.

In [6], it is proven that there is a function that is *potential* for this game; a maximum of this function is a (pure) equilibrium. This is an additional motivation to analyze our framework in this problem: it can be compared with the maximization of the associated potential.

5.2 Implementation details

Both our implementations of the m-SGM and SGM use the following specialized functions.

Initialization(IPG) The m-SGM stops once an equilibrium is computed. Therefore, the equilibrium computed will depend on its initialization as the following example illustrates.

Example 4 *Consider an instance of the two-player competitive lot-sizing game with the following parameters:* $T = 1$, $a_1 = 15$, $b_1 = 1$, $M_1^1 = M_1^2 = 15$, $C_1^1 = C_1^2 = H_1^1 = H_1^2 = 0$, $F_1^1 = F_1^2 = 15$. *It is a one-period game, therefore the inventory variables,* h_1^1 *and* h_1^2*, can be removed and the quantities produced are equal to the quantities placed in the market (that is,* $x_1^1 = q_1^1$ *and* $x_1^2 = q_1^2$*). Given the simplicity of the players optimization programs* (11)*, we can analytically compute the players' best reactions that are depicted in Figure 4.*

The game possesses two (pure) equilibria: $\hat{x} = (\hat{x}^1, \hat{y}^1, \hat{x}^2, \hat{y}^2) = (0, 0, 7.5, 1)$ *and* $\tilde{x} = (\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2) = (7.5, 1, 0, 0)$. *Thus, depending on the initialization of m-SGM, it will terminate with* $\hat{x}$ *or* $\tilde{x}$*: Figure 4 depicts the convergence to* $\hat{x}$ *when the initial sampled game is* $\mathbb{S} = \{(2, 1)\} \times \{(5, 1)\}$ *and to* $\tilde{x}$ *when the initial sampled game is* $\mathbb{S} = \{(4, 1)\} \times \{(1, 1)\}$.

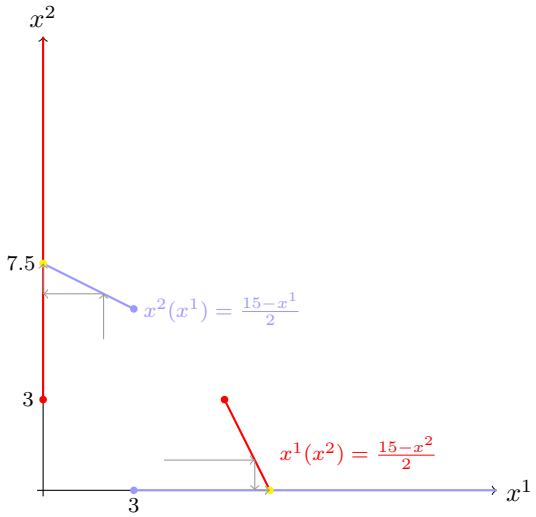

Figure 4: Players' best reaction functions.

In an attempt to keep as small as possible the size of the sampled games (*i.e.*, number of strategies explicitly enumerated), the initialization implemented computes a unique strategy for each player. We experimented initializing the algorithm with the social optimal strategies (strategies that maximize the total

players' payoffs), pure equilibrium for the potential part of the game[2] and optimal strategies if the players were alone in the game (*i.e.*, opponents' variables were set to be zero). There was no evident advantage for one of these initializations. This result was somehow expected, since, particularly for the knapsack game instances, it is not evident whether the game has an important coordination part (in the direction of social optimum) or an important conflict part. Therefore, our implementation initializes with the players' strategies that are optimal when they were alone in the game.

$PlayerOrder(\mathbb{S}_{dev_0}, \ldots, \mathbb{S}_{dev_k})$ The equilibrium returned by our algorithm depends on the players' order when we check their incentives to deviate: for the equilibrium σ_k of the sampled game k, there might be more than one player with incentive to deviate from σ_k, thus the successor will depend on the player that is selected. If players' index order is considered, the algorithm may take longer to converge to an equilibrium: it will be likely that it first finds an equilibrium of the game restricted to players 1 and 2, then an equilibrium of the game restricted to players 1, 2 and 3, and so on. Thus, this implementation sorts the players by decreasing order of number of previous iterations without receiving a new strategy.

$DeviationReaction(p, \sigma_k^{-p}, \Pi^p(\sigma_k), \varepsilon, IPG)$ When checking if a player p has incentive to deviate, it suffices to determine whether she has a strategy that strictly increases her payoff when she unilaterally deviates to it. Nowadays, there are software tools that can solve mixed integer linear and quadratic programming problems[3] effectively. Thus, our implementation solves the player p's best reaction problem (1) to σ_k^{-p}. We use Gurobi 5.6.3 [2] to solve these reaction problems.

$SortSizes(\sigma_0, \ldots, \sigma_{k-1})$ The authors of PNS [25] recommend that the support strategies' enumeration starts with support sizes ordered, first, by total size ($\sum_{p=1}^{m} s^p$), and, second, by a measure of balance (except, in case of a 2-players game where the criteria importance is reversed). However, in our method, from one sampled game to its successor or predecessor, the sampled game at hand just changes by one strategy, and thus we expect that the equilibria will not change too much either (in particular, the support sizes of consecutive sampled games are expected to be close). Therefore, our criterion to sort the support sizes is by increasing order of:

For $m = 2$: first, balance, second, maximum player's support size distance to the one of the previously computed equilibrium, third, maximum player's support size distance to the one of the previously computed equilibrium plus 1 and, fourth, sum of the players' support sizes;

For $m \geq 3$: first, maximum player's support size distance to the one of the previously computed equilibrium, second, maximum player's support size distance to the one of the previously computed equilibrium plus 1, third, sum of the players' support sizes and, fourth, balance.

For the initial sampled game, the criteria coincide with PNS.

$SortStrategies(\mathbb{S}, \sigma_0, \ldots, \sigma_{k-1})$ Following the previous reasoning, the strategies of the current sampled game are sorted by decreasing order of their probability in the predecessor equilibrium. Thus, the algorithm will prioritize finding equilibria using the support strategies of the predecessor equilibrium.

