Computation of equilibria on integer programming games

Maria Margarida da Silva Carvalho

Tese de Doutoramento apresentada à
Faculdade de Ciências da Universidade do Porto
Ciência de Computadores
2016

Computation of equilibria on integer programming games

Maria Margarida da Silva Carvalho

Doutoramento em Ciência de Computadores
Departamento de Ciência de Computadores
2016

Orientador
João Pedro Pedroso, Professor Auxiliar, Faculdade de Ciências da Universidade do Porto

Coorientador
Andrea Lodi, Professor Catedrático, Università di Bologna e École Polytechnique de Montréal

Esta tese foi financiada por uma bolsa de doutoramento da Fundação para a Ciência e a Tecnologia (referência SFRH/BD/79201/2011) no âmbito do programa POPH, financiado pelo Fundo Social Europeu e pelo Governo Português.

Acknowledgments

I would like to acknowledge the support of the Portuguese Foundation for Science and Technology (FCT) through a PhD grant number SFRH/BD/79201/2011 (POPH/FSE program).

The last four years were an exciting adventure to me. I learned, I got puzzled, I got frustrated, I conquered results, I had tons of fun. This journey was accompanied and made possible due to amazing people.

I have no words to João Pedro. I am grateful that he proposed me a challenging research plan. João Pedro was a tireless supervisor, always available and extremely supportive. I am deeply thankful to my co-supervisor Andrea for his guidance and for making me feel confident about myself. I enjoyed amazing scientific experiences thanks to Andrea and his department DEI, Università di Bologna. I highlight, how lucky I was for having the pleasure to collaborate with Alberto Caprara and Gerhard Woeginger while I lived in Bologna. My year in Bologna was one of the happiest years I ever lived! The role of Ana Viana during my Ph.D. was not only of a collaborator. I am grateful to Ana for sharing knowledge, including me in her research projects, being tremendously friendly and advising me. I kindly thanks Mathieu Van Vyve and Claudio Telha for receiving me warmly in Louvain-la-Neuve and enjoying with me puzzling afternoons.

My scientific accomplishments could not have been possible without the strength transmited by family and friends. I express a sincere feeling of gratitude to the friends that I made at DCC, DEI, CORE and INESC TEC. A special thanks goes to Amaya and Ana for being my family in Italy and spoiling the Ph.D. life events to me.

Agradeço profundamente aos meus pais, à minha irmã e ao Ricardo por todo o carinho e paciência infinita comigo durante estes anos. Mil obrigadas a todas e todos que equilibraram o meu dia-a-dia com o seu amor, amizade, encorajamento, tempo para ouvir problemas matemáticos e não matemáticos, em particular, Isa, Yellow Hat Sisters, Ângela, Mariana, Mari Sol e Inês.

Many more people have been important during the Ph.D., I apologize for not mentioning everybody.

Resumo

O problema da mochila, o problema de emparelhamento máximo e o problema de dimensionamento de lotes são exemplos clássicos de modelos de otimização combinatória que têm sido amplamente estudados na literatura. Nos últimos anos têm sido investigadas versões mais intrincadas, o que resulta numa melhor aproximação dos problemas do mundo real e num aperfeiçoamento das técnicas de solução. O objetivo desta tese de doutoramento é estender as ferramentas algorítmicas que resolvem problemas combinatórios com apenas um decisor para jogos, isto é, para problemas combinatórios com vários decisores. Frequentemente um processo de decisão depende de parâmetros que são controlados por decisores externos. Por conseguinte, os jogos combinatórios são uma linha de investigação fundamental, uma vez que refletem a realidade destes problemas.

Focamo-nos na classificação da complexidade computacional e no desenho de algoritmos para determinar equilíbrios de jogos em programação inteira com utilidades quadráticas. Num jogo em programação inteira, o objetivo de um jogador é formulado usando terminologia de programação matemática. Cada jogador tem o intuito de maximizar a sua utilidade, uma função que depende das suas variáveis de decisão (estratégias) e das dos restantes. Iremos concentrar-nos em jogos onde as funções de utilidade de cada jogador são quadráticas nas suas variáveis de decisão.

De forma a que esta tese seja auto-contida, começamos por fornecer as bases essenciais da teoria de complexidade computacional, da programação matemática e da teoria dos jogos. Seguir-se-á a apresentação das nossas contribuições, as quais estão divididas em duas partes: competição de Stackelberg e jogos em simultâneo.

A primeira parte é sobre competições de Stackelberg (também conhecidas por programação com dois níveis), onde os jogadores jogam de forma sequencial. Estudamos um dos modelos mais simples de competição de Stackelberg combinatória, o qual é baseado no problema da mochila. Caracterizamos a complexidade de calcular um equilíbrio e desenhamos um algoritmo novo para atacar um problema de interdição com dois níveis, o problema da mochila com restrições de interdição. Recentemente, a classe de problemas de interdição tem recebido uma grande atenção por parte da comunidade de investigação.

A segunda parte é sobre jogos em simultâneo, isto é, jogos em que os jogadores selecionam as suas estratégias ao mesmo tempo. Esta definição dá já uma ideia dos obstáculos que iremos encontrar na determinação de estratégias racionais para os jogadores, uma vez que as estratégias dos seus rivais terão de ser previstas antecipadamente. Neste contexto, investigamos a estrutura de 3 jogos em particular: o jogo de coordenação da mochila (baseado no problema da mochila), o jogo das trocas de rins (baseado no problema de emparelhamento máximo) e o jogo de dimensionamento de lotes (baseado no problema de dimensionamento de lotes).

Em jeito de conclusão, depois do estudo destes três jogos olhamos para a situação mais complexo, focando a nossa atenção no caso geral de jogos em simultâneo. Estabelecemos a relação entre os jogos em simultâneo e competições de Stackelberg, provando que encontrar uma solução para um jogo em simultâneo é pelo menos tão difícil como resolver uma competição de Stackelberg. Por fim, construímos um algoritmo para aproximar um equilíbrio para jogos em simultâneo.

Palavras-chave: Equilíbrios de Nash; jogos em programação inteira; competições de Stackelberg; jogos em simultâneo.

Abstract

The knapsack problem, the maximum matching problem and the lot-sizing problem are classical examples of combinatorial optimization models that have been broadly studied in the literature. In recent years, more intricate variants of these problems have been investigated, resulting in better approximations of real-world problems and in improvements in solution techniques. The goal of this Ph.D. thesis is to extend the algorithmic tools for solving these (single) decision-maker combinatorial problems to games, that is, to combinatorial problems with several decision makers. It is frequent for a decision process to depend on parameters that are controlled by external decision makers. Therefore, combinatorial games are a crucial line of research since they reflect the reality of these problems.

We focus in understanding the computational complexity and in designing algorithms to find equilibria to integer programming games with quadratic utilities. In an integer programming game, a player's goal is formulated by using the mathematical programming framework. Each player aims at maximizing her utility, a function of her and other players' decision variables (strategies). We will concentrate in games with quadratic utilities on each player's decision variables.

In order to make this thesis self-contained, we start by covering the essential background in computational complexity, mathematical programming and game theory. It is followed by the presentation of our contributions, which are fleshed out in two parts: Stackelberg competition and simultaneous games.

The first part concerns Stackelberg competitions (also known as bilevel programming), where players play sequentially. We study the most simple to model combinatorial Stackelberg competitions, which are based on the knapsack problem. We characterize the complexity of computing equilibria and we design a novel algorithm to tackle a bilevel interdiction problem, the knapsack problem with interdiction constraints, a special class of problems which have recently received significant attention in the research community.

The second part deals with simultaneous games, *i.e.*, games in which players select their strategies at the same time. This definition already gives a hint of the obstacles involved in finding players' rational strategies, since the opponents strategies have to be predicted. In this context, we investigate the structure of three particular games: the coordination knapsack game (based on the knapsack problem), the kidney-exchange game (based on the maximum matching problem) and the lot-sizing game (based on the lot-sizing problem).

To conclude, after investigating these particular games, we move on to the more complex case: general simultaneous games. We establish the connection of simultaneous games with Stackelberg competitions, and prove that finding a solution to a simultaneous game is at least as hard as solving a Stackelberg competition; finally, we devise an algorithm to approximate an equilibrium for simultaneous games.

Keywords: Nash equilibria; integer programming games; bilevel programming; Stackelberg competition; simultaneous games.

Contents

Chapter 1

Introduction

1.1 Context: Mathematical Programming and Game Theory

This section succinctly provides an overview of mathematical programming and game theory in order to introduce the problems that will be addressed. Chapter 2 will present in detail the background and previous work in these fields.

Mathematical programming is the field that studies optimization. The focus is any mathematical problem that implies maximizing or minimizing a function of many *decision variables*, called *objective function*, possibly subject to a set of constraints defining the so-called *feasible region*. It is suitable for modeling decision processes; therefore, it has been broadly applied in management science and operations research. There are powerful mathematical programming algorithms for solving *linear programming problems*, *i.e.*, problems for which the objective function and constraints are linear. The same holds for *concave quadratic programming problems*, where the goal is to maximize a concave quadratic objective function subject to a set of linear constraints. Recently, the research community concentrates on *mixed integer programming problems*, for which some constraints require part of the decision variables to be integer. This enables modeling situations in which decision variables take discrete values (*e.g.* when a company has to decide how many persons to employ, the company cannot employ a fraction of a person). The drawback is that whereas for linear programming there are known algorithms which are computational efficient - *i.e.*, which require resources that are bounded by a polynomial in the instance size - no such algorithms are known for general integer programming problems. Solving general integer programming problems has been proven to be at least as hard as solving any problem in NP, which is a complexity class believed to contain hard problems. Nevertheless, in the last decade there has been a huge scientific advance both in this setting and in computational power, resulting in software tools able to tackle (in practice) large integer programming instances.

Game theory concerns games: situations where there is more than one decision maker (*player*), and players' decisions (*strategies*) have influence in each others *utilities*. Game theory is especially used to analyze economic models. In economic markets, the participants' strategies will influence the market outcomes. There are many varieties of games implying distinct research approaches, we name some. A game may have a finite number of players or not; players may cooperate or not; there can exist full information about each player's utility and set of strategies or not; a game representation can vary: games can be classified into situations in which each player's set of strategies is finite and explicitly given (class of *finite games*), or situations in which the set of strategies is uncountable or not given explicitly (for example, in *continuous games*, each player set of strategies can be a closed interval of $\mathbb{R}$); players select strategies simultaneously or sequentially. In this thesis, we concentrate on the case of full information non-cooperative continuous games with a finite number of players, and in both two round and simultaneous games. In order to define a game, one must describe the players, their strategies and their utilities, as well as the game dynamic. A widely accepted solution to a game is the concept of *equilibrium*, which is a *profile of strategies*, one for each player, such that each player has no incentive to deviate from the equilibrium strategy if the opponents play according to that equilibrium strategies. There are results concerning sufficient conditions for a game to possess an equilibrium. Generally, however, the existence proofs are inefficient. In fact, for a large class of games, it has been proven that the problem of computing one equilibrium is at least as hard as solving any problem in the complexity class PPAD, which contains problems believed to be computationally hard.

Note that in game theory each player aims at selecting the most rational strategy; in other words, a player seeks her optimal decision. Thus, each player has an optimization problem to solve; this merges the mathematical programming and the game theory frameworks. Games using mixed integer programming formulations to describe a player's optimization problem have been seldom addressed. We call this category *integer programming games*. In this context, there are four natural research questions. Do integer programming games model real-world situations? Are there equilibria for integer programming games? How to compute equilibria? What is the computational complexity of computing equilibria? The literature in this context is scarce, focusing on special cases, using situation-specific structure or using solution concepts different from equilibrium.

1.2 Organization and Contributions

We close this chapter by outlining the thesis organization and research contributions to answer the questions raised above.

Chapter 2. The fundamental background material and notation are presented in Chapter 2, which is divided into three parts. In the first part, Section 2.1, the relevant complexity classes are defined: *polynomial time* P, *nondeterministic polynomial time* NP, *second level of the polynomial hierarchy* Σ_2^p and *Polynomial Parity Arguments on Directed graphs* PPAD. In Section 2.2 important mathematical programming definitions, well-known techniques to solve relevant optimization problems and available software tools are presented. Section 2.2.1 complements the mathematical programming introduction through the presentation of pertinent classical integer programming examples: the *maximum matching in a graph*, *knapsack problem* and *lot-sizing problem*. These formulations are later used to define a player's goal in the games at hand. The third part, Section 2.3, introduces central game theory concepts, establishes the connection with mathematical programming and formally defines integer programming games, the main topic of this thesis. That section has two parts. The first part, Section 2.3.1, defines two-round sequential games, known as Stackelberg competition or bilevel programming (under pessimistic and optimistic assumptions), Stackelberg equilibria and interdiction problems, and it also describes the challenges of computing these games' solutions. It concludes with literature review, which motivates further research in this field, and thus, our work in this context. The second part, Section 2.3.2, defines simultaneous games, Nash equilibrium, and presents known results about existence and characterization of equilibria. It follows the relevant literature review about simultaneous games, which points out the novelty of studying integer programming games. We conclude this chapter with the available solvers for games.

Chapter 3. Chapter 3 presents our contributions on Stackelberg competitions. In these games, there is a player called the *leader* that takes first her decision and another called the *follower* that can observe the leader's strategy prior to playing. In Section 3.1, three natural generalizations of the knapsack problem to two levels are modeled, which have in common the follower's optimization program: a knapsack problem. The following variants are considered: the follower's knapsack capacity is decided by the leader; the follower shares the knapsack capacity with the leader; and the items available for the follower are decided by the leader. In Section 3.2, we prove that: these bilevel knapsack variants are complete for the second level of the polynomial hierarchy under binary encondings; two of them become polynomial solvable under unary encondings, whereas the third becomes NP-complete; the third variant has a polynomial time approximation scheme, whereas the other two cannot be approximated in polynomial time within any constant factor, assuming P $\neq$ NP. Additionally, in Section 3.3, for the third variant of the bilevel knapsack problem (an interdiction problem) a novel algorithm is proposed and tested in order to show its practical effectiveness. Furthermore, it gives insights about generalizing the presented ideas to interdiction problems with real-world applicability.

Chapter 4. Chapter 4 focuses on simultaneous games. Section 4.1 starts by presenting our contributions for the simplest integer programming game that we could devise: the coordination knapsack game. In Section 4.2 an application of game theory in the context of health care is presented: *Competitive Two-Player Kidney Exchange Game*. This game has good properties, in the sense that an equilibrium can be computed efficiently and players would agree on the equilibrium to be played (a game may have multiple equilibria). Moreover, the work developed expands results concerning matchings in graphs. A classical game in economics, *Cournot competition*, is generalized in Section 4.3 in order to include a lot-sizing problem for each player in the market. The complexity of this game is investigated, allowing to identify cases in which an equilibrium for the game can be computed efficiently. Finally, Section 4.4 tackles the general case of simultaneous integer programming games. We start by proving that deciding about the existence of an equilibrium to a simultaneous integer programming game (even with only two players and linear utilities) is at least as hard as solving bilevel knapsack variants of Section 3.1, enabling us to relate sequential and simultaneous games. We derive sufficient conditions to guarantee equilibria. The section finishes proposing an algorithm to approximate equilibria in finite time, as well as the associated computational results. To the best of our knowledge, there are no previous algorithms in the literature capable of treating games with such a general form and therefore, we hope our contribution to be a stepping stone to future results in this context.

Chapter 5. The thesis concludes in Chapter 5, summarizing our contributions and presenting future research directions.

Chapter 2

Background

In this chapter, we provide the essential background that supports our contributions. We start defining the complexity classes P, NP, Σ_2^p and PPAD that will be employed later as a way of classifying the (in)tractability of the solution computation for the games under our study. The games in this thesis are represented by mathematical programming formulations, which are introduced next, along with duality theory (which is frequently at the base of algorithmic approaches in this context, including the algorithm proposed in Chapter 3). Computational complexity, most common used methods and solvers for mathematical programming problems are also presented in this chapter, which terminates with a game theory background, its connection to mathematical programming, and a literature review.

2.1 Complexity: P, NP, Σ_2^p and PPAD classes

The first developments in complexity theory are traced back to 1965 [71]. It was the frequency of intractable-looking problems faced by algorithm designers, that led to the development of complexity theory in computer science.

In this section, we introduce the basic computational complexity concepts required for the understanding of the work in this thesis. We refer the reader to Garey and Johnson [56], Papadimitriou [101] and Stockmeyer [121] for a comprehensive and relevant background in computational complexity.

A *decision problem* $\mathcal{A}$ consists of a set $D_{\mathcal{A}}$ of instances and a subset $Y_{\mathcal{A}} \subseteq D_{\mathcal{A}}$ of YES instances; the problem is to determine whether a given instance is a YES instance or not. A *deterministic algorithm* solves problem $\mathcal{A}$, if it halts for all input instances in $D_{\mathcal{A}}$ and returns the answer YES if and only if the instance is in $Y_{\mathcal{A}}$, otherwise, returns the answer NO. If the number of steps executed by the deterministic algorithm is bounded by a polynomial in the input size, then it is a *polynomial time deterministic algorithm*; we say that such algorithm is efficient. The *polynomial time* complexity class, denoted by P, consists of all decision problems for which a polynomial time deterministic algorithm exists. Cobham [24] and Edmonds [47] were the first to identify the relevance of studying the concept of efficient solvability, that is, to recognize the problems belonging to P.

A *nondeterministic algorithm* solves a decision problem $\mathcal{A}$ if the following two properties hold for all instances $\mathcal{I} \in D_{\mathcal{A}}$:

1. If $\mathcal{I} \in Y_{\mathcal{A}}$, then there exists a certificate $\mathcal{S}$ that, when guessed for input $\mathcal{I}$, will lead the algorithm to respond YES for $\mathcal{I}$ and $\mathcal{S}$;
2. If $\mathcal{I} \notin Y_{\mathcal{A}}$, then there exists no certificate $\mathcal{S}$ that, when guessed for input $\mathcal{I}$, will lead the algorithm to respond YES for $\mathcal{I}$ and $\mathcal{S}$.

A nondeterministic algorithm that solves a decision problem $\mathcal{A}$ is said to operate in polynomial time if there exists a polynomial p such that, for every instance $\mathcal{I} \in Y_{\mathcal{A}}$, there is some guess $\mathcal{S}$ that checks whether the response is YES for $\mathcal{I}$ and $\mathcal{S}$ within time $p(|\mathcal{I}|)$ (where $|I|$ is the size of $\mathcal{I}$). The *nondeterministic polynomial time* complexity class, denoted by NP, consists of all decision problems that can be solved by polynomial time nondeterministic algorithms. The class NP contains the problems in P and it is believed to strictly contain P, *i.e.*, that there are problems which cannot be solved efficiently. For some problems in NP for which it is not known an efficient algorithm, there are *pseudo-polynomial time algorithms*. An algorithm that solves problem $\mathcal{A}$ is called a pseudo-polynomial time algorithm for $\mathcal{A}$ if its time complexity is bounded above by a polynomial function of two variables: input size and magnitude of the largest number in the input. For sake of simplicity, whenever it is said polynomial time, we mean deterministic polynomial time.

A *polynomial transformation* (also called a *reduction*) from a decision problem $\mathcal{A}_1$ to a decision problem $\mathcal{A}_2$ is a function $f : D_{\mathcal{A}_1} \to D_{\mathcal{A}_2}$ that can be executed by a polynomial time deterministic algorithm such that for all instance $\mathcal{I} \in D_{\mathcal{A}_1}$, $\mathcal{I}$ is a YES instance for $\mathcal{A}_1$ ($\mathcal{I} \in Y_{\mathcal{A}_1}$) if and only if $f(\mathcal{I})$ is also a YES instance for $\mathcal{A}_2$ ($f(\mathcal{I}) \in Y_{\mathcal{A}_2}$).

A decision problem $\mathcal{A}$ is *complete* for a complexity class $\mathcal{C}$ if $\mathcal{A} \in \mathcal{C}$ and there is a polynomial transformation from $\mathcal{A}'$ to $\mathcal{A}$ for all $\mathcal{A}' \in \mathcal{C}$. Therefore, complete problems are the most difficult problems in their class. It was the difficulty at finding efficient algorithms to solve some NP problems that was at the base of NP-*completeness* theory, which is attributed to Cook [27]. Even conceptually simple problems can be NP-complete.

Such an NP-complete example is the famous PARTITION problem [56].

Problem: PARTITION

Instance: A sequence $a_1, a_2, \ldots, a_n$ of positive integers.

Question: Does there exist a set $S \subseteq \{1, 2 \ldots, n\}$ such that (PP)

$$\sum_{i \in S} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i?$$

The study of complexity classes that lie beyond NP was motivated by natural problems for which their precise classification in terms of the known complexity classes failed. The *polynomial hierarchy* was introduced by Meyer and Stockmeyer [91] in an attempt to properly classify decision problems that appear to be harder than NP-complete. In this thesis we focus on the second level of the polynomial hierarchy, denoted by Σ_2^p, built on top (lower level) of NP-complete problems. The Σ_2^p class consists in all decision problems that are solvable by polynomial time nondeterministic algorithms with access to an NP oracle. An NP oracle outputs the correct answer for problems in NP and each call to the oracle is counted as one computational step. Equivalently, Σ_2^p contains all decision problems in the form $\exists x \forall y \, P(x, y)$, that is, as a logical formula starting with an existential quantifier, followed by a universal quantifier, followed by a Boolean predicate $P(x, y)$ that can be evaluated in deterministic polynomial time. Lately, more and more problems have been proven to be Σ_2^p-complete; see Johannes [70]. An example of a Σ_2^p-complete problem is the SUBSET-SUM-INTERVAL decision problem (see Eggermont and Woeginger [48]).

Problem: SUBSET-SUM-INTERVAL

Instance: A sequence $q_1, q_2, \ldots, q_k$ of positive integers; two positive integers R and r with $r \leq k$. (SSI)

Question: Does there exist an integer S with $R \leq S < R + 2^r$ such that none of the subsets $I \subseteq \{1, \ldots, k\}$ satisfies $\sum_{i \in I} q_i = S$?

In this thesis, for some of our problems there is a proof of existence of a solution, but the existence proof does not provide an efficient algorithm. These are intractable problems of very different kind than decision problems and, thus, not suitable to be computationally classified through the previously defined classes. This is the case for the END-OF-THE-

Line problem.

Problem: End-Of-The-Line

Instance: A directed graph G; a specified unbalanced vertex (*i.e.*, the number of incoming arcs differs from the number of outgoing arcs). (ETL)

Question: Which is another unbalanced vertex?

For this problem, a solution is guaranteed to exist, by a parity argument. However, note that simply investigating the remaining vertices in order to find the unbalanced one cannot be guaranteed to be done in polynomial time ,since there is no specification on how G is given in the input. To see this, consider the case that G has 2^n vertices, one for every binary string of length n, and the vertices adjacency are given through two boolean circuits of polynomial size in n, call them predecessor and successor, such that, given a vertex, the predecessor returns a list of the incoming edges and the successor returns a list of the outgoing edges. In order to address the issue of giving a computational complexity classification for this different type of problems, the class *Polynomial Parity Arguments on Directed graphs*, denoted by PPAD, was introduced by Papadimitriou [102]. PPAD is a class of problems that can be reduced to the End-Of-The-Line. Therefore, a problem is PPAD-complete if End-Of-The-Line can be reduced to it. The PPAD class is believed to contain hard computational problems (such as fixed point problems); in particular, it is conjectured that P $\neq$ PPAD.

For some hard problems, it is possible to compute an "arbitrarily close solution" within polynomial time. In this thesis, we essentially study optimization problems; when these problems are hard, finding an approximate optimal solution efficiently is relevant. We conclude this section by defining *approximation scheme* for an optimization problem $\mathcal{A}$ to be an algorithm that takes as input both an instance $\mathcal{I} \in D_{\mathcal{A}}$ and an accuracy requirement $\varepsilon > 0$, and that then outputs a candidate solution with value $Approx(\mathcal{I})$ such that

$$\begin{aligned} \frac{OPT(\mathcal{I})}{Approx(\mathcal{I})} &\leq 1+\varepsilon \quad \text{for maximization problems} \\ \frac{Approx(\mathcal{I})}{OPT(\mathcal{I})} &\leq 1+\varepsilon \quad \text{for minimization problems} \end{aligned}$$

where $OPT(\mathcal{I})$ is the optimal value for instance $\mathcal{I}$. An algorithm is a *polynomial time approximation scheme* if, for each fixed $\varepsilon > 0$, it returns the approximate solution in polynomial time.

2.2 Mathematical Programming

Definitions and Basic Results. In mathematical programming, a problem is defined by a vector of *decision variables*, a function of that vector to be maximized or minimized, called *objective function*, and a set of *constraints* defining the feasible region for the decision variables. We denote the set of feasible vectors by X. The aim in a maximization problem (respectively, minimization) is to find an *optimal solution*, *i.e.*, a feasible vector for the decision variables such that the corresponding objective function is maximized (respectively, minimized). A problem is *feasible* if the set of feasible vectors for the decision variables is not empty, otherwise, it is *infeasible*; a maximization problem (respectively, minimization) is *bounded* if the objective function cannot assume arbitrarily large positive (respectively, negative) values at feasible vectors, otherwise, it is said to be *unbounded.*

A linear programming problem (LP) can be expressed as

$$\underset{x}{\text{maximize}} \quad (\max_{x})\; c^{\mathsf{T}}x \tag{2.2.1a}$$

$$\text{subject to (s.t.)} \quad Ax \leq b \tag{2.2.1b}$$

$$x_i \geq 0 \quad \text{for } i = 1, \ldots, n, \tag{2.2.1c}$$

where x is an n dimensional column vector of decision variables (decision vector), $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, A is an m-by-n real matrix and $(\cdot)^{\mathsf{T}}$ is the transpose operator. The objective function is defined in (2.2.1a). Constraints (2.2.1b) and (2.2.1c) define a *polyhedron* in $\mathbb{R}^n$, the feasible region X.

A set of points $\mathcal{P}$ is *convex* if for any set of points $z_1, z_2, \ldots, z_k \in \mathcal{P}$ and $\lambda_1, \lambda_2, \ldots, \lambda_k \in \mathbb{R}_+$ with $\sum_{i=1}^{k} \lambda_i = 1$, the *convex combination* $\sum_{i=1}^{k} \lambda_i z_i$ is in $\mathcal{P}$ (it is called *affine combination* if $\lambda_i \in \mathbb{R}$). The *dimension* of a convex set is n if and only if it has $n+1$, but no more, affinely independent points (*i.e.*, none of these points is an affine combination of the others). The polyhedron X defining the feasible region of an LP is a convex set. A *face* of X is a set $\{x \in X : \alpha^{\mathsf{T}}x = \beta\}$ for some $\alpha \in \mathbb{R}^n$, $\beta \in \mathbb{R}$ so that the inequality $\alpha^{\mathsf{T}}x \leq \beta$ holds for all $x \in X$. A *vertex* of X is the unique element of a zero dimensional face of X. It is a well-known result that if an LP has an optimum, then there is a vertex of X that is an optimal solution. A *facet* of a n dimensional polyhedron is a face of dimension $n-1$. See Nemhauser and Wolsey [95] for details in polyhedral theory.

Duality Theory (see Dantzig [34]) plays an important role in the context of linear programming. Introduced by von Neumann [129], the *dual problem* of the LP (2.2.1) is

$$\begin{aligned} \underset{x}{\text{minimize}} \quad (\underset{x}{\min}) \quad & b^\intercal y && (2.2.2a)\\ \text{s.t} \quad & A^\intercal y \geq c && (2.2.2b)\\ & y_i \geq 0 \quad \text{for } i = 1, \ldots, m. && (2.2.2c) \end{aligned}$$

In this context, LP (2.2.1) is called the *primal problem.* In what follows we summarize the primal-dual relationships.

Property 2.2.1 (Weak duality). *If x is a feasible solution for the primal problem* (2.2.1) *and y is a feasible solution for the dual problem* (2.2.2)*, then $c^\intercal x \leq b^\intercal y$.*

Property 2.2.2 (Strong duality). *If x^* is an optimal solution for the primal problem* (2.2.1) *and y^* is an optimal solution for the dual problem* (2.2.2)*, then $c^\intercal x^* = b^\intercal y^*$.*

Property 2.2.3 (Complementary slackness property). *If x is a feasible solution for the primal problem* (2.2.1) *and y is a feasible solution for the dual problem* (2.2.2)*, then x and y are optimal for their respective problems if and only if $x^\intercal (A^\intercal y - c) = 0_n$ and $y^\intercal (Ax - b) = 0_m$, where 0_k is k-dimensional column vector of zeros.*

Gale *et al.* [55] formulated the Duality Theorem.

Theorem 2.2.4 (Duality Theorem). *The following are the only possible relationships between the primal problem* (2.2.1) *and its dual problem* (2.2.2).

1. *If one problem has feasible solutions and a bounded objective function (and so has an optimal solution), then so does the other problem, so both the weak and strong duality properties are applicable.*
2. *If one problem has feasible solutions and an unbounded objective function (and so no optimal solution), then the other problem has no feasible solutions.*
3. *If one problem has no feasible solutions, then the other problem has either no feasible solutions or unbounded objective function.*

A *mixed integer programming problem* (MIP) has the following additional constraints with respect to an LP (problem (2.2.1)):

$$x_i \in \mathbb{Z}, \qquad \text{for } i = 1, \ldots, B, \tag{2.2.3}$$

where $B < n$. If $B = n$ it is an integer programming problem (IP). The *convex hull* of a set $\mathcal{P}$, denoted by $\text{conv}(\mathcal{P})$, is the set of all convex combinations of points in $\mathcal{P}$:

$$\text{conv}(\mathcal{P}) = \left\{ \sum_{i=1}^{k} z_i \lambda_i : \sum_{i=1}^{k} \lambda_i = 1 \text{ and } z_i \in \mathcal{P}, \lambda_i \geq 0 \text{ for all i} \right\}.$$

The convex hull of the feasible region for an MIP is a polyhedron (*i.e.*, it can be described by a system of inequalities). It is easy to see that if an MIP has an optimum, then there is a vertex of the convex hull of the feasible region for this MIP that is an optimal solution.

A *quadratic programming problem* (QP) has the following term added to the objective function of an LP (problem (2.2.1)):

$$-\frac{1}{2}x^{\mathsf{T}}Qx, \tag{2.2.4}$$

where Q is an n-by-n real symmetric matrix. If integer requirements are added to the constraints of an QP, we call the problem a *mixed integer quadratic programming problem* (MIQP).

Solving Optimization Problems. If the objective function of a maximization (minimization) problem over a polyhedron X is concave (convex), typically, it means that it can be solved efficiently, while the reverse, non-concave (non-convex) objective function, usually, leads to intractability. LP's were proven to be solvable efficiently through the ellipsoid algorithm by Khachiyan [75]. However, a remarkably fast procedure is more used in practice: the simplex method, by Dantzig [32] (which has worst-case exponential time). In case the objective function of a (maximization) QP is a concave (which is equivalent to the condition

$$x^{\mathsf{T}}Qx \geq 0, \qquad \text{for all } x,$$

i.e., Q is a *positive semidefinite* matrix), then it can be solved in polynomial time, *e.g.*, through the ellipsoid method.

The decision version of an optimization problem is to ask whether there is a feasible value for x such that the corresponding objective function is better than a predefined value. In order to simplify the text and whenever the context makes it obvious, we will simply say that an optimization problem is or not in NP according to its decision version.

If an QP is not concave (*i.e.*, if matrix Q is not positive semedefinite), the problem is NP-complete. The difficulty comes from the fact that QP can have multiple local optima (x is a local optimum if there is a neighborhood $\text{Viz}(x) \subseteq X$ such that for any $y \in \text{Viz}(x)$,

$$c^{\mathsf{T}}x - \frac{1}{2}x^{\mathsf{T}}Qx \geq c^{\mathsf{T}}y - \frac{1}{2}y^{\mathsf{T}}Qy$$

holds). IP is NP-complete, which implies that MIP and MIQP are NP-hard, since IP is a special case of these problems. In practice, there are powerful software tools that implement branch-and-bound, cutting planes and branch-and-cut techniques to tackle these problems with the integer requirements.

- The branch-and-bound scheme, proposed by Land and Doing [45], starts by solving the *continuous relaxation* of the problem (*i.e.*, solving the problem without integrality requirements); given an optimal solution x^* of the continuous relaxation with a fractional value x_i^*, for some $1 \leq i \leq B$, the problem is divided into two subproblems: one with constraint $x_i \leq \lfloor x_i^* \rfloor$ and another with $x_i \geq \lfloor x_i^* \rfloor + 1$; each subproblem is a node of the branch-and-bound tree. The process is repeated for each node, until the continuous relaxation of the subproblem is infeasible, integer feasible or the upper bound value of the subproblem is worse than the current best found feasible solution (under these three cases, the node is fathomed).
- The cutting plane approach, presented by Gomory [60], also starts solving the continuous relaxation. Given any solution x^* of the continuous relaxation, a separation problem is solved, *i.e.*, a problem whose aim is to find a valid linear inequality (cut) that cuts off x^* (an inequality which holds for any $x \in \text{conv}(X)$ but is not satisfied by x^*). The continuous relaxation with the addition of that inequality is solved, and the process repeats until a solution satisfying the integer requirement is found, or the problem is proven to be infeasible. See Cornuéjols [28] for a unified description of different groups of cuts.
- Branch-and-cut combines the two methods just described in order to integrate their advantages in a process which has been proven to be very effective; see Padberg and Rinaldi [100].

We refer the interested reader to Jünger *et al.* [72] for a survey and state-of-the-art of methods to solve MIPs.

Mathematical Programming Solvers. We restrict our attention to solvers for linear and concave (maximization) problems since, in this thesis, the optimizations at hand belong to one of these two classes.

As mentioned in the previous section, the difficulties of solving IPs, MIPs and MIQPs come from the consideration of integer requirements in the decision variables. However, recent software tools, both commercial and non-commercial, can in practice efficiently and reliably tackle some of these optimization problems. In this context, the fastest solvers are the open-source SCIP [116] (with SoPlex) and Cbc [20], and the commercial software Xpress [135], CPLEX [67] and Gurobi [63]. In order to analyze these solvers' evolution and compare their performance, the research community has created archives of benchmark instances. Since their foundation (SCIP in 2002; Cbc in 2005; Xpress in 1983; CPLEX in 1988; Gurobi in 2009) up to their current versions (SCIP 3.2.0 (with SoPlex 2.2.0); Cbc 2.0.4; Xpress 7.9.0; CPLEX 12.6.2; Gurobi 6.0.0) we can observe that there has been a significant advance in terms of improving computational times and including new

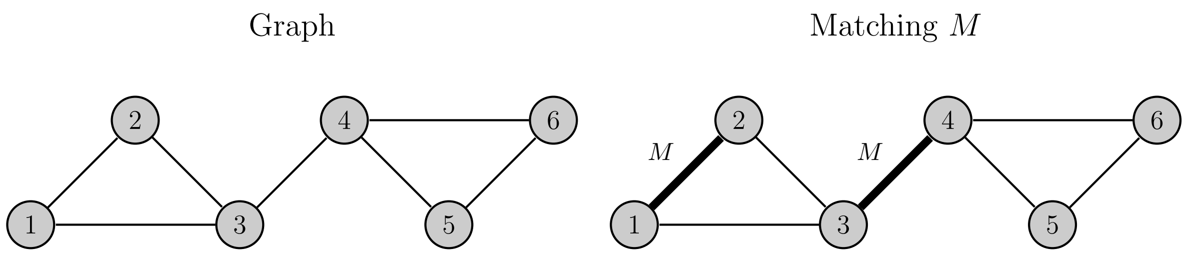


Figure 2.2.1: Matching in a Graph.

features (like solving MIP's and MIQP). The commercial solvers are in general faster than the referred open-source ones. On the other hand, the open-source solvers allow a better understanding of the underlying methods, as well as their modification to implement and test new algorithmic ideas.

The success of most of these solvers is not only due to the increase in computational power, but rather to improvements in the implementation of a branch-and-cut structure merged with sophisticated preprocessing and heuristic techniques.

2.2.1 Classical Examples

In this section, we will present three classical problems extensively studied in the literature of combinatorial optimization. We first present maximum matching problem in a graph, which is an IP that can be solved in polynomial time (Section 2.2.1.1). Then, in Section 2.2.1.2, we describe a model of the knapsack problem, which is also an IP, but is known to be NP-complete. Section 2.2.1.3 concludes with an MIP model for the lot-sizing problem, which is NP-complete but under some conditions can be solved in polynomial time.

2.2.1.1 Maximum Matching in a Graph

A *graph* $G = (V, E)$ is described by a set of *vertices* V and a set E of unordered pairs of vertices, called the *edges*. A subset M of E is called a *matching* of a graph G if no two edges in M share the same vertex. See Figure 2.2.1 for an illustration of a graph and a possible matching.

The maximum matching in a graph (MMG) is the problem of finding a matching of maximum carnality. For instance, the matching M in Figure 2.2.1 is not maximum; the set M together with edge $(5, 6)$ is a maximum matching.

Let us define some concepts of graph theory in matching and review some results. For a matching M in graph $G = (V, E)$, an *M-alternating path* is a path whose edges are alternately in $E \setminus M$ and M. An *M-augmenting path* is an M-alternating path whose origin and destination are M-unmatched vertices. Next, we present a simple property often used in this context.

Property 2.2.5. *Let M be a maximum matching of a graph $G = (V, E)$. Consider an arbitrary $R \subset M$ and the subgraph H of G induced by removing the R-matched vertices. The union of any maximum matching of H with R is a maximum matching of G.*

Next, we recall Berge's theorem [11].

Theorem 2.2.6 (Berge). *A matching M of a graph G is maximum if and only if it has no augmenting path.*

Berge's theorem is constructive, leading to an algorithm to find a maximum matching: start with an arbitrary matching M of G; while there is an M-augmenting path p, switch the edges along the path p from in to out of M and vice versa: update M to $M \oplus p$, where $\oplus$ denotes the symmetric difference of two sets (*i.e.*, the set of elements which are in either of the sets but not in their intersection). The updated M is a matching with one more edge, where the previously matched vertices are maintained matched.

Edmonds [47] proved that the problem of computing a maximum matching can be solved in polynomial time for any graph. Edmonds built a polynomial time algorithm to find an augmenting path for a matching. This algorithm together with Berge's theorem leads to a polynomial time iterative method: successively apply augmenting paths in a matching until there is none and, thus a maximum matching was found.

See Chapter 5 of [13] for details about matching on graphs.

2.2.1.2 The Knapsack Problem

The knapsack problem is one of the most fundamental problems in combinatorial optimization. It has been studied extensively, as certified for example by the books by Martello and Toth [87] and by Kellerer, Pferschy and Pisinger [74].

Consider a set of n items numbered from 1 to n. For each item i there is an associated profit $p_i > 0$ and weight $w_i > 0$. The knapsack problem (KP) consists in finding which items must be packed in a knapsack such that its capacity C is not exceeded and the profit is maximized. KP can be written as the following IP:

$$\max_x \quad \sum_{i=1}^{n} p_i x_i \tag{2.2.5a}$$

$$\text{s. t.} \quad \sum_{i=1}^{n} w_i x_i \leq C, \tag{2.2.5b}$$

$$x_i \in \{0,1\}, \qquad \text{for } i = 1,\dots,n, \tag{2.2.5c}$$

where x_i is the decision variable associated with packing item i ($x_i = 1$) or not ($x_i = 0$). The objective (2.2.5a) is to maximize the total profit for the packed items. Constraint (2.2.5b) ensures that the knapsack capacity is not exceeded and constraints (2.2.5c) guarantee that the decision variables are binary. It is assumed that p_i, w_i and C are positive integers.

Let us recall some standard concepts and results in this context. Assume that the items are ordered by non-increasing profit-to-weight ratio, *i.e.*,

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}. \tag{2.2.6}$$

The item c defined by

$$c = \min\{j : \sum_{i=1}^{j} w_i > C\},$$

is called the *critical item* of the knapsack instance.

A famous property established by Dantzig [33] can be used to solve the continuous relaxation of KP.

Theorem 2.2.7 (Dantzig [33]). *Suppose that the items are ordered as in* (2.2.6). *An optimal solution x^* of the continuous relaxation of problem* (2.2.5) *is given by*

$$x_i^* = 1 \textit{ for } i = 1,\dots,c-1$$

$$x_i^* = 0 \textit{ for } i = c+1,\dots,n$$

$$x_c^* = \left(C - \sum_{i=1}^{c-1} w_i\right) / w_c.$$

The continuous relaxation of KP immediately provides an upper bound.

Corollary 2.2.8. *A trivial upper bound to KP* (2.2.5) *is given by*

$$U = \sum_{i=1}^{c-1} p_i + x_c^* p_c.$$

Although solving the continuous relaxation of KP can be done in polynomial time, the same is unlikely to hold for KP itself since it is an NP-complete problem (see [56]).