Note that the function $\text{PNS}_{adaptation}(\mathbb{S}, x(k), \mathbb{S}_{dev_{k+1}}, Sizes_{ord}, Strategies_{ord})$ is specific for the m-SGM. The basic SGM calls PNS without any requirement on the strategies that must be in the support of the next equilibrium to be computed; in other words, $x(k)$ and $\mathbb{S}_{dev_{k+1}}$ are not in the input of the PNS.

5.3 Computational results

In this section, we will present the computational results for the application of the modified SGM and SGM to the knapsack and competitive lot-sizing games in order to define a benchmark and to 3validate the importance

[2] We only experimented this for the knapsack game, since the competitive lot-sizing is already potential. We consider the potential part of the knapsack game, when the parameters $c^p_{k,i}$ in each player's payoff function are replaced by $\frac{1}{2}(c^p_{k,i} + c^k_{p,i})$ in player p's payoff.

[3] In the knapsack game, a player's best reaction problem is an integer linear programming problem. In the competitive lot-sizing problem, a player best reaction problem is a mixed integer quadratic programming problem (it becomes continuous and concave once the binary variables y^p are fixed).

of the modifications introduced. For the competitive lot-sizing game, we further compare these two methods with the maximization of the game's potential function (which corresponds to a pure equilibrium). For building the game's data, we have used the Python's random module; see [9]. These instances are available upon request from the first author (M.C.). All algorithms have been coded in Python 2.7.2. Since for both the knapsack and competitive lot-sizing games the Feasibility Problems are linear (due to the bilateral interaction of the players in each of their objective functions), we use Gurobi 5.6.3 to solve them. The experiments were conducted on a Quad-Core Intel Xeon processor at 2.66 GHz and running under Mac OS X 10.8.4.

5.3.1 Knapsack Game

In our computations, the value of ε was zero since this is a purely integer programming game. The parameters v_i^p, $c_{k,i}^p$, and w_i^p are drawn independently from a uniform distribution in the interval $[-100, 100] \cap \mathbb{Z}$. For each value of the pair (n, m), 10 independent instances were generated. The budget W^p is set to $\lfloor \frac{\text{INS}}{11} \sum_{i=1}^{n} w_i^p \rfloor$ for the instance number "INS".

Tables 2 and 3 report the results of m-SGM and SGM algorithms. The tables show the number of items (n), the instance identifier ("INS"), the CPU time in seconds ("time"), the number of sampled games ("iter"), the type of equilibrium computed, pure ("pNE") or strictly mixed ("mNE"), in the last case, the support size of the NE is reported, the number of strategies in the last sampled game ($\prod_{p=1}^{m} |\mathbb{S}^p|$) and the number of backtrackings ("numb. back"). We further report the average results for each set of instances of size n. The algorithms had a limit of one hour to solve each instance. Runs with "tl" in the column time indicate the cases where algorithms reached the time limit. In such cases, the support size of the last sampled game's equilibrium is reported and we do not consider them in the average results row.

As the instance size grows, both in the size n and in the number of players m, the results make evident the advantage of the m-SGM. Since a backward step is unlikely to take place and the number of sampled games is usually equal for both algorithms, the advantage is in the support enumeration: m-SGM reduces the support enumeration space by imposing at iteration k the strategy $x(k)$ to be in the support of the equilibrium, while SGM does not. Later in this section, we discuss the reasons why backtracking is unlikely to occur.

In Table 2, we can observe that for instance 4 with $n = 100$, the m-SGM performed more iterations than SGM. This atypical case is due to the fact that both algorithms have different support enumeration priorities, and therefore, they compute the same equilibria on their initial iterations, but at some point, the algorithms may determine different equilibria, leading to different successor sampled games, and thus, terminating with different outputs. This event is more likely to occur on games with several equilibria.

We note that the bound n for the players' support sizes in an equilibrium (recall Lemma 9) did not contribute to prune the search space of PNS support enumeration, since the algorithm terminates with sampled games of smaller size.

5.3.2 Competitive lot-sizing game

Through dynamic programming, a player p's best reaction (11) for a fixed $(y^{-p}, x^{-p}, q^{-p}, h^{-p})$ can be computed in polynomial time if there are no production capacities, neither inventory costs [6]. For this reason, we decided to concentrate on this simplest instances, and to consider neither bounds in the production capacities (in practice, the production capacities are very large) nor inventory costs.

In our computations, the value of ε was 10^{-6}. The parameters a_t, b_t, F_t^p and C_t^p are drawn independently from a uniform distribution in the intervals $[20, 30] \cap \mathbb{Z}$, $[1, 3] \cap \mathbb{Z}$, $[10, 20] \cap \mathbb{Z}$, $[5, 10] \cap \mathbb{Z}$, respectively. For each value of the pair (m, T), 10 instances were generated.

For easiness of implementation and fair comparison with the computation of the potential function optimum, we do not use the dynamic programming procedure to solve a player best reaction problem, but Gurobi 5.6.3.

As previously mentioned, Section 5.1.2, the ULSG is potential, which implies the existence of a pure equilibrium. In particular, each sampled game of the competitive lot-sizing game is potential and thus has a

		m-SGM						SGM					
n	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$	numb. back	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$	
20	0	0.04	4	0	[2,2]	[3, 2]	0	0.03	4	0	[2,2]	[3, 2]	
	1	0.08	7	0	[2,2]	[5, 3]	0	0.07	7	0	[2,2]	[5, 3]	
	2	0.35	15	0	[4,4]	[9, 7]	0	0.41	15	0	[4,4]	[9, 7]	
	3	0.75	13	0	[5,5]	[7, 7]	0	0.97	13	0	[5,5]	[7, 7]	
	4	0.04	4	1	0	[3, 2]	0	0.04	4	1	0	[3, 2]	
	5	0.03	3	1	0	[2, 2]	0	0.03	3	1	0	[2, 2]	
	6	0.09	8	0	[3,3]	[5, 4]	0	0.09	8	0	[3,3]	[5, 4]	
	7	0.62	15	0	[4,4]	[8, 8]	0	0.99	15	0	[4,4]	[8, 8]	
	8	0.05	5	0	[2,2]	[3, 3]	0	0.05	5	0	[2,2]	[3, 3]	
	9	0.08	7	0	[3,3]	[4, 4]	0	0.07	7	0	[3,3]	[4, 4]	
		time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	pNE	mNE	time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	pNE	mNE
	avg.	0.21	8.10	4.90	4.20	0.2	0.8	0.27	8.10	4.90	4.20	0.2	0.8
n	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$	numb. back	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$	
40	0	1.09	16	0	[5,5]	[8, 9]	0	1.58	16	0	[5,5]	[8, 9]	
	1	0.31	11	0	[3,3]	[6, 6]	0	0.33	11	0	[3,3]	[6, 6]	
	2	0.37	12	0	[4,4]	[7, 6]	0	0.42	12	0	[4,4]	[7, 6]	
	3	0.67	15	0	[4,4]	[8, 8]	0	0.92	15	0	[4,4]	[8, 8]	
	4	0.08	5	0	[2,2]	[3, 3]	0	0.08	5	0	[2,2]	[3, 3]	
	5	0.16	8	0	[3,3]	[5, 4]	0	0.16	8	0	[3,3]	[5, 4]	
	6	0.17	8	0	[3,3]	[5, 4]	0	0.16	8	0	[3,3]	[5, 4]	
	7	0.54	15	0	[5,5]	[7, 9]	0	0.58	15	0	[5,5]	[7, 9]	
	8	0.08	5	0	[2,2]	[3, 3]	0	0.08	5	0	[2,2]	[3, 3]	
	9	0.23	9	0	[2,2]	[6, 4]	0	0.22	9	0	[2,2]	[6, 4]	
		time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	pNE	mNE	time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	pNE	mNE
	avg.	0.37	10.40	5.80	5.60	0.0	1.0	0.45	10.40	5.80	5.60	0.0	1.0
n	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$	numb. back	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$	
80	0	3.43	16	0	[5,6]	[8, 9]	0	5.45	16	0	[5,6]	[8, 9]	
	1	0.59	12	0	[4,4]	[7, 6]	0	0.58	12	0	[4,4]	[7, 6]	
	2	0.71	13	0	[4,4]	[8, 6]	0	0.87	13	0	[4,4]	[8, 6]	
	3	73.72	19	0	[7,7]	[10, 10]	0	134.31	19	0	[7,7]	[10, 10]	
	4	152.74	24	0	[7,7]	[12, 13]	0	325.08	24	0	[7,7]	[12, 13]	
	5	94.00	21	0	[7,7]	[12, 10]	0	163.71	21	0	[7,7]	[12, 10]	
	6	116.15	23	0	[6,6]	[15, 9]	0	215.98	23	0	[6,6]	[15, 9]	
	7	11.89	20	0	[4,4]	[10, 11]	0	23.08	20	0	[4,4]	[10, 11]	
	8	65.78	22	0	[7,7]	[11, 12]	0	110.19	22	0	[7,7]	[11, 12]	
	9	4.07	17	0	[4,4]	[10, 8]	0	6.99	17	0	[4,4]	[10, 8]	
		time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	pNE	mNE	time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	pNE	mNE
	avg.	52.31	18.70	10.30	9.40	0.0	1.0	98.62	18.70	10.30	9.40	0.0	1.0
n	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$	numb. back	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$	
100	0	tl	26	0	[7, 7]	[14, 13]	0	tl	25	0	[6, 6]	[13, 13]	
	1	tl	28	0	[8, 8]	[17, 12]	0	tl	27	0	[7, 7]	[16, 12]	
	2	tl	27	0	[8, 8]	[14, 14]	0	tl	27	0	[8, 8]	[14, 14]	
	3	tl	25	0	[6, 6]	[13, 13]	0	tl	25	0	[6, 6]	[13, 13]	
	4	667.49	24	0	[9,9]	[13, 12]	0	605.92	23	0	[9,9]	[12, 12]	
	5	1547.82	25	0	[9,9]	[13, 13]	0	2464.84	25	0	[9,9]	[13, 13]	
	6	tl	30	0	[8, 8]	[17, 14]	0	tl	26	0	[7, 7]	[14, 13]	
	7	1.97	16	0	[5,5]	[9, 8]	0	2.57	16	0	[5,5]	[9, 8]	
	8	tl	27	0	[8, 8]	[14, 14]	0	tl	26	0	[7, 7]	[14, 13]	
	9	tl	27	0	[8, 8]	[15, 13]	0	tl	27	0	[8, 8]	[15, 13]	
		time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	pNE	mNE	time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	pNE	mNE
	avg.	739.09	21.67	11.67	11.00	0.0	0.3	1024.44	21.33	11.33	11.00	0.0	0.3

Table 2: Computational results for the **knapsack game** with $m = 2$.