2.2.1.3 The Lot-Sizing Problem

Production planning is a classical problem in operations research, given its practical applications and the related challenging models in mixed integer programming. In this section, we focus on the simplest case, with only one machine, and planning the production of only one item. The lot-sizing problem (LSP) can be described as follows. There is a finite planning horizon of $T > 0$ periods. For each period $t = 1, \dots, T$, the demand is $D_t \geq 0$, the unit production cost (also known as variable cost) is $C_t \geq 0$, the unit inventory cost is $H_t \geq 0$, the fixed set-up cost is $F_t \geq 0$ and the production capacity is M_t. The goal is to find a production plan such that the demand of each period is satisfied and the total cost is minimized. Thus, we can model the problem as the following MIP:

$$\begin{aligned}
\min_{x,h,y} \quad & \sum_{t=1}^{T} C_t x_t + \sum_{t=1}^{T} H_t h_t + \sum_{t=1}^{T} F_t y_t & & \text{(2.2.7a)}\\
\text{s. t.} \quad & x_t + h_{t-1} = D_t + h_t & \text{for } t = 1, \dots, T \qquad & \text{(2.2.7b)}\\
& 0 \leq x_t \leq M_t y_t & \text{for } t = 1, \dots, T \qquad & \text{(2.2.7c)}\\
& h_0 = h_T = 0 & & \text{(2.2.7d)}\\
& h_t, x_t \geq 0 & \text{for } t = 1, \dots, T \qquad & \text{(2.2.7e)}\\
& y_t \in \{0, 1\} & \text{for } t = 1, \dots, T \qquad & \text{(2.2.7f)}
\end{aligned}$$

where, for each period $t = 1, \dots, T$, x_t is the production quantity, h_t is the quantity in inventory in the end of that period and y_t indicates if there was production ($y_t = 1$) or not ($y_t = 0$). The objective (2.2.7a) is to minimize the total production cost. Constraints (2.2.7b) model the conservation of product. Constraints (2.2.7c) ensure that the quantities produced are non-negative, satisfy the production limit, and assure that whenever there is production ($x_t > 0$), the binary variable y_t is set to 1, implying the payment of the set-up cost. Constraint (2.2.7d) fixes the initial and final inventory quantities to be 0, which is a simplification that does not reduce generality. Moreover, through equation (2.2.7b), the objective function could be alternatively written without the inventory costs; we assume such simplification from now on.

In the uncapacitated lot-sizing problem (ULSP), for each period t the production capacity M_t does not limit production, and the problem can be solved in polynomial time through dynamic programming, as presented next. A well-known property that reveals the structure of the ULSP is the following.

Proposition 2.2.9. *There exists an optimal solution to ULSP in which $h_{t-1}x_t = 0$ for all t.*

This proposition allows to describe the optimal solution to ULSP as follows.

Proposition 2.2.10. *There exists an optimal solution to ULSP characterized by*

i. *a subset of periods* $1 \leq t_1 < \ldots t_r \leq T$ *in which production takes place; the amount produced in* t_j *is* $D_{t_j} + \ldots + D_{t_{j+1}-1}$ *for* $j = 1, \ldots, r$ *with* $t_{r+1} = T + 1$*;*

ii. *a subset of periods* $R \subseteq \{1, \ldots, T\} \backslash \{t_1, \ldots, t_r\}$*; there is a set-up in periods* $\{t_1, \ldots, t_r\} \cup R$*.*

Proposition 2.2.10 shows that an optimal solution can be decomposed into a sequence of intervals, $[t_1, t_2 - 1]$, $[t_2, t_3 - 1]$, ..., $[t_r, T]$, plus some additional set-ups without production (periods in R). Let $G(t)$ be the minimum cost of solving ULSP over the first t periods, that is, satisfying the demands D_1, ..., D_t, and ignoring the demands after period t, and let $\phi(k, t)$ be the minimum cost of solving the problem over the first t periods subject to the additional condition that the last set-up and production period is $k \leq t$. From the definition, it follows that

$$G(t) = \min_{k:k \leq t} \phi(k, t). \tag{2.2.8}$$

Using the optimal solution description by Proposition 2.2.10 it is easy to conclude that the value of $\phi(k, t)$ is equal to the minimum cost of solving the problem over the first $k-1$ periods plus the costs associated with producing in period k to satisfy the demand up to period t. Therefore,

$$\phi(k, t) = G(k - 1) + F_k + C_k \sum_{j=k}^{t} D_t. \tag{2.2.9}$$

Now we have the tools to describe the dynamic programming procedure. Start with $G(0) = 0$ and calculate $G(1)$, $G(2)$, ..., terminating with the optimal value, $G(T)$ through the recursion

$$G(t) = \min_{k:k \leq t} \left[G(k - 1) + F_k + C_k \sum_{j=k}^{t} D_t \right]. \tag{2.2.10}$$

In order to recover the optimal solution, some additional information must be kept. These calculations can be done polynomially, in $O(T^2)$ computing time. In fact, the computation can be further improved in order to run in $O(T \log T)$.

When the production capacities are constant over time, LSP remains polynomially solvable. However, if capacities are time-varying the problem becomes NP-complete. The interested reader is referred to Pochet and Wolsey [106] for a complete treatment of production planning problems.

2.3 Game Theory

Basic Definitions. Game theory (Fudenberg and Tirole [53], Owen [99]) is a generalization of decision theory where players are concerned about finding their "best" strategies subject to the fact that each controls some, but not all, actions that can take place. It can be applied in a wide range of fields such as economics, political science, operations research and evolutionary biology; in short, whenever multiple agents interact. In a game, each player is a decision-maker and her utility is influenced by the other participants' decisions. A game is described by a set of players M, each player $p \in M$ having a (nonempty) set of feasible strategies X^p and a real-valued utility function Π^p over all combinations of the players' feasible strategies, *i.e.*, the domain is $X = \prod_{k \in M} X^k$. We call each $x^p \in X^p$ and $x \in X$ a player p *pure strategy* and a *pure profile of strategies*, respectively. In this thesis, it is assumed that the games are *non-cooperative* (*i.e.*, players have no compassion for the opponents), players are rational and there is *complete information*, *i.e.*, players have full information about each other utilities and strategies.

In a finite game, the set of strategies for each player p is finite and explicitly enumerated, that is, $X^p = \{1, 2, \dots, n_p\}$. Usually, it is represented in *normal-form* (or *strategic-form*), this is, through a multidimensional matrix with an entry for each pure profile of strategies $x \in X$, where that entry is an m dimensional vector of the players' utilities associated with x. The following example serves to illustrate the concepts just described.

Example 2.3.1. *In the well-known "rock-scissors-paper" game there are two players, $M = \{1, 2\}$. The set of feasible strategies for each player $p \in M$ is $X^p = \{rock, scissors, paper\}$. The players' utilities for each possible game outcome are given in the bimatrix of Table 2.1. Player 1 is the row player and player 2 is the column player; for each combination of the players' strategies there is an entry in the bimatrix which is a vector of their utilities: the first value is player 1 utility and the second value is player 2 utility. When the pure strategy "rock" is played against "scissor", the player selecting "rock" receives a utility of 1 paid by the opponent (who gets -1); when the pure strategy "scissors" is played against "paper", the player selecting "scissor" receives a utility of 1 paid by the opponent (who gets -1); when the pure strategy "paper" is played against "rock", the player selecting "rock" receives a utility of 1 paid by the opponent (who gets -1).*

In continuous games, broader sets of strategies with respect to finite games are considered: each player p strategy set X^p is a nonempty compact metric space and the utility Π^p is continuous[1]. In particular, in continuous games, X^p can be a set with an exponential (in

[1] All finite games are continuous games: a finite set is a compact metric space under the discrete metric and any function whose domain is endowed with the discrete metric is automatically continuous.

		Player 2		
		rock	scissors	paper
	rock	(0,0)	(1,-1)	(-1,1)
Player 1	scissors	(-1,1)	(0,0)	(1,-1)
	paper	(1,-1)	(-1,1)	(0,0)

Table 2.1: Rock-scissors-paper game

the size of the representation of the game) or uncountable number of feasible strategies. Next, we give an example of a continuous game that is not finite.

Example 2.3.2. *There are two firms (the players), $M = \{A, B\}$, producing a homogeneous good and competing in the same market. Firm A and firm B decide the quantities to produce, x^A and x^B, respectively. There is an associated unit production cost $C_p > 0$ and production capacity W_p, for each firm $p \in M$. The unit price function $P(\cdot)$ depends on the quantity of good that is put in the market; it is linear and decreasing, therefore $P(x^A + x^B) = a - b(x^A + x^B)$ with $a, b > 0$ and parameter a is greater than $2C_A$ and $2C_B$. Thus, the utility of firm $p \in M$ is*

$$\Pi^p(x^A, x^B) = \left(a - b(x^A + x^B)\right) x^p - C_p x^p$$

and the feasible set of strategies is $X^p = \{x^p : 0 \leq x^p \leq W_p\}$ (that is, the quantity x^p produced by firm p must be non negative and cannot exceed the production capacity).

In order to find "rational" strategies, the following definitions are commonly used. Let the operator $(\cdot)^{-p}$ for some $p \in M$ denote $(\cdot)$ for all players except player p. A strategy $\tilde{x}^p \in X^p$ is *dominated* if there is $\hat{x}^p \in X^p$ such that for all $x^{-p} \in X^{-p}$

$$\Pi^p(\tilde{x}^p, x^{-p}) \leq \Pi^p(\hat{x}^p, x^{-p}). \tag{2.3.1}$$

A strategy $\tilde{x}^p \in X^p$ is *conditionally dominated* given a profile of set of strategies $R^{-p} \subseteq X^{-p}$ for the remaining players, if there is $\hat{x}^p \in X^p$ satisfying

$$\Pi^p(\tilde{x}^p, x^{-p}) < \Pi^p(\hat{x}^p, x^{-p}) \quad \forall x^{-p} \in R^{-p}. \tag{2.3.2}$$

A profile of strategies is said to be *Pareto efficient* if it is not dominated [114].

Up to this point, only pure strategies have been considered. However, there are games for which pure strategies seem insufficient in providing a "rational strategy". The next example demonstrates this by showing that each pure profile of strategies is unstable.

Example (Continuation of Example 2.3.1). *In this game, both players decide simultaneously a strategy. The question is: which strategy should each player select such that* *her utility is maximized? The maximum gain that player 1 can guarantee to herself through a pure strategy is -1 and the minimum loss that player 2 can guarantee to herself through a pure strategy is 1. In other words, assume that player 1 and player 2 are* pessimistic*. Then, player 1 determines her* max-min strategy*: for each of player 1's strategies, determine her minimum gain and select player 1's strategy that maximizes her minimum gain (in this game, the three strategies lead to a minimum of -1 and thus, the maximum gain that can be guaranteed is -1). Analogously, player 2 determines her* min-max strategy*: for each of player 2's strategies, determine her maximum loss and select player 2's strategy that minimizes her maximum loss. Player 1 max-min strategy leads to a gain of -1 while player 2 min-max strategy leads to a loss of 1. Since these utility values do not coincide (i.e., the gain of player 1's max-min strategy is not the loss of player 2's min-max strategy), we conclude that none of the 6 pure profiles of strategies (game outcomes) leads to a stable situation: each player has incentive to unilaterally deviate. However, if we allow the use of more complex strategies it is possible to achieve an equilibrium, that is, a strategy for each player such that both are simultaneously maximizing their utilities.*

Motivated by Example 2.3.1, we introduce basic concepts of measure theory to formalize the use of a probability distribution among a set of strategies. Let Δ^p denote the space of Borel probability measures (see Fremlin [52]) over X^p and $\Delta = \prod_{p \in M} \Delta^p$. Similarly to pure strategy and profile definitions, $\sigma^p \in \Delta^p$ and $\sigma \in \Delta$ are called player p *mixed strategy* and *mixed profile of strategies*, respectively. In a *strict mixed strategy* no pure strategy is played with probability 1. For the sake of simplicity, whenever the context makes it clear, we use the term strategy to refer to a pure one. We make the standard *von Neumann-Morgenstern expected utility assumption* [130] that each player's utility under a profile of mixed strategies is the expected utility when all players choose their strategies according to their respective probability distributions in an independent way. Therefore, for $\sigma \in \Delta$, each player p expected utility is

$$\Pi^p(\sigma) = \int_{X^p} \Pi^p(x) d\sigma. \tag{2.3.3}$$

A player p *best reaction* (or best response) to a (fixed) strategy $\sigma^{-p} \in \Delta^{-p}$ of the opponents is a solution to

$$\max_{x^p \in X^p} \quad \Pi^p(x^p, \sigma^{-p}). \tag{2.3.4}$$

Example (Continuation of Example 2.3.1)**.** *If both players decide to assign a probability of $\frac{1}{3}$ for each of their three strategies, applying the probability theory definition of expected value, player 1 and player 2 could both guarantee an expected utility of 0 and none could unilaterally improve it by deviating to a different strategy.*

Connecting Mathematical Programming and Game Theory. Until the famous book by von Neumann and Morgenstern in 1944 [130], there were almost no papers about game theory, except for the contributions of Borel in the early 1920's [14–16] and von Neumann in 1928 [127] and 1937 [128]. It was in the fall of 1947 that von Neumann connected linear programming with games [34].

The observation of the players' best strategies presented in the Example 2.3.1 is in fact an application of von Neumann's min-max Theorem for two-player *zero-sum* games, *i.e.*, games where the sum of the players' utilities for each profile of strategies is zero

$$\sum_{p \in M} \Pi^p(x) = 0 \quad \forall x \in X.$$

Theorem 2.3.3 (von Neumann's min-max Theorem)**.** *Consider a two-player finite game with $M = \{1,2\}$, $X^1 = \{1,2,\ldots,n_1\}$ and $X^2 = \{1,2,\ldots,n_2\}$. Let the game be a zero-sum game, i.e, $\Pi^1(i,j) = -\Pi^2(i,j)$. Then, there are probability distributions q^1 and q^2 (i.e., $\sum_{i=1}^{n_1} q_i^1 = 1$, $\sum_{j=1}^{n_2} q_j^2 = 1$, $q_i^1 \geq 0$, $q_j^2 \geq 0$), satisfying*

$$\max_{q^1} \min_{q^2|q^1} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \Pi^1(i,j) q_i^1 q_j^2 = \min_{q^2} \max_{q^1|q^2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \Pi^1(i,j) q_i^1 q_j^2 \tag{2.3.5a}$$

$$\Leftrightarrow \max_{q^1} \min_{j} \sum_{i=1}^{n_1} \Pi^1(i,j) q_i^1 = \min_{q^2} \max_{i} \sum_{j=1}^{n_2} \Pi^1(i,j) q_j^2 \tag{2.3.5b}$$

where $q^1|q^2$ is to be read q^1 given q^2.

Theorem 2.3.3 allows to find the equilibria strategies for two-player zero-sum finite games through linear programming. The right hand side of equation (2.3.5b) is equivalent to solving

$$\begin{aligned}
\min_{q^2,v} \quad & v && (2.3.6a)\\
\text{s. t.} \quad & \Pi^1(1,1)q_1^2 + \Pi^1(1,2)q_2^2 + \ldots + \Pi^1(1,n_2)q_{n_2}^2 \leq v && (2.3.6b)\\
& \Pi^1(2,1)q_1^2 + \Pi^1(2,2)q_2^2 + \ldots + \Pi^1(2,n_2)q_{n_2}^2 \leq v && (2.3.6c)\\
& \vdots && (2.3.6d)\\
& \Pi^1(n_1,1)q_1^2 + \Pi^1(n_1,2)q_2^2 + \ldots + \Pi^1(n_1,n_2)q_{n_2}^2 \leq v && (2.3.6e)\\
& q_1^2 + q_2^2 + \ldots + q_{n_2}^2 = 1 && (2.3.6f)\\
& q_1^2 \geq 0, \quad q_2^2 \geq 0, \quad \ldots, \quad q_{n_2}^2 \geq 0 && (2.3.6g)
\end{aligned}$$

and the associated dual optimal solution gives the q^1 of the min-max Theorem 2.3.3 (recall the duality results of Section 2.2).

This was the first relationship between linear programming and game theory pointed out. However, for non two-players zero-sum finite games, von Neumann theorem does not necessarily hold, and alternative ways of computing "rational" strategies are required.

Integer Programming Games. Next, we define the particular representation of X^p characterizing integer programming games. Based on the definition presented by Köppe *et al.* [77], we define an integer programming game (IPG) as a non-cooperative game where the feasible set of strategies for each player p is characterized through linear inequalities and integer requirements on player p's decision variables

$$X^p = \{x^p \in \mathbb{R}^{n_p} : A^p x^p \leq b^p, x_i^p \in \mathbb{N} \text{ for } i = 1, \dots, B_p\} \tag{2.3.7}$$

with $B_p \leq n_p$. In this thesis, we will restrict our attention to IPGs. The example below is an IPG.

Example 2.3.4. *Consider Example 2.3.2, but now include set-up costs: whenever a firm $p \in M$ produces a positive quantity, $x^p > 0$, a fixed cost F_p must be paid. Then, the utility of firm $p \in M$ becomes*

$$\Pi^p(x^A, x^B) = \left(a - b(x^A + x^B)\right) x^p - C_p x^p - F_p y^p$$

and the feasible set of strategies is $X^p = \{(x^p, y^p) : y^p \in \{0,1\}, 0 \leq x^p \leq W_p y^p\}$, that is, the quantity x^p produced by firm p must be non negative, cannot exceed the production capacity and whenever $x^p > 0$, the set-up cost is paid ($y^p = 1$).

Remark: Note that Example 2.3.4 is also a continuous game, because each player set of strategies is bounded and thus, a compact metric space. Example 2.3.1 is in the so-called normal-form representation. However, game 2.3.1 could easily be formulated as an IPG by associating a binary variable to each player pure strategy (which would model the strategy selected), adding a constraint summing the decision variables up to one (this ensures that one strategy is selected) and formulating the players' objective functions according to the utility values for combinations of the binary variables. In fact, this transformation applied to any normal-form game leads to an equivalent IPG. Figure 2.3.1 depicts the relation between the aforementioned game classes; we highlight that an IPG contains all finite games and, if X is bounded and utility functions are continuous, it is a continuous game; as in this thesis we restrict our attention to quadratic utility functions, the continuity of the utilities is guaranteed.

In the next two sections, we distinguish between sequential games with two rounds (Section 2.3.1), Stackelberg competition, and simultaneous games (Section 2.3.2). Although in both types of games each player goal is to maximize her utility, the approach to find a solution significantly varies.

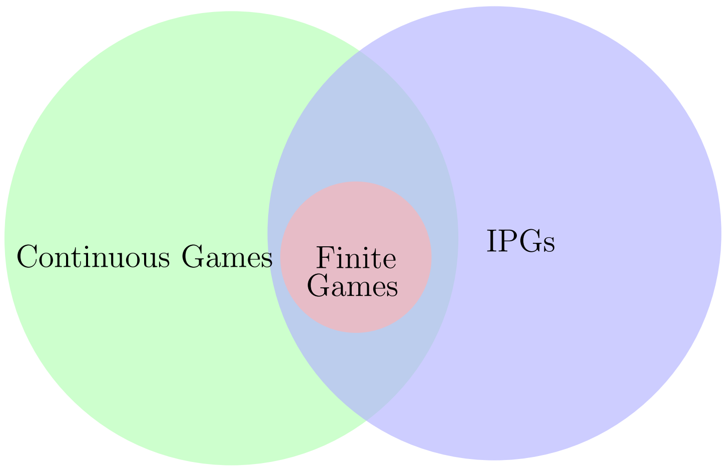


Figure 2.3.1: Games classes.

2.3.1 Stackelberg Competition

Basic Definitions. In the Stackelberg competition [131], also known as bilevel programming (BP), there are two players that play two rounds. In the first round the so-called leader takes action, and in the second round the other player (called the follower) observes the leader's decision and selects her strategy. The decision variables are split into two groups, those that are controlled by the leader (on the upper level) and those controlled by the follower (on the lower level). Both decision makers have an objective function (utility) of their own and a set of constraints on their variables that define the set of feasible strategies. Furthermore, there are coupling constraints that connect the decision variables of leader and follower. Let the leader and follower decision vectors be x and y, respectively. A mathematical formulation for a bilevel problem is

$$\begin{aligned}
\max_{x,y} \quad & \Pi^l(x,y) && \text{(2.3.8a)}\\
\text{s. t.} \quad & x \in X \subseteq \mathbb{R}^{n_x} && \text{(2.3.8b)}\\
& \text{where } y \text{ solves the follower's problem} && \text{(2.3.8c)}\\
& \max_{y} \quad \Pi^f(x,y) && \text{(2.3.8d)}\\
& \text{s. t.} \quad y \in Y(x) \subseteq \mathbb{R}^{n_y}, && \text{(2.3.8e)}
\end{aligned}$$

where the objective (2.3.8a) is the leader's utility who controls the decision vector x, and the objective (2.3.8d) is the follower's utility who controls the decision vector y. Note that problem (2.3.8) does not fully determine the follower's behavior: there might be many follower's optimal solutions for a fixed leader decision that yield different objective values for the leader. Which one will the follower choose? In the *optimistic* scenario the follower always picks an optimal solution that yields the best objective value for the leader, and in the *pessimistic* scenario she picks a solution that yields the worst objective value for the leader. In Section 3.3, we tackle a BP characterized by the fact that both leader and follower share the same objective function (although with different optimization directions) and this distinction about optimistic and pessimistic scenarios is not needed; when leader and follower share the same objective function but the leader aims to minimize, and the follower to maximize it, the problem is called *min-max programming* problem.

A Stackelberg equilibrium is an optimal solution for the BP (2.3.8); note that the objective function of an BP is the leader's utility function. Let us give a classical example of a game with two rounds, in order to clarify the concepts presented so far.

Example 2.3.5. *The classical Stackelberg competition is modeled according to Example 2.3.2, but without production capacity limitations. Let firm A be the leader and firm B the follower. Thus, we aim at finding the solution (Stackelberg equilibrium) for*

$$\max_{x^A} \quad \left(a - b(x^A + x^B)\right) x^A - C_A x^A \tag{2.3.9a}$$

$$s.\ t. \quad x^A \geq 0 \tag{2.3.9b}$$

$$\text{where } x^B \text{ solves the follower's problem} \tag{2.3.9c}$$

$$\max_{x^B} \quad \left(a - b(x^A + x^B)\right) x^B - C_B x^B \tag{2.3.9d}$$

$$s.\ t. \quad x^B \geq 0. \tag{2.3.9e}$$

Once the leader's strategy x^A is chosen, the follower selects her best reaction (which is easy to compute, since the follower's utility is concave), playing

$$x^B(x^A) = \frac{(a - C_B - bx^A)^+}{2b} \tag{2.3.10}$$

where $\alpha^+ = \max(0, \alpha)$. Then, since we assume that the leader is rational and can predict $x^B(x^A)$, she replaces in her utility function x^B by $x^B(x^A)$ and computes the optimal quantity x^{*A}

$$x^{*A} = \frac{a - 2C_A + C_B}{2b}. \tag{2.3.11}$$

In conclusion, $(x^{*A}, x^B(x^{*A}))$ is the Stackelberg equilibrium or, equivalently, the optimal solution for the bilevel problem (2.3.9).

Interdiction problems (see Israel [68]) are a special type of BP that have received large attention in the research community. These are min-max BPs where for each follower's variable, there is a leader's binary variable and an *interdiction constraint* in the lower level problem that enables the leader to make that follower's variable unavailable. Formally, in an interdiction problem, the follower's constraints (2.3.8e) include a set of interdiction constraints

$$y \leq U^\intercal (1 - x), \tag{2.3.12}$$

where U is a vector column of dimension $n_x = n_y$. In Section 3.3 of this thesis, we focus on an interdiction problem.

Solving Bilevel Problems. When the follower's problem is an LP, through strong duality (recall Properties 2.2.2 and 2.2.3) applied to the follower's optimization problem, one can compute a single level programming problem equivalent to the BP. If the follower's optimization problem is a concave QP then an equivalent single level programming problem can be obtained by replacing her optimization problem by appropriate Karush-Kuhn-Tucker (KKT) conditions (for details in these conditions see Karush [73] and Kuhn and Tucker [79]). The introduction of integer requirements, leading to mixed integer bilevel programs (MIBP), even if restricted to the follower's decision variables, is enough to make the transformation above not valid.

The decision version of a BP asks whether there exists an action of the leader such that for any follower's reaction, the leader's objective value is guaranteed to be at least as good as a predefined bound. The complexity class Σ_2^p is the natural hotbed for bilevel problems that are built on top of NP-complete single-level problems. If a problem is Σ_2^p-complete, there is no way of formulating it as a single-level integer problem of polynomial size unless the polynomial hierarchy collapses (a highly unlikely event which would cause a revolution in complexity theory, quite comparable to the revolution that would be caused by a proof that P=NP). In fact, even for the simplest MIBP with the leader's problem as an LP, the problem is Σ_2^p-complete (as is the case for the problem of Dempe and Richter [40] which we prove to be Σ_2^p-complete in Section 3.2.1).

It is a well-known fact in MIBP research that the techniques that successfully work on (classical, single-level) MIPs are not straightforward to generalize to the bilevel case. Indeed, the BP obtained by relaxing the integrality restrictions does not provide an upper bound on the maximization version of the original problem, and even if its solution is integral, it is not necessarily optimal for the original problem. This is illustrated in the following example.

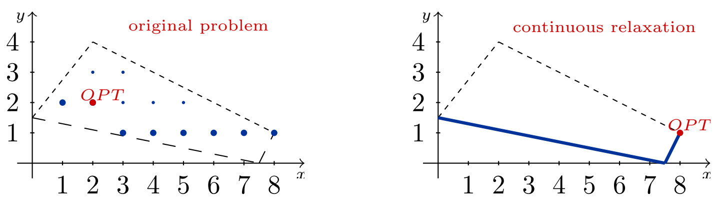


Figure 2.3.2: Blue represents the feasible region for Problem (2.3.13) and associated continuous relaxation.

Example 2.3.6 (Example from Moore and Bard [93]). *Consider the BP*

$$\begin{aligned}
\min_{x} \quad & -x-10y \\
s.\ t. \quad & x \in Z_+ \\
& \textit{where } y \textit{ is optimal to} \\
& \min_{y} \quad y \\
& s.\ t. \quad 5x-4y \geq -6 \\
& \qquad -x-2y \geq -10 \\
& \qquad -2x+y \geq -15 \\
& \qquad 2x+10y \geq 15 \\
& \qquad y \in \mathbb{Z}_+.
\end{aligned}$$

Observe Figure 2.3.2 which depicts the feasible region to our problem and to the associated continuous relaxation. An optimal solution is $(x^*, y^*) = (2,2)$ *with objective value (leader's utility) equal to* -22. *An optimal solution for the continuous relaxation is attained when* $(\hat{x}, \hat{y}) = (8,1)$ *with objective value equal to -18. Observe that two important properties used to prune the search space in the branch-and-bound scheme to MIPs do not hold in this case. Namely,*

- *the continuous relaxation optimal value does not provide a lower bound to problem* (2.3.13)*;*
- *the solution for the continuous relaxation satisfies the integrality constraints, however, it is not optimal to problem* (2.3.13)*.*

The only property that holds is the following: if the continuous relaxation for an MIBP is infeasible, then the MIBP itself is infeasible.

Next, we review the literature about Stackelberg competition. Note that the class of Stackelberg competitions we aim to tackle is in combinatorial optimization, and thus, is more studied in the context of mathematical programming. For this reason, the term bilevel programming will be used more often.

2.3.1.1 Previous Work

Generally speaking, multilevel optimization programs are extremely difficult from the computational point of view and cannot be expressed in terms of classical integer programming (which can only handle a single level of optimization). A ground-breaking paper by Jeroslow [69] established that several multilevel problems are complete for various levels of the polynomial hierarchy in computational complexity theory. Further hardness results for a broad range of families of multilevel optimization problems are due to Deng [43] and Dudás, Klinz and Woeginger [46].

The optimization literature only contains a handful of results on the solution of general MIBPs. Moore and Bard [93] adapt the classical branch-and-bound scheme for MIPs to MIBPs, and propose a number of simple heuristics. Their approach is fairly basic and can only handle small instances, with up to 20 integer variables. The main reason for the lack of success with this adaptation is the failure of two of the pruning criteria, used for solving MIPs, which do not hold for MIBPs (as Example 2.3.1 highlights). The challenge is in computing upper bounds (maximization version) with good quality to MIBPs. The usual approach is to solve the so-called *high-point problem*, which consists in dropping the follower's optimality condition and integrality constraints. This may provide good upper bounds for problems in which the leader's objective function takes (in some way) into account the follower's reaction. Unfortunately, for min-max problems, the lower bound provided by solving the associated high-point problem is generally considerably far from the optimum, so that the branch-and-bound tree is likely to be extremely big (this is pointed out in the survey by Ben-Ayed [8]). Moore and Bard [93] procedure, applied to a maximization version of an MIBP, in the root of the branch-and-bound tree, solves the high point problem and proceeds as in the MIP approach by branching in order to satisfy the integrality requirement and generating two subproblems; for each promising node (integer solution) it solves the corresponding continuous relaxation of the bilevel program; whenever an integer solution is computed, it verifies its bilevel feasibility by solving the lower level problem for the fixed leader's decision, to obtain a lower bound (because a feasible solution is obtained).

The first significant advances to the MIBP branch-and-bound scheme are due to DeNegre's dissertation [41], which added a number of interesting ingredients, leading to a branch-and-cut scheme, and in particular considered the so-called interdiction constraints. DeNegre also provides some heuristics to improve the solutions obtained through the branch-and-cut method.

Hemmati *et al.* [65] consider a more general bilevel interdiction problem on networks. An effective cutting plane algorithm in the spirit of the one described in Section 3.3.1 is proposed and enhanced with valid inequalities that are specific to the considered problem on networks. Links to the general interdiction literature, especially from a homeland security perspective, are provided by Smith [118] and Smith and Lim [119].

For an overview of this area, we refer the reader to the book edited by Dempe [38], and also to the annotated bibliographies of Vicente and Calamai [132], Dempe [39], and

Colson, Marcotte and Savard [25]. For a comprehensive survey on solution methodologies for MIBPs, we refer the reader to Saharidis *et al.* [113].

In this thesis, we start by classifying in terms of complexity three “simple” MIBP’s which have a formulation based on a natural generalization of the knapsack problem with two levels. As expected for combinatorial optimization problems with two levels, these bilevel knapsack variants are proven to be Σ_2^p-complete. For one of the bilevel knapsack variants with interdiction constraints, we propose a novel algorithmic approach that takes advantage from the fact that the problem is *(i)* min-max optimization and *(ii)* has interdiction constraints. Therefore, the algorithmic methodology employed presents interesting features for an adaptation to solve general interdiction problems.

2.3.2 Simultaneous Games

Basic Definitions. In a simultaneous game, players strategies are revealed at the same time. The solution concept that will be used is the famous *Nash equilibrium.* A Nash equilibrium (NE) is a profile of strategies $\sigma \in \Delta$ such that for each player $p \in M$ the following inequalities hold:

$$\Pi^p(\sigma) = \Pi^p(\sigma^p, \sigma^{-p}) \geq \Pi^p(x^p, \sigma^{-p}) \quad \forall x^p \in X^p. \tag{2.3.14}$$

The equilibria inequalities (2.3.14) reflect the nonexistence of incentive for each player p to deviate unilaterally to a strategy different from σ^p because there is no increase in the utility value. In other words, each player p best reaction to σ^{-p} is σ^p.

Next, we present two examples of simultaneous IPGs: a finite game (Example 2.3.7) and a continuous game (Example 2.3.8), as well as the computation of their equilibria.

Example 2.3.7 (Prisoner’s dilemma)**.** *The prisoner’s dilemma is a well-known game theory example. The players are two prisoners of a criminal gang that are suspected to have committed a crime. Due to the lack of evidence to convict either, the police needs them to testify against each other and, thus, interrogates them in separate rooms. Each of the suspects has two possible strategies: (*Defect*) testify against the other, which results in receiving a reward; and (*Cooperate*) keep silence. The bimatrix of Table 2.2 displays the four possible pure outcomes for the game with the players utilities: if both cooperate, they are released (both get 1); if only one testifies against (*Defect*), she is released and collects a reward (gets 2), while the other goes to the prison (gets -1); if both testify against, both go to prison, but they will still collect a reward for testifying. Observe that for each player, the strategy “*Defect*” strictly dominates “*Cooperate*”. Thus, (*Defect, Defect*) is the unique equilibrium of the game.*

		Prisoner II	
		Cooperate	Defect
Prisoner I	Cooperate	(1,1)	(-1,2)
	Defect	(2,-1)	(0,0)

Table 2.2: Prisoner's dilemma

Example 2.3.8 (Cournot duopoly)**.** *One of the earliest examples of game analysis is due to Antoine A. Cournot in his model of duopoly [29]. We present the classical formulation, which is modeled through Example 2.3.2 but without production capacity limitations; the players play simultaneously. Each player $p \in \{A, B\}$ aims to solve*

$$\max_{x^p} \quad \left(a - b(x^A + x^B)\right) x^p - C_p x^p \tag{2.3.15a}$$

$$\textit{subject to} \quad x^p \geq 0. \tag{2.3.15b}$$

In order to find the players' optimal solutions, apply derivatives on their objective functions and find their zeros (note that this is valid because both objective functions are concave). In this way, we get the equilibrium $(x^A, x^B) = \left(\frac{a+C_B-2C_A}{3b}, \frac{a+C_A-2C_B}{3b}\right)$.

Computing pure NE. A game is *potential* [92] if there is a real-valued function $\Phi : X \longrightarrow \mathbb{R}$ such that its value increases strictly when a player unilaterally switches to a strategy that strictly increases her utility. A potential function is *exact* when this increase is equal to the player's utility increase. Potential games are guaranteed to have pure NE.

Lemma 2.3.9 (Monderer and Shapley [92])**.** *The maximum of a potential function for a game is a pure Nash equilibrium.*

Proof. By contradiction, suppose that there is a profile of strategies for which the potential function attains its maximum value and it is not an NE. Then, at least one of the players would have advantage in switching to a new strategy, which would imply that the potential function would strictly increase its value in this new profile. However, that contradicts the fact that the previous profile was a potential function optimum. □

The proof of Lemma 2.3.9 suggests a method to compute an equilibrium. **Tâtonnement process or adjustment process:** assign a profile of strategies for the players; while there is a player with incentive to unilaterally deviate from the current profile of strategies, replace her strategy by one that improves that player's utility; otherwise, an equilibrium was found. If a game is potential, its potential function value at a profile of strategies strictly increases as this process iterates to new profiles of strategies. If the potential function has a maximum, this process converges to a pure NE.

There is no general procedure to decide if a game is potential and to compute a potential function of it. However, many games satisfy the *bilateral symmetric interaction* property, which is sufficient for a game to be potential. If in a game the utility function of each player $p \in M$ has the form

$$\Pi^p(x) = \sum_{i \in M} w_{p,i}(x^p, x^i), \tag{2.3.16}$$

where $w_{p,i}(x^p, x^i)$ is a function with $w_{p,i}(x^p, x^i) = w_{i,p}(x^p, x^i)$ for all $i \in M$ then, the bilateral symmetric interaction is satisfied. A bilateral symmetric interaction game is one where utility functions can be decomposed into symmetric interaction terms, which are bilaterally determined together with the term depending only on the players' own strategy. The Cournot Competition of Example 2.3.8 satisfies the bilateral symmetric interaction game property: $w_{A,B} = w_{B,A} = -bx^Ax^B$, $w_{A,A} = (a - bx^A - C_A)x^A$ and $w_{B,B} = (a - bx^B - C_B)x^B$.

Proposition 2.3.10 (Ui [123]). *A bilateral symmetric interaction game is potential. A potential function is*

$$\Phi(x) = \frac{1}{2} \sum_{i \in M} \sum_{j \in M} w_{i,j}(x^i, x^j), \tag{2.3.17}$$

where each player p's utility function has the form (2.3.16).

Proof. It is sufficient to prove that the difference in the potential function value when a player p unilaterally deviates is equal to that player's difference in the utility value

$$\begin{aligned}
\Phi(x) - \Phi(x^{-p}, \hat{x}^p) =& \frac{1}{2} \sum_{i \in M} \sum_{j \in M} w_{i,j}(x^i, x^j) - \frac{1}{2} \sum_{i \in M\setminus\{p\}} \sum_{j \in M\setminus\{p\}} w_{i,j}(x^i, x^j) && \text{(2.3.18a)}\\
& - \frac{1}{2} \sum_{j \in M} w_{p,j}(\hat{x}^p, x^j) - \frac{1}{2} \sum_{i \in M} w_{i,p}(x^i, \hat{x}^p) && \text{(2.3.18b)}\\
=& \sum_{j \in M} w_{p,j}(x^p, x^j) - \sum_{j \in M} w_{p,j}(\hat{x}^p, x^j) && \text{(2.3.18c)}\\
=& \Pi^p(x) - \Pi^p(\hat{x}^p, x^{-p}). && \text{(2.3.18d)}
\end{aligned}$$

□

The Cournot Competition of Example 2.3.8 is a potential game where a potential function is $\Phi(x^A, x^B) = (a - bx^A - C_A)x^A + (a - bx^B - C_B)x^B - bx^Ax^B$.

Computing NE for finite games. It has been argued that pure NE are more natural game outcomes than mixed equilibria, given their simplicity, the difficulty of computing equilibria and, thus, the players' limitations in determining them. However, games might fail to have pure equilibria. For example, in the famous children's game "rock-scissors-paper" (see Example 2.3.1) the only equilibrium is to uniformly randomize over the 3 strategies. Therefore, it is important to consider mixed equilibria when analyzing a game. Furthermore, a game may have no equilibria. Nash [94] proved that a game possesses an equilibrium if the set of strategies is finite.

Theorem 2.3.11 (Nash [94]). *A finite game has an equilibrium.*

The existence proof of this theorem does not provide a polynomial time algorithm for determining an equilibrium. There are general algorithms to compute NE for finite games, but they fail to be polynomial. See Nisan *et al.* [96] for comprehensive material in algorithmic game theory. In fact, Daskalakis *et al.* [35] proved that finding an NE for finite games is PPAD-complete (this is true even with only two players, see Chen *et al.* [23]).

The algorithms for finite games rely heavily on the following result.

Proposition 2.3.12. *Consider a finite game and a profile of strategies $\sigma \in \Delta$. Then, σ^p is player p best reaction to σ^{-p} if and only if for all $\hat{x}^p \in X^p$*

$$\sigma^p(\hat{x}^p) > 0 \quad \textit{implies} \quad \Pi^p(\hat{x}^p, \sigma^{-p}) = u^p, \tag{2.3.19}$$

where $u^p = \max_{x^p \in X^p} \Pi^p(x^p, \sigma^{-p})$ and $\sigma^p(x^p)$ is the probability assigned to the pure strategy x^p.

Proof. Note that

$$\Pi^p(\sigma) \leq u^p, \tag{2.3.20}$$

since $\Pi^p(\sigma)$ is a convex combination: $\Pi^p(\sigma) = \sum_{x^p \in X^p} \sigma^p(x^p)\Pi^p(x^p, \sigma^{-p})$. Therefore, $\Pi^p(\sigma) = u^p$ if and only if $\sigma^p(\hat{x}^p) > 0$ implies $\Pi^p(\hat{x}^p, \sigma^{-p}) = u^p$. □

By Proposition 2.3.12, σ^p is a player p's best reaction to the opponents strategies σ^{-p} if and only if all player p's pure strategies with positive probability assigned in σ^p are equally good (pure best responses) to σ^{-p}.

The *support of a strategy* $\sigma^p \in \Delta^p$, denoted as $\text{supp}(\sigma^p)$, is the set of all strategies $x^p \in X^p$ such that $\sigma^p(x^p) > 0$. Proposition 2.3.12 allows to reduce the problem of computing an equilibrium σ to determining its support strategies and then, computing its probabilities by solving the *Feasibility Problem* depicted in Figure 2.3.3. Constraints (2.3.21a) ensure that the strategies played with positive probability by player p have equal utility value

Feasibility Problem

Input: for all $p \in M$ a set of strategies A^p to be the support

Output: NE σ, if there exists both a mixed strategy profile σ and a utility value u^p for all $p \in M$ such that

$$u^p = \Pi^p(\hat{x}^p, \sigma^{-p}) \quad \forall p, \quad \forall \hat{x}^p \in A^p \tag{2.3.21a}$$

$$u^p \geq \Pi^p(x^p, \sigma^{-p}) \quad \forall p, \quad \forall x^p \in X^p \tag{2.3.21b}$$

$$\sum_{x^p \in A^p} \sigma^p(x^p) = 1 \quad \forall p \tag{2.3.21c}$$

$$\sigma^p(x^p) \geq 0 \quad \forall p, \quad \forall x^p \in A^p \tag{2.3.21d}$$

$$\sigma^p(x^p) = 0 \quad \forall p, \quad \forall x^p \in X^p - A^p, \tag{2.3.21e}$$

where $\Pi^p(\hat{x}^p, \sigma^{-p}) = \sum_{x \in A^{-p}} \Pi^p(\hat{x}^p, x) \prod_{k \in M - \{p\}} \sigma^k(x^k)$.

Figure 2.3.3: Feasibility Problem for finite games.

(Proposition 2.3.12); Constraints (2.3.21b) are the Nash equilibria conditions (2.3.14); Constraints (2.3.21c) to (2.3.21e) guarantee that σ^p is a probability distribution for each $p \in M$.

In the literature there are many algorithmic approaches for computing equilibria of finite games. These methods essentially differ in the way of enumerating supports for the equilibria. One of the approaches to compute NE for finite games that performs better in practice is PNS, developed by Porter, Nudelman and Shoham [107]. PNS enumerates support sets and solves the associate Feasibility Problem until it is feasible and thus, an NE of it was found. In order to possibly reduce the support enumeration search space, an additional step eliminating conditionally dominated strategies from being in the supports is included, decreasing the number of Feasibility Problems to be solved. In this thesis, our goal is not restricted to finite games; we refer the reader interested on finite games to the surveys and state-of-the-art algorithms collected in [133]. Note that the algorithms for finite games when applied to IPG imply the explicit enumeration of all profiles of strategies, which can be exponential in the size of the game representation or even unsuitable when the set of feasible strategies is uncountable.