		m-SGM							SGM						
n	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$	numb. back		time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$		
10	0	0.10	7	0	[2, 1, 2]	[4, 2, 3]	0		0.08	7	0	[2, 1, 2]	[4, 2, 3]		
	1	0.09	6	0	[2, 1, 2]	[3, 2, 3]	0		0.09	6	0	[2, 1, 2]	[3, 2, 3]		
	2	0.15	8	0	[3, 2, 2]	[4, 3, 3]	0		0.15	8	0	[3, 2, 2]	[4, 3, 3]		
	3	0.21	10	0	[2, 1, 2]	[5, 3, 4]	0		0.21	10	0	[2, 1, 2]	[5, 3, 4]		
	4	0.05	4	1	0	[2, 2, 2]	0		0.04	4	1	0	[2, 2, 2]		
	5	1.67	13	0	[3, 3, 3]	[5, 6, 4]	0		2.54	13	0	[3, 3, 3]	[5, 6, 4]		
	6	0.08	6	0	[2, 2, 1]	[3, 3, 2]	0		0.07	6	0	[2, 2, 1]	[3, 3, 2]		
	7	0.33	11	0	[2, 1, 2]	[5, 4, 4]	0		0.41	11	0	[2, 1, 2]	[5, 4, 4]		
	8	0.20	10	0	[2, 2, 1]	[4, 5, 3]	0		0.31	10	0	[2, 2, 1]	[4, 5, 3]		
	9	0.05	4	1	0	[2, 2, 2]	0		0.04	4	1	0	[2, 2, 2]		
		time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	$\lvert S^3\rvert$	pNE	mNE	time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	$\lvert S^3\rvert$	pNE	mNE
	avg.	0.29	7.90	3.70	3.20	3.00	0.2	0.8	0.39	7.90	3.70	3.20	3.00	0.2	0.8
n	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$	numb. back		time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$		
20	0	0.20	8	0	[2, 2, 1]	[4, 4, 2]	0		0.21	8	0	[2, 2, 1]	[4, 4, 2]		
	1	0.40	10	0	[2, 1, 2]	[4, 4, 4]	0		0.52	10	0	[2, 1, 2]	[4, 4, 4]		
	2	6.22	19	0	[2, 2, 3]	[7, 6, 8]	0		11.55	19	0	[2, 2, 3]	[7, 6, 8]		
	3	15.06	23	0	[4, 5, 3]	[8, 11, 6]	0		26.17	23	0	[4, 5, 3]	[8, 11, 6]		
	4	0.21	9	0	[2, 1, 2]	[5, 2, 4]	0		0.19	9	0	[2, 1, 2]	[5, 2, 4]		
	5	0.18	8	0	[2, 1, 2]	[4, 3, 3]	0		0.17	8	0	[2, 1, 2]	[4, 3, 3]		
	6	97.26	21	0	[4, 2, 5]	[9, 5, 9]	0		212.14	21	0	[4, 2, 5]	[9, 5, 9]		
	7	0.16	8	0	[2, 1, 2]	[4, 3, 3]	0		0.15	8	0	[2, 1, 2]	[4, 3, 3]		
	8	0.65	15	0	[3, 3, 1]	[6, 8, 3]	0		0.74	15	0	[3, 3, 1]	[6, 8, 3]		
	9	0.29	10	0	[2, 2, 2]	[4, 4, 4]	0		0.28	10	0	[2, 2, 2]	[4, 4, 4]		
		time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	$\lvert S^3\rvert$	pNE	mNE	time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	$\lvert S^3\rvert$	pNE	mNE
	avg.	12.06	13.10	5.50	5.00	4.60	0.0	1.0	25.21	13.10	5.50	5.00	4.60	0.0	1.0
n	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$	numb. back		time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$		
40	0	26.08	25	0	[2, 3, 4]	[8, 7, 12]	0		52.65	25	0	[2, 3, 4]	[8, 7, 12]		
	1	0.78	12	0	[2, 2, 3]	[5, 4, 5]	0		0.91	12	0	[2, 2, 3]	[5, 4, 5]		
	2	tl	29	0	[4, 5, 4]	[11, 11, 9]	0		tl	27	0	[4, 4, 4]	[10, 10, 9]		
	3	tl	29	0	[5, 5, 5]	[10, 11, 9]	1		tl	27	0	[5, 6, 5]	[10, 10, 9]		
	4	382.06	22	0	[4, 3, 6]	[9, 6, 9]	0		792.33	22	0	[4, 3, 6]	[9, 6, 9]		
	5	806.95	28	0	[5, 3, 5]	[10, 8, 12]	0		1585.39	28	0	[5, 3, 5]	[10, 8, 12]		
	6	tl	25	0	[6,3,4]	[9, 9, 9]	0		tl	23	0	[4,2,3]	[9, 8, 8]		
	7	1133.06	23	0	[5, 5, 6]	[9, 8, 8]	0		1897.04	23	0	[5, 5, 6]	[9, 8, 8]		
	8	1151.67	24	0	[7, 3, 7]	[10, 6, 10]	0		1743.75	24	0	[7, 3, 7]	[10, 6, 10]		
	9	14.14	22	0	[2, 4, 4]	[6, 8, 10]	0		20.36	22	0	[2, 4, 4]	[6, 8, 10]		
		time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	$\lvert S^3\rvert$	pNE	mNE	time	iter	$\lvert S^1\rvert$	$\lvert S^2\rvert$	$\lvert S^3\rvert$	pNE	mNE
	avg.	502.11	22.29	8.14	6.71	9.43	0.0	0.7	870.35	22.29	8.14	6.71	9.43	0.0	0.7

Table 3: Computational results for the **knapsack game** with $m = 3$.

pure equilibrium. In fact, our algorithms will return a pure equilibrium: both m-SGM and SGM start with a sampled game with only one strategy for each player and thus, one pure equilibrium; this equilibrium is given to the input of our PNS implementation, which implies that players' supports of size one will be prioritized leading to the computation of a pure equilibrium; this pure equilibrium will be in the input of the next PNS call, resulting in a pure equilibrium output; this reasoning propagates through the algorithms' execution. Even though our algorithms find a pure equilibrium, it is expected that the potential function maximization method will provide an equilibrium faster than our methods, since our algorithms deeply depend on the initialization (which in our implementation does not take into account the players' interaction).

Table 4 reports the results for the m-SGM, SGM and potential function maximization. The table displays the number of periods (T), the number of players (m) and the average CPU time in seconds ("time"). For our methods, a column reports the averages for the number of sampled games ("avg. iter"), the number of strategies in the last sampled game ("avg. $|\mathbb{S}^p|$") and the number of backtrackings ("avg. numb. back"). The columns "numb. pNE" display the number of instances solved by each method. In this case all instances were solved within the time frame of one hour.

In this case, m-SGM does not present advantages with respect to SGM. This is mainly due to the fact that the sampled games always have pure equilibria and our improvements have more impact when many mixed equilibria exist. The maximization of the potential functions allowed the computation of equilibria to be faster. This highlights the importance of identifying if a game is potential. On the other hand, the potential function maximization allows the determination of one equilibrium, while our method with different *Initialization* and/or *PlayerOrder* implementations may return different equilibria and, thus, allows larger exploration of the set of equilibria.

Algorithm *PlayerOrder* has a crucial impact in the number of sampled games to be explored in order to compute one equilibrium. In fact, when comparing our implementation with simply keeping the players' index order static, the impact on computational times is significant. Solving the players' best reactions by dynamic programming could improve further the computing times of our algorithms.