Computing NE for IPG. For IPG, if there is at least a player p for whom not all variables are bounded, or there are continuous variables (*i.e.*, $B_p < n_p$), Nash's Theorem 2.3.11 does not apply, since the set of strategies becomes infinite. In this case,

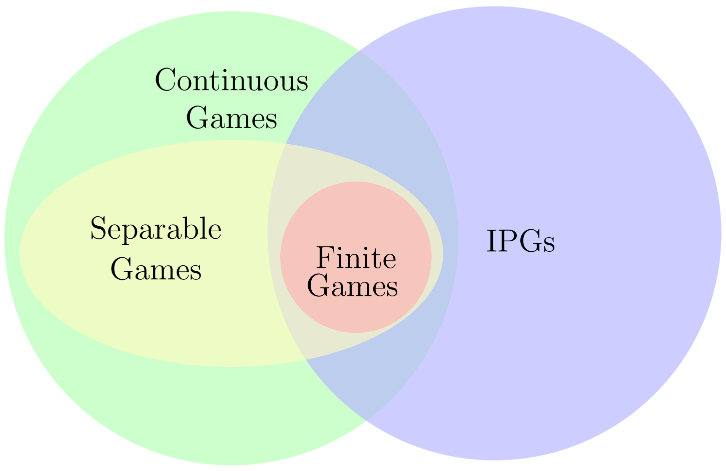


Figure 2.3.4: Games classes.

the most common existence theorem used is the following.

Theorem 2.3.13 (Glicksberg [58]). *Every continuous game has a Nash equilibrium.*

Therefore, an IPG is guaranteed to have an equilibrium if the players' objective functions are continuous and the set of strategies X is bounded (since IPG becomes a continuous game).

Let the set of players be $M = \{1, \dots, m\}$. A *separable game* is a continuous game with utility functions $\Pi^p : X \longrightarrow \mathbb{R}$ taking the form

$$\Pi^p(x) = \sum_{j_1=1}^{k_1} \dots \sum_{j_m=1}^{k_m} a^p_{j_1 \dots j_m} f^1_{j_1}(x^1) \dots f^m_{j_m}(x^m), \tag{2.3.22}$$

where $a^p_{j_1 \dots j_m} \in \mathbb{R}$ and the $f^p_j : X^p \longrightarrow \mathbb{R}$ are continuous. See Figure 2.3.4 for a clear picture of the games classes relations. Separable games have the following property.

Theorem 2.3.14 (Stein *et al.* [120]). *In a separable game, for every mixed strategy σ^p there is a finitely supported mixed strategy τ^p such that $f^p_j(\sigma^p) = f^p_j(\tau^p)$ for all j and $|supp(\tau^p)| \leq k_p + 1$. Moreover, if σ^p is countably-supported τ^p can be chosen with $supp(\tau^p) \subseteq supp(\sigma^p)$.*[2]

Combining Stein *et al.* and Glicksberg's Theorems 2.3.14 and 2.3.13, Stein *et al.* [120] conclude the following:

[2]Extend f^p to the space of all finite-valued signed measures in X^p:

$$f^p(\sigma^p) = \left(\int f^p_1(x^p) d\sigma^p, \dots, \int f^p_{k_p}(x^p) d\sigma^p \right).$$

Corollary 2.3.15. *Every separable game has a Nash equilibrium. Moreover, for every Nash equilibrium σ there is a Nash equilibrium τ such that each player p mixes among at most $k_p + 1$ pure strategies and $\Pi^p(\sigma) = \Pi^p(\tau)$.*

Therefore, in case an IPG is a separable game, the search for an equilibrium can be reduced to finding a finite set of strategies for each player p of size at most $k_p + 1$ and check through the Feasibility Problem (2.3.21) if they are the support on an equilibrium. The utility functions of the games to be analyzed in this thesis are linear or quadratic and, thus, the utilities are written in the form (2.3.22).

2.3.2.1 Previous Work

The literature on IPG is scarce and often focused on the particular structure of specific games. Moreover, typically, the analysis is restricted to pure Nash equilibria.

Kostreva [78] provides the first attempt to address the computation of pure NE to IPG. Kostreva [78] describes a theoretical approach to tackle IPG for which players' utility functions and constraints are polynomial, and integer variables are required to be binary. For each player's binary variable x the penalty $Mx(1-x)$ is added to her utility[3], where M is a suitably large positive number. Then, the Karush-Kuhn-Trucker (KKT) conditions are applied to each player's continuous relaxation 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 [136] or Gröbner basis [30]. Additionally, it must be verified which of the system's solutions are equilibria[4], implying solving each player's best response problem and resulting in long computational times. Gabriel *et al.* [54] developed an optimization model for which the optimal solution is a pure Nash equilibrium of a game that approximates an IPG with concave utilities when integer constraints are relaxed. In [54], the players' continuous relaxations 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 integer requirements. On the few experimental results presented, this approach leads to the computation of a pure NE for the original game. However, there is neither theoretical nor computational evidence showing the applicability of these ideas to the general case. Deciding the existence of pure equilibria in games with an exponential number of actions per player with general utility functions (expressed as Turing machines or Boolean circuits) was proven to be Σ_2^p-complete in Álvarez *et al.* [2] and Schoenebeck *et al.* [115].

[3]Note that the penalty $Mx(1-x)$ makes a player's best reaction problem non-concave.

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

Lee and Baldick [81] study the computation of mixed NE 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 normal-form (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 an NE will be. Stein *et al.* [120] restrict their attention to separable games. The authors are able to provide bounds on the cardinality of the support of equilibrium strategies (Theorem 2.3.14) and present a polynomial-time algorithm for computing ϵ-equilibria of two-player separable games with fixed strategy spaces and utility functions satisfying the Hölder condition.

We expand the class of problems introduced by Köppe, Ryan and Queyranne [77] as integer programming games (recall the strategy set formulation (2.3.7)); the difference is in the fact that we allow continuous decision variables in addition to integer variables. The utility functions in [77] are differences of piecewise-linear concave functions. This is not the case for our models; *e.g.*, for the IPG studied in Section 4.3, each player's objective function is quadratic in her decision variables. Moreover, since generating functions of integer points inside of polytopes (bounded polyhedron) are used to study pure NE, their approach would only be suitable if the players' strategy sets are countable (which is not the case when there are continuous variables). Finally, the application of Köppe, Ryan and Queyranne's results rely on computational implementations that are still in preliminary stage, although theoretically it 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).

As we have seen in the previous section, the class of IPGs contains finite games for which it has been proven that computing an equilibrium is PPAD-complete. Adding to this, the fact that deciding if a profile of strategies is an equilibrium is itself an NP-complete problem (since it implies to solve each player best reaction problem (2.3.4), which can, in turn, be an IP) reveals the difficulty of tackling this class of problems.

In this thesis, the first simultaneous IPG that we present has a special structure associated with the classical knapsack problem that enables to reduce the computation of pure equilibria to solving a two-objective optimization problem. Then, we analyze a game modeling the two-player kidney exchange markets for which our generalization of the maximum matching theory enable us to efficiently compute an equilibrium in which the players agree to play. The last particular game to be analyzed generalizes the classical Cournot competition model by merging it with the lot-sizing problem, and illustrates the difficulties of computing equilibria. Finally, in Section 4.4, general simultaneous IPGs with quadratic objective functions are studied. Note that IPGs may have no equilibria;

take, for instance, the case with a single player in which her optimization problem is unbounded. To the best of our knowledge, there is no previous study classifying the complexity of deciding if an IPG has an equilibrium. We prove that it is a Σ_2^p-complete problem, even in the case with only two players and linear utility functions. We also prove that deciding the existence of pure NE is Σ_2^p-complete. We conclude with an algorithm and the associated computational validation for simultaneous IPGs.

2.3.3 Game Theory Solvers

As far as we know, MibS [109] is the only solver available to tackle general MIBP's. Essentially, the generality of the algorithmic approaches to solve an MIBP reduce to determining optimal solutions for series of LP's and/or MIP's, and thus, the solvers mentioned in Section 2.2 are integrated in these algorithms framework.

To the best of our knowledge, there are no general solvers for IPG. Thus, as mentioned in the previous section, in the literature, either in the games analyzed there are no integrality requirements to be satisfied or these are somehow taken into account in the players' objective functions. This allows the problem of computing an equilibrium to be reduced to solving a constrained programming problem for which the solvers in Section 2.2 might be applied. However, the resulting constrained programming problems can be hard to solve and only enable the computation of pure equilibria.

Alternatively, IPG equilibria have been approximated by enumerating part of the players' feasible strategies and solving the resulting normal-form game (finite game). The most well-known and up-to-date game theory solver for normal-form games is the open-source Gambit [90], which results from a project initiated in the mid-1980's by Richard McKelvey at the California Institute of Technology. Gambit includes famous algorithmic approaches, like Lemke-Howson [82], Govindan-Wilson [61, 62], Simplicial Subdivision [126] and PNS [107]. In resemblance with the mathematical programming instances, there is a computational testbed for normal-form games: GAMUT [97].

Chapter 3

Stackelberg Competition: Bilevel Knapsack

[1]

Bilevel programming includes the classical single-level programming and therefore, it is expected to be more intricate. For this reason, in this chapter, we concentrate in studying the simplest mixed integar bilevel programming problems that one could devise.

The knapsack problem has been a fundamental "playground" for understanding single-level programming. Thus, this methodological motivation together with the simplicity of the KP model, lead us to study natural generalizations of KP to bilevel programming which are formulated in Section 3.1. In particular, we study these problems computational complexity (Section 3.2) and suggest a novel viable algorithmic approach for one of the bilevel knapsack variants that have seldom address in the literature (Section 3.3).

3.1 Bilevel Knapsack Variants

Over the last few years, a variety of authors has studied certain bilevel variants of the knapsack problem. Dempe and Richter [40] considered the variant where the leader controls the weight capacity of the knapsack, and where the follower decides which items are packed into the knapsack (Section 3.1.1). Mansi *et al.* [85] consider a bilevel knapsack variant where the item set is split into two parts, one of which is controlled by the leader and one controlled by the follower (Section 3.1.2). DeNegre [41] suggests yet another variant, where both players have a knapsack of their own; the follower can only choose from those items that the leader did not pack (Section 3.1.3). This section gives precise definitions for these variants and provides further information on them.

[1]The results of this chapter appears in:
A. Caprara, M. Carvalho, A. Lodi, G. J. Woeginger. A Study on the Computational Complexity of the Bilevel Knapsack Problem, SIAM Journal on Optimization 24(2), 2014, 823-838.
A. Caprara, M. Carvalho, A. Lodi, G. J. Woeginger. Bilevel knapsack with interdiction constraints, INFORMS Journal on Computing, Volume 28, Issue 2, Spring 2016, 319-333.

Throughout, we use a_i, a'_i, b_i, b'_i, c_i, c'_i and A, B, C, C' to denote item weights, cost coefficients, upper bounds, and lower bounds. All these numbers are assumed to be non-negative integers (or rationals). As usual, we will sometimes use the notation $a(I) = \sum_{i\in I} a_i$ for an index set I, and $a(x) = \sum_i a_i x_i$ for a 0-1 vector x.

3.1.1 The Dempe-Richter (DeRi) variant

The first occurrence of a bilevel knapsack problem in the optimization literature seems to be due to Dempe and Richter [40]. In their problem variant DeRi, as depicted in Figure 3.1.1, the leader controls the capacity x of the knapsack while the follower controls all items and decides which of them are packed into the knapsack. The objective function of the leader depends on the knapsack capacity x as well as on the packed items, whereas the objective function of the follower solely depends on the packed items.

$$\max_{x\in\mathbb{N}} \; f_1(x,y) \;=\; A\,x + \sum_{i=1}^{n} a_i y_i \tag{3.1.1a}$$

$$\text{s. t.} \quad C \;\le\; x \;\le\; C' \tag{3.1.1b}$$

where $y_1,\ldots,y_n$ solves the follower's problem

$$\max_{y\in\{0,1\}^n} \sum_{i=1}^{n} b_i y_i \quad \text{s.t.} \; \sum_{i=1}^{n} b_i y_i \le x \tag{3.1.1c}$$

Figure 3.1.1: The bilevel knapsack problem DeRi.

All decision variables in this bilevel programming problem are integers; the knapsack capacity x is integer, and the variables $y_1, \ldots, y_n \in \{0,1\}$ encode whether item i is packed into the knapsack ($y_i = 1$) or not ($y_i = 0$). We note that in the original model in [40] the knapsack capacity x is continuous; one nasty consequence of this continuous knapsack capacity is that the problem (3.1.1a)–(3.1.1c) may fail to have an optimal solution (Example 3.1.1 illustrates such case). The computational complexity of the problem remains the same, no matter whether x is integral or continuous.

Example 3.1.1. *Consider the DeRi instance with $n = 2$, $A = 1$, $C = 2$, $C' = 3$, $a_1 = 3$, $a_2 = 1$, $b_1 = 2$ and $b_2 = 3$. If $x < 3$, the follower only has the feasible strategy $y = (1,0)$, leading to $f_1(x,(1,0)) = x + 3$ which is greater or equal to 5; if $x = 3$, the follower optimal solution is $y = (0,1)$ which leads to $f_1(3,(0,1)) = 3 + 1 = 4$. It follows that the leader* *would choose x as close as possible to 3 in order to maximize her objective value f_1. This shows that there is no optimal solution for this instance.*

Dempe and Richter [40] discuss approximation algorithms for DeRi, and furthermore design a dynamic programming algorithm that solves variant DeRi in pseudo-polynomial time. Brotcorne, Hanafi and Mansi [17] derive another (simpler) dynamic program with a much better running time. Plyasunov [105] provides conditions under which the problem is non-degenerate and reduces to a series of linear programming problems.

3.1.2 The Mansi-Alves-de-Carvalho-Hanafi (MACH) variant

Mansi *et al.* [85] consider a bilevel knapsack variant where both players pack items into the knapsack. There is a single common knapsack for both players with a prespecified capacity of C. The item set is split into two parts, which are, respectively, controlled by the leader and the follower. The leader starts the game by packing some of her items into the knapsack, and then the follower adds some further items from her set. The objective function of the leader depends on all items packed by leader and follower, whereas the objective function of the follower solely depends on her own items. Figure 3.1.2 specifies the bilevel problem MACH.

$$\max_{x\in\{0,1\}^m} f_2(x,y) \;=\; \sum_{j=1}^{m} a_j x_j + \sum_{i=1}^{n} a'_i y_i \tag{3.1.2a}$$

$$\text{s. t.}\quad y_1,\dots,y_n \text{ solves the follower's problem}$$

$$\max_{y\in\{0,1\}^n} \sum_{i=1}^{n} b'_i y_i \quad \text{s.t.} \quad \sum_{i=1}^{n} c'_i y_i \le C - \sum_{j=1}^{m} c_j x_j \tag{3.1.2b}$$

Figure 3.1.2: The bilevel knapsack problem MACH.

Mansi *et al.* [85] describe several applications of their problem in revenue management, telecommunication, capacity allocation, and transportation. Variant MACH has also been studied in a more general form by Brotcorne, Hanafi and Mansi [18], who reduced the model to one-level in pseudo-polynomial time.

3.1.3 DeNegre (DNeg) variant

DeNegre [41] proposes another bilevel knapsack variant where both players hold their own private knapsacks and choose items from a common item set. First, the leader packs some of the items into her private knapsack, and then the follower picks some of the remaining items and packs them into her private knapsack. The objective of the follower is to maximize the profit of the items in her knapsack, and the objective of the hostile leader is to minimize this profit.

$$\min_{x\in\{0,1\}^n} f_3(x,y) = \sum_{i=1}^{n} b_i y_i \tag{3.1.3a}$$

$$\text{s. t.} \quad \sum_{i=1}^{n} a_i x_i \le A \tag{3.1.3b}$$

where $y_1, \dots, y_n$ solves the follower's problem

$$\max_{y\in\{0,1\}^n} \sum_{i=1}^{n} b_i y_i \quad \text{s.t.} \sum_{i=1}^{n} b_i y_i \le B \quad \text{and} \tag{3.1.3c}$$

$$y_i \le 1 - x_i \quad \text{for } i = 1, \dots, n \tag{3.1.3d}$$

Figure 3.1.3: The bilevel knapsack problem DNeg.

Figure 3.1.3 depicts the bilevel problem DNeg. The 0-1 variables $x_1, \dots, x_n$ (for the leader) and $y_1, \dots, y_n$ (for the follower) encode whether the corresponding item is packed into the knapsack. The interdiction constraint $y_i \le 1 - x_i$ in (3.1.3d) enforces that the follower cannot take item i once the leader has picked it. Note that leader and follower have exactly opposing objectives.

In Section 3.3, we will actually study a slightly more general version, where the constraint $\sum_{i=1}^{n} b_i y_i \le B$ in (3.1.3c) reads $\sum_{i=1}^{n} w_i y_i \le B$, and thus has cost coefficients that differ from the coefficients in the objective functions of leader and follower.

3.2 Computational Complexity

Recall the background Section 2.1 for essential concepts in the understanding of what follows.

In Section 3.2.1, we will show that all three bilevel knapsack variants are complete for the complexity class Σ_2^p. The second line of investigation is presented in Section 3.2.2 where we study these variants under so-called *unary* encodings (an integer n is represented as a string of n ones). The classical knapsack problem becomes much easier and polynomial solvable if the input in encoded in unary, and it is only natural to expect a similar behavior from our bilevel knapsack problems. Indeed, two of them become polynomial solvable if the input is encoded in unary, and thus show exactly the type of behavior that one would expect from a knapsack variant. The third variant, however, behaves stubbornly and becomes NP-complete under unary encodings, which is not the behavior one would expect. Our third line of results in Section 3.3, studies the approximability of the three bilevel variants. As a rule of thumb Σ_2^p-hard problems do not allow good approximation algorithms. Indeed, the literature only contains negative results in this direction that establish the inapproximability of various Σ_2^p-hard optimization problems (see [76] and [124, 125]). Of particular interest is the paper [125] by Umans that derives strong inapproximability results for Σ_2^p-hard optimization problems from certain error-correcting codes. Two of our bilevel knapsack variants (actually the same ones that are easy under unary encodings) behave exactly as expected and do not allow polynomial time approximation algorithms with finite worst case guarantee, assuming P$\neq$NP. For the third variant, however, we derive a polynomial time approximation scheme. This is the first approximation scheme for a Σ_2^p-hard optimization problem in the history of approximation algorithms, and from the technical point of view it is the most sophisticated result in this section. Section 3.2.4 concludes by summarizing our results.

3.2.1 Hardness Results under Binary Encodings

Throughout this section we consider bilevel knapsack problems where the input data is encoded in binary. As usual, we consider the decision versions of these optimization problems: “Does there exist an action of the leader that makes her objective value at least as good as some given bound?”

The decision versions of our bilevel problems DeRi, MACH, DNeg ask whether there exists a way of fixing the variables controlled by the leader, such that all possible settings of the variables controlled by the follower yield a good objective value for the leader. Since this question is exactly of the form $\exists x \forall y\, P(x, y)$, we conclude that all three considered bilevel knapsack variants are indeed contained in Σ_2^p. Next, we prove that these variants are Σ_2^p-hard. The Σ_2^p-hardness proofs in this section will all be done by reductions from the decision problem Subset-Sum-Interval (SSI), which has been proved to be Σ_2^p-complete by Eggermont and Woeginger [48].

Theorem 3.2.1. *The decision versions of the following bilevel problems (in binary encoding) are Σ_2^p-complete, both under the optimistic and under the pessimistic scenario:*

(a) The Dempe-Richter (DeRi) variant.
(b) The Mansi-Alves-de-Carvalho-Hanafi (MACH) variant.
(c) The Caprara-Carvalho-Lodi-Woeginger (DNeg) variant.

Proof. It remains to show that the three bilevel knapsack variants encapsulate the full difficulty of class Σ_2^p which will be done from reduction to problem Subset-Sum-Interval. In our reductions, all feasible solutions that are optimal for the follower will yield the same objective value for the leader. Hence the constructed instances do not depend on whether the follower behaves benevolently or malevolently towards the leader, and the theorem holds unconditionally under the optimistic scenario as well as under the pessimistic scenario.

The hardness proof for DeRi. Our reduction starts from an instance of Subset-Sum-Interval. We construct the following instance of DeRi.

- We set $A = 0$, $C = R$, and $C' = R + 2^r - 1$.
- For $i = 1, \ldots, k$, we create a so-called ordinary item i with leader's profit $a_i = 0$ and follower's profit/weight $b_i = q_i$.
- Furthermore there is a special magic item 0 with leader's profit $a_0 = 1$ and follower's profit $b_0 = 1/2$.

We claim that in the constructed instance of DeRi the leader can make her objective value ≥ 1 if and only if the Subset-Sum-Interval instance has answer YES.

(Proof of if). Assume that the Subset-Sum-Interval instance has answer YES, and consider the corresponding integer S that cannot be represented as a subset sum. Then a good strategy for the leader is to choose $x = S$ for the knapsack capacity. Suppose for the sake of contradiction that the follower does not pack the magic item. Then the weight of the packed set (and hence the follower's profit) is at most $S - 1$, which she could improve by adding the magic item to it. This contradiction shows that the magic item must be packed by the follower, which yields a profit of 1 for the leader.

(Proof of only if). Now assume that the Subset-Sum-Interval instance has answer NO, and consider the optimal knapsack capacity x for the leader. There exists a subset $I \subseteq \{1, \ldots, k\}$ with $\sum_{i \in I} q_i = x$, and the corresponding set of ordinary items brings a profit of x to the follower. If the follower packs the magic item, then her profit is at most $(x - 1) + 1/2 = x - 1/2$. Consequently the follower will not pick the magic item, and the objective value of the leader is 0. This completes the proof of Theorem 3.2.1.(a).

The hardness proof for MACH. We will essentially recycle and imitate the hardness argument from the preceding proof. Hence let us take an instance of Subset-Sum-Interval and construct the following instance of MACH from it.

- For $j = 0, \dots, r-1$ we create a so-called padding item j that is owned by the leader. The jth padding item has profit $a_j = 0$ and weight $c_j = 2^j$.
- For $i = 1, \dots, k$, we create a so-called ordinary item i that is owned by the follower. The ith ordinary item has profit $a'_i = 0$ for the leader and profit/weight $b'_i = c'_i = q_i$ for the follower.
- There is a magic item 0 owned by the follower, with profit $a'_0 = 1$ for the leader and profit/weight $b'_0 = c'_0 = 1/2$ for the follower.
- The knapsack capacity is $C = R + 2^r - 1$.

This completes the construction of the MACH instance. Now let us discuss the possible actions of leader and follower.

The leader decides which of the padding items are to be packed into the knapsack. Note that the overall weight of a subset of padding items can take any value between 0 and $2^r - 1$, and note, furthermore, that padding items bring no profit to the leader. Hence the decision power of the leader boils down to deciding how much of the knapsack capacity should be consumed by padding items; the remaining knapsack capacity after the leader's move can be any number between $C - (2^r - 1) = R$ and $C - 0 = R + 2^r - 1$. This means that the leader has essentially the same decision power as in previous reduction.

Then the follower has to react. The follower selects some of the ordinary items and possibly the magic item for the knapsack. As these items with their weights and profits are identical to those used in the previous reduction, also the follower has the same decision power. Summarizing, we see that leader and follower both face the same situation as in the proof of Theorem 3.2.1.(a). This completes the proof of Theorem 3.2.1.(b).

The hardness proof for DNeg. We consider an instance of Subset-Sum-Interval, and we define $Q = \sum_{i=1}^{k} q_i$. We construct the following instance of DNeg.

- For $j = 0, \dots, r-1$ we create a padding item p_j with $a(p_j) = 1$ and $b(p_j) = Q + 2^j$.
- For $j = 0, \dots, r-1$ we create a dummy item d_j with $a(d_j) = 1$ and $b(d_j) = Q$.
- For $i = 1, \dots, k$, we create an ordinary item o_i with $a(o_i) = r + 1$ and $b(o_i) = q_i$.
- The knapsack capacities are $A = r$ and $B = R + 2^r - 1 + rQ$.

We claim that in the constructed instance of DNeg the leader can make her objective value $\leq B - 1$ if and only if the Subset-Sum-Interval instance has answer YES.

(Proof of if). Assume that the integer S with $R \leq S < R + 2^r$ cannot be represented as a subset sum of the q_i. Then we make the leader pick r items among the padding items and dummy items whose b-values add up to a total of $rQ + (S - R)$. How does the follower react to this? We distinguish two cases. First, if the follower does not pick all r remaining padding items and dummy items, then her objective value is at most the b-value of the r most valuable padding items plus the b-value of all ordinary items; this b-value is smaller than B. Second, if the follower does pick all r remaining padding items and dummy items, then she picks a total b-value of $rQ + (2^r - 1) + (R - S) = B - S$. The remaining capacity in the follower's knapsack hence equals S, and by the definition of S there is no way of filling this remaining capacity with the ordinary items. Hence, the followers objective value always remains strictly below B.

(Proof of only if). Now assume that the Subset-Sum-Interval instance has answer NO. The leader must pack her knapsack with at most r padding items and dummy items, and she must leave at least r of the padding items and dummy items for the follower. The follower may react as follows. She arbitrarily picks r of the remaining padding items and dummy items, whose total b-value will lie somewhere between rQ (if all of them are dummy items) and $rQ + 2^r - 1$ (if all of them are padding items). Then the remaining capacity S in the follower's knapsack lies between $B - (rQ + 2^r - 1) = R$ and $B - rQ = R + 2^r - 1$. Since the Subset-Sum-Interval instance has answer NO, there exists a subset of the numbers q_i that adds up to S. The follower picks the corresponding ordinary items and fills her knapsack up to its limit B. This completes the proof of Theorem 3.2.1.(c). $\square$

3.2.2 Complexity Results under Unary Encodings

Throughout this section we consider bilevel knapsack problems where the input data is encoded in unary. As the Σ_2^p-complete problem Subset-Sum-Interval from Section 3.2.1 is solvable in polynomial time under unary encodings (Eggermont and Woeginger [48]), the hardness results in Theorem 3.2.1 do not carry over to the unary bilevel knapsack versions. We will show that variants DeRi and MACH under unary encodings are solvable in polynomial time, whereas variant DNeg under unary encodings is NP-complete.

A polynomial time solution for unary-DeRi. We consider the bilevel knapsack variant DeRi in (3.1.1a)–(3.1.1c). Our main tool is the polynomial time algorithm for the standard knapsack problem under unary encodings; see for instance Martello and Toth [87].

The leader simply checks all values x in the interval $C \leq x \leq C'$. For every fixed value of x, the optimization problem of the follower is a standard knapsack problem in unary encoding, and hence can be solved in polynomial time. The leader determines the corresponding optimal objective value $V(x)$ of the follower, and then computes the resulting objective value for herself under the optimistic and under the pessimistic scenario; this amounts to solving another standard knapsack problem under unary encoding. In the end the leader chooses the value x that brings her the best objective value.

This result is essentially due to Dempe and Richter [40]. A more sophisticated analysis of the approach yields the time complexity in the following theorem.

Theorem 3.2.2. *(Brotcorne, Hanafi and Mansi [17])*
The bilevel knapsack problem DeRi in unary encoding can be solved to optimality in polynomial time $O(nC')$, both for the optimistic scenario and the pessimistic scenario.

A polynomial time solution for unary-MACH. Next let us turn to variant MACH in (3.1.2a)–(3.1.2b). In a preprocessing phase we compute the following auxiliary information; note that the 0-1 variables $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$ in these auxiliary problems have the same meaning as in the problem (3.1.2a)–(3.1.2b).

- For $z = 0, \ldots, C$ we determine the maximum value $g(z)$ of $\sum_{j=1}^{m} a_j x_j$ subject to the constraint $\sum_{j=1}^{m} c_j x_j = z$.
- For $t = 0, \ldots, C$, we determine the maximum value $h(t)$ of $\sum_{i=1}^{n} b'_i y_i$ subject to the constraint $\sum_{i=1}^{n} c'_i y_i \leq t$.
- For $u = 0, \ldots, \sum_{i=1}^{n} b'_i$ and $v = 0, \ldots, C$, we determine the maximum value $k_{\max}(u, v)$ and the minimum value $k_{\min}(u, v)$ of $\sum_{i=1}^{n} a'_i y_i$ subject to the constraints $\sum_{i=1}^{n} b'_i y_i = u$ and $\sum_{i=1}^{n} c'_i y_i \leq v$.

The computations of the values $g(z)$ and $h(t)$ are again standard knapsack problems under unary encoding, and hence solvable in polynomial time. The computation of the values $k_{\max}(u, v)$ and $k_{\min}(u, v)$ can also be done in polynomial time by routine dynamic programming methods; we omit the straightforward details.

What are the options of the leader? The leader will pack a certain subset of her items into the knapsack, whose overall weight we want to denote by $z := \sum_{j=1}^{m} c_j x_j$. Then the follower is left with a remaining knapsack capacity of $C - z$. The follower will pick an item set that gives her the largest possible personal profit, which by definition equals $h(C-z)$. The follower's item set gives the leader a resulting profit of $k_{\max}(h(C-z), C-z)$ in the optimistic scenario and a profit of $k_{\min}(h(C-z), C-z)$ in the pessimistic scenario. Summarizing, once the leader has chosen her value of z, then her maximum profit in the optimistic scenario equals

$$g(z) \;+\; k_{\max}(h(C-z), C-z), \tag{3.2.1}$$

whereas her maximum profit in the pessimistic scenario equals

$$g(z) \;+\; k_{\min}(h(C-z), C-z). \tag{3.2.2}$$

Hence the decision making of the leader boils down to picking a value z from the range $0 \le z \le C$ that maximizes the expression in (3.2.1), respectively, (3.2.2). And as all the data is encoded in unary, this once again can be done in polynomial time. We summarize our findings in the following theorem.

Theorem 3.2.3. *The bilevel knapsack problem MACH in unary encoding can be solved in polynomial time, both for the optimistic scenario and the pessimistic scenario.*

NP-completeness of unary-DNeg. Our reduction is from the standard Vertex-Cover problem in undirected graphs; see Garey and Johnson [56].

Problem: Vertex-Cover

Instance: An undirected graph $G = (V, E)$; an integer bound t. (VC)

Question: Does G possess a vertex cover of size t, that is, a subset $T \subseteq V$ such that every edge in E has at least one of its vertices in T?

A *Sidon sequence* is a sequence $s_1 < s_2 < \cdots < s_n$ of positive numbers in which all pairwise sums $s_i + s_j$ with $i < j$ are different. Erdős and Turán [49] showed that for any odd prime p, there exists a Sidon sequence of p integers that all are below $2p^2$. The argument in [49] is constructive and yields a simple polynomial time algorithm for finding Sidon sequences of length n whose elements are bounded by $O(n^2)$. For more information on Sidon sequences, the reader is referred to O'Bryant [98].

We start our polynomial time reduction from an arbitrary instance $G = (V, E)$ and k of Vertex-Cover. Let $n = |V| \ge 10$, and let $v_1, \ldots, v_n$ be an enumeration of the vertices in V. We construct a Sidon sequence $s_1 < s_2 < \cdots < s_n$ whose elements are polynomially bounded in n. We define $S = \sum_{i=1}^{n} s_i$ as the sum of all numbers in the Sidon sequence, and we construct the following instance of DNeg as specified in (3.1.3a)–(3.1.3d).

- For every vertex v_i, we create a corresponding vertex-item with leader's weight $a(v_i) = 1$ and follower's weight $b(v_i) = S + s_i$.
- For every edge $e = [v_i, v_j]$, we create a corresponding edge-item with leader's weight $a(e) = t + 1$ and follower's weight $b(e) = 5S - s_i - s_j$.

- The capacity of the leader's knapsack is $A = t$, and the capacity of the follower's knapsack is $B = 7S$.

We claim that in the constructed instance of DNeg the leader can make her objective value $\leq 7S - 1$ if and only if the Vertex-Cover instance has answer YES.

(Proof of if). Assume that there exists a vertex cover T of size $|T| = t$. Then a good strategy for the leader is to put the t vertex-items that correspond to vertices in T into her knapsack, which fills her knapsack of capacity $A = t$ to the limit. Suppose for the sake of contradiction that afterwards the follower can still fill her knapsack with total weight $7S$. Then the follower must pick at least one edge-item (she can pack at most six vertex-items, and their weight would stay strictly below $7S$). Furthermore, the follower cannot pick two edge-items (since every edge-item has weight greater than $4S$). Consequently the follower must pick exactly one edge-item that corresponds to some edge $e = [v_i, v_j]$. The remaining space in the follower's knapsack is $2S + s_i + s_j$ and must be filled by two vertex-items. By the definition of a Sidon sequence, the only way of doing this would be by picking the two vertex-items corresponding to v_i and v_j. But that's impossible, as at least one of the vertices v_i and v_j is in the cover T so that the item has already been picked by the leader. This contradiction shows that the follower cannot reach an objective value of $7S$.

(Proof of only if). Now let us assume that the graph G does not possess any vertex cover of size t, and let us consider the game right after the move of the leader. Since the leader can pack at most t vertex-items, there must exist some edge $e = [v_i, v_j]$ in E for which the leader has neither picked the item corresponding to v_i nor the item corresponding to v_j. Then the follower may pick the vertex-item v_i, the vertex-item v_j, and the edge-item e, which brings her a total weight of $7S$.

Theorem 3.2.4. *The decision version of the bilevel knapsack problem DNeg in unary encoding is NP-complete, both for the optimistic scenario and the pessimistic scenario.*

Proof. The above construction can be performed in polynomial time. As the elements in the Sidon sequence are polynomially bounded in $|V|$, also their sum S and all the integers in our construction are polynomially bounded in $|V|$. In particular, this yields that the unary encoding length of the constructed DNeg instance is polynomially bounded in $|V|$. Together with the above arguments, this implies that DNeg in unary encoding is NP-hard.

It remains to show that DNeg in unary encoding is contained in NP. We use the optimal move of the leader as NP-certificate. This certificate is short, as it just specifies a subset of the items. To verify the certificate, we have to check that the follower cannot pick any item set of high weight. Since all weights are encoded in unary, this checking amounts to solving a standard knapsack problem in unary encoding, which can be done in polynomial time. □

3.2.3 Approximability and inapproximability

Our Σ_2^p-completeness proofs in Section 3.2.1 have devastating consequences for the polynomial time approximation of problems DeRi and MACH. Recall that our reduction for problem DeRi yields the following: it is Σ_2^p-hard to distinguish the DeRi instances in which the leader can reach an objective value of 1 from those DeRi instances in which the leader can only reach objective value 0. An analogous statement holds for problem MACH. As a polynomial time approximation algorithm with finite worst case guarantee would be able to distinguish between these two instance types, we get the following result.

Corollary 3.2.5. *Problems DeRi and MACH do not possess a polynomial time approximation algorithm with finite worst case guarantee, unless P=NP holds (which is equivalent to P=Σ_2^p).*

The statement in Corollary 3.2.5 is not surprising at all: the literature on the approximability of Σ_2^p-hard optimization problems consists entirely of such negative statements that show the inapproximability of various problems; see Ko and Lin [76] and Umans [124]. The following theorem breaks with this old tradition, and presents the first approximation scheme for a Σ_2^p-hard optimization problem in the history of approximation algorithms.

Theorem 3.2.6. *Problem DNeg has a polynomial time approximation scheme.*

The rest of this section is dedicated to the proof of Theorem 3.2.6. We apply and extend a number of rounding tricks from the seminal paper [80] by Lawler, we use approximation schemes from the literature as a black box, and we also add a number of new ingredients and rounding tricks.

Throughout the proof we will consider a fixed instance $\mathcal{I}$ of problem DNeg. Without loss of generality (*w.l.o.g.*) we assume that no item i in the instance satisfies $b_i > B$: such items could never be used by the follower, and hence are irrelevant and may as well be ignored. Let ε with $0 < \varepsilon < 1/3$ be a small positive real number; for the sake of simplicity we will assume that the reciprocal value $1/\varepsilon$ is integer.

Our global goal is to determine in polynomial time a feasible solution for the leader that yields an objective value of at most $(1+\varepsilon)^4$ times the optimum ($\frac{Approx(\mathcal{I})}{OPT(\mathcal{I})} \leq (1+\varepsilon)^4$). This will be done by a binary search over the range $0, 1, \ldots, B$ that (approximately) sandwiches the optimal objective value between a lower and an upper bound. Whenever we bisect the search interval between these bounds at some value U, we have to decide whether the optimal objective value lies below or above U. If the optimal objective value lies below U, then Lemma 3.2.8 and Lemma 3.2.9 (both derived next) show how to find and how to verify in polynomial time an approximate solution for the leader whose objective value is bounded by $(1+\varepsilon)^3\,U$. If these lemmas succeed then we make U the new upper bound. If the lemmas fail to produce an approximate objective value of at most $(1+\varepsilon)^3\,U$, then we make U the new lower bound. The binary search process terminates as soon as the upper bound comes within a factor of $1+\varepsilon$ of the lower bound. Note that we then lose a factor of $1+\varepsilon$ between upper and lower bound, and that we lose a factor of at most $(1+\varepsilon)^3$ by applying the lemmas. All in all, this yields the desired approximation guarantee of $(1+\varepsilon)^4$ and completes the proof of Theorem 3.2.6. See Figure 3.2.1 for an illustration of these ideas.

$$Approx(\mathcal{I}) \le U(1+\varepsilon)^3, \quad U \le L(1+\varepsilon), \quad L \le OPT(\mathcal{I})$$

$$\Rightarrow Approx(\mathcal{I}) \le OPT(\mathcal{I})(1+\varepsilon)^4$$

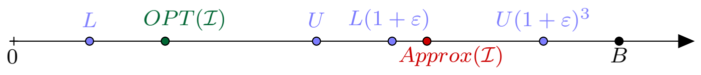


Figure 3.2.1: Approximation of the optimal value for a DNeg instance $\mathcal{I}$. Let L and U be a lower and upper bound, respectively, for $OPT(\mathcal{I})$.

How do handle the central cases. We start by assuming that U is an upper bound on the optimal objective value of the considered instance with

$$B/2 \;\le\; U \;\le\; B/(1+\varepsilon). \tag{3.2.3}$$

The items $i = 1, \ldots, n$ are partitioned according to their b-values into so-called *large* items that satisfy $U < b_i$, into *medium* items that satisfy $\varepsilon U < b_i \le U$, and into *small* items that satisfy $b_i \le \varepsilon U$. We denote by L, M, S respectively the set of large, medium, small items. Furthermore a medium item i belongs to class $\mathcal{C}_k$, if it satisfies

$$k\varepsilon^2 U \;\le\; b_i \;<\; (k+1)\varepsilon^2 U.$$

Note that only classes $\mathcal{C}_k$ with $1/\varepsilon \le k \le 1/\varepsilon^2$ play a role in this classification. By (3.2.3) the overall size of $2/\varepsilon$ medium items exceeds the capacity of the follower's knapsack. Hence the follower can fit at most $2/\varepsilon$ medium items into her knapsack.

In the following we will analyze two scenarios. In the first scenario, the solution x^* used by the leader and the solution y^* for the follower both will carry a superscript*. The sets of large, medium, small items packed by x^* into the leader's knapsack will be denoted respectively by L_x^*, M_x^*, S_x^*, and the corresponding sets for y^* and the follower will be denoted L_y^*, M_y^*, S_y^*. In the second scenario we use analogous notations with the superscript$^\#$. The first scenario is centered around an optimal solution x^* for the leader. The second scenario considers another feasible solution $x^\#$ for the leader that we call the *aligned* version of x^*.

- Solution $x^\#$ packs all large items into the knapsack; hence $L_x^\# = L$.
- Solution $x^\#$ packs the following medium items from class $\mathcal{C}_k$ (note that $M_x^\# \subseteq M_x^*$):
  (i.) If $|\mathcal{C}_k - M_x^*| \le 2/\varepsilon$, then solution $x^\#$ packs all items in $M_x^* \cap \mathcal{C}_k$.
  (ii.) If $|\mathcal{C}_k - M_x^*| > 2/\varepsilon$, then $x^\#$ packs an item $i \in M_x^* \cap \mathcal{C}_k$ if and only if there are at most $2/\varepsilon$ items $j \in \mathcal{C}_k - M_x^*$ with smaller b-value $b_j \le b_i$. (By this choice, the $2/\varepsilon$ items with smallest b-value in $\mathcal{C}_k - M_x^*$ coincide with the $2/\varepsilon$ items with smallest b-value in $\mathcal{C}_k - M_x^\#$.)
- For the small items we first determine a $(1+\varepsilon)$-approximate solution to the following auxiliary problem (Aux): find a subset $Z \subseteq S$ of the small items that minimizes $b(Z)$, subject to the covering constraint $a(Z) \ge a(L_x^\# \cup M_x^\#) + a(S) - A$. Solution $x^\#$ then packs the complementary set $S_x^\# = S - Z$.

This completes the description of $x^\#$, which is easily seen to be a feasible action for the leader. Note that also the optimal solution x^* packs all the large items, as otherwise the follower may pack a large item and push the objective value above the bound U. Then $L_x^\# = L_x^*$ and $M_x^\# \subseteq M_x^*$ imply $a(L_x^* \cup M_x^*) \ge a(L_x^\# \cup M_x^\#)$, which yields

$$A \;\ge\; a(L_x^* \cup M_x^* \cup S_x^*) \;\ge\; a(L_x^\# \cup M_x^\#) + a(S_x^*). \tag{3.2.4}$$

As $a(S_x^*) = a(S) - a(S - S_x^*)$, we conclude from (3.2.4) that the set $S - S_x^*$ satisfies the covering constraint in the auxiliary problem (Aux). Hence, the optimal objective value of (Aux) is upper bounded by $b(S - S_x^*)$, and any $(1+\varepsilon)$-approximate solution Z to (Aux) must satisfy $b(Z) \le (1+\varepsilon)\, b(S - S_x^*)$, which is equivalent to

$$b(S - S_x^\#) \;\le\; (1+\varepsilon)\, b(S - S_x^*). \tag{3.2.5}$$

The following lemma demonstrates that the aligned solution $x^\#$ is almost as good for the leader as the underlying optimal solution x^*.