5.3.3 Final Remarks

In the application of our two methods in all the studied instances of these games, backtracking never occurred. Indeed, we noticed that this is a very unlikely event (even though it may happen, as shown in Example 5). This is the reason why both m-SGM and SGM, in general, coincide in the number of sampled games generated: it is in the support enumeration for each sampled game that the methods differ; the fact that the last added strategy is mandatory to be in the equilibrium support of the m-SGM makes it faster. The backtracking will reveal useful for problems in which it is "difficult" to find the strategies of a sampled game that enable to define an equilibrium of an IPG. At this point, for the games studied, in comparison with the number of pure profiles of strategies that may exist in a game, not too many sampled games had to be generated in order to find an equilibrium, meaning that the challenge is to make the computation of equilibria for sampled games faster.

		m-SGM							SGM						Potential Function Maximization	
			avg.					numb.		avg.				numb.	avg	numb.
m	T	time	iter	$\|S^1\|$	$\|S^2\|$	$\|S^3\|$	numb. back	pNE	time	iter	$\|S^1\|$	$\|S^2\|$	$\|S^3\|$	pNE	time	pNE
2	10	0.58	14.90	8.00	7.90		0	10	0.49	14.90	8.00	7.90		10	0.01	10
	20	1.14	15.60	8.60	8.00		0	10	1.00	15.60	8.60	8.00		10	0.01	10
	50	3.33	16.00	9.00	8.00		0	10	3.02	16.00	9.00	8.00		10	0.03	10
3	10	2.57	30.60	11.40	10.80	10.40	0	10	2.20	30.60	11.40	10.80	10.40	10	0.01	10
	20	4.51	32.00	12.00	11.10	10.90	0	10	3.88	32.00	12.00	11.10	10.90	10	0.03	10
	50	10.69	33.10	12.10	11.50	11.50	0	10	9.36	33.10	12.10	11.50	11.50	10	0.08	10

Table 4: Computational results for the **competitive uncapacitated lot-sizing game**.

Comparison: m-SGM and PNS In the case of the knapsack game, the number of strategies for each player is finite. In order to find an equilibrium of it, we can explicitly determine all feasible strategies for each player and, then apply directly PNS. In Tables 5 and 6, we compare this procedure with m-SGM, for $n = 5$, $n = 7$

and $n = 10$ (in these cases, each player has at most $2^5 = 32$, $2^7 = 128$ and $2^{10} = 1024$ feasible strategies, respectively). We note that the computational time displayed in these tables under the direct application of PNS does not include the time to determine all feasible strategies for each player (although, for $n = 5$, $n = 7$ and $n = 10$ is negligible). Based on these results it can be concluded that even for small instances, m-SGM already performs better than the direct application of PNS, where all strategies must be enumerated.

			m-SGM							direct PNS			
n	m	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert S^p \rvert$	numb. back		time	pNE	mNE	$\prod_{p=1}^{m} \lvert S^p \rvert$
5	2	0	0.00	1	1	0	[1, 1]	0		0.02	1	0	[31, 11]
		1	0.01	2	1	0	[1, 2]	0		0.01	1	0	[10, 7]
		2	0.01	2	1	0	[1, 2]	0		0.03	1	0	[29, 21]
		3	0.01	3	0	1	[2, 2]	0		0.11	0	1	[16, 16]
		4	0.02	4	1	0	[3, 2]	0		0.01	1	0	[17, 12]
		5	0.01	2	1	0	[2, 1]	0		0.01	1	0	[16, 17]
		6	0.01	2	1	0	[2, 1]	0		0.02	1	0	[17, 16]
		7	0.01	2	1	0	[2, 1]	0		0.01	1	0	[17, 15]
		8	0.02	4	1	0	[3, 2]	0		0.01	1	0	[15, 16]
		9	0.00	1	1	0	[1, 1]	0		0.01	1	0	[16, 9]
			avg.					numb.		avg.			numb.
			time	iter	$\lvert S^1 \rvert$	$\lvert S^2 \rvert$		pNE	mNE	time	$\lvert S^1 \rvert$	$\lvert S^2 \rvert$	pNE mNE
			0.01	2.30	1.80	1.50		9	1	0.03	18.40	14.00	9 1
	m	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert S^p \rvert$	numb. back		time	pNE	mNE	$\prod_{p=1}^{m} \lvert S^p \rvert$
	3	0	0.03	4	0	1	[1, 3, 2]	0		0.07	0	1	[1, 6, 17]
		1	0.01	1	1	0	[1, 1, 1]	0		0.07	1	0	[10, 22, 29]
		2	0.02	3	1	0	[2, 2, 1]	0		0.05	1	0	[13, 29, 9]
		3	0.02	4	1	0	[2, 2, 2]	0		0.04	1	0	[22, 21, 8]
		4	0.02	3	1	0	[2, 1, 2]	0		0.07	1	0	[16, 17, 16]
		5	0.06	6	0	1	[2, 2, 4]	0		0.83	0	1	[15, 16, 16]
		6	0.02	3	1	0	[2, 2, 1]	0		0.07	1	0	[16, 16, 18]
		7	0.01	2	1	0	[2, 1, 1]	0		0.03	1	0	[11, 12, 15]
		8	0.01	2	1	0	[2, 1, 1]	0		0.08	1	0	[12, 24, 12]
		9	0.02	3	1	0	[2, 1, 2]	0		0.11	1	0	[13, 19, 18]
			avg.					numb.		avg.			numb.
			time	iter	$\lvert S^1 \rvert$	$\lvert S^2 \rvert$	$\lvert S^3 \rvert$	pNE	mNE	time	$\lvert S^1 \rvert$	$\lvert S^2 \rvert$	$\lvert S^3 \rvert$ pNE mNE
			0.02	3.10	1.80	1.60	1.70	8	2	0.14	12.90	18.20	15.80 8 2
n	m	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert S^p \rvert$	numb. back		time	pNE	mNE	$\prod_{p=1}^{m} \lvert S^p \rvert$
7	2	0	0.03	2	1	0	[1, 2]	0		0.61	1	0	[69, 120]
		1	0.03	4	0	1	[2, 3]	0		212.07	0	1	[103, 72]
		2	0.02	4	0	1	[3, 2]	0		24.31	0	1	[53, 64]
		3	0.01	2	1	0	[2, 1]	0		0.27	1	0	[82, 99]
		4	0.01	3	1	0	[2, 2]	0		0.07	1	0	[43, 45]
		5	0.02	4	0	1	[2, 3]	0		0.27	1	0	[62, 57]
		6	0.01	3	1	0	[2, 2]	0		0.18	1	0	[69, 62]
		7	0.03	5	0	1	[3, 3]	0		106.34	0	1	[88, 68]
		8	0.01	2	1	0	[2, 1]	0		0.19	1	0	[82, 49]
		9	0.01	3	1	0	[2, 2]	0		0.32	1	0	[34, 60]
			avg.					numb.		avg.			numb.
			time	iter	$\lvert S^1 \rvert$	$\lvert S^2 \rvert$		pNE	mNE	time	$\lvert S^1 \rvert$	$\lvert S^2 \rvert$	pNE mNE
			0.02	3.20	2.10	2.10		6	4	34.46	68.50	69.60	7 3
	m	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert S^p \rvert$	numb. back		time	pNE	mNE	$\prod_{p=1}^{m} \lvert S^p \rvert$
	3	0	0.03	4	0	1	[1, 3, 2]	0		0.45	1	0	[1, 85, 25]
		1	0.12	7	0	1	[3, 4, 2]	0		tl	0	0	[91, 65, 18]
		2	0.01	2	1	0	[2, 1, 1]	0		4.79	1	0	[80, 35, 65]
		3	0.03	4	1	0	[2, 2, 2]	0		3.03	1	0	[24, 39, 61]
		4	0.03	4	1	0	[3, 2, 1]	0		2.70	1	0	[48, 69, 32]
		5	0.03	4	1	0	[2, 2, 2]	0		1.44	1	0	[64, 66, 66]
		6	0.02	3	1	0	[2, 2, 1]	0		5.48	1	0	[64, 64, 67]
		7	0.02	3	1	0	[2, 1, 2]	0		6.29	1	0	[59, 59, 95]
		8	0.02	3	1	0	[1, 2, 2]	0		1.27	1	0	[46, 42, 62]
		9	0.14	9	0	1	[4, 4, 3]	0		tl	0	0	[69, 94, 69]
			avg.					numb.		avg.			numb.
			time	iter	$\lvert S^1 \rvert$	$\lvert S^2 \rvert$	$\lvert S^3 \rvert$	pNE	mNE	time	$\lvert S^1 \rvert$	$\lvert S^2 \rvert$	$\lvert S^3 \rvert$ pNE mNE
			0.05	4.30	2.20	2.30	1.80	7	3	3.18	48.25	57.37	59.12 8 0