Lemma 3.2.7. *Given an optimal solution (x^*, y^*) with $f_3(x^*, y^*) \le U$, let $x^\#$ be the solution aligned to x^*. If the leader uses $x^\#$, then every feasible reaction $y^\#$ for the follower yields an objective value $f_3(x^\#, y^\#) \le (1+2\varepsilon)\, U$.*

Proof. Suppose for the sake of contradiction, that there exists a reaction $y^{\#}$ for the follower that yields an objective value of $f_3(x^{\#}, y^{\#}) > (1+2\varepsilon)\,U$. Based on $y^{\#}$ we will construct another solution y^* for the follower:

- Solution y^* does not use any large item; hence $L_y^* = \emptyset$.
- Solution y^* picks the same number of items from every class $\mathcal{C}_k$ as $y^{\#}$ does. It avoids items in x^* and selects the $|\mathcal{C}_k \cap M_y^{\#}|$ items in $\mathcal{C}_k - M_x^*$ that have the smallest b-values.
- Finally we add small items from $S - S_x^*$ to the follower's knapsack, until no further item fits or until we run out of items.

Solution $y^{\#}$ packs at most $2/\varepsilon$ medium items, and hence uses at most $2/\varepsilon$ items from $\mathcal{C}_k$. By our choice of medium items for $x^{\#}$ we derive $b(\mathcal{C}_k \cap M_y^*) \le b(\mathcal{C}_k \cap M_y^{\#})$ for every k, which implies

$$b(M_y^*) \;\le\; b(M_y^{\#}) \;\le\; B. \tag{3.2.6}$$

Solution y^* only selects items that are not used by x^*, and inequality (3.2.6) implies that all the selected items indeed fit into the follower's knapsack. Hence, y^* constitutes a feasible reaction of the follower if the leader chooses x^*.

Next, let us quickly go through the item types. First of all, neither solution y^* nor solution $y^{\#}$ can use any large item, so that we have

$$b(L_y^*) \;=\; b(L_y^{\#}) \;=\; 0. \tag{3.2.7}$$

For the medium items, the ratio between the smallest b-value and the largest b-value in class $\mathcal{C}_k$ is at least $k/(k+1) \ge 1-\varepsilon$. Hence, we certainly have $b(\mathcal{C}_k \cap M_y^*) \ge (1-\varepsilon)\, b(\mathcal{C}_k \cap M_y^{\#})$, which implies

$$b(M_y^*) \;\ge\; (1-\varepsilon)\, b(M_y^{\#}). \tag{3.2.8}$$

Let us turn to the small items. Suppose that y^* cannot accommodate all small items from $S - S_x^*$ in the follower's knapsack. Then some small item i with $b_i < \varepsilon U$ does not fit, which with (3.2.3) leads to $b(y^*) > B - \varepsilon\, U \ge U$. As this violates our upper bound U on the optimal objective value, we conclude that y^* accommodates all such items and satisfies $S_y^* = S - S_x^*$. This relation together with (3.2.5) and the disjointness of the sets $S_x^{\#}$ and $S_y^{\#}$ yields

$$b(S_y^*) \;=\; b(S - S_x^*) \;\ge\; \frac{b(S - S_x^{\#})}{1+\varepsilon} \;\ge\; \frac{b(S_y^{\#})}{1+\varepsilon} \;>\; (1-\varepsilon)\, b(S_y^{\#}). \tag{3.2.9}$$

Now let us wrap things up. If the leader chooses x^*, the follower may react with the feasible solution y^* and get an objective value

$$\begin{aligned} f_3(x^*, y^*) &= b(L_y^*) + b(M_y^*) + b(S_y^*) \\ &> (1-\varepsilon)\, b(L_y^{\#}) + (1-\varepsilon)\, b(M_y^{\#}) + (1-\varepsilon)\, b(S_y^{\#}) \\ &= (1-\varepsilon)\, f_3(x^{\#}, y^{\#}) \;>\; (1-\varepsilon)(1+2\varepsilon)\, U \;>\; U. \end{aligned}$$

Here we used the estimates in (3.2.7), (3.2.8), and (3.2.9). As this objective value violates the upper bound U, we have reached the desired contradiction. □

Lemma 3.2.8. *Given an upper bound U on the objective value that satisfies (3.2.3), one can compute in polynomial time a feasible solution x for the leader, such that every reaction y of the follower has $f_3(x, y) \le (1+\varepsilon)^3\, U$.*

Proof. If we did not only know the bound U but also an optimal solution x^*, then we could simply determine the corresponding aligned solution $x^{\#}$ and apply Lemma 3.2.7. We will bypass this lack of knowledge by checking many candidates for the set $M_x^{\#}$. Let us recall how the aligned solution $x^{\#}$ picks medium items from class $\mathcal{C}_k$.

- If $|\mathcal{C}_k - M_x^*| \le 2/\varepsilon$ then $M_x^{\#} \cap \mathcal{C}_k = M_x^* \cap \mathcal{C}_k$. Note that there are only $O(|\mathcal{C}_k|^{2/\varepsilon})$ different candidates for $M_x^{\#} \cap \mathcal{C}_k$.
- If $|\mathcal{C}_k - M_x^*| > 2/\varepsilon$ then $M_x^{\#} \cap \mathcal{C}_k$ is a subset of M_x^*; an item i from $M_x^* \cap \mathcal{C}_k$ enters $M_x^{\#}$ if there are at most $2/\varepsilon$ items $j \in \mathcal{C}_k - M_x^*$ with $b_j \le b_i$. Note that $M_x^{\#} \cap \mathcal{C}_k$ is fully determined by the $2/\varepsilon$ items with smallest b-value in $\mathcal{C}_k - M_x^*$. As there are only $O(|\mathcal{C}_k|^{2/\varepsilon})$ ways for choosing these $2/\varepsilon$ items, there are only $O(|\mathcal{C}_k|^{2/\varepsilon})$ different candidates for $M_x^{\#} \cap \mathcal{C}_k$.

Altogether there are only $O(|\mathcal{C}_k|^{2/\varepsilon})$ ways of picking the medium items from class $\mathcal{C}_k$. As every class satisfies $|\mathcal{C}_k| \le n$ and as there are only $1/\varepsilon^2$ classes to consider, we get a polynomial number $O(n^{2/\varepsilon^3})$ of possibilities for choosing the set $M_x^{\#}$ in the aligned solution. Summarizing, we only need to check a polynomial number of candidates for set $M_x^{\#}$.

How do we check such a candidate $M_x^{\#}$? The aligned solution always uses $L_x^{\#} = L$, and the auxiliary problem (Aux) is fully determined once $M_x^{\#}$ and $L_x^{\#}$ have been fixed. We approximate the auxiliary problem by standard methods (see for instance Pruhs and Woeginger [108]), and thus also find the set $S_x^{\#}$ in polynomial time. This yields the full corresponding aligned solution $x^{\#}$. It remains to verify the quality of this aligned solution for the leader, which amounts to analyzing the resulting knapsack problem at the follower's level. We use one of the standard approximation schemes for knapsack as for instance described by Lawler [80], and thereby get a $(1+\varepsilon)$-approximate solution for the follower's problem.

While checking and scanning through the candidates, we eventually must hit a good candidate $M_x^\#$ that yields the correct aligned version x of an optimal solution. By Lemma 3.2.7 the corresponding objective value $f_3(x, y)$ is bounded by $(1 + 2\varepsilon)\, U$. Then the approximation scheme finds an objective value of at most $(1+\varepsilon)(1+2\varepsilon)\, U \leq (1+\varepsilon)^3 U$. This completes the proof of the lemma. □

How do handle the boundary cases. Finally let us discuss the remaining cases where U does not satisfy the bounds in (3.2.3). The first case $U > B/(1 + \epsilon)$ is trivial, as the objective value never exceeds the follower's knapsack capacity B; hence in this case the objective value will always stay below $(1 + \epsilon)\, U$. The second case $U < B/2$ is settled by the following lemma.

Lemma 3.2.9. *Given an upper bound $U < B/2$ on the objective value, one can compute in polynomial time a feasible solution x for the leader, such that every reaction y of the follower has $f_3(x, y) \leq (1 + \epsilon)\, U$.*

Proof. If the objective value is below $B/2$, then the leader must pick all items i with $b_i \geq B/2$; otherwise the follower could pick one and push the objective value to $B/2$ or more. Once the leader has chosen her solution x, all remaining items will fit into the follower's knapsack: the knapsack has free capacity of at least $B - U > B/2$, and hence every item i with $b_i < B/2$ will fit there.

With these observations, the goal of the leader boils down to the following: partition the item set into two parts Z_l and Z_f such that the value $b(Z_f)$ is minimized subject to the condition that the items in Z_l altogether fit into the leader's knapsack. This minimization problem belongs to the class of subset selections problems studied by Pruhs and Woeginger [108]: determine a subset Z_f of items that has minimum cost $b(Z_f)$ subject to the feasibility constraint that the total size of all items outside Z_f is at most the size of the leader's knapsack. This subset selection problem can be solved in pseudopolynomial time by routine dynamic programming; the resulting time complexity is bounded in B, in n, and in the logarithm of A. With this, Theorem 1.2 in [108] yields the existence of an approximation scheme which yields the desired solution x for the leader. □

3.2.4 Summary

We have analyzed the computational complexity of three bilevel knapsack problems from the literature. All three problems DeRi, MACH, DNeg turn out to be Σ_2^p-complete under the standard binary encoding of the input. Our results provide strong evidence that bilevel knapsack problems cannot be formulated as a classical single-level integer problem of polynomial size; otherwise the entire polynomial hierarchy would collapse to its first level which is considered to be extremely unlikely in the area of computational complexity theory. Furthermore, we have settled the complexity of these three bilevel knapsack problems under *unary* encodings of the input: unary-DeRi and unary-MACH are polynomially solvable, whereas unary-DNeg is NP-complete. Finally, we studied the approximability of the three problems. DeRi and MACH turned out to be inapproximable, whereas DNeg has a polynomial time approximation scheme.

Our investigations provide a complete and clean picture of the complexity landscape of the considered bilevel knapsack problems. We expect that our results will also be useful in classifying and understanding other bilevel problems, and that our hardness proofs will serve as stepping stones for future results.

3.3 Bilevel Knapsack with Interdiction Constraints

In this section, we will investigate a more general version of the bilevel knapsack variant defined in Section 3.1.3 which can be modeled through the following bilevel formulation:

$$\text{(DNeg)} \quad \min_{(x,y)\in\{0,1\}^n\times\{0,1\}^n} \sum_{i=1}^{n} b_i y_i \tag{3.3.1a}$$

$$\text{s. t.} \quad \sum_{i=1}^{n} a_i x_i \le A \tag{3.3.1b}$$

where $y_1, \ldots, y_n$ solves the follower's problem

$$\max_{y\in\{0,1\}^n} \sum_{i=1}^{n} b_i y_i \quad \text{s.t.} \quad \sum_{i=1}^{n} w_i y_i \le B \quad \text{and} \tag{3.3.1c}$$

$$y_i \le 1 - x_i \quad \text{for } i = 1, \ldots, n, \tag{3.3.1d}$$

where x and y are the binary decision vectors controlled by the leader and the follower, respectively. Since it is in fact this general version that was originally suggested in the PhD thesis of DeNegre [41], we keep the label DNeg to designate it. Without loss of generality, we will throughout make the following three assumptions:

$$b_i,\ a_i,\ w_i,\ A \text{ and } B \text{ are positive integers} \tag{3.3.2}$$

$$a_i < A \quad \text{and} \quad w_i < B \quad \text{for all } i \tag{3.3.3}$$

$$\sum_{i=1}^{n} a_i > A \quad \text{and} \quad \sum_{i=1}^{n} w_i > B. \tag{3.3.4}$$

In addition to our methodological motivation, DNeg can model a real-world application, called Corporate Strategy problem which is described in [41]: a Company B wishes to determine its marketing strategy for the upcoming fiscal year. Company B has to decide which demographic or geographic regions to target, subject to a specified marketing budget. There exists a cost to establish a marketing campaign for each target region and an associated benefit. Company B's goal is to maximize its marketing benefit. The larger Company A has market dominance; whenever Company A and Company B target the same region, Company B is unable to establish a worthwhile marketing campaign. In other words, Company A can interdict regions for the marketing problem to be solved by Company B.

Our goal is to end up with an algorithm to find the exact optimal solution. In Section 3.3.1, we review the algorithmic approaches to bilevel knapsack variants highlighting the difficulties that general MIBP methods encounter when solving DNeg and then propose one straightforward scheme for problem DNeg. In Section 3.3.2, we devise our algorithm for DNeg which is the central contribution in this section. Section 3.3.3 presents the computational results for our algorithm when applied on new randomly generated instances and on instances from the literature.

3.3.1 Knapsack Bilevel Algorithms

Brotcorne *et al.* [18] consider a bilevel knapsack problem in which the decision of the leader only modifies the budget available for the follower. The algorithm in [18] may be summarized as follows: compute an upper bound for the follower's budget, by ignoring the resources consumed by the leader; solve the follower's 0–1 KP considering this budget bound through the standard knapsack dynamic programming approach (see for instance [86]). More precisely, the best follower's reactions for all her possible budgets from 0 to the bound are computed. (Note that in this case, different decisions of the leader may yield the same subproblem for the follower.) With this, the authors are able to define the follower's best reaction set for any fixed leader's decision through linear constraints, reducing the problem to single-level. If we mimic this procedure for problem DNeg, we would have to consider all the leader's interdictions that imply different reactions of the follower. However, in this case for every possible decision of the leader, the follower's KP is modified in terms of the (not interdicted) items available and not in terms of her budget. Since different decisions of the leader always yield different problems for the follower, the number of lower level subproblems for the follower grows with the number 2^n of item subsets and hence is exponential. In short, this is the reason why the methods developed in [18] cannot be applied to DNeg.

Yet another bilevel knapsack variant occurs in the work of Chen and Zhang [22], where the leader's decision only interferes with the follower's objective function, but not with the follower's feasible region. This variant is computationally much easier, since the leader wants to maximize the social welfare (total profit) that leads to a coordination and alignment of the leader's and the follower's interests.

Next, we focus in the general MIBP algorithms when applied to DNeg. As mentioned in Section 2.3.1.1, these methods adapt the Branch-and-Bound techniques for single level optimization to the bilevel case. In the root node, the high-point problem is solved which in the case of being applied to DNeg has an optimal value of zero, and hence does not provide an interesting lower bound for solving DNeg. Under this approach, the method continues by standard variable branching, and once a node has an integer solution verify its bilevel feasibility (which amounts to solving a KP for the follower). A bilevel feasible solution represents an upper bound and therefore helps to prune some nodes. Unfortunately, for all possible leader's decisions the high-point problem may have its optimum equal to zero if $y = 0$ (thus, these nodes are not pruned), meaning that the method would enumerate all the possible leader's decisions. Note, that the number of feasible leader's solutions is $\Theta\left(2^n\right)$, so that this all boils down to a standard brute force approach.

DeNegre [41] considers interdiction problems, and constructs a Branch-and-Cut scheme by adding some new ingredients to the basic method. (In [41] the disjunction is stated for the general interdiction problems, but for sake of clarity, we explicitly show it here for the DNeg problem.) Consider a node t where the optimal solution (x^t, y^t) is integer but not bilevel feasible (that is, the best follower's reaction to x^t is $\widehat{y}$ with $\sum_{i=1}^n b_i \widehat{y}_i > \sum_{i=1}^n b_i y_i^t$). In such a node t, the method either adds valid inequalities (cuts) such that x^t becomes infeasible (the so-called nogood cuts), or exploits the interdiction structure of the problems by branching on the following disjunction: either the leader packs a set of items such that $\sum_{i:x_i^t=0} x_i \geq 1$ or the leader packs a set of items such that $\sum_{i:x_i^t=0} x_i \leq 0$ and the follower has a profit $\sum_{i=1}^n b_i y_i \geq \sum_{i=1}^n b_i \widehat{y}_i$. In Section 3.3.2, we will build a method that uses this disjunction idea to solve DNeg, but in a more sophisticated and efficient way.

Hemmati *et al.* [65] proposed a cutting plane scheme for an interdiction problem in the context of networks. Next, we describe a natural cutting plane approach to solve DNeg exactly. The ideas of this approach will be an ingredient of our algorithm that will be stated in Section 3.3.2.

Cutting plane approach Problem DNeg is equivalent to the following single-level linear optimization problem:

$$(BKP) \quad \min_{(p,x)\in R\times\{0,1\}^n} p \tag{3.3.5a}$$

$$\text{subject to} \quad \sum_{i=1}^{n} a_i x_i \leq A \tag{3.3.5b}$$

$$p \geq \sum_{i=1}^{n} y_i b_i \left(1 - x_i\right) \quad \forall y \in \mathbb{S} \tag{3.3.5c}$$

Here $\mathbb{S}$ is the collection of all feasible packings for the follower. As the size of $\mathbb{S}$ is $O\left(2^n\right)$, the use of the cutting plane approach is the standard method to apply; see Algorithm 3.3.1.1. In Algorithm 3.3.1.1, the function *BestReaction* receives as input the leader's decision x^k from the optimal solution of a BKP with $\mathbb{S}$, and computes a rational reaction $y\left(x^k\right)$ for the follower, that is, the KP optimum to interdiction x^k.

Note that this type of single-level reformulation works for all interdiction problems where the lower level optimization problem can be replaced by a set of constraints explicitly taking into account all possible reactions to the leader's strategy. Note furthermore that this reformulation is exponential in size.

Algorithm 3.3.1.1 CP- Cutting Plane Approach

Input: An instance of DNeg.

Output: Optimal value and an optimal solution to DNeg.

```
 1: $k \leftarrow 1$
 2: Initialize $\mathbb{S}$ (e.g., with the best follower's reaction when there is no interdiction)
 3: Let $\left(p^k, x^k\right)$ be an optimal solution to BKP with $\mathbb{S}$
 4: $y(x^k) \leftarrow \mathit{BestReaction}\left(x^k\right)$
 5: while $p^k < \sum_{i=1}^{n} b_i y_i(x^k)$ do
 6:     Add constraint $p \geq \sum_{i=1}^{n} y_i(x^k) b_i \left(1 - x_i\right)$ to BKP // update $\mathbb{S}$
 7:     $k \leftarrow k + 1$
 8:     Solve $BKP$ and let $\left(p^k, x^k\right)$ be the optimal solution
 9:     $y(x^k) \leftarrow \mathit{BestReaction}\left(x^k\right)$
10: end while
11: return $p$, $\left(x^k, y(x^k)\right)$
```

3.3.2 CCLW Algorithm: a Novel Scheme

Motivated by the previous section, we propose a new approach to tackle DNeg. The algorithm initialization is studied by computing **an upper bound for DNeg**. Then, we will construct a naive **iterative method** for solving DNeg exactly. This basic scheme will be enhanced through a sequence of improvements in what follows. One such improvement takes into account the ideas of the cutting plane approach presented in the previous section, thus mixing the advantages of this method with ours.

An Upper bound for DNeg. The unsuccessful search for *dual lower* bounds in bilevel optimization motivated us to try a completely different approach, which first computes a *primal upper* bound. In practice, this approach is very effective and enabled us to quickly find an optimal solution in almost all our experiments.

The following theorem formulates the first upper bound for DNeg that our algorithm computes. The underlying idea is simple: the set of follower's feasible strategies is extended (through the continuous relaxation of her optimization program) and, consequently, the follower's profit is greater than or equal to the one obtained with the original set of strategies. This provides an upper bound to DNeg.

Theorem 3.3.1. *The optimal solution value of the following continuous bilevel formulation provides an upper bound on the optimal solution value of problem DNeg:*

$$
\begin{aligned}
(UB) \qquad & \min_{(x,y)\in\{0,1\}^n\times[0,1]^n} \sum_{i=1}^{n} b_i y_i && \text{(3.3.6a)}\\
& \textit{s. t.} \quad \sum_{i=1}^{n} a_i x_i \le A && \text{(3.3.6b)}\\
& \quad \textit{where } y_1,\dots,y_n \textit{ solves the follower's problem} &&\\
& \quad \max_{y\in[0,1]^n} \sum_{i=1}^{n} b_i y_i \quad \textit{s.t.} \sum_{i=1}^{n} w_i y_i \le B \quad \textit{and} && \text{(3.3.6c)}\\
& \quad y_i \le 1 - x_i \quad \textit{for } i = 1,\dots,n && \text{(3.3.6d)}
\end{aligned}
$$

Proof. The follower's problem (3.3.6c)-(3.3.6d) is a relaxation of problem (3.3.1c)-(3.3.1d) since the binary requirement on the variables y is removed. Therefore, given any fixed leader's interdiction x, the optimal value of problem (3.3.6c)-(3.3.6d) is greater or equal than the optimal value of problem (3.3.1c)-(3.3.1d) and thus, provides an upper bound.

To complete the proof note that problems DNeg and UB both are always bilevel feasible which implies that UB always provides an upper bound to DNeg. □

From the last proof, it is easy to see that an analogous result holds for any (general) min-max MIBP. Our motivation for introducing UB is that it can be written as a single-level MIP, thus leading to the possibility of applying effective solution methods as well as reliable software tools.

Theorem 3.3.2. *The bilevel problem UB is equivalent to the following:*

$$
\begin{aligned}
(MIP^1) \qquad \min_{x\in\{0,1\}^n, z\in[0,\infty)^{n+1}, u\in[0,\infty)^n} \quad & z_0 B + \sum_{i=1}^{n} u_i && && (3.3.7a)\\
\textit{s. t.} \quad & \sum_{i=1}^{n} a_i x_i \leq A && && (3.3.7b)\\
& u_i \geq 0 && \textit{for } i = 1,\ldots,n && (3.3.7c)\\
& u_i \geq z_i - b_i x_i && \textit{for } i = 1,\ldots,n && (3.3.7d)\\
& w_i z_0 + z_i \geq b_i && \textit{for } i = 1,\ldots,n. && (3.3.7e)
\end{aligned}
$$

Proof. The two main ingredients of our proof are the use of duality theory (presented in Section 2.2) and the convex relaxation by McCormick [89].

The follower's optimization problem (continuous relaxation of her KP) is feasible and bounded for any x. Hence, it always has an optimal solution. In this way, according to the strong duality principle (Property 2.2.2), we can write the single-level formulation equivalent to UB in the following way:

$$
\begin{aligned}
\min_{x\in\{0,1\}^n, z\in[0,\infty)^{n+1}, y\in[0,1]^n} \quad & \sum_{i=1}^{n} b_i y_i && && (3.3.8a)\\
\text{s. t.} \quad & \sum_{i=1}^{n} a_i x_i \leq A && && (3.3.8b)\\
& z_0 B + \sum_{i=1}^{n} (1 - x_i)\, z_i = \sum_{i=1}^{n} b_i y_i && && (3.3.8c)\\
& \sum_{i=1}^{n} w_i y_i \leq B && && (3.3.8d)\\
& x_i + y_i \leq 1 && \text{for } i = 1,\ldots,n && (3.3.8e)\\
& w_i z_0 + z_i \geq b_i && \text{for } i = 1,\ldots,n, && (3.3.8f)
\end{aligned}
$$

where the new variables z_i are the dual variables of the follower's continuous relaxation problem.

Note that we can further simplify the above formulation by removing the decision vector y,

$$\min_{x\in\{0,1\}^n, z\in[0,\infty)^{n+1}} \quad z_0 B + \sum_{i=1}^{n} (1-x_i)\, z_i \tag{3.3.9a}$$

$$\text{s. t.} \quad \sum_{i=1}^{n} a_i x_i \leq A \tag{3.3.9b}$$

$$w_i z_0 + z_i \geq b_i \quad \text{for } i = 1, \dots, n \tag{3.3.9c}$$

Let us clarify this equivalence. Observe that any feasible solution (x^*, z^*, y^*) of (3.3.8) implies that (x^*, z^*) is feasible for (3.3.9) and thus, (3.3.9) provides a lower bound to (3.3.8). On the other hand, given any optimal solution (x^*, z^*) of (3.3.9), we may consider x^* fixed in the follower's continuous relaxation problem and obtain an associated primal optimal solution y^*. This ensures that (x^*, z^*, y^*) is feasible to (3.3.8) and, in particular, optimal.

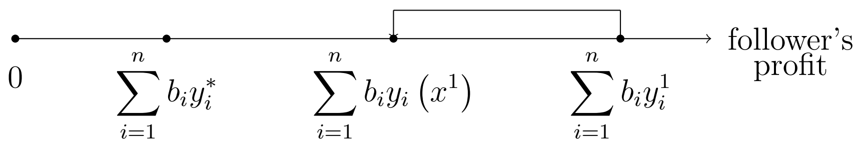
Finally, the bilinear terms $x_i z_i$ are linearized by adding the extra variables $u_i = (1 - x_i)\, z_i$ and the associated McCormick constraints (3.3.7c) and (3.3.7d). $\square$

Before showing how the solution of MIP^1 will be used to obtain an algorithm for problem DNeg, it is worth noting that UB can be alternatively written as

$$\begin{aligned}
\min_{(x,y)\in\{0,1\}^n\times[0,1]^n} \quad & \sum_{i=1}^{n} b_i y_i (1-x_i) \\
\text{s. t.} \quad & \sum_{i=1}^{n} a_i x_i \leq A \\
& \text{where } y_1, \dots, y_n \text{ solves the follower's problem} \\
& \max_{y\in[0,1]^n} \sum_{i=1}^{n} b_i y_i (1-x_i) \quad \text{s.t.} \sum_{i=1}^{n} w_i y_i \leq B.
\end{aligned}$$

It is easy to verify that this is a reformulation of UB (same optimal solution value) and, that for any fixed vector x we can use strong duality to obtain an equivalent single-level optimization problem. Indeed, for any fixed vector x, the interdiction constraints are embedded into the objective function, by setting to 0 the profit of all interdicted items. The advantage of this reformulation is that no variables of the leader do appear in the right hand side of the follower's constraints, which implies that there are no bilinear terms in its dual. However, in practice the reformulation does not have a significant impact on the computation times.

So far, we have built a Mixed Integer Linear Problem MIP^1 to compute an upper bound on DNeg. The first step of our algorithm is to solve MIP^1 to optimality and to obtain



0
$\sum_{i=1}^{n} b_i y_i^*$
$\sum_{i=1}^{n} b_i y_i\left(x^1\right)$
$\sum_{i=1}^{n} b_i y_i^1$
follower's profit

Figure 3.3.1: Illustration of the upper bounds to DNeg, where (x^*, y^*) is an optimal solution to DNeg, (x^1, y^1) is an optimal solution to MIP^1 and $(x^1, y(x^1))$ is the corresponding bilevel feasible solution.

the leader's decision vector x^1. This then is followed by solving the following KP, which is the follower's best reaction to x^1:

$$(KP^1) \quad \max_{y \in \{0,1\}^n} \quad \sum_{i=1}^{n} b_i y_i \tag{3.3.11a}$$

$$\text{s. t.} \quad \sum_{i=1}^{n} w_i y_i \leq B \tag{3.3.11b}$$

$$y_i \leq 1 - x_i^1 \quad \text{for } i = 1, \ldots, n, \tag{3.3.11c}$$

Let $y(x^1)$ be an optimal solution of KP^1. Then $\sum_{i=1}^{n} b_i y_i(x^1)$ is our new upper bound. Figure 3.3.1 provides a pictorial illustration of the relationships between these solutions.

We will see in Section 3.3.3 that on our randomly-generated test instances, $(x^1, y(x^1))$ provides a very tight approximation of the optimal solution value to DNeg. Before continuing, we note that if in the optimal solution of UB the follower's vector y is binary, then that solution is bilevel feasible but *not* necessarily optimal for DNeg.

Example 3.3.3. *Consider an instance with 3 items where*

$$b = (4,3,3), \quad a = (2,1,1), \quad w = (4,3,2), \quad A = 2 \text{ and } B = 4.$$

It is easy to check that the optimal solution for UB is binary with $x = (0,1,1)$ and $y = (1,0,0)$ with value 4. However, the optimal solution for DNeg has $x = (1,0,0)$ and $y = (0,1,0)$ (or $y = (0,0,1)$) with value 3. Indeed, when $x = (1,0,0)$ and the follower has the possibility of packing fractions of items, then the follower's reply is $y = \left(0, \frac{2}{3}, 1\right)$ with value 5.

Iterative method. The basic scheme to solve problem DNeg is given by Algorithm 3.3.2.1. It consists of iteratively computing upper bounds by solving, at each iteration k, the MIP proposed in the previous section amended by a *nogood constraint*

(NG_0) that forbids the leader to repeat her last strategy x^{k-1} (see for instance [7] or [31]):

$$\sum_{i:x_i^k=1}(1-x_i)+\sum_{i:x_i^k=0}x_i \geq 1. \tag{3.3.12}$$

In this way, essentially the leader's strategies are enumerated until the last MIP is proven infeasible.

Algorithm 3.3.2.1 Basic Iterative Method

Input: An instance of DNeg.

Output: Optimal value and an optimal solution to DNeg.

```
 1: $k \leftarrow 1$; $BEST \leftarrow +\infty$;
 2: Build $MIP^k$
 3: while $MIP^k$ is feasible do
 4:     $x^k \leftarrow \arg\min\{MIP^k\}$
 5:     $y\left(x^k\right) \leftarrow BestReaction\left(x^k\right)$ // solves the follower's KP by fixing $x^k$
 6:     if $\sum_{i=1}^n b_i y_i\left(x^k\right) < BEST$ then
 7:         $BEST \leftarrow \sum_{i=1}^n b_i y_i\left(x^k\right)$;
 8:         $\left(x^{BEST}, y^{BEST}\right) \leftarrow \left(x^k, y\left(x^k\right)\right)$
 9:     end if
10:     $MIP^{k+1} \leftarrow$ add $(NG_0)$ in $x^k$ to $MIP^k$

            $$\sum_{i:x_i^k=1}(1-x_i)+\sum_{i:x_i^k=0}x_i \geq 1$$

11:     $k \leftarrow k+1$
12: end while
13: $OPT \leftarrow BEST$; $\left(x^{OPT}, y^{OPT}\right) \leftarrow \left(x^{BEST}, y^{BEST}\right)$;
14: return OPT, $\left(x^{OPT}, y^{OPT}\right)$
```

In Algorithm 3.3.2.1, as in Algorithm 3.3.1.1, function *BestReaction* receives as input the leader's decision x^k from the optimal solution of an MIP^k, and computes a rational reaction $y\left(x^k\right)$ for the follower, that is, the KP optimum to interdiction x^k. It is easy to see that Algorithm 3.3.2.1 finds an optimal solution to DNeg. However, it is a very inefficient process and a number of improvements can be applied to make it more effective both in theory and in practice. More precisely, we will propose several improvements that lead to an enhanced and substantially faster version of Algorithm 3.3.2.1; this final version is presented in the end of this section.

Throughout the paper we use the notation of Algorithm 3.3.2.1. The leader interdiction computed in iteration k is denoted by x^k, the follower's optimal solution to x^k is denoted by

$y(x^k)$, $BEST$ and (x^{BEST}, y^{BEST}) are the minimum value and associated solution among all bilevel feasible values computed up to iteration k, and OPT and (x^{OPT}, y^{OPT}) are DNeg optimal value and associated solution. Denote by y^k the follower's optimal relaxed solution to x^k which, although not used from the algorithmic point of view, theoretically, it will play an important role.

Strengthening the Nogood Constraints. Let us first concentrate on strengthening the nogood constraints.

A feasible strategy x^k for the leader is *maximal*, if $\nexists j \in \{i : x_i^k = 0\}$ such that $\sum_{i=1}^n a_i x_i^k + a_j \leq A$. A strategy for the leader is maximal, if she does not have enough budget left to pick more items. A maximal strategy dominates an associated non-maximal strategy, since it leaves the follower with a smaller set of options: at least one further item cannot be taken by the follower due to the interdiction constraints. Algorithm 3.3.2.2 takes a not necessarily maximal strategy and turns it into a maximal one.

Algorithm 3.3.2.2 MakeMaximal

Input: An instance of DNeg and a leader's feasible solution x^k of it.
Output: A leader's maximal feasible solution containing the items of the input x^k.

```
1: $Residual \leftarrow A - \sum_{i=1}^n a_i x_i^k$
2: $i \leftarrow 1$
3: while $i \leq n$ and $Residual > 0$ do
4:    if $x_i^k = 0$ and $Residual - a_i \geq 0$ then
5:       $Residual \leftarrow Residual - a_i$
6:       $x_i^k \leftarrow 1$
7:    end if
8:    $i \leftarrow i + 1$
9: end while
10: return $x^k$
```

Once a strategy x^k for the leader and its corresponding bilevel solution $\left(x^k, y\left(x^k\right)\right)$ have been evaluated, there is no need to keep x^k feasible, because we want to concentrate in new bilevel feasible solutions potentially decreasing the follower's profit. If x^k is a maximal strategy for the leader, then $\sum_{i:x_i^k=0} x_i \geq 1$ is called a *strong maximal constraint* (NG_1). It is easy to see that a NG_1 constraint dominates a NG_0 one when both are associated with the same leader interdiction.

The strong maximal constraints can be strengthened further in the following way. Let $\left(x^k, y\left(x^k\right)\right)$ denote a bilevel feasible solution for DNeg. There is no point in generating new solutions for the leader where the set of items picked by the follower in $y\left(x^k\right)$ is available, as the follower would have a profit at least as high as the previous one. If x^k is a maximal strategy for the leader, then $\sum_{i:y_i\left(x^k\right)=1} x_i \geq 1$ is called a *nogood constraint for the follower* (NG_2).

It is easy to see that given a maximal strategy for the leader, the corresponding strong maximal constraint is dominated by the associated nogood constraint for the follower, as $y_i\left(x^k\right) = 1$ implies $x_i^k = 0$. If $\left(x^k, y(x^k)\right)$ is not the optimal solution of DNeg then, under the strategy $y(x^k)$, the follower is packing an item interdicted in any optimal solution. This establishes the validity of the nogood constraints for the follower.

Thus, at each iteration k of the algorithm in which the (standard) nogood cuts are replaced by the follower's nogood cuts, either an optimal solution has already been obtained or any optimal strategy for the leader satisfies all the follower's nogood constraints already added. This shows the correctness of the substitution of (standard) nogood with follower's nogood constraints.

A further strengthening of the follower's nogood constraints can be achieved by paying close attention to the cutting plane approach described in Section 3.3.1.

Theorem 3.3.4. *Consider an iteration k of Algorithm 3.3.2.1. If $BEST$ is not the optimal value of problem DNeg, then there is an optimal admissible interdiction x^* for the leader such that*

$$\sum_{i=1}^{n} b_i y_i \left(1 - x_i^*\right) \leq BEST - 1 \qquad \forall y \in \{0,1\}^n \text{ such that } \sum_{i=1}^{n} w_i y_i \leq B. \tag{3.3.13}$$

Proof. Let (x^*, y^*) be an optimal solution of DNeg. Then

$$\sum_{i=1}^{n} y_i b_i \left(1 - x_i^*\right) \leq \sum_{i=1}^{n} b_i y_i^* \qquad \forall y : \quad \sum_{i=1}^{n} w_i y_i \leq B.$$

Moreover, if $BEST$ at iteration k is not an optimal value of DNeg, then $\sum_{i=1}^{n} b_i y_i^* \leq BEST - 1$. □

With the help of Theorem 3.3.4, it is easy to derive the following new type of valid constraints, to be introduced in each iteration k to strengthen MIP^k:

$$(NG_3) \quad \textit{cutting plane constraint} \qquad \sum_{i=1}^{n} y_i\left(x^k\right) b_i \left(1 - x_i\right) \leq BEST - 1. \tag{3.3.14}$$

In this way, whenever $BEST$ is updated in the iterative procedure, also the right-hand-sides of the previous cutting plane constraints are updated.

It is easy to show that a cutting plane constraint dominates a follower's nogood constraint when associated with the same leader interdiction. Indeed, after solving MIP^k in an arbitrary iteration k, a best reaction of the follower to x^k is computed and then it is checked whether this leads to a better solution for DNeg. At that point, the following inequality holds:

$$\sum_{i=1}^{n} b_i y_i \left(x^k\right) \geq BEST.$$

Hence, in order to satisfy the associated cutting plane constraint

$$\sum_{i=1}^{n} y_i \left(x^k\right) b_i \left(1 - x_i\right) \leq BEST - 1,$$

the leader must interdict at least one item packed with the strategy $y\left(x^k\right)$.

Next, the general dominance of the cutting plane constraints over the remaining presented ones is established.

Proposition 3.3.5. *Consider Algorithm 3.3.2.1 amended by making the leader's strategy maximal (after step 4) (call it Algorithm$_0$) and replacing the nogood constraint (step 10) by*

- *Algorithm$_1$: the strong maximal constraint;*
- *Algorithm$_2$: the follower's nogood constraint;*
- *Algorithm$_3$: the cutting plane constraint.*

Assume that if in an iteration k, Algorithm$_2$ and Algorithm$_3$ have a common optimal interdiction x^k then, both select x^k and the same associated best reaction $y(x^k)$. Then, for $i = 1, 2, 3$, Algorithm$_i$ returns the optimal solution after a number of iterations less or equal than Algorithm$_{i-1}$.

Proof. For Algorithm$_i$ denote as $MIP^{k,i}$ and $F^{k,i}$ the optimization problem MIP^k and the associated feasible region for the leader maximal interdictions at iteration k. Define $F^{k,i}$ as equal to the empty set if Algorithm$_i$ had returned the optimal solution in a number of iterations less or equal to k. Denote $x^{k,i}$ as the leader optimal solution to $MIP^{k,i}$.

For each Algorithm$_i$ note that the purpose of each iteration k is to cut off non optimal leader's maximal interdictions, therefore it is enough to concentrate on the set $F^{k,i}$. In other words, it is sufficient to show that $F^{k,i} \subseteq F^{k,i-1}$ holds for any iteration k since it directly implies that Algorithm$_i$ enumerates a less or equal number of bilevel feasible solutions in comparison with Algorithm$_{i-1}$. We will prove that this result holds for $i = 1, 2$ through induction in k.

In the first iteration, $k = 1$, all algorithms solve the same MIP^1 and thus, $F^{1,2} = F^{1,1} = F^{1,0}$.

Next, assume that $F^{m,i} \subseteq F^{m,i-1}$ holds for $m = k$. The induction hypothesis implies that the optimal solution value of $MIP^{m,i-1}$ is a lower bound to $MIP^{m,i}$.

Recall that we have argued before that for the same leader interdiction: the nogood constraint is dominated by the strong maximal constraint; the strong maximal constraint is dominated by the follower's nogood constraint.

By contradiction, suppose that $F^{m+1,i} \not\subseteq F^{m+1,i-1}$. This implies the existence of $x \in F^{m+1,i}$ such that $x \notin F^{m+1,i-1}$. Since $F^{m+1,i} \subset F^{m,i} \subseteq F^{m,i-1}$, then $x \in F^{m,i-1}$. Therefore, x only violates the additional constraint of $F^{m+1,i-1}$ associated with $F^{m,i-1}$. This is only possible if x is the optimal solution of $MIP^{m,i-1}$. Because $MIP^{m,i-1}$ provides a lower bound to $MIP^{m,i}$ and $x \in F^{m,i}$, x is the optimal solution of $MIP^{m,i}$. However, this means that x will be cut off from $F^{m,i}$ and thus $x \notin F^{m+1,i}$, leading to a contradiction.

It remains to prove that Algorithm$_3$ finishes in a number of iterations less or equal than Algorithm$_2$. To this end the following assumption is necessary.

As mentioned before, in the first iteration $MIP^{1,2} = MIP^{1,3}$ and thus, by the proposition assumption, $y(x^{1,2}) = y(x^{1,3})$. This fact, implies that $MIP^{2,2} = MIP^{2,3}$ since $BEST = \sum_{i=1}^{n} p_i y_i(x^{1,2})$ means that the NG_3 constraint is equivalent to NG_2 with respect to $y(x^{1,2})$. Moreover, $y(x^{2,2}) = y(x^{3,2})$ and, consequently, the associated NG_3 constraint dominates NG_2. We conclude that $F^{3,3} \subseteq F^{3,2}$. At this point, Algorithm$_3$ has advantage over Algorithm$_2$ because the set of interdictions $F^{3,3}$ is at most as large as $F^{2,3}$. Note that if there is an iteration $k \geq 3$ such that $y(x^{k,3}) \neq y(x^{k,2})$ then, Algorithm$_3$ is reducing the set of feasible interdictions through NG_3 associated with $y(x^{k,3})$ and Algorithm$_3$ might end up computing $y(x^{k,2})$ latter on in an iteration $m > k$ which shows that Algorithm$_3$ progresses more or as fast as Algorithm$_2$. $\square$

We conclude this series of cut improvements with two observations. First, the improvements described above are purely based on the fact that we are dealing with an interdiction problem. Hence, any type of interdiction problem for which we can prove an adaptation of Theorem 3.3.2 can be attacked by the basic iterative method with cutting plane constraints. Secondly, all constraints described so far depend solely on the decision variables of the leader. Therefore, the statement of Theorem 3.3.2 also applies to all improvements, and each MIP^k is equivalent to a bilevel optimization problem in which the follower solves a relaxed knapsack problem.

Stopping Criteria. Our next goal is to add a condition for the whole algorithm to stop. Let $b_{max} = \max_{i=1,\dots,n} b_i$.

Proposition 3.3.6. *At an iteration k of the Basic Iterative Method, $BEST$ cannot be decreased in the current and forthcoming iterations if*

$$\sum_{i=1}^{n} b_i y_i^{BEST} + b_{max} \leq \sum_{i=1}^{n} b_i y_i^k.$$

Proof. Let OPT be the optimal value to DNeg and assume that the proposition condition holds. For any leader's optimal solution x^*, Corollary 2.2.8 implies that the optimal value of the follower's continuous knapsack with interdiction x^* lies within the interval

$$[OPT, OPT + b_{max}] \,. \tag{3.3.15}$$

Because y^{BEST} is the follower's strategy corresponding to the best solution computed up to iteration k, obviously,

$$\sum_{i=1}^{n} b_i y_i^{BEST} + b_{max} \geq OPT + b_{max}.$$

Then $\sum_{i=1}^{n} b_i y_i^k$ is not in the range (3.3.15) which implies that x^k is not an optimal interdiction. Furthermore, since the optimal value of the MIPs is monotonically increasing with the algorithm iterations, none of the upcoming iterations returns a leader's optimal solution. $\square$

In other words, the quantity b_{max} is an upper bound on the amount by which the continuous solution value of any follower's reaction can decrease. If $\sum_{i=1}^{n} b_i y_i^k - b_{max}$ is already bigger than the current incumbent solution value, then no further improvement is possible since (of course) $\sum_{i=1}^{n} b_i y_i^{k+1} \geq \sum_{i=1}^{n} b_i y_i^k$.

Saving some Knapsack Computations. In an iteration k of our algorithm, the leader's interdiction just built may lead to an improvement if the following necessary condition holds. The following observation follows from Corollary 2.2.8.

Proposition 3.3.7. *At an iteration k, the pair $\left(x^k, y\left(x^k\right)\right)$ does not decrease $BEST$ if*

$$\sum_{i=1}^{n} b_i y_i^k - b_{c^k} y_{c^k}^k \geq \sum_{i=1}^{n} b_i y_i^{BEST},$$

where c^k is the critical item for the follower's continuous knapsack with interdiction x^k.

Thus, whenever the above condition is violated, we do not need to compute the best reaction by solving the associated KP. Our next goal is to embed the condition of Proposition 3.3.7 as a constraint inside MIP^k. For that purpose, the following lemma and theorem will be crucial. Lemma 3.3.8 follows from Corollary 2.2.8.

Lemma 3.3.8. *Let x^k be a leader's interdiction. Then*

$$\sum_{i=1}^{n} b_i y^k - \sum_{i=1}^{n} b_i y_i \left(x^k\right) \leq b_{c^k}.$$

Note that b_{c^k} provides yet another upper bound on the value of the improvement due to *BestReaction*. The following theorem makes the upper bound independent of the critical item computation. Let $w_{max} = \max_{i=1,\dots,n} w_i$.

Theorem 3.3.9. *Let x^k be a leader's interdiction. Then, for the corresponding follower's relaxed rational reaction to x^k there exists a dual solution that satisfies*

$$z_0^k w_{max} \geq \sum_{i=1}^{n} b_i y_i^k - \sum_{i=1}^{n} b_i y_i \left(x^k\right).$$

Proof. By Theorem 2.2.7, there exists a solution in which *at most one* entry of y^k is not binary in the relaxed best reaction to x^k; furthermore, if such an entry does exist then its value equals $y^k_{c^k}$. By the complementary slackness Property 2.2.3, there is a corresponding optimal dual solution with $z^k_{c^k} = 0$. The c^k dual constraint (3.3.8f) implies

$$z_0^k w_{c^k} \geq b_{c^k} \Rightarrow z_0^k w_{max} \geq z_0^k w_{c^k} \geq b_{c^k}.$$

By using Lemma 3.3.8, we get

$$z_0^k w_{max} \geq z_0^k w_{c^k} \geq b_{c^k} \geq \sum_{i=1}^{n} b_i y^k - \sum_{i=1}^{n} b_i y_i \left(x^k\right).$$

Otherwise, if all follower's variables are binary

$$\sum_{i=1}^{n} b_i y^k - \sum_{i=1}^{n} b_i y_i \left(x^k\right) = 0 \leq z_0^k w_{max},$$

because $z_0^k \geq 0$. $\square$

In order to use the upper bound derived above, the following proposition establishes yet another necessary condition which is similar in spirit to Proposition 3.3.7.

Proposition 3.3.10. *At an iteration k, $BEST$ will not decrease if*

$$\sum_{i=1}^{n} b_i \lfloor y_i^k \rfloor > BEST - 1.$$

In other words, if we round down the relaxed rational reaction of the follower to the leader strategy x^k, then the resulting feasible solution for the follower has a profit strictly smaller than the best bilevel feasible bound known. Because of Theorem 3.3.9

$$\sum_{i=1}^{n} b_i y^k - \sum_{i=1}^{n} b_i y_i \left(x^k\right) \leq \sum_{i=1}^{n} b_i y^k - \sum_{i=1}^{n} b_i \lfloor y_i^k \rfloor \leq b_{c^k} \leq z_0^k w_{max},$$

and it is easy to see that also the following holds:

$$z_0^k B + \sum_{i=1}^{n} \overbrace{\left(1 - x_i^k\right) z_i^k}^{u_i^k} - z_0^k w_{max} \leq BEST - 1. \tag{3.3.16}$$

The following theorem turns condition (3.3.16) into an inequality that can be added to MIP^k.

Theorem 3.3.11. *In the end of iteration k, the strong cut*

$$z_0 B + \sum_{i=1}^{n} u_i - z_0 w_{max} \leq BEST - 1,$$

is valid for MIP^{k+1}.

Proof. The dual of the follower's relaxed problem with the introduction of the strong cut (and replacing u_i) is

$$(Dual)\ \min_{z \geq 0} \quad z_0 B + \sum_{i=1}^{n} \left(1 - x_i^k\right) z_i \tag{3.3.17a}$$

$$\text{s. t.} \quad w_i z_0 + z_i \geq b_i \qquad \text{for } 1 = 1, \ldots, n. \tag{3.3.17b}$$

$$z_0 B + \textstyle\sum_{i=1}^{n} \left(1 - x_i^k\right) z_i - z_0 w_{max} \leq BEST - 1, \tag{3.3.17c}$$

and the follower's relaxed problem is

$$(Primal)\ \max_{y \geq 0} \quad \sum_{i=1}^{n} y_i b_i - (BEST - 1)\, y_{n+1} \tag{3.3.18a}$$

$$\text{s. t.} \quad \textstyle\sum_{i=1}^{n} y_i w_i - (B - w_{max})\, y_{n+1} \leq B \tag{3.3.18b}$$

$$y_i - \left(1 - x_i^k\right) y_{n+1} \leq 1 - x_i^k \quad \text{for } i = 1, \ldots, n. \tag{3.3.18c}$$

Essentially, we are dealing with a new item $n+1$ whose profit $-(BEST-1)$ and weight $-(B-w_{max})$ are both negative. We will show that no optimal solution will use this new item: then $y^k_{n+1}=0$ holds, and the above primal problem collapses to the previous KP continuous relaxation for which the critical item exists. Hence, let us first ignore the new item and solve the continuous knapsack as before. Let c be the critical item, and let S be the set of (indices of) items that are fully taken. Then, clearly

$$\frac{\sum_{i\in S} b_i}{\sum_{i\in S} w_i} \geq \frac{b_c}{w_c}.$$

Moreover, by Proposition 3.3.10 we may assume $\sum_{i\in S} b_i \leq BEST-1$. Finally $\sum_{i\in S} w_i \geq B - w_{max}$, as otherwise c would not be the critical item. Altogether, this yields

$$\frac{BEST-1}{B-w_{max}} \geq \frac{\sum_{i\in S} b_i}{\sum_{i\in S} w_i} \geq \frac{b_c}{w_c}.$$

As the profit-to-weight ratio of the new item is at least as large as the profit-to-weight ratio of the critical item and as profit and weight of the new item are negative, the new item will not be used in an optimal solution. □

In the next section we will show that this cut is crucial in practice, as it significantly reduces the number of leader interdictions in the enumeration. This is the reason why the iterative approach is currently superior to the cutting plane (CP) approach. It is relatively easy to embed additional conditions to reduce the search space of the iterative approach, whereas additional cutting planes to enhance CP seem difficult to be developed.

Pre-processing. For the approach developed so far, it is crucial to compute good upper bounds to the profit and weight of the items that may act as critical item. We describe a pre-processing routine that tightens these bounds and hence leads to a stronger approach.

Recall that in this context we are dealing with the continuous relaxation of KP for the follower. Suppose that the follower could pack all the items from 1 to $c-1$ as illustrated in Figure 3.3.2. Since the follower has incentive to fully pack the available items from 1 to $c-1$, these items can never be critical. Another interesting observation is that some of the less valuable items for the follower are never packed by her and hence are not critical: this occurs because the follower uses all her budget on the most valuable available items. All in all, we are interested in computing a bound on the maximum follower's weight interdicted by the leader. This trivially can be achieved by solving the following relaxed

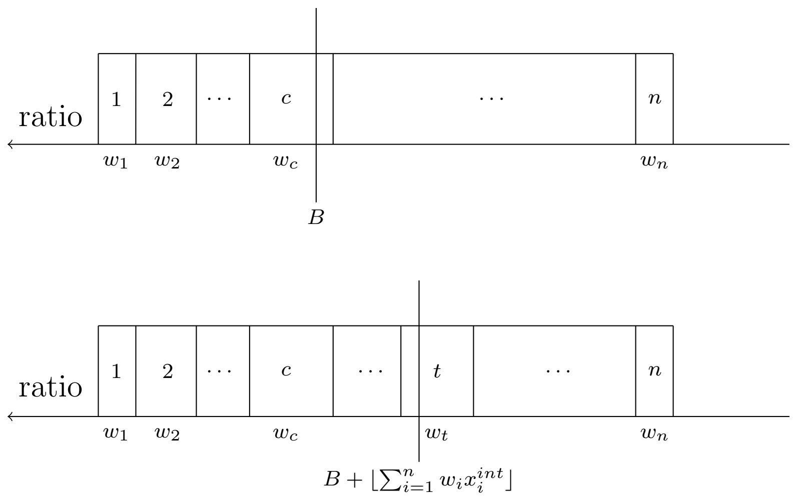


Figure 3.3.2: Illustration of the follower's preferences when her knapsack is relaxed: items from 1 to $c-1$ and from $t+1$ to n are never critical.

KP:

$$x^{int} = \arg\max_{x\in[0,1]^n} \sum_{i=1}^n w_i x_i \tag{3.3.19a}$$

$$\text{s. t.} \qquad \sum_{i=1}^n a_i x_i \leq A. \tag{3.3.19b}$$

Therefore, the leader interdicts at most $\lfloor\sum_{i=1}^n w_i x_i^{int}\rfloor$ of the total available weight of the follower. It is easy to see from Figure 3.3.2 that the items from $t+1$ to n are never critical. In conclusion, with $t = \min\{j : B + \lfloor\sum_{i=1}^n w_i x_i^{int}\rfloor \leq \sum_{i=1}^j w_i\}$ we have

$$b_{max} = \max_{i=c,\dots,t} b_i \qquad \text{and} \qquad w_{max} = \max_{i=c,\dots,t} w_i.$$

The running time of this pre-processing is $O(n \log n)$, and hence slightly more expensive than the simple $O(n)$ procedure by computing b_{max} and w_{max} by taking all n items.

We could improve these bounds even further by adding so-called sensitive intervals for identifying the critical item candidates; see [18]. However, this comes at the cost of adding more constraints to our MIPs. For that reason, we will apply this improvement only to the very hard instances as explained in Section 3.3.3.

CCLW algorithm. Our main algorithm is summarized in Algorithm 3.3.2.3. For ease of reference, we call it the Caprara-Carvalho-Lodi-Woeginger Algorithm (CCLW).

Algorithm 3.3.2.3 CCLW

Input: An instance of DNeg.

Output: Optimal value and an optimal solution to DNeg.

```
 1: Compute $b_{max}, w_{max}$ according to the Pre-processing
 2: $k \leftarrow 1$; $BEST \leftarrow +\infty$;
 3: Build $MIP^k$
 4: while $MIP^k$ is feasible do
 5:    $x^k \leftarrow \arg\min\{MIP^k\}$
 6:    if $BEST + b_{max} \leq$ Optimal value of $MIP^k \left(= \sum_{i=1}^{n} b_i y_i^k\right)$ then
 7:       STOP;
 8:    else
 9:       $x^k \leftarrow MakeMaximal\left(x^k\right)$
10:       $y\left(x^k\right) \leftarrow BestReaction\left(x^k\right)$ // solves the follower's KP by fixing $x^k$
11:       if $\sum_{i=1}^{n} b_i y_i\left(x^k\right) < BEST$ then
12:          $BEST \leftarrow \sum_{i=1}^{n} b_i y_i\left(x^k\right)$;
13:          $\left(x^{BEST}, y^{BEST}\right) \leftarrow \left(x^k, y\left(x^k\right)\right)$
14:          $MIP^{k+1} \leftarrow$ if $k = 1$ add *strong cut*

                 $$z_0 B + \sum_{i=1}^{n} u_i - z_0 w_{max} \leq BEST - 1,$$

             otherwise update the right hand side of the *strong cut* and $NG_3$s with $BEST$-1.
15:       end if
16:       $MIP^{k+1} \leftarrow$ add $NG_3$ in $y\left(x^k\right)$ to the $MIP^k$ :

                 $$\sum_{i: y_i\left(x^k\right)=1} b_i\left(1 - x_i\right) \leq BEST - 1$$

17:    end if
18:    $k \leftarrow k + 1$
19: end while
20: $OPT \leftarrow BEST$;$\left(x^{OPT}, y^{OPT}\right) \leftarrow \left(x^{BEST}, y^{BEST}\right)$;
21: return $OPT$, $\left(x^{OPT}, y^{OPT}\right)$
```

3.3.3 Computational Results

In this section we computationally evaluate the algorithms from the preceding section in two phases. First, we compare CCLW with CP. There we also discuss the importance of the main ingredients of algorithm CCLW, as well as the structural difficulty of DNeg instances with respect to our algorithms. Secondly, we compare CCLW with the results of [41] and [42].

All algorithms have been coded in Python 2.7.2, and each MIP has been solved with Gurobi 5.5.0. The experiments were conducted on a Quad-Core Intel Xeon processor at 2.66 GHz and running under Mac OS X 10.8.4.

Method Comparisons CP and CCLW will be compared against each other. Moreover, we will discuss the structural difficulty of bilevel knapsack instances with respect to the performance of CCLW.

Generation of instances. For building the follower's data, we have used the knapsack generator described in [86]; the profits b_i and weights w_i are taken with uncorrelated coefficients from the interval $[0, 100]$. For each value n, 10 instances were generated; these instances are available upon request. According to [86], the budget B is set to $\lceil \frac{\text{INS}}{11} \sum_{i=1}^{n} w_i \rceil$ for the instance number "INS". The leader's data, a_i and A all were generated by using Python's random module; see [51]. In particular, a_i and A were chosen uniformly at random from $[0, 100]$ and $[B - 10, B + 10]$, respectively. Note that if the leader's budget is significantly smaller than the follower's budget, then there are fewer feasible solutions for the leader and the instance would be easier. On the other hand, if the leader's budget is significantly bigger than the follower's budget, then all the items may be packed by leader and follower together, and again the instance would be easier. We will see below that CCLW is very efficient for these cases.

CP versus CCLW. In an attempt of asserting the importance of each ingredient of algorithm CCLW, we performed some tests with its basic scheme (Algorithm 3.3.2.1). It turned out that within one hour of CPU time, the Basic Iterative Method can only solve instances with up to 15 items. Although this is comparable to the size of problems reported in [41, 42] (discussed in detail in the end of this section), both CP and CCLW can go much higher in terms of number of items. For this reason, no detailed results for Algorithm 3.3.2.1 are reported here.

Table 3.1 reports the results of algorithms CP and CCLW. For each instance, the table shows the number of items ($n \in \{35, 40, 45, 50\}$), the instance identifier ("INS"), and the optimal value ("OPT"). For algorithm CP, we further report the number of cutting plane iterations ("#It.s"), and the CPU time in seconds ("time"), while for algorithm CCLW we report the value of MIP^1 ("ObjF"), the number of iterations ("#MIPs"), the iteration in which the optimal solution has been found ("OPT_{Iter}"), and the CPU time in seconds ("time"). Finally, for algorithm CCLW we also report some data on the most expensive MIP solved, namely the CPU time in seconds ("WMIP time") and the number of nodes ("WMIP nodes"). The algorithms had a limit of one hour to solve each instance. The red entries (in square brackets) mark the cases where algorithm CP reached the time limit, and in such cases we report the lower bound value instead of the computing time.

The results in Table 3.1 clearly illustrate that algorithm CCLW is superior to algorithm CP. In particular, CCLW usually finds an optimal solution within 2 iterations, which shows that in practice we will find the optimum very early and the only challenge is to prove optimality. Looking at the number of MIPs solved and at the computing times, we observe that for any number of items algorithm CCLW is extremely powerful for instances with INS ≥ 5. An optimal solution is computed by MIP^1 and optimality is proved by MIP^2, except in three cases with INS $= 5$. Considering the way in which the instances are generated, the next theorem shows that this behavior is structural.

Theorem 3.3.12. *If for any leader's maximal interdiction the follower can pack the remaining items, then CCLW solves DNeg in two iterations.*

Proof. Given that the follower is able to pack all the items left by any maximal interdiction of the leader, we get that the follower's budget constraint is not binding. In particular, the solution of the follower's relaxed problem to any leader's maximal interdiction is binary. Hence, the MIPs' optimal values are bilevel feasible and the DNeg optimum is consequently found in the first iteration of CCLW.

In the second iteration, MIP^2 uses the additional strong cut

$$z_0 B + \sum_{i=1}^{n} u_i - z_0 w_{max} \leq BEST - 1.$$

The dual variable z_0 corresponds to the follower's budget constraint (3.3.6c). As initially noted, constraint (3.3.6c) is not binding which together with the complementary slackness Property 2.2.3 implies that the associated optimal dual solution has $z_0 = 0$. However, with $z_0^2 = 0$ the strong cut imposes

$$\sum_{i=1}^{n} u_i^2 \leq BEST - 1.$$

			CP		CCLW					
									WMIP	WMIP
n	INS	OPT	#It.s	time	ObjF	#MIPs	OPT$_{\text{iter}}$	time	time	nodes
35	1	279	16	0.34	288.07	14	2	0.79	0.05	14
	2	469	40	1.59	474.00	33	1	2.57	0.09	171
	3	448	253	55.61	455.88	203	1	40.39	0.50	1,635
	4	370	397	495.50	374.56	11	1	1.48	0.14	363
	5	467	918	[451]	472.00	5	2	0.72	0.19	660
	6	268	155	71.43	268.00	2	1	0.06	0.03	0
	7	207	298	144.46	207.00	2	1	0.06	0.03	0
	8	41	11	0.25	41.00	2	1	0.04	0.01	0
	9	80	25	0.97	80.00	2	1	0.03	0.00	0
	10	31	8	0.12	31.00	2	1	0.03	0.00	0
40	1	314	24	0.66	326.12	21	1	1.06	0.05	60
	2	472	77	6.67	483.78	67	2	7.50	0.19	805
	3	637	338	324.61	644.78	244	1	162.80	2.52	4,521
	4	388	530	1,900.03	396.56	3	1	0.34	0.13	165
	5	461	653	[457]	466.18	2	1	0.22	0.15	66
	6	399	534	2,111.85	399.00	2	1	0.09	0.04	0
	7	150	254	83.59	150.00	2	1	0.05	0.02	0
	8	71	33	1.73	71.00	2	1	0.04	0.01	0
	9	179	404	137.16	179.00	2	1	0.08	0.03	4
	10	0	2	0.03	0.00	2	1	0.03	0.00	0
45	1	427	45	1.81	434.60	33	1	2.37	0.08	74
	2	633	97	13.03	642.36	74	1	11.64	0.25	903
	3	548	845	[547]	558.69	387	1	344.01	2.86	10,638
	4	611	461	[566]	624.84	108	1	38.90	1.01	8,611
	5	629	462	[568]	630.00	15	7	3.42	0.30	1,179
	6	398	639	3,300.76	398.00	2	1	0.07	0.03	0
	7	225	141	60.43	225.00	2	1	0.04	0.01	0
	8	157	221	60.88	157.00	2	1	0.05	0.01	0
	9	53	23	0.83	53.00	2	1	0.05	0.01	0
	10	110	11	0.40	110.00	2	1	0.05	0.01	0
50	1	502	58	2.86	514.12	39	1	4.55	0.12	114
	2	788	733	1,529.16	798.0	695	2	1,520.56	7.29	6,352
	3	631	467	[612]	638.47	212	1	105.59	2.03	7,909
	4	612	310	[586]	621.04	17	1	3.64	0.32	954
	5	764	287	[657]	768.88	3	1	0.60	0.27	369
	6	303	385	1,046.85	303.00	2	1	0.05	0.01	0
	7	310	617	2,037.01	310.00	2	1	0.09	0.04	0
	8	63	49	2.79	63.00	2	1	0.05	0.01	0
	9	234	717	564.97	234.00	2	1	0.10	0.05	3
	10	15	5	0.09	15.00	2	1	0.04	0.01	0

Table 3.1: Comparison between CP and CCLW.

This means that the optimal value of MIP^2 is strictly better then the value obtained in MIP^1. But this is absurd, as MIP^2 equals MIP^1 plus an additional constraint (the strong cut). Consequently MIP^2 is infeasible, and CCLW stops in the second iteration. □

As INS increases its value, larger budget capacities are associated with the leader and the follower. Therefore, it is likely that these instances fall into the condition of Theorem 3.3.12.

Strength of the CCLW Ingredients. In order to evaluate the effectiveness of CCLW main algorithmic ingredients, we have performed two additional sets of experiments. First, we considered what happens to the basic enumerative scheme (Algorithm 3.3.2.1) if it is strengthened by the nogood cuts (NG_3) described in Section 3.3.2. The results are reported in Table 3.2 for instances with $n \in \{30, 35\}$.

							WMIP	WMIP
n	INS	OPT	ObjF	#MIPs	OPT_{iter}	time	time	nodes
30	1	272	282.80	13	2	0.27	0.02	9
	2	410	423.29	34	1	0.95	0.04	223
	3	502	513.63	110	1	10.56	0.28	1,036
	4	383	385.00	151	2	36.65	1.06	7,094
	5	308	308.00	301	1	121.27	1.85	7,730
	6	223	223.00	239	1	44.22	0.81	5,580
	7	146	146.00	121	1	8.32	0.15	1,072
	8	88	88.00	70	1	2.03	0.05	281
	9	113	113.00	83	1	2.71	0.07	674
	10	82	82.00	73	1	1.99	0.04	276
35	1	279	288.07	19	2	0.72	0.04	16
	2	469	474.00	53	1	3.20	0.08	524
	3	448	455.88	303	1	102.23	1.31	2,673
	4	370	374.56	474	1	1,203.90	19.49	74,265
	5	467	472.00	1,152	2	tl	9.30	26,586
	6	268	268.00	234	1	222.66	5.78	35,510
	7	207	207.00	471	1	321.08	3.97	28,962
	8	41	41.00	42	1	1.24	0.04	49
	9	80	80.00	98	1	5.28	0.09	285
	10	31	31.00	33	1	0.85	0.03	9

Table 3.2: Algorithm 3.3.2.1 with strengthened nogood constraints (NG_3).

The results in Table 3.2 show that this (simple) strengthening already allows us to double the size of the instances that the basic scheme can settle (recall the discussion at the beginning of the previous section). More precisely, all instances with 30 items can be solved to optimality in rather short computing times, whereas size 35 becomes troublesome.

If we compare these results to the corresponding results in Table 3.1, we note that the number of MIPs needed to prove optimality is much bigger, in particular for the cases INS $\geq$ 3. This behavior becomes dramatic for INS $\geq$ 5 where CCLW generally proves optimality in 2 iterations (as suggested by Theorem 3.3.12), whereas the improved version of the basic scheme still needs a large number of iterations. The difference in behavior seems to be mainly caused by the strong cut presented in Theorem 3.3.11. This observation is also confirmed by our second set of experiments, in which we removed the strong cut from algorithm CCLW. The corresponding results are reported in Table 3.3. Indeed, the results in Table 3.3 illustrate that without the strong cut, the number of MIPs required by CCLW blows up significantly. The algorithm is only slightly better (because of the stopping criteria) than the basic iterative scheme with strengthened nogood cuts (see Table 3.2).

							WMIP	WMIP
n	INS	OPT	ObjF	#MIPs	OPT$_{\text{iter}}$	time	time	nodes
35	1	279	288.07	14	2	0.89	0.04	16
	2	469	474.00	33	1	1.76	0.05	207
	3	448	455.88	218	1	43.27	0.50	1,443
	4	370	374.56	277	1	216.96	2.40	14,651
	5	467	472.00	1,152	2	tl	9.26	26,586
	6	268	268.00	59	1	3.76	0.10	756
	7	207	207.00	202	1	25.86	0.27	1,667
	8	41	41.00	21	1	0.62	0.03	49
	9	80	80.00	30	1	1.06	0.04	207
	10	31	31.00	2	1	0.03	0.00	0

Table 3.3: CCLW without the strong cut.

Solving Large(r) Instances. What are the computational limits of Algorithm CCLW? How does it scale to larger values of n? Table 3.4 provides some partial answers to these questions by displaying the results for CCLW on instances with 55 items. Again, we see that MIP^1 is very effective in computing the leader's strategy, as in most of the cases we obtain the optimal DNeg solution already at iteration 1. In general, the machinery discussed in the previous sections seems to be able to keep the enumeration of leader strategies under control: CCLW succeeds in solving all but two instances. The two exceptions are the instances with INS $\in \{3, 4\}$, on which CCLW exceeded its time limit of 1 hour of CPU time (the "tl" entries in the table).

			CCLW					
							WMIP	WMIP
n	INS	OPT	ObjF	#MIPs	OPT_{iter}	time	time	nodes
55	1	480	489.21	103	2	18.57	0.37	1,090
	2	702	706.15	419	1	443.53	4.33	11,097
	3	778	783.67	926	1	tl	8.85	21,491
	4	889	899.34	787	1	tl	14.67	41,813
	5	726	726.00	2	1	0.24	0.13	158
	6	462	462.00	2	1	0.09	0.04	0
	7	370	370.00	2	1	0.08	0.03	0
	8	387	387.00	2	1	0.10	0.04	0
	9	104	104.00	2	1	0.06	0.01	0
	10	178	178.00	2	1	0.06	0.02	0

Table 3.4: CCLW computational results on instances with $n = 55$.

For the most challenging instances, we implemented a pre-processing step based on the idea of computing sensitive intervals (as done in [18]). Ideally, in each iteration k of CCLW we would like to know the profit b_{c^k} of the critical item in the optimal solution for the follower's continuous knapsack. (Recall Theorem 3.3.9 which shows that $z_0^k w_{max}$ is an upper bound on b_{c^k} in each iteration k.) To reach this goal, we compute sensitive intervals with the function

$$\phi(Z_0^+ \longrightarrow Z_0^+) : \sum_{i=1}^{c} w_i x_i \longrightarrow \max_{i=c',\dots,t} b_i, \tag{3.3.20}$$

where $c' = \min\{j : \sum_{i=1}^{c} w_i x_i + B \leq \sum_{i=1}^{j} w_i\}$. In this way, instance INS = 4 in Table 3.4 was solved within the time limit. The computation took 2,796.20 CPU seconds, and the speed-up was mainly due to a strong reduction in the number of MIPs (693 versus *at least* 787). In principle, sensitivity interval pre-processing could achieve the same kind of reduction in all considered instances. Note however that this pre-processing adds 5 constraints and up to n binary variables to every MIP solved by CCLW. Hence, there is a tradeoff between performing fewer iterations and working with larger MIPs, and this is also the reason why we decided not to include sensitivity interval pre-processing in the standard version of CCLW: it slightly slows down the computing time, whereas only few additional hard instances can be solved with it. (Note that it does not manage to solve the instance $n = 55$ and INS = 3 to optimality.)

All in all, we conclude that new algorithmic ideas will be needed to attack the hard instances with INS ≤ 4 for larger values of n. For instance for $n = 100$, computation times of 1 hour CPU time (as we reached for the smaller instances in this section) seem currently out of reach.

Literature Comparison. DeNegre [41] and DeNegre and Ralphs [42] solved knapsack interdiction instances by using the Branch-and-Cut procedure described in Section 2.3.1.1. These authors present two branching strategies: maximum infeasibility and strong branching. We compare our method CCLW against these two procedures in Table 3.5 (the instances have kindly been provided by the authors of [41, 42]). The data in the table averages over 20 instances, and the computing times for [42] refer to an Intel Xeon 2.4GHz processor with 4GB of memory. A "-" indicates that due to memory requirements, no instance of the corresponding size was solved.

	Branch and Cut [42]		
	Maximum Infeasibility	Strong Branching	CCLW
n	Avg CPU time	Avg CPU time	Avg CPU time
10	3.17	4.69	0.009
11	6.63	9.13	0.009
12	13.27	17.50	0.009
13	27.54	35.84	0.010
14	60.08	71.90	0.011
15	124.84	145.99	0.011
16	249.19	296.16	0.014
17	516.65	-	0.013

Table 3.5: Summary of results for instances in [41, 42].

Although it is always difficult to compare different computing codes running on different computers, we believe that from the results in Table 3.5 it is safe to conclude that, for these instances, CCWL outperforms the Branch-and-Cut method. In particular, the highest average number of Branch-and-Bound nodes explored by Gurobi for solving the MIPs is 4.55 for the instances with $n = 16$, thus the impact of the parallelism associated with our computing platform to be Quad-Core is negligible. We noticed that in all the instances introduced in [41, 42], CCLW executes only two iterations and the optimum is always found in the first iteration. The second iterations are only needed to prove optimality, due to the fact that both leader and follower have enough capacity to pack all the items. Theorem 3.3.12 shows that in these cases the strong cut makes MIP^2 infeasible.

3.3.4 Summary

We have analyzed a special class of interdiction problems and proposed an exact algorithm for solving it. Our method uses a new way of generating (enumerating) solutions, which seems to hit the optimal solution at a very early stage and thus allows us to concentrate on techniques for proving optimality. This behavior is quite different from classical Branch-and-Bound methods, which usually starts from infeasible (super-optimal) solutions and apply extensive enumerations. Of course, the classical branch-and-bound scheme has proven very effective for classical MIPs, whereas our results might indicate that this is not the case for MIBPs. Furthermore, we introduce a new cut for the leader's variables which seems to be much stronger than the ones used in the literature and which significantly decreased the number of enumerated bilevel feasible solutions. Also cuts limiting the objective function range had a big impact in speeding up the method.

We were able to solve instances with up to 100 binary variables, which is significantly larger than the size of instances solved in the literature. Our method is very efficient on instances where both leader and follower have a large budget. Consequently, the challenging and hard instances are those in which the budget of both leader and follower forces them to evaluate a large number of strategies.

The comparison of our algorithm CCLW with the best ones from the literature demonstrates its advantage, and stresses the importance that problem-specific algorithms currently have in solving bilevel programming. A promising line for future research on general interdiction problems is to exploit the follower's integrality relaxation; this is in harsh contrast to the classical high-point relaxation where the follower is forgotten as a decision-maker.

Chapter 4

Simultaneous Games

In this chapter, we will focus in simultaneous integer programming games. To warm-up, Section 4.1 investigates a simple IPG, called the *coordination knapsack game*, where each player's optimization problem is a knapsack problem. Section 4.2 describes a game in the context of kidney exchange, called the *competitive two-player kidney exchange game*, and generalizes results from matching on graphs in order to solve the game efficiently. The *competitive uncapacitated lot-sizing game* is modeled in Section 4.3 through the generalization of the classical Cournot Competition to a finite time horizon and inclusion of lot-sizing decisions in the optimization programs of each firm (player) participating in the market. The chapter concludes in Section 4.4 by classifying the complexity of simultaneous IPGs and proposing a general algorithmic approach for computing at least one approximate equilibrium.

4.1 Two-Player Coordination Knapsack Game

[1]

Our Game Model. In resemblance to the methodological motivation to study bilevel knapsack variants, we start the study of simultaneous IPGs by modeling a game which is very simple to describe. Again, the knapsack problem is in the base of our game. The two-player coordination knapsack game (CKG) consists in a game played by a set of players $M = \{A, B\}$, where each player's goal is to maximize the individual valuation over a set of n items. The optimization problem for each player $p \in M$ is

$$\max_{x^p \in \{0,1\}^n} \sum_{i=1}^{n} c_i^p x_i^A x_i^B \tag{4.1.1a}$$

$$\text{s. t.} \quad \sum_{i=1}^{n} w_i^p x_i^p \leq W^p. \tag{4.1.1b}$$

The objective (4.1.1a) models the fact that player $p \in M$ gets profit $c_i^p \geq 0$ associated with item i if and only if $x_i^A = x_i^B = 1$. In other words, the benefit is only perceived for items which are chosen by both players. Each player p has to select a subset of items that does not exceed the capacity constraint (4.1.1b) of her knapsack.

Another motivation to study the CKG is that it models situations in which a firm has to decide a set of new technologies to invest in, subject to a budget constraint and taking into account that the revenue is restricted to the technologies that were also adopted by other firms.

Literature Review. In the literature, to the best of our knowledge, the most similar game to CKG that has been studied is the two-group knapsack game by Wang *et al.* [134]. Wang *et al.* [134] consider a game in which two groups simultaneously bid (select) on a common pool of potential projects (items); the profit of a particular project can be wholly taken by the sole bidder group or shared proportionally by two group bidders according to each group power in the market. The main difference between this game model [134] and CKG is twofold: *(1)* in [134] there is a profit for sole bidders and *(2)* shared projects (items) benefit the two groups proportionally, enabling existence conditions for the game to be potential and thus, to have at least a pure equilibrium (recall Lemma 2.3.9).

[1]The results of this chapter appear in:
M. Carvalho, J. P. Pedroso. Two-Player Coordination Knapsack Game, working paper.

Our Contributions and Organization of the Section. In general, CKG is not potential; see Example 4.1.1 below. Moreover, potential function arguments only allow to determine one (pure) equilibrium. In this work, we are able to prove the existence of pure equilibria and to characterize the equilibria set.

Example 4.1.1 (CKG is not potential)**.** *Consider an CKG instance with $n = 4$ items, profits equal to $c^A = (13, 7, 5, 6)$ and $c^B = (5, 6, 7, 10)$, weights equal to $w^A = (1, 1, 1, 1)$ and $w^B = (1, 1, 1, 2)$, and total capacities equal to $W^A = 2$ and $W^B = 3$. Observe the players' utilities for the following profiles of strategies:*

$$\begin{aligned}
x^A &= (1,0,0,1) \text{ and } x^B = (1,1,1,0), & \Pi^A &= 13 \text{ and } \Pi^B = 5\\
x^A &= (1,0,0,1) \text{ and } x^B = (0,1,0,1), & \Pi^A &= 6 \text{ and } \Pi^B = 10\\
x^A &= (0,1,1,0) \text{ and } x^B = (0,1,0,1), & \Pi^A &= 7 \text{ and } \Pi^B = 6\\
x^A &= (0,1,1,0) \text{ and } x^B = (1,1,1,0), & \Pi^A &= 12 \text{ and } \Pi^B = 13.
\end{aligned}$$

By the definition of potential function $\Phi(x^A, x^B)$, the above utility' values imply that it satisfies

$$\begin{aligned}
\Phi((1,0,0,1),(1,1,1,0)) &< \Phi((1,0,0,1),(0,1,0,1))\\
\Phi((1,0,0,1),(0,1,0,1)) &< \Phi((0,1,1,0),(0,1,0,1))\\
\Phi((0,1,1,0),(0,1,0,1)) &< \Phi((0,1,1,0),(1,1,1,0))\\
\Phi((0,1,1,0),(1,1,1,0)) &< \Phi((1,0,0,1),(1,1,1,0)),
\end{aligned}$$

which is impossible.

In Section 4.1.1, we prove the existence of pure NE and reduce the computation of Pareto efficient pure NE to a two-objective optimization problem. To conclude, in Section 4.1.2, it will be shown that the utilities for any mixed equilibria of CKG lie in the convex hull formed by the utilities associated with the game pure NE.

4.1.1 Computing Pure Equilibria

If player A packs the set of items S^A, then it is easy to see that player B can restrict her best reaction to this set; let a player B's optimal response to S^A be $S^B \subseteq S^A$. It is feasible for both players to pack the items S^B and this is an equilibrium.

Lemma 4.1.2. *If selecting the set of items S satisfies constraint* (4.1.1b) *for all $p \in M$, then it is an equilibrium for both players to only pack the items S.*

As a consequence of this lemma, we conclude that any CKG has a pure equilibrium, since it is feasible for both players to pack the set of items $S = \emptyset$.

Corollary 4.1.3. *CKG has a pure equilibrium.*

As an implication of Lemma 4.1.2, the search of pure equilibria can be restricted to the strategies in which the players select exactly the same items. A profile $x = (x^A, x^B) \in X$ is called *coordination profile* if $x^A = x^B$. Given $x \in X$, a *coordination profile of* x is $\tilde{x} \in X$ such that $\tilde{x}_i^A = \tilde{x}_i^B = x_i^A x_i^B$.

Corollary 4.1.4. *For any pure equilibrium $\hat{x} \in X$, there is another equilibrium $\tilde{x} \in X$ which is a coordination profile of $\hat{x}$, and $\Pi^p(\hat{x}^A, \hat{x}^B) = \Pi^p(\tilde{x}^A, \tilde{x}^B)$ for all $p \in M$.*

The reason why this game has "coordination" in its name is based on Lemma 4.1.2 and Corollary 4.1.4: when the players choose the same set of items to pack (coordinate), an equilibrium is attained.

Each player has potentially $O(2^n)$ feasible strategies, thus the potential number of equilibria is also $O(2^n)$. In the presence of multiple equilibria, the concept of Nash equilibrium may be refined. We will concentrate in Pareto efficient equilibria (defined in Section 2.3). Pure Pareto efficient equilibria which are coordination profiles can be described through the computation of the Pareto frontier of a two-objective optimization program.

Theorem 4.1.5. *Each pure Pareto efficient equilibrium for CKG has a coordination profile which is an equilibrium and a solution of the following two-objective optimization programming problem:*

$$\max_{x \in \{0,1\}^n} \quad \left(\sum_{i=1}^n c_i^A x_i, \sum_{i=1}^n c_i^B x_i \right) \tag{4.1.4a}$$

$$s.\ t. \quad \sum_{i=1}^n w_i^A x_i \leq W^A \tag{4.1.4b}$$

$$\sum_{i=1}^n w_i^B x_i \leq W^B. \tag{4.1.4c}$$

Proof. This is a direct consequence of Corollary 4.1.4. □

If the data is integer, in order to solve the optimization (4.1.4), one could reduce this search to solving a series of MIPs: simply remove one of the objective functions, say player B's objective function, and solve the resulting one-objective optimization problem (this will provide the preferable pure NE for player A); add the constraint that player B must have a profit of at least one unit greater than the one just computed and solve this new one-objective optimization problem; repeat this process until the optimization problem becomes infeasible (player B cannot get higher profits). Alternatively, it could be applied the dynamic programming method proposed by Delort and Spanjaard [37]. This problem may be solved, *e.g.*, by SYMPHONY [122], which is a software tool that tackles two-objective MIP's.

4.1.2 Summary

We have shown that the coordination knapsack game possesses a pure Nash equilibrium, and that for each Pareto efficient profile there is an associate pure NE, which is in the set of solutions of a two-objective mixed integer programming problem. The literature is rich in proposing methods capable of handling two-objective MIPs, which is out of the scope of this thesis.

The developed work enables us to reach some conclusions about mixed equilibria of the CKG. The expected utilities for mixed equilibria are convex combinations of utilities evaluated for pure profiles. Thus, by definition of convex hull for a set, the expected utilities for mixed equilibria lie in the convex hull of the set of utilities for pure profiles. This convex hull can easily be determined given the Pareto frontier for the two-objective program (4.1.4) and the fact that each player's utility is never negative; see Figure 4.1.1. We were not able to experimentally find any instance of CKG with a Pareto efficient mixed equilibrium, thus we have the following conjecture:

Conjecture 4.1.6. *The Pareto efficient equilibria of an CKG is completely defined by its pure equilibria.*

In Section 4.4, CKG is generalized to allow more than two players, to data that may be non-positive, and to adding to each player's utility independent profits/costs for each item. This general version will be rich in (strictly) mixed equilibria. Indeed, for some instances there is no pure equilibrium.

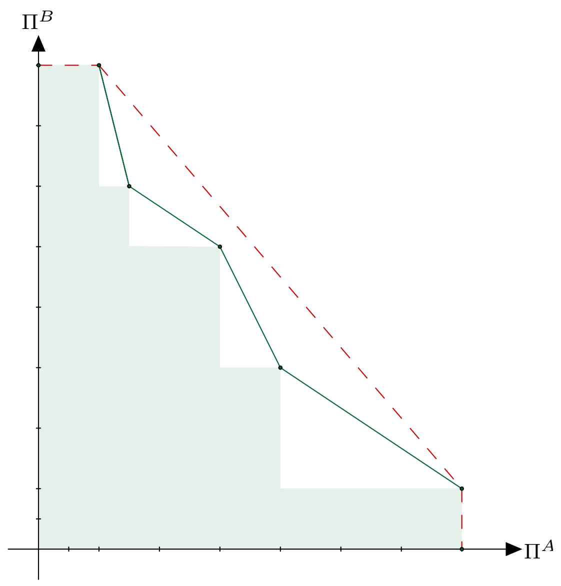


Figure 4.1.1: Pareto frontier of a CKG. The green dots represent the players' utilities in a Pareto efficient pure equilibrium; the grey area represents all the utilities that are dominated; the dashed line in red is the convex hull boundary for the set of utilities' values.

4.2 Competitive Two-Player Kidney Exchange Game

[2]

The Context. The *kidney exchange problem* can be described as follows. A patient suffering from renal failure can see her life quality improved through the transplantation of a healthy kidney. Whenever possible, a patient receives a kidney transplant from a deceased donor, or from a compatible living donor that is a patient's relative or friend. Unfortunately, these two possibilities of transplantation can only satisfy a tiny fraction of the demand, since deceased donors are scarce and patient-donor incompatibilities may occur.