Table 5: Computational results for the m-SGM and PNS to the knapsack game with $n = 5, 7$.

			m-SGM							direct PNS			
n	m	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert S^p \rvert$	numb. back		time	pNE	mNE	$\prod_{p=1}^{m} \lvert S^p \rvert$
10	2	0	0.04	4	0	1	[2, 3]	0		tl.	0	0	[792, 436]
		1	0.01	2	1	0	[2, 1]	0		6.87	1	0	[253, 385]
		2	0.05	7	0	1	[4, 4]	0		tl.	0	0	[924, 883]
		3	0.05	6	0	1	[3, 4]	0		51.00	1	0	[382, 396]
		4	0.01	2	1	0	[2, 1]	0		11.10	1	0	[426, 489]
		5	0.02	3	1	0	[2, 2]	0		10.59	1	0	[468, 474]
		6	0.01	2	1	0	[1, 2]	0		9.93	1	0	[511, 481]
		7	0.02	4	1	0	[3, 2]	0		12.75	1	0	[470, 510]
		8	0.03	5	0	1	[3, 3]	0		tl.	0	0	[803, 482]
		9	0.03	4	0	1	[2, 3]	0		tl.	0	0	[293, 811]
			avg.					numb.		avg.			numb.
			time	iter	$\lvert S^1 \rvert$	$\lvert S^2 \rvert$		pNE	mNE	time	$\lvert S^1 \rvert$	$\lvert S^2 \rvert$	pNE mNE
			0.03	3.90	2.40	2.50		5	5	17.04	418.33	455.83	6 0
	m	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \lvert S^p \rvert$	numb. back		time	pNE	mNE	$\prod_{p=1}^{m} \lvert S^p \rvert$
	3	0	2.65	19	0	1	[5, 7, 9]	0		1228.25	1	0	[26, 806, 282]
		1	0.21	11	0	1	[5, 5, 3]	0		tl.	0	0	[318, 762, 879]
		2	0.70	12	0	1	[4, 6, 4]	0		tl.	0	0	[458, 263, 455]
		3	0.04	5	1	0	[2, 3, 2]	0		1136.29	1	0	[334, 529, 690]
		4	0.08	7	0	1	[3, 4, 2]	0		tl.	0	0	[351, 555, 659]
		5	0.05	6	1	0	[3, 3, 2]	0		tl.	0	0	[610, 480, 518]
		6	0.06	7	1	0	[3, 3, 3]	0		2453.11	1	0	[462, 520, 513]
		7	0.05	6	1	0	[3, 3, 2]	0		437.29	1	0	[519, 375, 342]
		8	0.09	8	0	1	[3, 5, 2]	0		tl.	0	0	[347, 698, 571]
		9	0.04	5	1	0	[3, 2, 2]	0		tl.	0	0	[716, 773, 817]
			avg.					numb.		avg.			numb.
			time	iter	$\lvert S^1 \rvert$	$\lvert S^2 \rvert$	$\lvert S^3 \rvert$	pNE	mNE	time	$\lvert S^1 \rvert$	$\lvert S^2 \rvert$	$\lvert S^3 \rvert$ pNE mNE
			0.40	8.60	3.40	4.10	3.10	5	5	1313.73	335.25	557.50	456.75 4 0

Table 6: Computational results for the m-SGM and PNS to the knapsack game with $n = 10$.

6 Conclusions and further directions

It is known that the problem of equilibria existence for IPGs is Σ_2^p-complete and, even if an equilibrium exists, its computations is at least PPAD-complete. This is not surprising, since verifying if a profile of strategies is an equilibrium for an IPG implies solving each player's best response optimization, which can be an NP-complete problem. Thus, the goal of this paper was to contribute with an algorithmic approach for the computation of equilibria with a reasonable running time in practice.