To potentially increase the number of kidney transplants, some countries' recent legislation (*e.g.*, United Kingdom [84], Netherlands [36]) allows a pairwise exchange: *e.g.*, for two patient-donor pairs P_1 and P_2 the patient of pair P_1 receives a kidney from the donor of pair P_2 and vice versa. The idea can be extended to allow more than two pairs to

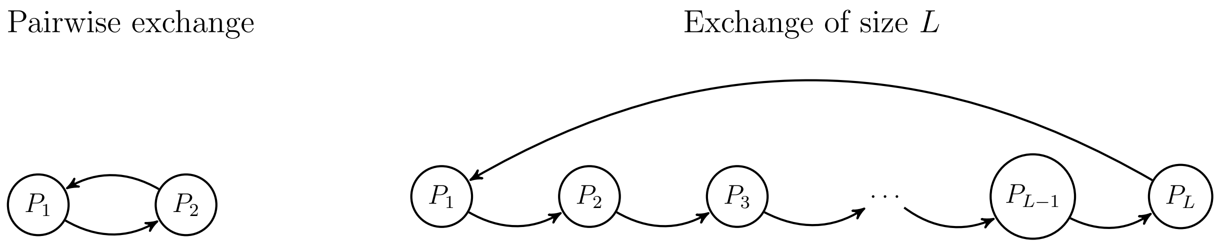


Figure 4.2.1: Kidney exchanges.

be involved in an exchange (for L-pairs, P_2 receives a kidney from the donor P_1, P_3 from the donor of P_2, etc, and, finally, P_1 from the donor of P_L, closing a cycle; see Figure 4.2.1), and to include undirected (altruistic) donors, as well as pairs with other characteristics [26]. The general aim is to define a match that maximizes the number of transplants in a pool. Because in most cases the operations must take place at the same time, for logistic reasons the number of pairs that can be involved in an exchange is limited to a maximum value, say L. Furthermore, because additional compatibility tests that must be performed prior to transplant may uncover new incompatibilities, resulting in the cancellation of all transplants involved in the cycle, it is preferable for the cycles to be shorter.

[2]The results of this section appear in:
M. Carvalho, A. Lodi, J. P. Pedroso, A. Viana. Nash Equilibria in the Two-Player Kidney Exchange Game, Mathematical Programming, 2016 (accepted).

Abraham *et al.* [1] formulated the kidney exchange program (KEP) as an integer programming problem with an exponential number of variables, which maximizes the number of vertices covered in a digraph by vertex-disjoint cycles of size at most L. In this model, the vertices of the digraph represent patient-donor pairs and the arcs represent the compatibilities between pairs. A compact model, where the number of variables and constraints increases polynomially with the problem size, has been proposed by Constantino *et al.* [26].

In the previous models, there is a centralized decision-maker deciding the exchange program. However, there are other potential decision makers to be considered that can influence the exchange program. In Cechlárová *et al.* [21], patient-donor pairs are the players in a cooperative kidney exchange game that is structurally different from what is presented in this paper because the players, the set of actions and utilities interact differently, as will be clear later with our game model description.

Multi-Agent Kidney Exchange. Although some countries have a national kidney exchange pool with the matches being done by a central authority, other countries have regional (or hospital) pools, where the matches are performed internally with no collaboration between the different entities. Since it is expected that as the size of a patient-donor pool increases more exchanges can take place, it became relevant to study kidney exchange programs involving several hospitals or even several countries. In such cases, each entity can be modeled as a self-interested agent that aims at maximizing the number of its patients receiving a kidney (see Ashlagi and Roth [5, 6]).

To the extent of our knowledge, work in this area concentrates on the search of a strategyproof mechanism that decides all exchanges to be performed in a multi-hospital setting. A mechanism is *strategyproof* if the participating hospitals do not have incentive to hide information from a central authority that decides the exchanges to be executed through that mechanism. For the 2-hospital kidney exchange program with pairwise exchanges, the deterministic strategyproof mechanism in Ashlagi *et al.* [4] provides a 2-approximation ratio on the maximum number of exchanges, while the randomized strategyproof mechanism in Caragiannis *et al.* [19] guarantees a $\frac{3}{2}$-approximation ratio. Additionally, Ashlagi *et al.* [4] built a randomized strategyproof mechanism for the multi-hospital case with approximation ratio 2, again only for pairwise exchanges. In these mechanisms, in order to encourage the hospitals to report all their incompatible pairs, social welfare is sacrificed. In fact, in [4] it is proven that the best lower bound for a strategyproof (randomized) mechanism is 2 ($\frac{8}{7}$), which implies that no mechanism returning the maximum number of exchanges is strategyproof. In this context, the question is whether, analyzing the hospitals interaction from a standpoint of a non-cooperative game, Nash equilibria would improve the program's social welfare.

A Game Model. We can formalize and generalize KEP to a competitive N-player kidney exchange game (N–KEG) with two sequential moves: first, simultaneously, each player n, for $n = 1, \dots, N$, decides the internal exchanges to be performed; second, an independent agent (IA) takes the first-stage unused pairs and decides the external exchanges to be done such that the number of pairs participating on it is maximized. Let us define V^n as the vertex set of player n, $V = \bigcup_{n=1}^{N} V^n$ and C as the set of cycles with size at most L. Let $C^n = \{c \in C : c \cap V^n = c\}$ be the subset of cycles involving only player n's patient-donor pairs, and $I = C \setminus \bigcup_{n=1}^{N} C^n$ be the subset of cycles, involving at least two patient-donor pairs of distinct players. Each player solves the following bilevel programming problem:

$$\max_{x^n \in \{0,1\}^{|C^n|}} \quad \sum_{c \in C^n} w_c^n x_c^n + \sum_{c \in I} w_c^n y_c \tag{4.2.1a}$$

$$\text{s. t.} \quad \sum_{c \in C^n : i \in c} x_c^n \leq 1 \quad \forall i \in V^n \tag{4.2.1b}$$

where y solves the problem

$$\max_{y \in \{0,1\}^{|I|}} \quad \sum_{c \in I} \sum_{n=1}^{N} w_c^n y_c \tag{4.2.1c}$$

$$\text{s.t.} \quad \sum_{c \in I : i \in c} y_c \leq 1 - \sum_{n=1}^{N} \sum_{c \in C^n : i \in c} x_c^n \quad \forall i \in V. \tag{4.2.1d}$$

Player n controls a binary decision vector x^n with size equal to the cardinality of C^n. An element x_c^n of x^n is 1 if cycle $c \in C^n$ is selected, 0 otherwise. Similarly, the IA controls the binary decision vector y with size equal to the cardinality of I. The objective function (4.2.1a) translates on the maximization of player n's patients receiving a kidney: w_c^n the number of player n's patient-donor pairs in cycle c (which is the size of c if it is an internal). Constraints (4.2.1b) ensure that every pair is in at most one cycle. The IA objective (4.2.1c) represents the maximization of patient-donor pairs receiving a kidney in the second-stage. Constraints (4.2.1d) are analogous to (4.2.1b), but also ensure that pairs participating in the first-stage exchanges are not selected by the IA.

In the way that we defined N–KEG, it is implicit that it is a complete information game. Initially, every player decides the pairs to reveal, and only revealed pairs will be considered in each player utility as well as in the second stage IA decision process. Note that there is no incentive for hiding information, as each player has complete control over her internal exchanges, and, therefore, can guarantee to be at least as good as if it was by herself. Moreover, if there were hidden pairs, they would not be considered in the IA decision, and thus, the players would not benefit from external exchanges including them. Consequently, the players do not have advantage in hiding information, and therefore, this is intrinsically a complete information game.

The formulation above brings up the following research question: is the generalization of KEP to N–KEG relevant? In particular, it is worth noting that the special case of KEP with $L = 2$ can be formulated as a maximum matching problem and consequently, solved in polynomial time. Moreover, the multi-agent kidney exchange literature focuses mainly in exchanges with size 2. Thus, the most natural and relevant extension to look at is 2–KEG with pairwise exchanges.

Our Contributions. In this section, we concentrate on the non-cooperative 2-player kidney exchange game (2–KEG) with pairwise exchanges (*i.e.*, $L = 2$). A player can be a hospital, a region or even a country. Under this setting it is inefficient to follow the classical normal-form game approach by specifying all the players' strategies. Note also that in our formulation of N–KEG, players' strategies are lattice points inside polytopes described by systems of linear inequalities. Thus, N–KEG and, in particular, 2–KEG belongs to the class of IPG.

We show that 2–KEG has always a pure Nash equilibrium (NE) and that it can be computed in polynomial time. Furthermore, we prove the existence of an NE that is also a *social optimum*, *i.e.*, the existence of an equilibrium where the maximum number of exchanges is performed. Finally, we show how to determine an NE that is a social optimum, always the preferred outcome of both players, and can be computed in polynomial time.

Our work indicates that studying the players interaction through 2–KEG turns the exchange program efficient both from the social welfare and players' point of view. In contrast, as mentioned before, there is no centralized mechanism that is strategyproof and at the same time guarantees a social optimum. Although we provide strong evidence that under 2–KEG the players' most rational strategy is a social optimum, we note the possibility of multiple equilibria. We show that the worst case Nash equilibrium in terms of social welfare is at least $\frac{1}{2}$ of the social optimum. Thus, the worst case outcome for our game is comparable with the one for the best deterministic strategyproof mechanism (recall that it guarantees a 2-approximation of the social optimum). The 2–KEG opens a new research direction in this field that is worth being explored.

Organization of the Section. Section 4.2.1 formulates 2–KEG in mathematical terms. Section 4.2.2 proves the existence of a Nash equilibrium that maximizes the social welfare and measures the Nash equilibria quality enabling the comparison of our game with strategyproof mechanisms. Section 4.2.3 proves that the players have incentive to choose Nash equilibrium that are socially optimal. Section 4.2.4 refines the concept of social welfare equilibria motivating for a unique rational outcome for the game. Section 4.2.5 discusses extensions to our model and Section 4.2.6 draws some conclusions.

4.2.1 Definitions and Preliminaries

We recall Section 2.2.1.1 where the essential background in matching is provided.

Let the players of 2–KEG be labeled player A and player B. For representing a 2–KEG as a graph, let V be a set of vertices representing the incompatible patient-donor pairs of players A and B, and E be the set of possible pairwise exchanges, *i.e.*, the set of edges (i, j) such that the patient of $i \in V$ is compatible with the donor of $j \in V$ and vice versa. For each player n, $V^n \subseteq V$ and $E^n \subseteq E$ are her patient-donor pairs and internal compatibilities, respectively. A player n's strategy set is the set of matchings in graph $G^n = (V^n, E^n)$. A profile of strategies is the specification of a matching M^n in $G^n = (V^n, E^n)$ for each player $n = A, B$. The independent agent controls the external exchanges $E^I \subseteq E$, *i.e.*, $(a, b) \in E^I$ if $a \in V^A$ and $b \in V^B$. Let $E^I(M^A, M^B)$ be a subset of E^I such that no edge is incident upon a vertex covered by M^A or M^B. For a player B's matching M^B define the *player A's reaction graph* $G^A(M^B) = (V, E^A \cup E^I(\emptyset, M^B))$ and for a player A's matching M^A define the *player B's reaction graph* $G^B(M^A) = (V, E^B \cup E^I(\emptyset, M^A))$. In the figures of this section, we will represent vertices that belong to V^A as gray circles and vertices that belong to V^B as white diamonds.

On the first stage of 2–KEG, each player n decides simultaneously a matching M^n of graph G^n to be executed. On the second stage of the game, given player A's first-stage decision M^A and player B's first-stage decision M^B, the IA decides the external exchanges to be performed such that the number of pairs covered by its decision is maximized. In other words, the IA finds a maximum matching $M^I(M^A, M^B)$ of $E^I(M^A, M^B)$. In the end of the game, player A's utility is $2|M^A| + |M^I(M^A, M^B)|$ and player B's utility is $2|M^B| + |M^I(M^A, M^B)|$.

An important factor for a game is that its rules are executed efficiently. For 2–KEG this means that the IA optimization problem must be easy to solve. As mentioned in Section 2.2.1.1, computing a maximum matching can be solved in polynomial time for any graph. Therefore, given the players' decisions, the IA optimization problem is solved in polynomial time.

A legitimate question that must be answered is if the game is well defined in the sense that the rules are unambiguous. Note that the utility of each player depends on the IA decision rule. In the general N–KEG case, there might be situations where there are multiple optimal IA's decisions that benefit the players differently. However, for 2–KEG that is not possible, because only pairwise exchanges are considered. That is, any IA matching leads to equal benefits for both players.

Proposition 4.2.1. *2–KEG is well defined.*

One apparent difficulty in the treatment of the game has to do with the bilevel optimization problem (4.2.1) of each player. However, computing a player's optimal strategy to a fixed matching of the other player can be simplified. From the standpoint of player A, the best reaction M^A to a player B's fixed strategy M^B can be computed by dropping the IA objective function (4.2.1c) (game rule) and solving the single level matching problem in the reaction graph $G^A(M^B)$. Basically, we are claiming that player A best reaction predicts the appropriate IA decision given M^A and M^B. This holds because IA's edges have a positive impact on the utility of player A.

Lemma 4.2.2. *Let M^B be a matching of player B in 2–KEG. Player A's best reaction to M^B can be achieved by solving a maximum weight matching problem on the graph $G^A(M^B)$, where the edges of G^A in E^A have weight 2 and those in $E^I(\emptyset, M^B)$ weight 1. The equivalent for player B also holds.*

4.2.2 Nash Equilibria and Social Welfare

In what follows, we will concentrate on pure equilibria. According with the equilibria conditions (2.3.14), a player A's matching M^A of G^A and a player B's matching M^B of G^B is a pure Nash equilibrium for 2–KEG if

$$2|M^A| + |M^I(M^A, M^B)| \geq 2|R^A| + |M^I(R^A, M^B)| \quad \forall \text{ matching } R^A \text{ of } G^A$$

$$2|M^B| + |M^I(M^A, M^B)| \geq 2|R^B| + |M^I(M^A, R^B)| \quad \forall \text{ matching } R^B \text{ of } G^B.$$

Along this section, we use NE to refer to pure Nash equilibria. A mixed-strategy Nash equilibrium attributes a probability distribution over the players' feasible decisions; therefore, its description may involve many players' strategies, which would be computationally unsuitable; furthermore, the pure equilibria study shows that their consideration is enough to achieve a good and efficiently computable outcome for both players.

In Section 4.2.2.1, we prove the existence of NE for 2–KEG and that it can be computed in polynomial time. Through these results, in Section 4.2.2.2 we prove the existence of an NE that maximizes the social welfare (sum of the players' utilities or, equivalently, number of vertices matched). In Section 4.2.2.3, we measure the quality of the NE in terms of social welfare. This analysis allow us to conclude that the worst case Nash equilibrium to 2–KEG and the best deterministic strategyproof mechanism guarantee that at least $\frac{1}{2}$ of the number of vertices matched in a social optimum is achieved.

4.2.2.1 Existence of a Pure Nash Equilibrium

In order to prove the existence of an NE we will use the concept of potential function to games, as defined in Section 2.3.2. For 2–KEG, a potential function Φ is a real-valued function over the set of player A's matchings in G^A and player B's matchings in G^B such that the value of Φ increases strictly when a player switches to a new matching that improves her utility.

Observe that a player A's decision does not interfere in the set of player B's matchings in G^B. In particular, player A cannot influence the part of player B's utility related with a matching in G^B. The symmetric observation holds for player B's decision. With this in mind, it is not difficult to find an exact potential function to 2–KEG.

Proposition 4.2.3. *Function* $\Phi(M^A, M^B) = 2|M^A|+2|M^B|+|M^I(M^A, M^B)|$ *is an exact potential function of 2–KEG.*

A profile of strategies for which the potential function maximum is attained is an NE (Lemma 2.3.9).

Theorem 4.2.4. *There exists at least one pure Nash equilibrium to 2–KEG and it can be computed in polynomial time.*

Proof. A matching corresponding to the maximum of the function Φ of Proposition 4.2.3 is an NE of 2–KEG. Computing a maximum to Φ is equivalent to solving a maximum weight matching problem, where the edges in E^A and E^B weight 2 and the edges in E^I weight 1. This can be done in polynomial time (see, *e.g.*, Papadimitriou and Steiglitz [103]). $\square$

Consider the 2–KEG instance represented in Figure 4.2.2. In this case, the NE achieved by computing the potential function maximum is $M^A = \{(4, 5)\}$, $M^B = \{(2, 3)\}$ (and thus, $M^I(M^A, M^B) = \emptyset$). There is another NE that does not correspond to a potential function maximum: $R^A = \emptyset$, $R^B = \emptyset$ and consequently $M^I(R^A, R^B) = \{(1, 2), (4, 3), (5, 6)\}$. The latter helps all the patient-donor pairs, and thus is more appealing to the players. This observation, motivates the need of studying efficient Nash equilibria that are possibly not achieved through the potential function maximum.

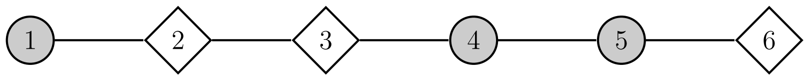


Figure 4.2.2: 2–KEG instance with two distinct Nash equilibria.

4.2.2.2 Social Welfare Equilibrium

In what follows, we introduce a refinement of the NE concept in 2–KEG: the social welfare equilibrium.

A *social optimum* of 2–KEG is a maximum matching of the overall graph game $G = (V, E)$, corresponding to an exchange program that maximizes the number of patients receiving a kidney. A *social welfare equilibrium* (SWE) is an NE that is also a social optimum.

Observe that any NE, and thus any SWE, is a local maximum of Φ if the neighborhood of a strategy profile consists of a player's unilateral deviation. In what follows, we will use this fact to prove the existence and efficient computation of an SWE.

Theorem 4.2.5. *There is always a social welfare equilibrium to 2–KEG.*

Proof. Let M be a maximum matching (and thus, a social optimum) of the graph G representing a 2–KEG, where $E^A \cap M$ and $E^B \cap M$ are players' A and B strategies, respectively. If M is not an NE, let us assume, *w.l.o.g.*, that player A has incentive to deviate from $E^A \cap M$, given player B's strategy $E^B \cap M$. Let M^A be player A's best reaction to $E^B \cap M$. Observe that we can assume that $M^A \cup M^I(M^A, E^B \cap M)$ is a maximum matching of A in the reaction graph $G^A(E^B \cap M)$. If it is not, by Berge's Theorem 2.2.6, there is a maximum matching such that it does not decrease the number of player A's matched vertices. Therefore, by Property 2.2.5, $|M^A| + |M^I(M^A, E^B \cap M)| + |E^B \cap M| = |M|$.

Given that A has incentive to deviate, it holds by definition of potential function that $\Phi(E^A \cap M, E^B \cap M) < \Phi(M^A, E^B \cap M)$. If M^A together with $E^B \cap M$ is not an NE, then we can repeat the procedure above (alternating the player) until an NE is obtained (as the tâtonnement process in Section 2.3.2). Note that the value of the potential function increases strictly, which means that no feasible profile of strategies is visited more than once and social welfare does not decrease. In addition, players have a finite number of feasible matchings, which implies that this process will terminate in an equilibrium. □

Besides the fact that an SWE is an appealing NE to the players, it also has the advantage of being computable in polynomial time through the algorithm of the last proof (translated to pseudo-code in Algorithm 4.2.2.1). It is a well-known result that weighed matching problems can be solved in polynomial time (see, *e.g.*, [103]). Therefore, it remains to prove that the number of iterations is polynomially bounded in the size of the instance. The next trivial result can be used to this end.

Lemma 4.2.6. *An upper bound to the maximum value of the 2–KEG potential function* $\Phi(M^A, M^B) = 2|M^A| + 2|M^B| + |M^I(M^A, M^B)|$ *is* $|V^A| + |V^B|$.

As noted before, the potential function Φ strictly increases whenever a player has incentive to unilaterally change her strategy. Therefore, our algorithm will in the worst case stop once the maximum value to Φ is reached, which is bounded by $|V^A| + |V^B|$. Taking into account that the value of Φ is always an integer number, the number of evaluations of Φ through the process is also bounded by $|V^A| + |V^B|$.

Theorem 4.2.7. *The computation of a social welfare equilibrium to 2–KEG can be done in polynomial time.*

Algorithm 4.2.2.1

Input: A 2–KEG instance G.

Output: A social welfare Nash equilibrium.

```
1: $M \leftarrow$ maximum matching of $G$
2: $M^A \leftarrow M \cap E^A$, $M^B \leftarrow M \cap E^B$, $M^I \leftarrow M \cap E^I$        initial matchings
3: while $\exists$ player $n \in \{A, B\}$ with incentive to deviate from $M^n$ do
4:     $R^n \leftarrow$ player $n$'s best reaction to $M^{-n}$ such that it is also a maximum matching of
       $G^n(M^{-n})$    solve a maximum weight matching on $G^n(M^{-n})$ and, after, apply (unweighted)
       augmenting paths to the solution until a maximum matching is obtained
5:     $M^n \leftarrow R^n$, $M^I \leftarrow M^I(R^n, M^{-n})$        update solution
6: end while
7: return $M^A$, $M^B$
```

4.2.2.3 Price of Stability and Price of Anarchy

In order to measure the quality of the Nash equilibria of a given game, we use the standard measures: price of stability and price of anarchy (see Chapter 17 of [96]). The *price of stability* (PoS) is the ratio between the highest total utilities value of one of its equilibria and that of a social optimum; the *price of anarchy* (PoA) is the ratio between the lowest total utilities value within its equilibria and that of a social optimum.

The following two results set PoS and PoA for 2–KEG.

Corollary 4.2.8. *The price of stability of the 2–KEG is 1.*

Proof. Since we proved existence of a social welfare equilibrium

$$\text{PoS} = \frac{\text{highest total utilities value among all Nash equilibria}}{\text{social optimum}} = 1.$$

□

Theorem 4.2.9. *The price of anarchy is $\frac{1}{2}$ for the 2–KEG.*

Proof. By the definition of price of anarchy, we have

$$\text{PoA} = \frac{\text{lowest total utilities value among all Nash equilibria}}{\text{social optimum}}.$$

Let M^A, M^B and $M^I(M^A, M^B)$ be the matchings of player A, B and the IA, respectively, that lead to the Nash equilibrium with lowest total utilities value, that is

$$z^* = 2|M^A| + 2|M^B| + 2|M^I(M^A, M^B)|.$$

Let M be a maximum matching of the game graph G. Therefore, the social optimum is equal to

$$\overline{z} = 2|M \cap E^A| + 2|M \cap E^B| + 2|M \cap E^I|.$$

By the definition of NE, we know that under M^A and M^B, none of the players has incentive to deviate, thus

$$\begin{aligned}
& z^* \geq 2|M \cap E^A| + |M^I(M \cap E^A, M^B)| + 2|M \cap E^B| + |M^I(M^A, M \cap E^B)| \\
\Leftrightarrow & z^* \geq 2|M \cap E^A| + 2|M \cap E^B| + 2|M \cap E^I| - 2|M \cap E^I| + |M^I(M \cap E^A, M^B)| \\
& \quad + |M^I(M^A, M \cap E^B)| \\
\Leftrightarrow & z^* \geq \overline{z} - \left(2|M \cap E^I| - |M^I(M^A, M \cap E^B)| - |M^I(M \cap E^A, M^B)|\right). \qquad (4.2.2a)
\end{aligned}$$

The set $M \cap E^I$ may include matchings of vertices also matched under M^A or M^B, therefore

$$2|M \cap E^I| \leq 2|M^A| + 2|M^B| + |R^A| + |R^B|,$$

where R^n is a subset of E considering all the edges in $M \cap E^I$ but not in M^n and incident with a vertex of V^n, for $n = A, B$. See Figure 4.2.3. The number of player B's vertices matched in $M^I\left(M \cap E^A, M^B\right)$ is equal or greater than R^B, because this external matching has available the vertices incident with the edges of R^B and can match them with any vertex not in $M \cap E^A$, thus

$$|R^B| - |M^I(M^A, M \cap E^B)| \leq 0.$$

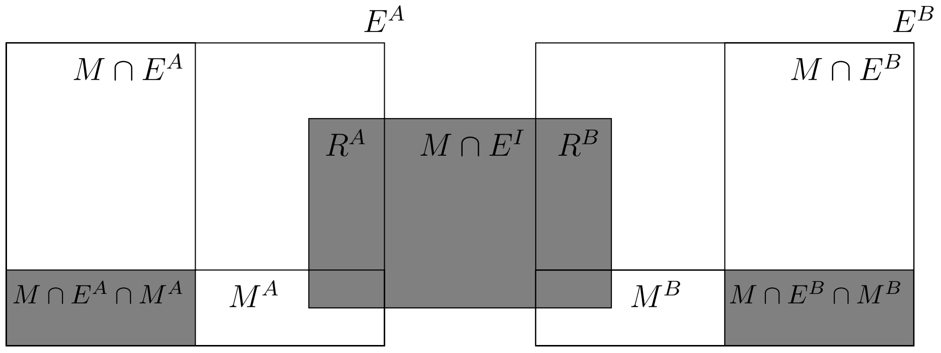


Figure 4.2.3: Illustration of the solutions associated with the worst Nash equilibrium and the social optimum.

In a completely analogous way, it can be shown that

$$|R^A| - |M^I(M \cap E^A, M^B)| \leq 0.$$

The inequalities above imply

$$2|M \cap E^I| - |M^I(M^A, M \cap E^B)| - |M^I(M \cap E^A, M^B)| \leq 2|M^A| + 2|M^B| \leq z^*,$$

which together with inequality (4.2.2a) results in

$$z^* \geq \overline{z} - z^* \Leftrightarrow \frac{z^*}{\overline{z}} \geq \frac{1}{2}.$$

Now, we will use an instance to prove that the bound $\frac{1}{2}$ is tight.

Consider a 2–KEG represented by the graph of Figure 4.2.4. It is easy to see that the worst Nash equilibrium in terms of total utilities is $M^A = \{(1,2)\}$, $M^B = \emptyset$ and $M^I\left(M^A, M^B\right) = \emptyset$ with a total of $z^* = 2$. On the other hand, the social optimum is $M = \{(1,3),(2,4)\}$ with a value of $\overline{z} = 4$. In this instance the price of anarchy is $\frac{z^*}{\overline{z}} = \frac{2}{4} = \frac{1}{2}$. □

4.2.3 Rational Outcome: Social Welfare Equilibrium

In this section, we will prove that the social welfare equilibria are Pareto efficient (defined in Section 2.3) and any NE that is not social optimal is dominated by an SWE. Consequently, from both the social welfare and the players' point of view, these equilibria are the most desirable game outcomes. Moreover, recall that in Section 4.2.2.2, we presented an algorithm that computes an SWE in polynomial time emphasizing its practicality.

Below we show that no SWE is dominated, *i.e.*, all SWE are Pareto efficient.

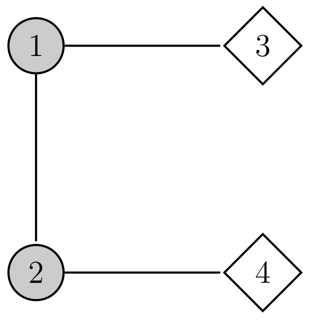


Figure 4.2.4: The price of anarchy is $\frac{1}{2}$.

Lemma 4.2.10. *In 2–KEG any social welfare equilibrium is Pareto efficient.*

Proof. Let M^A and M^B be players' A and B strategies, respectively, in a SWE. Assume that this SWE is not Pareto efficient, that is, there is a player A's feasible strategy R^A and a player B's feasible strategy R^B that dominate this equilibrium. Without loss of generality, these assumptions translate into:

$$2|M^A| + |M^I(M^A, M^B)| \leq 2|R^A| + |M^I(R^A, R^B)|$$

$$2|M^B| + |M^I(M^A, M^B)| < 2|R^B| + |M^I(R^A, R^B)|.$$

Summing the two inequalities above and simplifying, we obtain:

$$|M^A| + |M^I(M^A, M^B)| + |M^B| < |R^A| + |M^I(R^A, R^B)| + |R^B|,$$

which contradicts the assumption that the equilibrium given by M^A and M^B is a social optimum (maximum matching). □

Note that this result also holds for more than two players which reinforces the interest of studying SWE.

In the next section, we prove any NE that is not a social optimum is dominated by an SWE. In order to achieve this result we need the following theorem, which fully characterizes a player's best reaction.

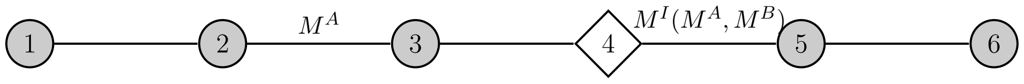
Theorem 4.2.11. *In 2–KEG, let M^B be a player B's fixed matching. A player A's matching M^A can be improved if and only if there is a $M^A \cup M^I(M^A, M^B)$-alternating path in $G^A(M^B)$ whose origin is a vertex in V^A, unmatched in this path, and the destination is a*

i. $M^A \cup M^I(M^A, M^B)$*-unmatched vertex belonging to* V^A*, or*
ii. $M^I(M^A, M^B)$*-matched vertex in* V^B*, or*
iii. $M^I(M^A, M^B)$*-unmatched vertex in* V^B*.*

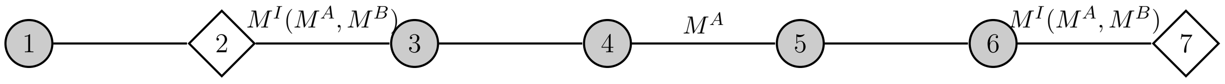
The symmetric result for player B also holds.

Proof. Consider a fixed match M^B of G^B.

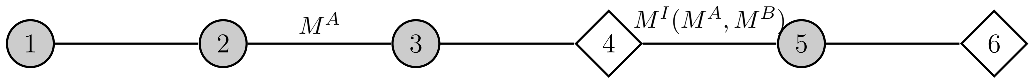
(Proof of if). Let M^A be a player A's strategy. Recall Lemma 4.2.2 in which we state that given M^B, we can assume that player A controls the IA decision. If there is a path p in $G^A(M^A)$ satisfying *i.*, *ii.* or *iii.*, then, $(M^A \cup M^I(M^A, M^B)) \oplus p$ improves player A's utility in comparison with $M^A \cup M^I(M^A, M^B)$; see Figure 4.2.5 for an illustration.



Case *i.* - The matching $\{(2,3),(4,5)\} \oplus \{(1,2),(2,3),(3,4),(4,5),(5,6)\}$ increases player A's utility by two units.



Case *ii.* - The matching $\{(2,3),(4,5),(6,7)\} \oplus \{(1,2),(2,3),(3,4),(4,5),(5,6),(6,7)\}$ increases player A's utility by one unit.



Case *iii.* - The matching $\{(2,3),(4,5)\} \oplus \{(1,2),(2,3),(3,4),(4,5),(5,6)\}$ increases player A's utility by one unit.

Figure 4.2.5: Possibilities for player A's to have an incentive to deviate from strategy M^A, given the opponent strategy M^B.

(Proof of only if). Let M^A be player A's best reaction to M^B and consider a feasible player A's strategy R^A that is not her best reaction to M^B. We will show that assuming that there is no $R^A \cup M^I(R^A, M^B)$-alternating path of $G^A(M^B)$ as stated in the theorem leads to a contradiction.

Note that given any two matchings M^1 and M^2 of a graph, in the induced subgraph with edges $M^1 \oplus M^2$, each vertex can be incident to at most two edges; hence, any connected component of $M^1 \oplus M^2$ is either an even cycle with edges alternately in M^1 and M^2, or a path with edges alternately in M^1 and M^2. Let us define H^A as the subgraph of G^A that results from considering the edges in $M^A \oplus R^A$, and H as the subgraph of $G^A(M^B)$ that results from considering the edges in $(M^A \cup M^I(M^A, M^B)) \oplus (R^A \cup M^I(R^A, M^B))$. Connected components of H^A and of H are either even cycles or paths.

If $|M^A| > |R^A|$, H^A has more edges of M^A than of R^A, and therefore there exists a path p of H^A that starts and ends with edges of M^A. If the origin and destination of p are $M^I(R^A, M^B)$-unmatched, then p is an $R^A \cup M^I(R^A, M^B)$-alternating path as stated in *i.*, which contradicts our assumption. Thus, for all paths of H^A starting and ending with edges of M^A, it holds that all their vertices are both M^A-matched and $R^A \cup M^I(R^A, M^B)$-matched (see Figure 4.2.6). Therefore, the advantage of $M^A \cup M^I(M^A, M^B)$ over $R^A \cup$

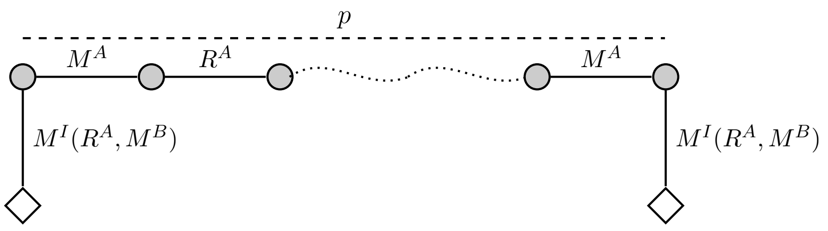


Figure 4.2.6: The path p is not an $R^A \cup M^I(R^A, M^B)$-alternating path of type *i.*

$M^I(R^A, M^B)$ must be outside H^A. Analogously, if $|M^A| \leq |R^A|$, we also conclude that the advantage of $M^A \cup M^I(M^A, M^B)$ over $R^A \cup M^I(R^A, M^B)$ must be outside H^A.

In this way, there is $a \in V^A$ and $b \in V^B$ such that $(a, b) \in M^I(M^A, M^B)$, but a is $R^A \cup M^I(R^A, M^B)$-unmatched. Then, since we assumed that there is no $R^A \cup M^I(R^A, M^B)$-alternating path as stated in the theorem (and the IA does not violate the game rules), the path of H starting in a must end in a vertex $a' \in V^A$ that is $R^A \cup M^I(R^A, M^B)$-matched and $M^A \cup M^I(M^A, M^B)$-unmatched. Therefore, the number of V^A vertices covered by $M^A \cup M^I(M^A, M^B)$ and $R^A \cup M^I(R^A, M^B)$ on this component is the same (see Figure 4.2.7). In conclusion, any path of H starting in a vertex of V^A that is

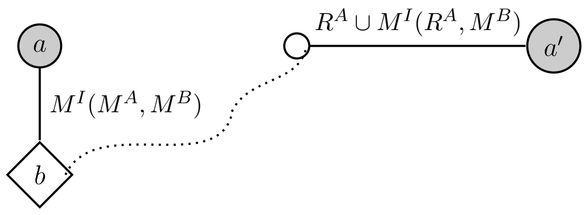


Figure 4.2.7: Path component of H. The white circle is a vertex for which it is not important to specify the player to which it belongs.

$R^A \cup M^I(R^A, M^B)$-unmatched and $M^I(M^A, M^B)$-matched does not give advantage to $M^A \cup M^I(M^A, M^B)$ over $R^A \cup M^I(R^A, M^B)$. This contradicts the fact that strategy R^A is not a player A's best reaction to M^B. $\square$

4.2.3.1 Computation of a Dominant SWE

We present in Algorithm 4.2.3.1 a method that, given a 2–KEG graph and a socially suboptimal Nash equilibrium, computes an SWE that we claim dominates the given equilibrium.

Algorithm 4.2.3.1

Input: A 2–KEG instance G and an NE M of G.

Output: M if it is an SWE, else an SWE dominating it.

```
1: $S \leftarrow$ a maximum matching of $G$
2: if $|M| = |S|$ then
3:     return $M$
4: end if
5: $t \leftarrow 1$
6: $\mathcal{P}^t \leftarrow$ paths from $M \oplus S$ with both extreme edges in $S$        $M$-augmenting paths
7: $M^t \leftarrow M \oplus p_1 \oplus \ldots \oplus p_r$ where $\{p_1, p_2, \ldots, p_r\} = \mathcal{P}^t$
8: while there is an $M^t$-alternating path $x = (v_0, v_1, \ldots, v_{2m})$ of type ii. in $G^n(M^t \cap E^{-n})$
   for some $n \in \{A, B\}$ do
9:     Assume $(v_0, v_1) \in E^I \cap M^t$ with $v_0 \in V^{-n}$ and $v_1 \in V^n$.
10:    $j \leftarrow \max_{i=0,\ldots,2m-1}\{i : (v_i, v_{i+1}) \in q$ for some $q \in \mathcal{P}^t\}$
11:    $y \leftarrow (u_0, u_1, \ldots, u_k, u_{k+1}, \ldots, u_f) \in \mathcal{P}^t$ used to determine $j$ with $(u_k, u_{k+1}) =$
       $(v_j, v_{j+1})$
12:    $z \leftarrow (v_{2m}, v_{2m-1}, \ldots, v_{j+1}, u_{k+2}, \ldots, u_f)$
13:    $M^{t+1} \leftarrow M^t \oplus y \oplus z$
14:    $\mathcal{P}^{t+1} \leftarrow (\mathcal{P}^t - \{y\}) \cup \{z\}$
15:    $t \leftarrow t + 1$
16:    $G' \leftarrow$ subgraph of $G^n(M^t \cap E^{-n})$ induced by considering only edges of $x$ from $v_0$ to
       $v_j = u_k$ and of $y$ from $u_0$ to $u_k = v_j$
17:    if there is a $x \leftarrow M^t$-alternating path of type ii. in $G'$ starting in $(v_0, v_1)$ go to
       step 10
18: end while
19: return $M^t$.
```

In what follows we provide a proof of the correctness of this algorithm. For sake of clarity, first of all, we provide an illustration of how the algorithm works by applying it to a 2–KEG instance.

Example 4.2.12. *Consider the 2–KEG instance represented in Figure 4.2.8.*

A Nash equilibrium M that is not a maximum matching is represented by bold edges in the top-left graph of Figure 4.2.9. The matching M is a Nash equilibrium, since there is

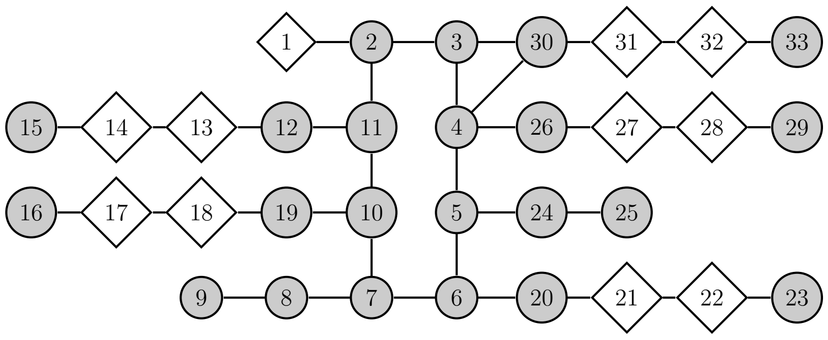


Figure 4.2.8: A 2–KEG instance.

no M-alternating path as stated in Theorem 4.2.11; and it is not a maximum matching because there are M-augmenting paths, e.g., $(25, 24, 5, 6, 20, 21, 22, 23)$*. We will apply Algorithm 4.2.3.1 to this NE in order to achieve one that is an SWE and dominates it.*

The algorithm starts by computing an arbitrary maximum matching S, represented in the top-right graph of Figure 4.2.9; the symmetric difference between M and S is represented in the center-left graph of that figure. There are 6 connected components in $S \oplus M$, three of which include M-augmenting paths:

$$\mathcal{P}^1 = \{(33, 32, 31, 30, 3, 4, 26, 27, 28, 29), (25, 24, 5, 6, 20, 21, 22, 23), \\ (15, 14, 13, 12, 11, 10, 19, 18, 17, 16)\}.$$

Therefore, at the end of step 7 we obtain a maximum matching M^1, represented at the center-right of Figure 4.2.9.

The algorithm proceeds searching for an M^1-alternating path of type ii. in $G^n(M^1 \cap E^{-n})$ for some $n \in \{A, B\}$, i.e., the algorithm will check if M^1 is an SWE. In this step, path $x = (1, 2, 3, 4, 5, 6, 7, 8, 9)$ is found, which shows that M^1 is not an equilibrium. The M-augmenting path $y = (25, 24, 5, 6, 20, 21, 22, 23)$ is replaced by $z = (9, 8, 7, 6, 20, 21, 22, 23)$, leading to matching M^2 represented at the bottom-left graph of Figure 4.2.9. Next, step 17 is used to verify if there is an M^2-alternating path of type ii. considering only the edges $(1, 2), (2, 3), (3, 4), (4, 5), (5, 24), (24, 25)$. There is: path $(1, 2, 3, 4, 5, 24, 25)$. The M-augmenting path $(33, 32, 31, 30, 3, 4, 26, 27, 28, 29)$ is modified into $(25, 24, 5, 4, 26, 27, 28, 29)$, obtaining M^3 represented in the lower-right graph of Figure 4.2.9. In the next iteration no M^3-alternating path of type ii. can be found, and thus the algorithm terminates. M^3 is an SWE that dominates M.

Next we will prove that for any socially suboptimal NE, the Algorithm 4.2.3.1 returns a dominant SWE.

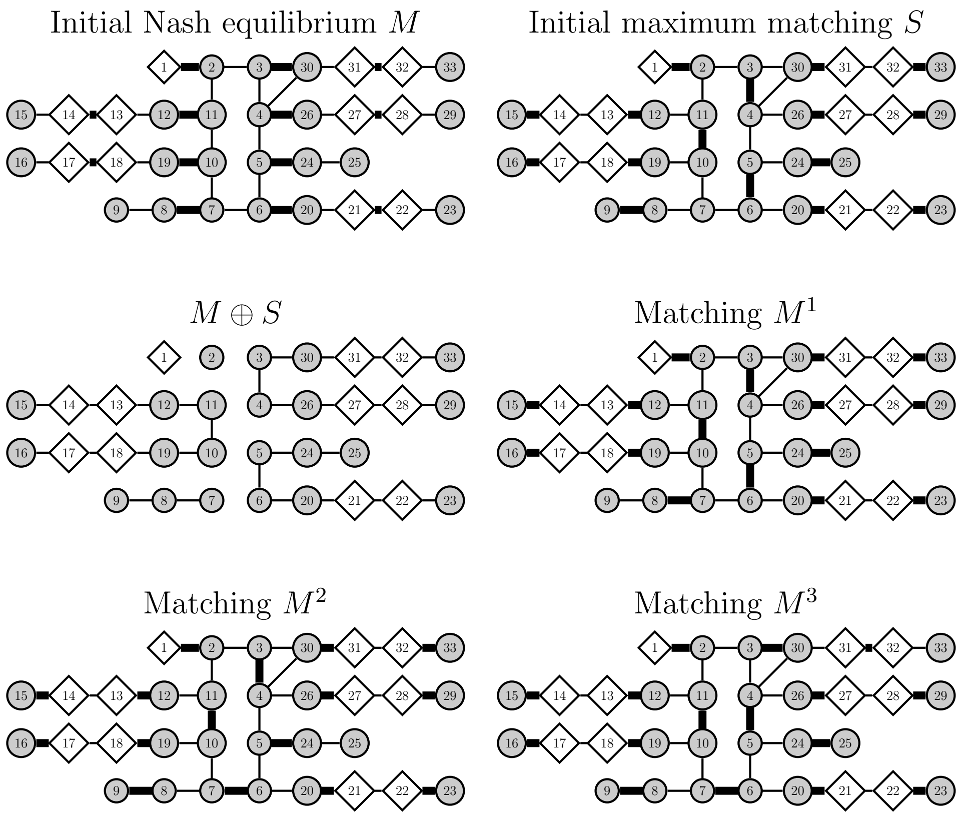


Figure 4.2.9: Computation of a dominant SWE in the 2–KEG instance of Figure 4.2.8 starting from the initial equilibrium in the top-left graph, and the initial maximum matching of top-right graph.

The algorithm starts by computing a maximum matching S. If the Nash equilibrium from the input is a maximum matching, the algorithm returns it and stops. Otherwise, it proceeds. At iteration t, $\mathcal{P}^t$ is the set of M-augmenting paths used to compute the maximum matching M^t. In this way, step 7 augments M in order to obtain a maximum matching M^1. Note that $|\mathcal{P}^1|$ augmenting paths of M are used in order to get M^1 and that the symmetric difference of a matching with an associated augmenting path only adds additional covered vertices. Therefore, none of the M-matched vertices is M^1-unmatched, which shows that the players' utilities associated with M^1 are equal to or greater than the ones achieved through M.

Note that if there is an M^1-alternating path of type *i.* or *iii.*, then it is also an augmenting path of M^1 contradicting the fact that M^1 is a maximum matching. Therefore, by Theorem 4.2.11, if M^1 is not a Nash equilibrium then there is an M^1-alternating path of type *ii.* in $G^A(M^1 \cap E^B)$ or $G^B(M^1 \cap E^A)$. In this case, the algorithm will remove the M^1-alternating path of type *ii.* through steps 8 to 15. In these steps an M-augmenting path $y \in \mathcal{P}^1$ is replaced by a new M-augmenting path z. Thus, it is obvious that the new maximum matching M^2 dominates the utilities achieved through M.

Suppose that in step 8 an M^t-alternating path x of type *ii.* is found. Since M is an NE, the path x cannot be M-alternating. Thus, x intersects at least one M^t-matched edge of a $y \in \mathcal{P}^t$. The algorithm picks such y accordingly with the one closest to v_{2m}, since this rule ensures that y never intersects x from $v_{j+1} = u_{k+1}$ to v_{2m}. Then, through step 13, v_{2m} is made M^{t+1}-matched, which eliminates the M^t-alternating path x of type *ii.*. See Figure 4.2.10 for illustration.

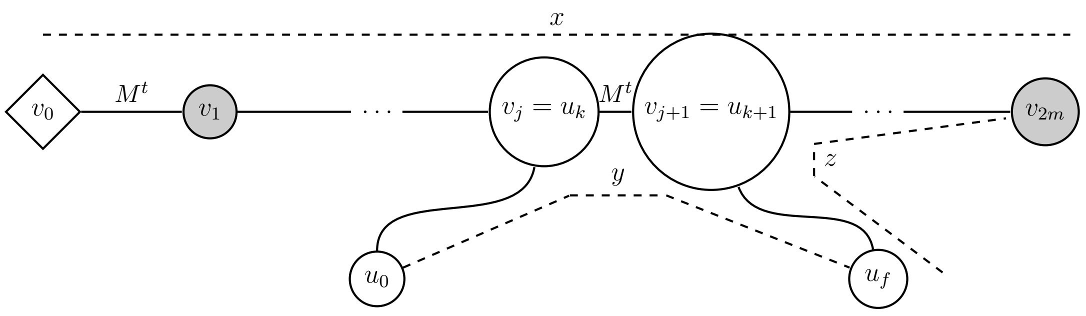


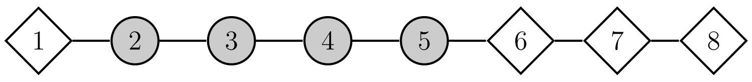
Figure 4.2.10: Modification of y to z through x. White circle vertices mean that there is no need to specify the player to which the vertices belong.

So far, we proved that at any iteration t of Algorithm 4.2.3.1, the current maximum matching M^t dominates M and that if there is an M^t-alternating path of type *ii.*, we eliminate it in the next maximum matching M^{t+1}. It remains to show that the elimination of paths of type *ii.* will stop, leading to an SWE.

By construction, the size of the augmenting path sets is maintained during the algorithm execution. Indeed, in each iteration, an M-augmenting path is replaced by a new one.

Lemma 4.2.13. $|\mathcal{P}^t| = |\mathcal{P}^k| \quad \forall t, k \geq 1$.

For an M-augmenting path $y = (u_0, u_1, \ldots, u_f)$, define $\sigma(y)$ as the number of times that y switches the player's graph plus one unit if the first internal edge that follows the extreme $u_0 \in V^i$ is in E^{-i}, and plus one unit if the last internal edge that precedes the extreme $u_f \in V^j$ is in E^{-j}. For instance, the path



has σ-value equal to 3: count two unities because, the first extreme vertex, 1, is in V^B while the following internal edge, $(2, 3)$, is in E^A and add 1 unit because the rest of the path is in E^B. Indeed, the σ-value of M-augmenting paths has to be greater or equal to two, otherwise it is not a Nash Equilibrium (*i.e.*, there is an M-alternating path as described in Theorem 4.2.11, or the independent agent is not choosing a maximum matching as obliged by the game rule). The following lemma states that the σ-value of the paths in $\mathcal{P}^t$ is non-increasing.

Lemma 4.2.14. *In an iteration t of Algorithm 4.2.3.1 $\sigma(y) \geq \sigma(z)$.*

Proof. Consider an arbitrary iteration t of Algorithm 4.2.3.1. Without loss of generality, assume that the M^t-alternating path x of type *ii.* found is in $G^A(M^t \cap E^B)$.

In step 11, $y = (u_0, u_1, \ldots, u_f)$ is the selected augmenting path in $\mathcal{P}^t$. In order to get z, the part of y from u_0 to u_k is replaced by a path that has all the edges in $E^A \cup E^I$. Note that there must be an internal edge in y after u_{k+1}, otherwise M is not an equilibrium: the path $(u_f, u_{f-1}, \ldots, u_{k+1}, v_{j+2}, v_{j+3}, \ldots, v_{2m})$ would be an M-alternating path in $G^A(M \cap E^B)$ satisfying one of the conditions of Theorem 4.2.11. Thus, we continue the proof by distinguishing two possible cases: the first internal edge in y after u_{k+1} is in E^B or E^A.

Case 1: The first internal edge in y after u_{k+1} is in E^B. Then, $\sigma(z)$ is equal to one plus the number of times that the path y from u_{k+1} to u_f switches the player's graph plus one unit if the last internal edge before $u_f \in V^i$ is in E^{-i}. Observe that $\sigma(y)$ is greater or equal to the number of times that the path y from u_{k+1} to u_f switches the player's graph plus one unit if the last internal edge before $u_f \in V^i$ is in E^{-i}. In order to get equal, the part of y from u_0 to u_{k+1} must have the edges in $E^B \cup E^I$ and $u_0 \in E^B$. However, this contradicts the fact that M is a Nash equilibrium: one of the vertices u_k or u_{k+1} has to be in V^A, otherwise y is not in player A's graph. If $u_{k+1} \in V^A$, then $u_{k+2} \in V^B$, which means that the part of x from v_{2m} to (u_{k+1}, u_{k+2}) is an M-alternating path of type *ii.* in $G^A(M \cap E^B)$. Otherwise, if $u_k \in V^A$, then $u_{k-1} \in V^B$ and the part of y from u_0 to u_k is an M-alternating path of type *ii.* in $G^B(M \cap E^A)$. In conclusion, $\sigma(y) \geq \sigma(z)$.

Case 2: The first internal edge in y after u_{k+1} is in E^A. Then, $\sigma(z)$ is equal to the number of times that the path y from u_{k+1} to u_f switches the player's graph plus one unit if the last internal edge before $u_f \in V^i$ is in E^{-i}. Note that $\sigma(y)$ is greater or equal to the number of times that the path y from u_{k+1} to u_f switches the player's graph plus one unit if the last internal edge before $u_f \in V^i$ is in E^{-i}. In conclusion, $\sigma(y) \geq \sigma(z)$.

□

An immediate consequence it the following corollary.

Corollary 4.2.15. *If $\sigma(y) > \sigma(z)$ holds in iteration t, then z will never evolve during the rest of the algorithm to be equal to y.*

Proof. Assume that $\sigma(y) > \sigma(z)$ in iteration t. By Lemma 4.2.14, if z is selected in a forthcoming iteration then the resulting (modified) path has a σ-value less or equal to $\sigma(z)$ and, in particular, less than $\sigma(y)$. Therefore, it is impossible that from iteration z this path evolves to y, since that contradicts Lemma 4.2.14. $\square$

Whenever Algorithm 4.2.3.1 at iteration t modifies y such that $\sigma(y) > \sigma(z)$, we get that the maximum matching M^t will never be computed again in later iterations.

Corollary 4.2.16. *Algorithm 4.2.3.1 can only cycle after iteration t if $\sigma(y) = \sigma(z)$.*

Now, we will prove that when a modification of an augmenting path y to z has $\sigma(y) = \sigma(z)$, then the algorithm finds an M^{t+1}-alternating path of type *ii.* in step 17. This particular search for such a path is the important ingredient for the algorithm to stop after a finite number of iterations. If we remove this step from Algorithm 4.2.3.1 and we simply arbitrarily search for the elimination of paths of type *ii.* then the algorithm can cycle. For instance, in Example 4.2.12, when we are in iteration 2 and we do not perform the search as stated in step 17, then we can compute the M^2-alternating path $(1, 2, 11, 10, 7, 6, 5, 24, 25)$ that would lead us to $M^3 = M^1$, making the algorithm to cycle.

Lemma 4.2.17. *If $\sigma(y) = \sigma(z)$ at the end of step 15 of Algorithm 4.2.3.1, then a path of type ii. is found in step 17.*

Proof. Suppose that the algorithm is in the end of step 15. Without loss of generality, the proof concentrates only on the case for which x is in $G^A(M^{t-1} \cap E^B)$, since for x in $G^B(M^{t-1} \cap E^A)$ the proof is analogous.

We will make use of Lemma 4.2.14 proof in order to conclude that under the lemma hypothesis, $\sigma(y) = \sigma(z)$, the edges of y from u_0 to u_k are in $E^A \cup E^I$. Case 1 of that proof implies that in order to get $\sigma(y) = \sigma(z)$, the edges of the path y from u_0 to u_k should be in $E^A \cup E^I$ and $u_0 \in V^A$. In order to get $\sigma(y) = \sigma(z)$ in case 2, we also get that the edges of the path y from u_0 to u_k should be in $E^A \cup E^I$ and $u_0 \in V^A$.

Next, we will show that there is an M^t-alternating path of type *ii.* from (v_0, v_1) to u_0 that only uses the edges of x from v_0 to v_j and y from u_0 to u_k. Therefore, for sake of clarity, consider $y' = (u_0, u_1, \ldots, u_k)$ and $x' = (v_0, v_1, v_2, \ldots, v_j)$. Recall that $u_k = v_j$.

In step 17, the new M^t-alternating path of type *ii.* x can be built as follows. Start to follow x' from v_0 until it intersects a vertex u_{j_1} in y' (note that y' intersects x' at least in $u_k = v_j$). Consider the following possibilities.

Case 1: If $(u_{j_1}, u_{j_1-1}) \in M^t$, then $x = (v_0, v_1, \dots, u_{j_1}, u_{j_1-1}, \dots, u_0)$ is an M^t-alternating path of type *ii.*.

Case 2: If $(u_{j_1}, u_{j_1+1}) \in M^t$, then $(u_{j_1}, u_{j_1-1}) \in M^{t-1}$ and $(u_{j_1}, u_{j_1-1}) \in x'$, which implies $u_{j_1+1} \notin x'$. Follow y' by index increasing order starting in u_{j_1+1} until it is reached a vertex $u_{j_2} = v_{i_1}$ of x' (note that such vertex exists since at least $u_k = v_j \in x'$, with $k > j_1 + 1$). The vertex $u_{j_2-1} \notin x'$, otherwise, we would have stopped in u_{j_2-1}. Thus, $(u_{j_2}, u_{j_2-1}) \notin M^{t-1}$. Otherwise, x' would not be an M^{t-1}-alternating path. In conclusion, $(u_{j_2}, u_{j_2-1}) \in M^t$.

Next, we follow x' by index decreasing order starting in $u_{j_2} = v_{i_1}$ until we intersect a vertex u_{j_3} of y' (which has to occur, since we noted before that at least u_{j_1-1} is in x'). If $(u_{j_3}, u_{j_3-1}) \in M^t$, then the rest of the M^t-alternating is found as in case 1. Otherwise, $(u_{j_3}, u_{j_3+1}) \in M^t$ and we proceed as in the beginning of case 2. This process will terminate in u_0 since we are always adding new vertices to our M^t-alternating path and the number of vertices is finite.

□

Corollary 4.2.18. *The algorithm can only cycle if it remains in steps 15 to 17.*

Theorem 4.2.19. *After a finite number of executions of steps 15 to 17, the algorithm fails to find such a path in step 17.*

Proof. The length of the path $(v_0, v_1, v_2, \dots, v_j)$ considered in step 17 strictly decreases in each consecutive execution of steps 15 to 17. □

As a corollary of the above Theorem we can now state the desired result.

Corollary 4.2.20. *After a finite number of iterations, the Algorithm 4.2.3.1 stops and finds an SWE that dominates the NE given in the input.*

4.2.4 Refinement of SWE

Although Algorithm 4.2.2.1 computes an SWE, the results obtained in Section 4.2.3 (see Theorem 4.2.11) allow the definition of a simpler polynomial time algorithm returning an SWE. Furthermore, the algorithm will solve another aspect left open in the previous sections where we discussed the advantage of SWE among the set of NE for 2–KEG. This refinement to select an NE is still not sufficient to get uniqueness, *i.e.*, there are 2–KEG instances for which there is more than one SWE. The algorithm presented in this section will solve this issue.

Example 4.2.21. *Consider the 2–KEG instance represented in Figure 4.2.11. There are four maximum matchings M^1 to M^4, of which matchings M^1 and M^2 are NE (SWE). Under M^1 player A has utility 4 and player B has utility 2; in contrast, under M^2 both players have utility 3.*

This instance has two distinct SWE, and by repeating the relevant pattern we can create instances with multiple distinct SWE. For example, the game of Figure 4.2.12 has eight SWE.

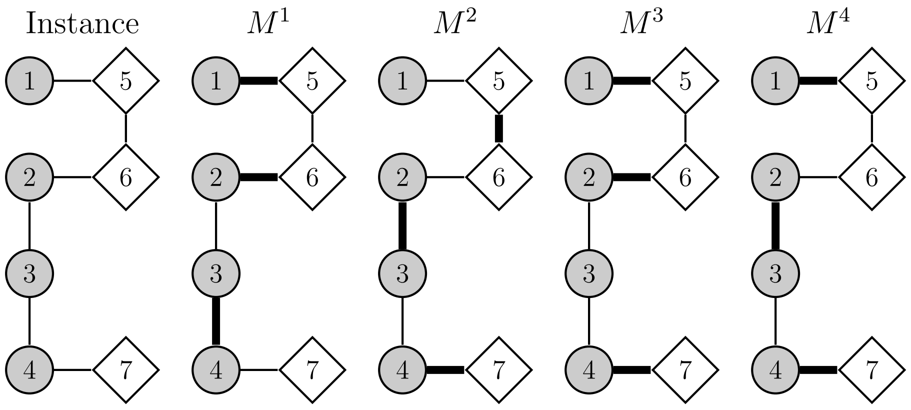


Figure 4.2.11: 2–KEG instance with four different maximum matchings, and two SWE, M^1 and M^2.

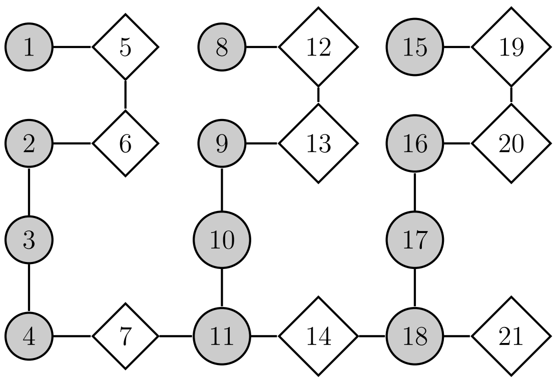


Figure 4.2.12: 2–KEG instance with eight SWE.

In this context it seems rational to search for the social welfare equilibrium that minimizes the number of external exchanges, since that decreases the dependency of the players on each other; in practice, this seems to be a more desirable solution. Therefore, in what follows, we will show how to find such an equilibrium in polynomial time.

Consider Algorithm 4.2.4.1. This algorithm based on the number of vertices, $|V|$, it associates weight $2 + 2|V|$ for internal edges and weight $1 + 2|V|$ for external edges.

Then, a maximum weight matching is returned. We will prove that this algorithm can be executed in polynomial time and that it computes a social welfare equilibrium that minimizes the number of external exchanges.

Algorithm 4.2.4.1

Input: A 2–KEG instance G.

Output: An SWE that minimizes the number of external exchanges.

```
1: for $e$ in $E^A \cup E^B$ do
2:     $w_e \leftarrow 2 + 2|V|$
3: end for
4: for $e$ in $E^I$ do
5:     $w_e \leftarrow 1 + 2|V|$
6: end for
7: $M \leftarrow$ maximum weight matching in $G$ given edge weights $w_e, \quad \forall e \in E$
8: return $M$
```

Lemma 4.2.22. *Algorithm 4.2.4.1 can be executed in polynomial time.*

Proof. It is a well-known result that weighed matching problems can be solved in polynomial time (see, *e.g.*, [103]). Therefore, step 7 can be executed in polynomial time. Additionally, the attribution of weights for the graph edges is linear in the number of edges. Therefore, the algorithm can run in polynomial time. □

In order to prove that Algorithm 4.2.4.1 outputs an SWE, we need to prove that M is a maximum matching and an NE.

Lemma 4.2.23. *Algorithm 4.2.4.1 returns a maximum matching.*

Proof. In step 7 of the algorithm, the maximum weight on an edge in the maximum weight matching problem considered is $2+2|V|$. Thus, any matching of size k has a total weight not greater than $k(2+2|V|)$. If that is not a maximum matching, *i.e.*, if $k < |S|$, where S is a maximum matching for G, the total weight is bounded above by

$$k(2+2|V|) = 2k(1+|V|) \leq 2(|S|-1)(1+|V|) = 2|S||V| + 2(|S|-|V|-1) < 2|S||V|,$$

where the last inequality comes from the fact that $|S| < |V|$.

A maximum matching on the graph game has a total weight at least equal to $|S|(1+2|V|) = |S| + 2|S||V|$. Therefore, a maximum matching has always a total weight greater than any non maximum matching. In conclusion, a maximum weight matching with the proposed edge weights is also a matching with maximum cardinality. □

Lemma 4.2.24. *Algorithm 4.2.4.1 returns an NE.*

Proof. Let M be the output of Algorithm 4.2.4.1.

By Lemma 4.2.23 we know that M is a maximum matching. If M is not an NE, then some player must have incentive to deviate; *w.l.o.g.*, assume that player A has incentive to deviate from $M \cap E^A$. Then, there must be an M-alternating path p of type *ii.* in $G^A(M \cap E^B)$ such that $M \oplus p$ increases player A's utility

$$2|(M \oplus p) \cap E^A| + |(M \oplus p) \cap E^I| > 2|M \cap E^A| + |M \cap E^I|.$$

On the other hand, the matching $|M \oplus p|$ must have a total weight not greater than the one associated with M, *i.e.*,

$$\begin{aligned}&(2+2|V|)|M \cap E^A| + (2+2|V|)|M \cap E^B| + (1+2|V|)|M \cap E^I| \geq \\ &(2+2|V|)|(M \oplus p) \cap E^A| + (2+2|V|)|(M \oplus p) \cap E^B| + (1+2|V|)|(M \oplus p) \cap E^I|.\end{aligned}$$

Since the path p only uses the edges in $E^A \cup E^I$, the set $M \cap E^B$ is equal to $(M \oplus p) \cap E^B$. Hence, in this inequality, we can remove the second term of both sides and rewrite as

$$\begin{aligned}&\overbrace{2|M \cap E^A| + |M \cap E^I| - 2|(M \oplus p) \cap E^A| - |(M \oplus p) \cap E^I|}^{<0} + \\ &2|V|\left(|M \cap E^A| + |M \cap E^I| - |(M \oplus p) \cap E^A| - |(M \oplus p) \cap E^I|\right) \geq 0.\end{aligned}$$

Player A's utility is bigger with $M \oplus p$ than with M. Thus, in this inequality the first four terms lead to a negative number. This implies that

$$|M \cap E^A| + |M \cap E^I| > |(M \oplus p) \cap E^A| + |(M \oplus p) \cap E^I| \geq 0,$$

which is impossible since, M and $M \oplus p$ have the same cardinality and, in particular, $|M \cap (E^A \cup E^I)| = |(M \oplus p) \cap (E^A \cup E^I)|$. □

Finally, it remains to prove that Algorithm 4.2.4.1 returns a matching that minimizes the number of external edges on it among the set of SWE.

Lemma 4.2.25. *Algorithm 4.2.4.1 outputs a matching that minimizes the number of external edges among the set of social welfare equilibria.*

Proof. Let M be the matching returned by Algorithm 4.2.4.1. We will prove by showing that assuming another SWE M' contains more internal exchanges than M leads to a contradiction. Since both M and M' are maximum matchings, M' has a total weight greater than M; but this contradicts the fact that the algorithm returns a maximum weight matching (where the internal edges weight more than the external ones). □

The next theorem concludes this section.

Theorem 4.2.26. *Algorithm 4.2.4.1 computes an SWE that minimizes the number of external exchanges in polynomial time.*

Unfortunately, for some 2–KEG instances this refinement of the SWE still does not lead to an unique solution.

Example 4.2.27. *Consider the 2–KEG instance of Figure 4.2.13. There are two SWE that minimize the number of external exchanges, M^1 and M^2. These matchings lead both players to an utility of 3.*

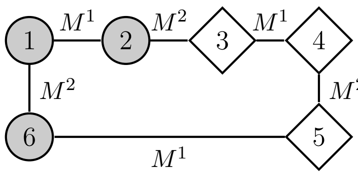


Figure 4.2.13: 2–KEG instance with two distinct SWE that lead both players to the same utility.

However, the players utilities under social welfare equilibria that minimize the number of external exchanges are unique as we will prove next.

Theorem 4.2.28. *In any SWE that minimizes the number of external exchanges, for a fixed instance, the players' utilities are always the same.*

Proof. Consider an instance of 2–KEG for which there are two different SWE minimizing the number of external exchanges, say M^1 and M^2, of Algorithm 4.2.4.1. The proof is by contradiction, by assuming that player A's utilities with M^1 and M^2 are different. Without loss of generality,

$$2|M^1 \cap E^A| + |M^1 \cap E^I| > 2|M^2 \cap E^A| + |M^2 \cap E^I|.$$

Build the subgraph H of G induced by the edges in the set $(M^1 \oplus M^2) \cap (E^A \cup E^I)$. As player A covers more of her vertices through M^1 than through M^2, there must be at least one vertex $a \in V^A$ such that a is M^1-matched and M^2-unmatched. Consider each distinct component p of H; p is a path starting in, say, vertex a. There are three possible cases. Namely,

Case 1: path p terminates in an M^2-matched vertex of V^A. Then, it is not this component that gives advantage to M^1.

Case 2: path p terminates in an M^2-matched vertex of V^B. Then, p is an M^2-alternating path of type *ii.*; by Lemma 4.2.24, this contradicts the fact that M^2 is an NE.

Case 3: path p terminates in an M^1-matched vertex. Then, p is an augmenting path to M^2; by Lemma 4.2.24, this contradicts the fact that M^2 is a maximum matching.

□

We finish this section by noting that another desirable SWE is that in which the difference of players' utilities is minimized, *i.e.*, the discrepancy of the players' utilities is minimized resulting in a more "fair" outcome. It is easy to show that the social welfare equilibrium introduced in this section, *i.e.*, that minimizing the number of external matchings achieves simultaneously the goal of minimizing the difference of players' utilities.

Theorem 4.2.29. *If M is an SWE with minimum number of external matchings then, it also an SWE that minimizes the difference of players' utilities.*

Proof. Let M^A, M^B and $M^I(M^A, M^B)$ be the social welfare equilibrium that minimizes the number of external matchings. Let R^A, R^B and $M^I(R^A, R^B)$ be the social welfare equilibrium that minimizes the difference in the players utilities, *i.e.*, the value of $|2|R^A| + |M^I(R^A, R^B)| - 2|R^B| - |M^I(R^A, R^B)|| = ||R^A| - |R^B||$ is the minimum among all social welfare equilibria.

If $|M^I(M^A, M^B)| = |M^I(R^A, R^B)|$, then the matching $R^A \cup R^B \cup M^I(R^A, R^B)$ is also an SWE that minimizes the number of external matchings. Thus, by the uniqueness of the players' utilities under this refinement of the SWE, $M^A \cup M^B \cup M^I(M^A, M^B)$ also minimizes the difference of players' utilities.

If $|M^I(M^A, M^B)| \neq |M^I(R^A, R^B)|$ then, $|M^A| + |M^B| > |R^A| + |R^B|$ since, by hypothesis $|M^I(M^A, M^B)| < |M^I(R^A, R^B)|$ and both matchings have maximum cardinality. Without loss of generality, there must be a path p that starts and ends in M^A-matched vertices and alternates between edges in M^A and edges in R^A. Matching $R^A \cup R^B \cup M^I(R^A, R^B)$ is an NE which implies that p cannot be a path as described in Theorem 4.2.11. Therefore, the extreme vertices of p must be $M^I(R^A, R^B)$-matched which does not show any advantage of $M^A \cup M^I(M^A, M^B)$ and $R^A \cup M^I(R^A, R^B)$ over each other in terms of player A's utility. In this way, it follows that both matchings lead to the same utility for both players. □

In conclusion, one may argue that the players will select social welfare equilibria since, given any Nash equilibrium, both players can improve their utilities through an SWE. Additionally, choosing an SWE that minimizes the number of external exchanges is a desirable propriety for both players, and we demonstrated that such equilibrium can be found in polynomial time. Moreover, players are indifferent among such equilibria, because utilities remain the same for any of them. Thus, it seems reasonable to consider that the players will agree in the SWE to be played.

4.2.5 Model Extensions

In what follows, we discuss extensions to the results when our assumptions (exchanges size, players' utilities and number of players) are relaxed.

A common problem of these extensions is that the IA decision may become undefined (in contrast with Proposition 4.2.1), in the sense that there might exist more than one optimal solution maximizing the number of external exchanges that would benefit the players differently. In order to deal with this issue, we could, for example, impose a public preference on the external exchanges to the IA, associate a probability for each equivalent optimal solution of the IA or assume that the players are pessimistic/optimistic about the IA decision.

Relaxation of Exchanges Maximum Size to $L > 2$. In the literature about kidney exchange programs, besides cycles of size two (matchings), typically cycles of size three (3-way exchanges) are allowed. In the latter case, we conjecture that (recall the notation introduced to N–KEG in Problem 4.2.1)

$$\Phi(x^A, x^B) = \sum_{c \in C^A} w_c x_c^A + \sum_{c \in C^B} w_c x_c^B + \sum_{c \in I : w_c^A = w_c^B = 1} y_c + \frac{3}{2} \sum_{c \in I : w_c^A = 2 \vee w_c^B = 2} y_c$$

is a (*non-exact*) potential function and thus, a maximum is an NE. However, for general values of L the game may fail to have a pure Nash equilibrium, as shown in Figure 4.2.14. The main difference when $L > 3$ is that in this case external cycles may help strictly more patients of a same player than an internal exchange, while for $L = 3$ an external exchange helps at most as many patients as an internal one.

Besides cyclic exchanges, researchers have also included *chains*, where, there is an altruistic donor starting the exchange (see Figure 4.2.15). Allowing exchanges beyond matchings ($L = 2$) and chains is an extension with positive impact in the social optimum, and it calls for studying the existence of pure Nash equilibria with good social properties.

Change in Players' Utilities. Investigating different players' utilities is of crucial importance. The literature on the kidney exchange program is rich of examples analyzing different solution selection criteria (*e.g.*, see [44]).

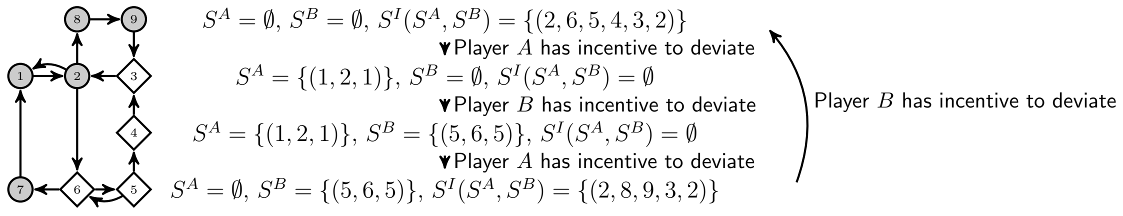

1
2
3
4
5
6
7
8
9
$S^A = \emptyset$, $S^B = \emptyset$, $S^I(S^A, S^B) = \{(2,6,5,4,3,2)\}$
Player A has incentive to deviate
$S^A = \{(1,2,1)\}$, $S^B = \emptyset$, $S^I(S^A, S^B) = \emptyset$
Player B has incentive to deviate
$S^A = \{(1,2,1)\}$, $S^B = \{(5,6,5)\}$, $S^I(S^A, S^B) = \emptyset$
Player A has incentive to deviate
$S^A = \emptyset$, $S^B = \{(5,6,5)\}$, $S^I(S^A, S^B) = \{(2,8,9,3,2)\}$
Player B has incentive to deviate

Figure 4.2.14: A game instance with $L = 5$. Player A can select $\{(1,2,1)\}$ or $\emptyset$; Player B can select $\{(5,6,5)\}$ or $\emptyset$. Let S^P be player P internal exchange program, for $P = A, B$, and $S^I(S^A, S^B)$ the IA external exchange program. The diagram on the right hand side of the graph shows that none of the (pure) game outcomes is a Nash equilibrium (implying that the game cannot be potential).

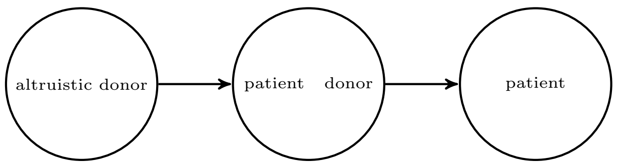


Figure 4.2.15: Example of a chain of length 2.

A simple extension would be to assume that the players prioritize maximum matchings that maximize “hard-to-match” vertices. In this case, we could still have an SWE. We first compute an SWE for 2–KEG. If this SWE is not an equilibrium for this extension, then, w.l.o.g., there is a M-unmatched vertex $a \in V^A$ hard-to-match and a $M^A \cup M^I(M^A, M^B)$-alternating path p that terminates in a player A M-matched vertex that is not hard-to-match. Because the maximum matching $M' = M \oplus p$ improves player A utility and does not create alternating paths of type *ii.* (see Theorem 4.2.11), we just need to repeat this process until no player has incentive to deviate.

However, for more complicated players’ utilities the game may fail to have pure Nash equilibria. For instance, consider the compatible graph of Figure 4.2.16. The IA behavior remains as before: maximize the number of external exchanges among the available vertices; be indifferent between the players’ evaluation of the different matchings; have a deterministic decision, that is, for any combination of the players’ strategies (internal matchings) the external exchange selected by the IA is known. In Figure 4.2.17, we have all the possible outcomes for the game. Observe that none of these 4 possible outcomes is a Nash equilibrium and thus, no pure equilibrium exists.

Another extension in this context is to Bayesian games. In this case, the players would not know their opponents evaluations/utilities for the exchanges. Under this incomplete information scenario, it would be interesting to explore how the players can build believes

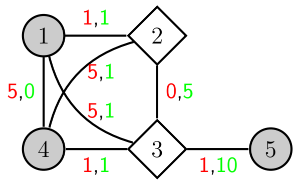


Figure 4.2.16: The players' utility of each matching is given by the numbers in the edges: player A value is in red and player B value in green.

$M^A = \emptyset$	$M^A = \emptyset$	$M^A = \{(1,4)\}$	$M^A = \{(1,4)\}$
$M^B = \emptyset$	$M^B = \{(2,3)\}$	$M^B = \emptyset$	$M^B = \{(2,3)\}$
$M^I(M^A, M^B) = \{(1,3),(2,4)\}$	$M^I(M^A, M^B) = \emptyset$	$M^I(M^A, M^B) = \{(3,5)\}$	$M^I(M^A, M^B) = \emptyset$
$\Pi^A = 10$	$\Pi^A = 0$	$\Pi^A = 6$	$\Pi^A = 5$
$\Pi^B = 2$	$\Pi^B = 5$	$\Pi^B = 10$	$\Pi^B = 5$

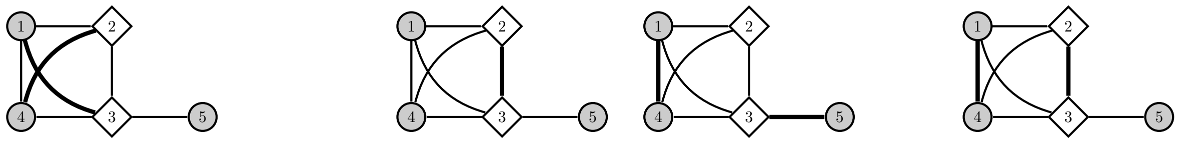


Figure 4.2.17: All possible outcomes for the game.

about the opponents' objectives by repeatedly observing the game outcomes and, thus, use them to compute (Bayesian) equilibria.

Increase Number of Players to $N > 2$. Extending our results about the existence of an NE and an SWE dominating it is immediate. Let $\{1, 2, \ldots, N\}$ be the set of players. Then, (by extending our notation in an obvious way)

$$\Phi(M^1, M^2, \ldots, M^N) = \sum_{P=1}^{N} 2|M^P| + |M^I(M^1, M^2, \ldots, M^N)|$$

is a (non-exact) potential function, and a optimum of it is an NE. The function is potential, since whenever a player increases her utility it is because she is increasing the number of internal exchanges. An increase in the number of internal exchanges has a greater impact in the value of Φ than external exchanges. The results in Section 4.2.3 remain valid in this setting. The ideas presented analyze each player's incentives for deviation, which hold for more than 2 players, because we can think of a player opponents' as a single one (reducing the study to 2–KEG).

It remains to investigate, if there is an NE which the players would agree to choose.

4.2.6 Summary

In this section, we have shown that the two-player kidney exchange game has always a pure Nash equilibrium and that it can be computed in polynomial time. Furthermore, we have proven the existence of a NE that is also a social optimum. Finally, and more importantly, we have shown that for any NE there is always a social welfare Nash equilibrium that is a preferred outcome for both players.

There is no uniqueness result for social welfare equilibria. In order to find rational guidelines for the players' strategies, we add to the social welfare equilibrium the requirement that it must be the one that minimizes the number of external exchanges. For this type of solution, we were able to prove uniqueness in terms of the players' utilities and to show that it can be efficiently computed, thus strengthening the fact that this is a realistic outcome for the game.

Although we show that a social welfare equilibrium can be computed in polynomial time, a full characterization of the Pareto frontier of social welfare equilibria (with respect to pure Nash equilibria) remains to be done. This is an interesting subject for future research.

Our work also indicates that studying the players interaction through 2–KEG turns the exchange program efficient both from the social welfare and the players' point of view. These results motivate further research in the generalization of the game to more than two players, to exchanges including more than two patient-donor pairs and to different evaluation metrics of the exchanges. Some of these generalizations have been preliminarily discussed in Section 4.2.5.

Additional inspiration for future research is given by the recent paper Hajaj *et al.* [64], where a strategyproof mechanism for a multi-period dynamic model was shown to lead to a global maximum matching that cannot be guaranteed by a mechanism for the static case. Therefore, given that 2–KEG already provides such solution as a rational outcome in the static case, investigating the 2–KEG by playing it repeatedly as the players' pools of patient-donor pairs change over time would be another line to explore in the future work.

4.3 Competitive Uncapacitated Lot-Sizing Game

[3]

Our Game Model. In this section, Pedroso and Smeers [104] Cournot competition model is analyzed. We investigate the authors' competitive uncapacitated lot-sizing game (ULSG) version. The ULSG is a game that merges the lot-sizing problem (see Section 2.2.1.3) with Cournot competition (see Example 2.3.8). A player is a firm with its own production facility, modeled as an uncapacitated lot-sizing problem. For each time period, instead of fixed demands to be satisfied by each player, a Cournot competition is played. The lot-sizing part turns the game combinatorial, *i.e.*, an IPG, and the Cournot competition models the players interaction.

Literature in Lot-Sizing Games. The generality of the lot-sizing games formulated in the literature have in common (with the ULSG) that players model their production through a lot-sizing programming problem, and differ in the way in which the players affect each others utilities. There is literature about lot-sizing games focusing on the underlying cooperative direction. In this type of games, instead of searching for a Nash equilibrium, the goal is to find coalitions between the players such that they do not have incentive to leave them (it would lead to an utility decrease); *e.g.* see Heuvel *et al.* [66]. To the best of our knowledge, the literature in non-cooperative (competitive) lot-sizing games significantly differs from our setting. Maskin and Tirole [88] analyze an oligopoly, where set-up costs are considered and firms are committed to a particular action in the short-run. In opposition to the model that we present in this section, in [88], firms move sequentially and set-up costs are considered to be sufficiently large so that no two firms can operate profitably. Federgruen and Meissner [50] analyze a Bertrand (price) competition. In this model, each player decides a market price which is maintained throughout the game. Given these market prices, the demand in each time period for each player is determined. The authors are able to get sufficient conditions for the existence and efficiency of computing one Nash equilibrium if the set-up costs are constant during the whole time horizon for each player. It is also mentioned the Cournot competition associated with this model. In this last case, a player's strategy reduces to deciding a basic deseasonalized target volume quantity through which the demand is determined for each time period (the authors note that this case is considerably more difficult). Li and Meissner [83] consider a lot-sizing game version in which the players' strategies are the

[3]The results of this chapter appear in:
M. Carvalho, M. Van Vyve, C. Telha. Competitive Uncapacitated Lot-Sizing Game, working paper.

production capacities purchased at the beginning of the time planning and, afterwards, each player solves a lot-sizing programming problem. The cost of buying capacity depends on the total capacity purchased by the players. After this choice is taken, the players' problem is just a single-item lot-sizing problem with limited capacity. The authors prove the existence of a capacity equilibrium under modest assumptions. In the models of these two papers ([50] and [83]), as well as in our model, the producers decide their strategies initially and stay committed to them until the end of the time horizon.

Pedroso and Smeers [104] apply a tâtonnement process (recall Section 2.3.2) in order to compute an equilibrium to the competitive lot-sizing game. In the authors' computational experiments, this process successfully computes an equilibrium. Thence, their work opens the questions of the method conditions to converge to an equilibrium and how efficient it is.

Our Contributions and Organization of the Section. In Section 4.3.1, we formalize the competitive uncapacitated lot-sizing game, which is a novel Cournot Competition model. Section 4.3.2 describes the players' best responses once the opponents' strategies are fixed, and, in particular, a dynamic programming method to find a player's best response in polynomial time. It is proven that ULSG is potential in Section 4.3.3, immediately implying the existence of a (pure) Nash equilibrium. For the case of a single period and for the case of only set-up costs, algorithms to find a pure NE in polynomial time are described in Section 4.3.4 and Section 4.3.5, respectively. There may exist multiple equilibria for an ULSG, and thus, refinements to the equilibrium concept are usually used (as we did for the two-player kidney exchange game); we show that it is NP-hard to find a pure NE for the single-period case with respect to a given linear objective function, but one can compute such an optimal equilibrium in pseudo-polynomial time. In Section 4.3.6 we remark that our results can be easily extended if inventory costs are considered. Section 4.3.7 summarizes the open questions.