To the best of our knowledge, the novel framework proposed and evaluated in this paper is the first addressing the computation of equilibria for non-cooperative simultaneous games in which the players' optimization problem is modelled through a mixed integer program. These games are of practical interest given that in real-world applications, each player decision problem is combinatorial.

Under our game model, each player's goal is described through a mathematical programming formulation. Therefore, we combined algorithms (and tools) from mathematical programming and game theory to devise a novel method to determine Nash equilibria. Our basic method, SGM, iteratively determines equilibria to finite games which are samples (in the space of the strategies) of the original game; in each iteration, by solving the player's best reactions to an equilibrium of the previous sampled game, it is verified if the determined equilibrium coincides with an ε-equilibrium of the original game. Once none of the players has incentive to deviate from the equilibrium of the current sampled game, the method stops and returns it. In order to make the algorithm faster in practice, special features were added. For this purpose, we devised the modified SGM. Our algorithms were experimentally validated through two particular games: the knapsack and the competitive lot-sizing games. For the knapsack game, the m-SGM provides equilibria to medium size instances within the time frame of one hour. The results show that this is a hard game which is likely to have strictly mixed equilibria. The hardness comes from the conflicts that projects selected by different players have in their payoffs: for some projects i a player p can benefit from player k simultaneous investment, while player k is penalized. For the competitive lot-sizing game, its property of being potential makes our algorithms' iterations fast (since there is always a pure equilibrium, that is, an equilibrium with a small support size) and, thus, the challenge is in improving the methods' initialization.

Note that for the instances solved by our algorithms, there is an exponential (knapsack game) or infinite (competitive lot-sizing game) number of pure profiles of strategies. However, by observing the computational results, a small number of explicitly enumerated pure strategies was enough to find an equilibrium. For this reason, the explicitly enumerated strategies (the sampled games) are usually "far" from describing (even partially) a player p's polytope $\text{conv}(X^p)$ and thus, at this point, this information is not used in PNS to speed up its computations. For instance, Corollary 2 and Lemma 9 did not reduce the number of supports enumerated by PNS in each iteration of m-SGM. Due to the fact that it is in PNS that our algorithms struggle the most, its improvement is the first aspect to further study; we believe that exploring the possibility of extracting information from each player's polytope of feasible strategies will be the crucial ingredient for this.

There is a set of natural questions that this work opens. Can we adapt m-SGM to compute all equilibria (or characterize the set of equilibria)? Can we compute an equilibrium satisfying a specific property (e.g., computing the equilibrium that maximizes the social welfare, computing a non-dominated equilibrium)? Will in practice players play equilibria that are "hard" to find? If a game has multiple equilibria, how to decide among them? From a mathematical point of view, the first two questions embody a big challenge, since there seems to be hard to extract problem structure to the general IPG class of games. The two last questions raise another one, which is the possibility of considering different solution concepts to IPGs.

Acknowledgments

Part of this work was performed while the first author was in the Faculty of Sciences University of Porto and INESC TEC. The first author thanks the support of Institute for data valorisation (IVADO), the Portuguese Foundation for Science and Technology (FCT) through a PhD grant number SFRH/BD/79201/2011,POPH/FSE.

References

[1] Dimitri P. Bertsekas, Asuman E. Ozdaglar, and Angelia Nedić. *Convex analysis and optimization*. Athena scientific optimization and computation series. Athena Scientific, Belmont (Mass.), 2003.

[2] R. Bixby. Progress in mathematical optimization solvers. CO @ Work 2015 summer school, 2015.

[3] Margarida Carvalho. *Computation of equilibria on integer programming games*. PhD thesis, Faculdade de Ciências da Universidade do Porto, 2016.

[4] Margarida Carvalho, Andrea Lodi, and João Pedro Pedroso. Existence of nash equilibria on integer programming games. In A. Ismael F. Vaz, João Paulo Almeida, José Fernando Oliveira, and Alberto Adrego Pinto, editors, *Operational Research*, pages 11–23, Cham, 2018. Springer International Publishing.

[5] Margarida Carvalho, Andrea Lodi, João Pedro Pedroso, and Ana Viana. Nash equilibria in the two-player kidney exchange game. *Mathematical Programming*, pages 1–29, 2016.

[6] Margarida Carvalho, João Pedro Pedroso, Claudio Telha, and Mathieu Van Vyve. Competitive uncapacitated lot-sizing game. *International Journal of Production Economics*, 204:148 – 159, 2018.

[7] Xi Chen and Xiaotie Deng. Settling the complexity of two-player Nash equilibrium. In *Foundations of Computer Science, 2006. FOCS '06. 47th Annual IEEE Symposium on*, pages 261–272, Oct 2006.

[8] D. A. Cox, J. Little, and D.l O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3e (Undergraduate Texts in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[9] Python Software Foundation. Python v2.7.3 documentation. http://docs.python.org/library/random.html, 2012.

[10] Steven A. Gabriel, Saule Ahmad Siddiqui, Antonio J. Conejo, and Carlos Ruiz. Solving discretely-constrained Nash-Cournot games with an application to power markets. *Networks and Spatial Economics*, 13(3):307–326, 2013.

[11] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

[12] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bull. Amer. Math. Soc.*, 64(5):275–278, 09 1958.

[13] R. Hemmecke, S. Onn, and R. Weismantel. Nash-equilibria and N-fold integer programming. *ArXiv preprint 0903.4577*, 2009.

[14] W. Karush. Minima of Functions of Several Variables with Inequalities as Side Constraints. Master's thesis, Dept. of Mathematics, Univ. of Chicago, 1939.

[15] Matthias Köppe, Christopher Thomas Ryan, and Maurice Queyranne. Rational generating functions and integer programming games. *Oper. Res.*, 59(6):1445–1460, November 2011.

[16] Michael M. Kostreva. Combinatorial optimization in Nash games. *Computers & Mathematics with Applications*, 25(10 - 11):27 – 34, 1993.

[17] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, Berkeley, Calif., 1951. University of California Press.

[18] Kwang-Ho Lee and R. Baldick. Solving three-player games by the matrix approach with application to an electric power market. *Power Systems, IEEE Transactions on*, 18(4):1573–1580, Nov 2003.

[19] C. E. Lemke and Jr. Howson, J. T. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):pp. 413–423, 1964.

[20] Hongyan Li and Joern Meissner. Competition under capacitated dynamic lot-sizing with capacity acquisition. *International Journal of Production Economics*, 131(2):535 – 544, 2011.

[21] John Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, September 1951.

[22] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[23] M.V. Pereira, S. Granville, M.H.C. Fampa, R. Dix, and L.A. Barroso. Strategic bidding under uncertainty: a binary expansion approach. *Power Systems, IEEE Transactions on*, 20(1):180–188, Feb 2005.

[24] Yves Pochet and Laurence A. Wolsey. *Production Planning by Mixed Integer Programming (Springer Series in Operations Research and Financial Engineering)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[25] Ryan Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a Nash equilibrium. *Games and Economic Behavior*, 63(2):642 – 662, 2008. Second World Congress of the Game Theory Society.

[26] Noah D. Stein, Asuman Ozdaglar, and Pablo A. Parrilo. Separable and low-rank continuous games. *International Journal of Game Theory*, 37(4):475–504, 2008.

[27] W.I. Zangwill and C.B. Garcia. *Pathways to solutions, fixed points, and equilibria*. Prentice-Hall series in computational mathematics. Prentice-Hall, 1981.

Appendix A Illustration of Backtracking Step

Example 5 *Consider the two-player knapsack game described by the following optimization problems*

$$\begin{aligned} Player A: \max_{x^A\in\{0,1\}^n} \quad & 15x_1^A + 8x_2^A - 3x_3^A + 43x_4^A - 15x_5^A + 39x_1^Ax_1^B - 90x_2^Ax_2^B \\ & + 11x_3^Ax_3^B - 84x_4^Ax_4^B - 43x_5^Ax_5^B \\ \text{subject to} \quad & 70x_1^A - 79x_2^A - 8x_3^A - 62x_4^A - 96x_5^A \leq -140 \end{aligned}$$

$$\begin{aligned} Player B: \max_{x^B\in\{0,1\}^n} \quad & 24x_1^B + 13x_2^B + 44x_3^B - x_4^A - 45x_5^B - 73x_1^Ax_1^B - 58x_2^Ax_2^B \\ & - 78x_3^Ax_3^B - 49x_4^Ax_4^B + 72x_5^Ax_5^B \\ \text{subject to} \quad & 69x_1^B + 25x_2^B - 39x_3^B - 74x_4^B + 70x_5^B \leq 40.8 \end{aligned}$$

In what follows, we go through each sampled game generated by m-SGM; Figure 5 display the players' strategies computed and associated players' payoffs.

Sampled game 0 *The NE is* $\sigma_0 = (1;1)$. *However, in the original game, player A has incentive to deviate to* $x(1) = (0,0,1,1,1)$.

Sampled game 1 *The NE is* $\sigma_1 = (0,1;1)$. *However, in the original game, player B has incentive to deviate to* $x(2) = (0,1,0,0,0)$.

Sampled game 2 *The NE is* $\sigma_2 = (0,1;0,1)$. *However, player A has incentive to deviate to* $x(3) = (0,0,0,1,1)$.

Sampled game 3 *The NE is mixed with* $supp(\sigma_3^A) = \{(0,0,1,1,1),(0,0,0,1,1)\}$ *and* $supp(\sigma_3^B) = \{(1,1,1,1,0),(0,1,0,0,0)\}$, $\sigma_3 = (0, \frac{3}{13}, \frac{10}{13}; \frac{3}{11}, \frac{8}{11})$. *However, in the original game, player B has incentive to deviate to* $x(4) = (0,0,1,0,1)$.

Sampled game 4 *The NE is* $\sigma_4 = (1,0,0;0,0,1)$. *However, in the original game, player A has incentive to deviate to* $x(5) = (0,1,1,1,0)$.

Sampled game 5 *There is no NE with* $x(5) = (0,1,1,1,0)$ *in the support of player A. Thus, initialize backtracking.*

Revisiting sampled game 4 *Keep the best reaction strategy* $x^A = (0,1,1,1,0)$ *that originated the sampled game 5, but do not consider it in the support enumeration (this strategy only appears in the Feasibility*

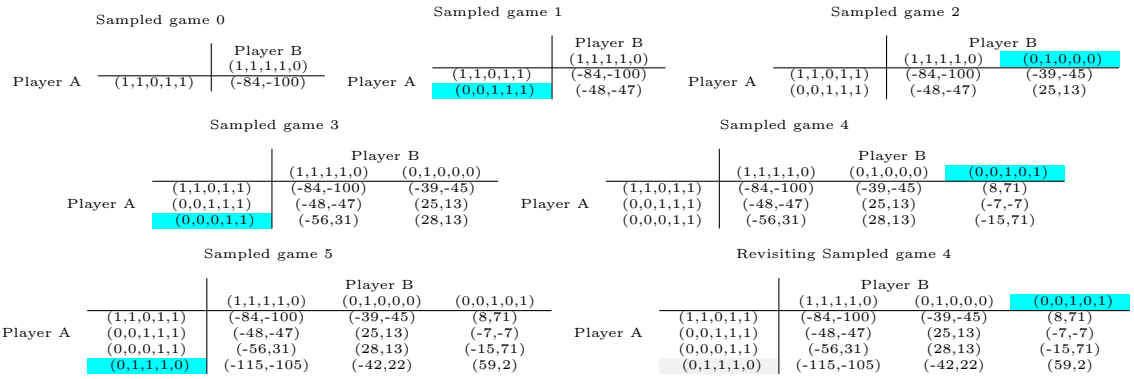

Figure 5: Modified SGM applied to Example 5. The strategies in cyan must be in the equilibrium support, while the strategies in gray are not considered in the support enumeration.

Problem in order to avoid the repetition of equilibria). A NE with $x^B = (0,0,1,0,1)$ *in the support is computed:* $\sigma_4 = (0, \frac{29}{39}, \frac{10}{39}, 0; 0, \frac{8}{11}, \frac{3}{11})$ *with supports* $supp(\sigma_4^A) = \{(0,0,1,1,1),(0,0,0,1,1)\}$ *and* $supp(\sigma_4^B) = \{(0,1,0,0,0),(0,0,1,0,1)\}$. *This NE is a NE of the original game.*