4.3.1 Model and Notation

The ULSG establishes the connection between the classical uncapacitated lot-sizing model and the Cournot competition. The model we have built has a discretized finite time horizon of T periods. In each period t there is a market for a homogeneous product. We assume that for each period t, the market unit price is P_t, represented by the demand function $P_t = (a_t - b_t q_t)^+$ where $\alpha^+ = \max(\alpha, 0)$, q_t is the total quantity placed in the market, and a_t , b_t are given parameters modeling the market size and the level of players interaction, respectively. The set of firms (players) competing in this multi-period market is $M = \{1, 2, \ldots, m\}$. The production structure of each firm is represented by an uncapacitated lot-sizing model. That is, each firm p has to decide how much to produce in each time period t (production variable x_t^p) and how much to place in the market (variable q_t^p); we assume that a firm is fully committed to a strategy for the finite time horizon T. For each firm p and period t, there are set-up and variable (linear) production costs, denoted by F_t^p and C_t^p, respectively, no upper limit on production quantities, and a producer can build inventory by producing in advance (inventory variable for period t is h_t^p). We assume that there are no inventory costs (in Section 4.3.6 this assumption is removed). In this way, we obtain the following model for each player (firm) $p = 1, 2, \ldots, m$:

$$\max_{y^p, x^p, q^p, h^p} \quad \Pi^p(y^p, x^p, h^p, q^p, q^{-p}) = \sum_{t=1}^{T} P_t(q_t) q_t^p - \sum_{t=1}^{T} C_t^p x_t^p - \sum_{t=1}^{T} F_t^p y_t^p \tag{4.3.1a}$$

$$\text{s. t.} \quad x_t^p + h_{t-1}^p = h_t^p + q_t^p \qquad \text{for } t = 1, \ldots, T \tag{4.3.1b}$$

$$0 \le x_t^p \le \mathcal{B} y_t^p \qquad \text{for } t = 1, \ldots, T \tag{4.3.1c}$$

$$h_0^p = h_T^p = 0 \tag{4.3.1d}$$

$$h_t^p, q_t^p \ge 0 \qquad \text{for } t = 1, \ldots, T \tag{4.3.1e}$$

$$y_t^p \in \{0, 1\} \qquad \text{for } t = 1, \ldots, T \tag{4.3.1f}$$

where $\mathcal{B}$ is a sufficient large number and $q_t = \sum_{i=1}^{m} q_t^i$ (total quantity introduced in the market of period t). The total quantity introduced in the market of period t is the responsible for the optimization program (4.3.1) to induce a game. The goal of player p is to maximize the utility (4.3.1a), which is simply the sum of her profit minus the production costs in each period t. Constraints (4.3.1b) represent the conservation of product. Constraints (4.3.1c) 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 set-up cost F_t^p. We assume that the initial and final inventory quantities are zero, which is captured by equations (4.3.1d). Inventory quantities and output quantities must be non-negative, Constraints 4.3.1e. The variables y_t^p are restricted to be binary through constraint (4.3.1f).

Let y^p, x^p, h^p be T dimensional vectors of player p's decision variables for each time period t. Finally, for theoretical purposes, let us assume that variable and set-up costs are positive integers, and define producing in period $T + 1$ as not participating in the game.

4.3.2 Best Responses

Recall from Section 2.2.1.3 that in ULSP the demand is fixed and the problem reduces to minimizing the costs. A well-known and fundamental property of ULSP is that it has an optimal solution with no inventory at the begin of a period with positive production (Proposition 2.2.9). The same property holds for a player p's optimal solution for (4.3.1).

Proposition 4.3.1. *Let $q^{-p} \in X^{-p}$ be fixed. There exists an optimal solution to* (4.3.1) *(best response to q^{-p}) in which $h^p_{t-1} x^p_t = 0$ for $t = 1, 2, \ldots, T$.*

Proof. Suppose that q^p is an optimal solution to (4.3.1) given q^{-p}. The optimal production plan to player p reduces to an ULSP with demand q^p. Therefore, Proposition 2.2.9 holds, and thus, there is an optimal solution such that $h^p_{t-1} x^p_t = 0$ for $t = 1, 2, \ldots, T$. □

Proposition 4.3.1 is the essential ingredient to determine the optimal output quantities for player p.

Proposition 4.3.2. *Let $q^{-p} \in X^{-p}$ and player p's positive production periods $t_1 < t_2 < \ldots, < t_r$ be fixed. There is an optimal solution to problem* (4.3.1) *satisfying*

$$\begin{aligned} &\overline{q}^p_t(q^{-p}) = 0, && \textit{for} \quad t = 1, 2, \ldots, t_1 - 1 \\ &\overline{q}^p_t(q^{-p}) = \frac{(a_t - b_t \sum_{i \neq p} q^i_t - C^p_{t_j})^+}{2b_t}, && \textit{for} \quad t = t_1, \ldots, T \\ \textit{with} \quad & j = \arg\max_{\substack{t_u \leq t \\ u=1,2,\ldots,r}} t_u. \end{aligned}$$

Proof. Let $T^p = \{t_1, t_2, \ldots, t_r\}$ be as stated in the proposition. By Proposition 4.3.1, in period $t \geq t_1$, the optimal output quantity q^p_t is produced in the latest production period t_j prior to t, so the production variable can be simply replaced by $x^p_{t_j} = \sum_{t=t_j}^{\min(t_{j+1},T)} q^p_t$. The optimal value for q^p_t in 4.3.1 can be determined by optimizing an univariate concave quadratic function (the part of the utility function associated with q^p_t), that is,

$$(a_t - b_t q^p_t - b_t \sum_{i \neq p} q^i_t) q^p_t - C^p_{t_j} q^p_t$$

leading to the formulas of this proposition. □

Recall from Section 2.2.1.3 that ULSP can be solved in polynomial time through dynamic programming. If $q^{-p} \in X^{-p}$ is fixed, a similar idea extends to efficiently compute an optimal production plan for player p.

Lemma 4.3.3. *Solving player p's best reaction* (4.3.1) *for q^{-p} can be done in polynomial time.*

Proof. Let $G^p(t, q^{-p})$ be the maximum utility of player p over the first t periods, given the opponents' strategies q^{-p}. Then, $G^p(t, q^{-p})$ can be written as player p's maximum utility when the last production period was k

$$G^p(t, q^{-p}) = \max_{k:k\leq t}\{G^p(k-1, q^{-p}) + \sum_{u=k}^{t}(a_u - b_u(\overline{q}_u^p + \sum_{j\neq p}^{m} q_u^j))\overline{q}_u^p - F_k^p - C_k^p \sum_{u=k}^{t} \overline{q}_u^p\},$$

where $\overline{q}_u^p$ is computed according with Proposition 4.3.2. Thus, computing $G^p(T, q^{-p})$, which is equivalent to solve the best reaction problem (4.3.1) for q^{-p}, can be done in $O(T^2)$ time (recall the dynamic programming method for ULSP described at the end of Section 2.2.1.3). □

In an equilibrium each player is selecting her best reaction (optimal solution of problem (4.3.1)) to the opponents' strategies on that equilibrium. Thus, once the players' production periods are fixed, we can apply Proposition 4.3.2 simultaneously for all the players, obtaining a system of equations in the output variables q which can be simplified and solved, resulting in the following proposition.

Proposition 4.3.4. *Let T^p be the set of production periods for each player p for an ULSG. Then, an optimal output quantity for player p is*[4]

$$\begin{aligned} \overline{q}_t^p &= 0, && \textit{for} \quad t = 1, 2, \ldots, \min\{T^p\} - 1 \\ \overline{q}_t^p &= \frac{(P_t(S_t) - C_{t_j^p}^p)^+}{b_t}, && \textit{for} \quad t = \min\{T^p\}, \ldots, T, \end{aligned}$$

where $t_j^p = \max_{u\in T^p, u\leq t} u$ (last production period prior to t for player p), $S_t = \{i : t \in T^i \text{ for } i = 1, 2, \ldots, m\}$ (players participating in the market of period t) and $P_t(S_t) = \frac{a_t + \sum_{i\in S_t} C_{t_j^i}^i}{|S_t|+1}$ (market price of period t). In particular, player p's utility is

$$\Pi^p(T^1, \ldots, T^m) = \sum_{t\in T^p} -F_t^p + \sum_{t=\min\{T^p\}}^{T} \frac{(P_t(S_t) - C_{t_j^p}^p)^+}{b_t}(P_t(S_t) - C_{t_j^p}^p). \qquad (4.3.4)$$

In conclusion, the sets of production periods for all the players are sufficient to describe an NE. This fact significantly simplifies the game analysis in Section 4.3.4 and Section 4.3.5.

[4] By optimal output quantities it must be understood the quantities of an NE for the game in which production periods are fixed beforehand.

In what follows, we use the notation of Proposition 4.3.4: S_t is the set of players participating in the market of period t and $P_t(S_t)$ is the unit market price of period t for the set of players S_t.

Proposition 4.3.4 leads to a natural variant of ULSG: restrict each player p's strategy to her set $T^p \subseteq \{1, \ldots, T, T+1\}$ of production periods and her utility is computed accordingly with utility (4.3.4); call this modified game ULSG-sim. Proposition 4.3.4 associates output quantities to each profile of strategies in ULSG-sim. Because these output quantities are optimal for the fixed sets of production in ULSG-sim, the set of NE of ULSG-sim propagates to the original ULSG:

Proposition 4.3.5. *Any NE of an ULSG-sim is an NE of the associated ULSG.*

In Section 4.3.5, we compute a NE for a special case of the ULSG-sim (and hence for the ULSG), using a potential function argument. The ULSG-sim, however, is not always a potential game like the ULSG (as we will show in the next section). Moreover, the latter can have even a larger set of NE. This shows the advantages and disadvantages of investigating ULSG through ULSG-sim. The following two examples illustrate that ULSG-sim may not be potential (Example 4.3.6) and that an NE for ULSG does not have to be an NE of ULSG-sim (Example 4.3.7).

Example 4.3.6 (ULSG-sim is not a potential game)**.** *Consider the instance of ULSG-sim with* $m = 2$, $T = 2$, $a_1 = 20$, $a_2 = 40$, $b_1 = b_2 = 1$, $F_1^1 = 17, F_2^1 = 10$, $F_1^2 = 18$, $F_2^2 = 10$, $C_1^1 = 7$, $C_2^1 = 5$, $C_1^2 = 17$ *and* $C_2^2 = 1$. *The following relations for the players' utilities:*

$$\begin{aligned}
&\Pi^1(\{1\},\{1\}) < \Pi^1(\{2\},\{1\})\\
&\Pi^2(\{2\},\{1\}) < \Pi^2(\{2\},\{3\})\\
&\Pi^1(\{2\},\{3\}) < \Pi^1(\{1\},\{3\})\\
&\Pi^2(\{1\},\{3\}) < \Pi^2(\{1\},\{1\})
\end{aligned}$$

imply that a potential function Φ *must satisfy* $\Phi(\{1\},\{1\}) < \Phi(\{1\},\{1\})$ *which is impossible.*

Example 4.3.7 (An NE for ULSG may not be an NE of ULSG-sim)**.** *Consider the following instance with* $m = 2$, $T = 2$, $a_1 = 12$, $a_2 = 9$, $b_1 = b_2 = 1$, $F_1^1 = 15, F_2^1 = 5$, $F_1^2 = 7$, $F_2^2 = 19$ *and* $C_1^1 = C_2^1 = C_1^2 = C_2^2 = 0$. *Note that the absence of variable costs implies that it is a dominant strategy to produce only once.*

In the original game, $x^1 = q^1 = (0, \frac{a_2}{3b_2}) = (0, 3)$ *and* $x^2 = (\frac{a_1}{2b_1} + \frac{a_2}{3b_2}, 0)$, $q^2 = (\frac{a_1}{2b_1}, \frac{a_2}{3b_2}) = (6, 3)$ *represents a profile of strategies that is a Nash equilibrium of ULSG with player 1's* *utility equal to 4 and player 2's utility equal to 38; if player 1 (player 2) does not participate in the game her utility decreases to zero, thus player 1 (player 2) does not have incentive to unilaterally deviate from the equilibrium and not produce; if player 1 decides to produce in period 1, then, by Proposition 4.3.2, she would produce* $x^1 = (\frac{a_1}{4b_1} + \frac{a_2}{3b_2}, 0)$ *and introduce in the market* $q^1 = (\frac{a_1}{4b_1}, \frac{a_2}{3b_2})$*, decreasing her utility to 3; if player 2 decides to produce in period 2, then, by Proposition 4.3.2, she would produce* $x^2 = (0, \frac{a_2}{3b_2})$ *and place on the market* $q^2 = (0, \frac{a_2}{3b_2})$*, decreasing her utility to -10.*

Let us verify if the profile of strategies in ULSG-sim associated with the NE to ULSG described above, $T^1 = \{2\}$ *and* $T^2 = \{1\}$*, is an NE for ULSG-sim. Player 1's utility for the profile of strategies under consideration is* 4*. Since player 1's utility is positive, the player has incentive to participate in the game. It remains to check if player 1 has incentive to produce in period 1. If player 1 deviates to* $T^1 = \{1\}$ *then the associated utility is* $-F_1^1 + \frac{a_1^2}{9b_1^2} + \frac{a_2^2}{9b_2^2} = -15 + 16 + 9 = 10$ *which is greater than when player 1 produces in period 2. Thus,* $T^1 = \{2\}$ *and* $T^2 = \{1\}$ *is not an equilibrium of ULSG-sim.*

4.3.3 Existence and Computation of Nash Equilibria

As pointed out in Simon [117] and Rubinstein [112], players tend to prefer simple strategies which might be sub-optimal. This reason together with the fact that ULSG has always a pure equilibrium (as we prove next), justify that we concentrate our investigation only in pure Nash equilibria.

If there were no set-up costs and $T = 1$, we would be under the classical Cournot competition where, clearly, the players with smallest variable costs will be the ones sharing the market; this will be treated in detail in Section 4.3.4. If we relax T to be arbitrary but keep the restriction of only variable costs, the problem is equivalent to solving the Cournot competition for each period t separately and considering the player p's variable cost in period t equal to $\min_{u=1,\dots,t} C_u^p$, this is, each player participates in market t by producing in advance in the least expensive period. In summary:

Theorem 4.3.8. *When* $F_t^p = 0$ *for* $p = 1, 2, \dots, m$ *and* $t = 1, 2, \dots, T$*, then the set of NE for ULSG, projected onto the variables* (x, h, q) *is contained in a polytope and the market price is equal for all the NE. Furthermore, unless the problem is degenerate (i.e., there are at least two players for which the production costs coincide with the market price in an equilibrium), there is only one NE, and it can be computed in polynomial time.*

Next, we investigate the effect on the equilibria search when set-up costs are introduced in the game.

In what follows, we show that our game possesses at least one NE through the concept of potential game (recall Section 2.3.2).

Proposition 4.3.9. *The ULSG is a potential game that contains NE, one of them being a maximizer in X of the game exact potential function*

$$\Phi(y,x,h,q) = \sum_{p=1}^{m}\sum_{t=1}^{T}\left[-F_t^p y_t^p - C_t^p x_t^p + \left(a_t - \frac{b_t}{2}(2q_t^p + \sum_{i\neq p} q_t^i)\right) q_t^p\right] \tag{4.3.6a}$$

$$= \sum_{p=1}^{m}\left[\Pi^p\left(y^p, x^p, q^p, q^{-p}\right) + \sum_{t=1}^{T}\left(\frac{q_t^p b_t}{2}\sum_{i\neq p} q_t^i\right)\right]. \tag{4.3.6b}$$

Proof. The fact that ULSG is a potential game and the function (4.3.6) is an exact potential of it is a direct result from Ui's Proposition 2.3.10 [123]. Lemma 2.3.9 by Monderer and Shapley states that a strategy maximizing the potential function of a potential game is a pure Nash equilibrium. More generally, if we define the neighborhood of a point $(y,x,h,q) \in X$ to be any point in X such that only one player modifies her strategy then, any local maximum of the potential function $\Phi(y,x,h,q)$ is an NE. It only remains to check that the potential function Φ has indeed a maximum in the domain of feasible strategies. This follows from the fact that Φ is a linear combination of binary variables (and hence, bounded) plus a concave function (see Appendix A). □

Given that ULSG is potential and its potential function has an optimum, the **tatônnement process** described in Section 2.3.2, when applied to ULSG, is guaranteed to compute (converge to) an NE. This process requires to solve the players' best reactions 4.3.1 in each of its iterations; although, each iteration can be performed in polynomial time (by Lemma 4.3.3), we could not prove that the number of iterations is polynomial in the size of the input which would imply that the tatônnement process runs in polynomial time.

Alternatively, in order to find an equilibrium, one could compute a maximum of the potential function $\Phi(y,x,h,q)$ in X which amounts to solve a concave MIQP, see the proof in Appendix A. Once the binary variables y are fixed, *i.e.*, production periods have been decided, maximizing the potential function amounts to solve a concave quadratic problem and therefore, a maximum can be computed efficiently. In particular, recall from Theorem 4.3.8, that if there are no set-up costs (which is equivalent to say that the binary variables y_t^p are set to zero and constraints (4.3.1c) are removed) there is (in general) a unique equilibrium which can be found in polynomial time. Once set-up costs are considered, the analyses seems to complicate as indicated by the fact that a player's advantage in the game is not anymore a mirror of her variable cost alone. Since computing an equilibrium through the potential function maximization implies solving an MIQP which in general is hard, we will restrict our study to simpler cases (single period and only set-up costs) in an attempt to get insight in the understanding of the game's equilibria.

4.3.4 Single Period

In all this section we restrict our attention exclusively to the case with a single period ($T = 1$). For simplicity, we drop the subscript t from our notation. Note that in this setting there is no inventory to consider (variables h^p disappear) and the quantities produced are exactly those placed in the market ($x^p = q^p$). Additionally, by Proposition 4.3.4, the problem of computing equilibria reduces to decide the set of players producing strictly positive quantities. We start by proving that an NE can be computed in polynomial time. Then, we show that characterizing the set of NE is a NP-complete problem, and that admits a pseudo-polynomial time algorithm. All these results follow from a simpler characterization of the equilibrium conditions that we now describe.

In an NE, a subset of producers $S \subseteq \{1, 2, \dots, m\}$ plays a strictly positive quantity. By the definition of NE, no player in S has incentive to stop producing (leave S) and a player not in S has no incentive to start producing (enter in S). Therefore, applying Proposition 4.3.4, a player p in S must have non-negative utility

$$-F^p + \frac{(P(S) - C^p)^+}{b}(P(S) - C^p) \geq 0 \quad \Leftrightarrow \quad P(S) \geq \sqrt{F^p b} + C^p, \tag{4.3.7}$$

while a player p not in S must have non-positive utility if she enters S, even if producing the optimal quantity $\frac{(P(S)-C^p)^+}{2b}$ given by Proposition 4.3.2

$$-F^p + \frac{(P(S) - C^p)^+}{2b}\frac{(P(S) - C^p)}{2} \leq 0 \quad \Leftrightarrow \quad P(S) \leq 2\sqrt{F^p b} + C^p. \tag{4.3.8}$$

To find one NE efficiently, we refer to Algorithm 4.3.4.1. In a nutshell, this algorithm uses the lower bounds to $P(S)$ given by conditions 4.3.7 to order the players in step 1. Starting from $S = \emptyset$, it adds a player to S whenever she has advantage to join the current S (step 4). Since a player p will only join S if her variable cost C^p is smaller than the market price, it is easy to see that $P(S)$ decreases whenever a player is added to S (note that $P(S)$ is simply the average of the variable costs together with the parameter a). Thus, once in iteration k, if player p did not had incentive to enter S then, she will never have it in the future updates of S. This shows that in the end of the algorithm, the players not in S do not have incentive to enter it. On the other hand, taking into account the order of the players, whenever player p has incentive to be added to S, we have $P(S \cup \{p\}) > \sqrt{F^p b} + C^p \geq \sqrt{F^i b} + C^i$ for all $i \in S$, ensuring condition (4.3.7). This shows that the algorithm outputs correctly an NE.

In Algorithm 4.3.4.1, step 1 involves ordering a set of numbers with size m which can be done in $O(m \log m)$ time. Then, a cycle follows which can cost $O(m)$ time. In this way, it is easy to conclude that the algorithm runs in time $O(m \log m)$.

Theorem 4.3.10. *Algorithm 4.3.4.1 outputs an NE and runs in $O(m \log m)$ time.*

Algorithm 4.3.4.1

Input: A single period ULSG instance.

Output: A subset S of players producing strictly positive quantities in an NE.

```
1: Assume that the players are ordered according with $\sqrt{F^1 b} + C^1 \leq \sqrt{F^2 b} + C^2 \leq \ldots \leq \sqrt{F^m b} + C^m$.
2: Initialize $S \leftarrow \emptyset$
3: for $1 \leq p \leq m$ do
4:    if $C^p + 2\sqrt{F^p b} < P(S)$ then
5:       $S \leftarrow S \cup \{p\}$
6:    else
7:       if $P(S \cup \{p\}) \geq \sqrt{F^p b} + C^p$ then
8:          Arbitrarily decide to set $p$ in $S$.
9:       end if
10:   end if
11: end for
12: return S
```

In particular, the last theorem implies that there is always (at least) one NE. To see that there can be more than one, consider an instance where all players have $C^p = 0$ and $F^p = F$. Then Algorithm 4.3.4.1 will stop adding elements when $P(S) = a/(|S| + 1) < 2\sqrt{Fb}$. But since the ordering is arbitrary, this means that any set S of cardinality $\lceil a/(2\sqrt{Fb}) \rceil - 1$ is a NE. Therefore it makes sense to define the optimization problem (decision version):

Problem: OPTIMIZE 1-PERIOD UNCAPACITATED LOT SIZING GAME

Instance: Positive reals a, b, B, vectors $C, F \in \mathbb{Z}^m_+$ and $p \in \mathbb{Z}^m$. (1P-LSG-OPT)

Question: Is there a subset S of $\{1, 2, \ldots, m\}$ such that

$$\sum_{i \in S} p^i \geq B \tag{4.3.9a}$$

$$C^p + \sqrt{F^p b} \leq P(S) \quad \forall k \in S \tag{4.3.9b}$$

$$C^p + 2\sqrt{F^p b} \geq P(S) \quad \forall k \notin S \quad ? \tag{4.3.9c}$$

It turns out that 1P-LSG-OPT is NP-complete and thus, likely to be an intractable problem. We prove this through a reduction from Partition (PP) (given a set of n positive integers, find if they can be split into two groups with identical sum), which is NP-complete [56].

Theorem 4.3.11. 1P-LSG-OPT *is NP-complete.*

Proof. Given a set $S \subseteq \{1, 2, \ldots, m\}$, constraints (4.3.9a), (4.3.9b) and (4.3.9c) can be verified in polynomial time in the size of the instance. Therefore, 1P-LSG-OPT is in NP.

We show that 1P-LSG-OPT is NP-complete by reducing Partition to it. Let $\{a_i\}_{i=1..m}$ be an instance of Partition. Set $A = \frac{1}{2}\sum_{i=1}^{m} a_i$ and $M = 1 + 2A$. We construct the following instance of 1P-LSG-OPT.

- Set $b = 1$, $a = Am$, and $B = M - A$.
- $\mathcal{I} = \{1, 2, \ldots, m\}$ is a set of m players such that for each element $i = 1, 2, \ldots, m-1$ set $C^i = a_i$, $F^i = (A - C^i)^2$ and $p^i = -a_i$, and $C^m = a_m$, $F^m = (A - C^m)^2$ and $p^m = -a_m + M$.
- $\mathcal{D} = \{m+1, m+2, \ldots, 2m-1\}$ is a set of $m-1$ dummy players such that for each element $i = m+1, m+2, \ldots, 2m, 2m-1$ set $C^i = 0$, $F^i = \left(\frac{A}{2}\right)^2$ and $p^i = 0$.
- Set an *upper bound* player UB with $C^{\text{UB}} = A$, $F^{\text{UB}} = 0$ and $p^{\text{UB}} = -3M$.

(Proof of if). For a YES instance of Partition, there is $Z \subseteq \{1, 2, \ldots, m\}$ so that $\sum_{i \in Z} a_i = A$ and $m \in Z$. Note that $S = Z \cup \{m+1, m+2, \ldots, 2m - |Z|\}$ is a solution to 1P-LSG-OPT, with $|S| = m$, and whose market price $P(S)$ equals

$$\frac{a + \sum_{i \in S} C^i}{|S| + 1} = \frac{Am + \sum_{i \in Z} a_i}{m+1} = \frac{Am + A}{m+1} = A.$$

Let us verify that the S is indeed a YES instance for 1P-LSG-OPT.

Inequality (4.3.9a) is satisfied: since $m \in Z \subseteq S$, then

$$M - \sum_{i \in Z} a_i = M - A \Rightarrow \sum_{i \in S} p^i = B.$$

Inequalities (4.3.9b) hold for S:

$$C^p + \sqrt{F^p b} = a_p + \sqrt{(A - a_p)^2} = A = P(S), \ \ \forall p \in S \cap \mathcal{I} = Z$$

$$C^p + \sqrt{F^p b} = 0 + \sqrt{\left(\frac{A}{2}\right)^2} = \frac{A}{2} \leq P(S), \ \ \forall p \in S \cap \mathcal{D}.$$

Inequalities (4.3.9c) hold: using $a_p < A$ for $p = 1, 2, \ldots, m$, it follows that

$$\begin{aligned} C^p + 2\sqrt{F^p b} = 2A - a_p \geq A = P(S), \ \ \forall p \in \overline{S} \cap \mathcal{I} \\ C^p + 2\sqrt{F^p b} = A = P(S), \ \ \forall p \in \overline{S} \cap \mathcal{D} \\ C^{\mathrm{UB}} + 2\sqrt{F^{\mathrm{UB}} b} = A = P(S). \end{aligned}$$

(Proof of only if). It is easy to check that the p^i values and B are set in such a way that any YES instance S of 1P-LSG-OPT must contain player m, but cannot contain the upper bound player UB.

Using inequalities (4.3.9b) and (4.3.9c) for players m and UB, respectively, it follows that $P(S)$ must be equal to A. In particular

$$P(S) = A \Rightarrow \frac{Am + \sum_{i \in S \cap \mathcal{I}} a_i}{|S| + 1} = A \Rightarrow \sum_{i \in S \cap \mathcal{I}} a_i = A(|S| + 1 - m).$$

Clearly $0 \leq \sum_{i \in S \cap \mathcal{I}} a_i \leq A$, but furthermore, the first inequality is strict, since $m \in S$. It follows that $m - 1 < |S| \leq m$, so $|S| = m$, and $\sum_{i \in S \cap \mathcal{I}} a_i = A$. □

Theorem 4.3.11 shows that maximizing a linear function over the set of NE is hard, assuming $P \neq NP$. Yet, we can build a pseudo-polynomial time algorithm to solve this problem: let $L_p = C^p + \sqrt{F^p b}$ and $U_p = C^p + 2\sqrt{F^p b}$ for $p = 1, 2, \ldots, m$. We propose to solve this problem using Algorithm 4.3.4.2, where $H(k, l, r, s, C)$ is the optimal value of the problem limited to players $\{1, 2, \ldots, k\}$, where $|S| = l$, the tightest lower bound is L_r, the tightest upper bound is U_s and $\sum_{i \in S} C^i = C$.

From each (k, l, r, s, C), we can choose to either add $k + 1$ or not to the set S, leading to the updates of lines 3 and 4, respectively. At the end, the optimal objective function value is given by the maximum entry $H(m, l, r, s, C)$ leading to a feasible solution. It is easy to build the optimal S by a standard backward pass of the underlying recursion.

Therefore we have established the following result.

Theorem 4.3.12. *Finding the optimal NE in the 1-period lot-sizing game can be solved in $\mathcal{O}(m^4 \sum_{k=1}^{m} C^k)$ time.*

Remark. The potential function (4.3.6) restricted to this case, *i.e.*, $T = 1$ and domain 2^m (power set of $\{1, 2, \ldots, m\}$), is submodular. It is well-known that general submodular functions are hard to maximize. This is the reason why we built an algorithm to compute an NE which is not based on this function.

Algorithm 4.3.4.2

Input: A single period ULSG instance and a vector $p \in \mathbb{Z}^m$.

Output: The optimal value of the input function associated with p over the set of NE.

```
1: Initialize $H(\cdot) \leftarrow -\infty$ but $H(0,0,0,0,0) \leftarrow 0$.
2: for $k = 0 : m-1$; $l, r, s = 0 : k$; $C = 0 : \sum_{i=0}^{k} C^i$ do
3:     $H(k+1, l+1, \arg\max_{i=k+1,r} L_i, s, C+C^k) \leftarrow$
           $\max(H(k+1, l+1, \arg\max_{i=k+1,r} L_i, s, C+C^k), H(k,l,r,s,C) + p^{k+1})$
4:     $H(k+1, l, r, \arg\min_{i=k+1,s} U_i, C) \leftarrow$
           $\max(H(k+1, l, r, \arg\min_{i=k+1,s} U_i, C), H(k,l,r,s,C))$
5: end for
6: return $\arg\max_{l,r,s,C}\{H(m,l,r,s,C) | L_r \leq \frac{a+C}{l+1} \leq U_s\}$.
```

4.3.5 Congestion Game Equivalence: only set-up costs

Throughout this section, we approach the ULSG with only set-up costs, *i.e.*, $C_t^k = 0$ for all $k = 1, 2, \ldots, m$ and $t = 1, 2, \ldots, T$.

There are two immediate important observations valid in this special case. One is that it is always optimal for a player to produce only once in order to minimize the set-up costs. Another is that the strategies in an NE depend only on the number of players sharing the market in each period. From Proposition 4.3.4, if S_t are the players participating in period t, then their revenue is $\frac{a_t^2}{b_t(|S_t|+1)^2}$, with a market price of $P_t(S_t) = \frac{a_t}{|S_t|+1}$.

These observations lead us to a connection with congestion games. A *congestion game* is one where a collection of players has to go from a (source) vertex in a digraph to another (sink) and the cost of using an arc of the graph depends on the number of players also selecting it in their paths; each player's goal is to minimize the cost of her path; see Rosenthal [111]. We can easily reformulate ULSG-sim as a congestion game: consider a digraph $G = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N} = \mathcal{S} \cup \mathcal{T}$ with $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$ and $\mathcal{T} = \{1, 2, \ldots, T, T+1\}$, and $\mathcal{A} = \mathcal{F} \cup \mathcal{P}$ with $\mathcal{F} = \{(s_k, t) : k = 1, 2, \ldots, m \text{ and } t = 1, 2, \ldots, T+1\}$ and $\mathcal{P} = \{(t, t+1) : t = 1, 2, \ldots, T\}$. The cost of arcs $(s_k, t) \in \mathcal{F}$ equals F_t^k; the cost of arcs $(t, t+1) \in \mathcal{P}$ equals $-\frac{a_t^2}{b_t(1+n)^2}$, where n is the number of players selecting this arc. Finally, for each player k the source vertex is s_k and the sink is $T+1$. Figure 4.3.1 illustrates this transformation. This reformulation has polynomial size since, the number of vertices is $m + T + 1$ and the number of arcs is $m(T+1) + T$ (note that the size of ULSG is $O(mT)$ since mT set-up costs are given).

Any congestion game is a potential game as proved by Rosenthal [111] (as well as the converse; see Monderer and Shapley [92]) and the author also provides a potential function.

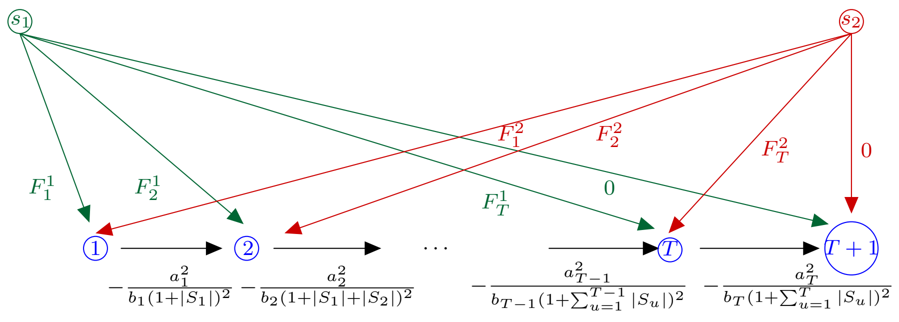

s_1
s_2
F_1^1
F_2^1
F_1^2
F_2^2
F_T^2
0
F_T^1
0
1
2
$\cdots$
T
$T+1$
$-\frac{a_1^2}{b_1(1+|S_1|)^2}$
$-\frac{a_2^2}{b_2(1+|S_1|+|S_2|)^2}$
$-\frac{a_{T-1}^2}{b_{T-1}(1+\sum_{u=1}^{T-1}|S_u|)^2}$
$-\frac{a_T^2}{b_T(1+\sum_{u=1}^{T}|S_u|)^2}$

Figure 4.3.1: Congestion game for ULSG-sim with $m = 2$.

In our case it is

$$\Phi(t^1, \dots, t^m) = \sum_{k=1}^{m} -F_{t^k}^k + \sum_{t=1}^{T} \sum_{k=1}^{n_t} \frac{a_t^2}{(k+1)^2 b_t}, \tag{4.3.13}$$

where $t^k \in \{1, 2, \dots, T+1\}$ is the period in which player k produces and $n_t = \#\{k : t^k \leq t, k = 1, \dots, m\}$. Using the same proof argument as for Proposition 4.3.6, one can prove that a maximizer of 4.3.13 is an NE for ULSG-sim and thus, by Proposition 4.3.5, for ULSG.

For this specific problem, maximizing the potential function (4.3.13) is equivalent to solve the min-cost flow problem in the following network (see Figure 4.3.2):

- consider a digraph $G = (\mathcal{N}', \mathcal{A}')$ where $\mathcal{N}' = \{s\} \cup \mathcal{S} \cup \mathcal{T}$ with $\mathcal{S} = \{s_1, s_2, \dots, s_m\}$ and $\mathcal{T} = \{1, 2, \dots, T, T+1\}$, and $\mathcal{A}' = \mathcal{I} \cup \mathcal{F} \cup \mathcal{P}'$ with $\mathcal{I} = \{(s, s_k) : k = 1, 2, \dots, m\}$, $\mathcal{F} = \{(s_k, t) : k = 1, 2, \dots, m$ and $t = 1, 2, \dots, T+1\}$ and $\mathcal{P}' = \{(t, t+1) : t = 1, 2, \dots, T$ and $k = 1, \dots, m\}$ (m parallel arcs).
- for $(s, s_k) \in \mathcal{I}$ the cost is 0 and capacity is 1;
- for $(s_k, t) \in \mathcal{F}$ the cost is F_t^k and capacity is 1; set $F_{T+1}^k = 0$;
- for $(t, t+1) \in \mathcal{P}'$ and $k = 1, \dots, m$, the cost is $-\frac{a_t^2}{b_t(1+k)^2}$ and capacity is 1;
- the supply is m in vertex s and the demand at $T+1$ is m.

Observe that this reformulation is polynomial in the size of an ULSG instance: the network has $1+m+T+1$ vertices and $m+m(T+1)+mT$ arcs. The advantage of this reformulation is that solving a min-cost flow problem can be done in polynomial time; Goldberg and Tarjan [59].

There is another alternative approach to compute a, possibly distinct, NE. A maximum of the potential function (4.3.6) is an NE and it is in the subset of strategies in which the players decide the production period and choose the optimal quantities accordingly with Proposition 4.3.4. Therefore, restricting function (4.3.6) to this subset of strategies,

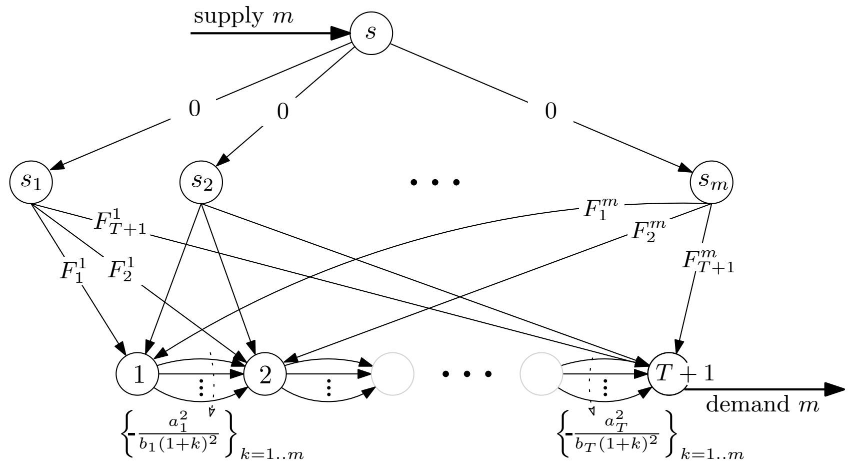

supply m
s
0
0
0
s_1
s_2
s_m
F^1_{T+1}
F^1_1
F^1_2
F^m_1
F^m_2
F^m_{T+1}
1
2
$T+1$
demand m
$\left\{-\frac{a_1^2}{b_1(1+k)^2}\right\}_{k=1..m}$
$\left\{-\frac{a_T^2}{b_T(1+k)^2}\right\}_{k=1..m}$

Figure 4.3.2: Minimum cost flow approach to optimize (4.3.13). All arcs have unit capacity.

it simplifies to

$$\Phi(t_1,t_2,\ldots,t_m) = \sum_{p=1}^{m}\left(-F^p_{t_p} + \sum_{t=t_p}^{T}\frac{a_t^2}{b_t(n_t\ +1)^2} + \sum_{t=t_p}^{T}\frac{a_t^2}{2(n_t+1)}(n_t-1)\frac{a_t^2}{(n_t+1)b_t}\right)$$

$$= \sum_{p=1}^{m} -F^p_{t_p} + \sum_{t=1}^{T}\frac{a_t^2}{2b_t(n_t+1)}n_t \tag{4.3.14a}$$

$$= \sum_{p=1}^{m} -F^p_{t_p} + \sum_{t=1}^{T}\sum_{i=1}^{n_t}\frac{a_t^2}{2i(i+1)b_t}. \tag{4.3.14b}$$

Once again, computing the maximum of (4.3.14b) is equivalent to solve a min-cost flow problem similar to the one in Figure 4.3.2 (the difference is in the cost of the arcs $(t,t+1)$ which are $\{\frac{a_t^2}{2k(k+1)b_t}\}_{k=1,\ldots,m}$ for $t=1,\ldots,T$).

We remark that there are instances for which the optimal solutions for the maximums of 4.3.13 and 4.3.14b do not coincide and thus, two distinct NE can be computed in polynomial time.

The results of this section are summarized in the following theorem.

Theorem 4.3.13. *When $C^k_t = 0$ for $k = 1,2,\ldots,m$ and $t = 1,2,\ldots,T$, an NE for an ULSG can be computed in polynomial time by solving a minimum-cost network flow problem.*

4.3.6 Extension: inventory costs

Recall from Section 2.2.1.3 that in the lot-sizing problem (2.2.7) inventory costs are taken into account, a natural aspect in real-world applications that influences the optimal production plan; as noted there, using the flow conservation constraints (2.2.7b), an ULSP can be transformed in an equivalent one without inventory costs which are included in the new variable costs. In ULSG, if each player p's objective (4.3.1a) considers inventory costs H^p_t for each period t, an analogous replacement of the inventory variables h^p_t (through constraint (4.3.1b)) results in new variable production costs, but also in new market prices; these market prices depend on each player's inventory costs; therefore, since in the results previously presented, we consider equal market prices for each player, the inclusion of inventory costs requires an adaption of them.

Proposition 4.3.14. *Consider an ULSG with each player p's utility function equal to*

$$\Pi^p(y^p, x^p, h^p, q^p, q^{-p}) = \sum_{t=1}^{T} P_t(q_t)q^p_t - \sum_{t=1}^{T} C^p_t x^p_t - \sum_{t=1}^{T-1} H^p_t h^p_t - \sum_{t=1}^{T} F^p_t y^p_t. \tag{4.3.15}$$

The results presented in Section 4.3.2 and Section 4.3.3 for each player p hold if a_t is replaced by $a^p_t = a_t + \sum_{u=t}^{T-1} H^p_u$, C^p_t is replaced by $\hat{C}^p_t = C^p_t + \sum_{u=t}^{T-1} H^p_u$ and $P_t(S_t)$ is replaced by $P^p_t(S_t) = a^p_t + \frac{\sum_{i\in S_t}\left(\hat{C}^i_{t^i_j} - a^i_t\right)}{|S_t|+1}$.

Proof. One can use constraints (4.3.1b) to eliminate the inventory variables in player p's objective function (4.3.15). Thus, using $h^p_t = \sum_{u=1}^{t}(x^p_t - q^p_t)$ in the objective function (4.3.15), leads to

$$\Pi^p(y^p, x^p, h^p, q^p, q^{-p}) = \sum_{t=1}^{T}(a^p_t - b_t q_t)^+ q^p_t - \sum_{t=1}^{T} \hat{C}^p_t x^p_t - \sum_{t=1}^{T} F^p_t y^p_t.$$

and the proof follows. □

4.3.7 Summary

In the uncapacitated lot-sizing game, the production cost of player p in period t depends on two parameters: the variable cost C^p_t and the set-up cost F^p_t. When we consider production costs with only one of these parameters or a single period, the problem of computing a pure equilibrium becomes tractable, although characterizing the set of pure equilibria is NP-complete. Table 4.1 summarizes our findings.

It remains open the question of whether it is a tractable problem to find an optimal NE when there are no variable costs and if an NE can be efficiently computed for the general

Problem	Compute one NE	Characterize the set of NE
ULSG with $T = 1$	P	NP-complete
ULSG with $F = 0$	P	P
ULSG with $C = 0$	P	?
ULSG	?	NP-complete

Table 4.1: Computational complexity of ULSG.

case. As we will see in the next section, in practice, it is fast to compute one equilibrium for ULSG.

A typical constraint in the lot-sizing problem 2.2.7 is the presence of positive initial and final inventory quantities, which for the uncapacitated case can be assumed to be 0, without loss of generality, by modifying the demands (see Pochet and Wolsey [106]). In one hand, considering positive initial and final inventory quantities in ULSG for each player does not interfere with the fact that the game is potential, since the the objective function does not change. On the other hand, this is problematic when characterizing each player's best response, since in the game there is no fixed demand to satisfy. Therefore, it is interesting to study the influence of relaxing the assumption that initial and final inventories are zero in future research.

When production capacities are introduced in LSP, it becomes NP-complete (see [106]). Thus, if there are players' production capacities for each period in our game, solving each player's best response becomes NP-complete. Note that this does not interfere in the formulation of a player's utility function, and thus the game remains potential with only the potential function domain reduced (set of pure profiles of strategies X).

Therefore, including more restrictions (*e.g.* positive initial and/or final inventory quantities, production capacities) on the lot-sizing model of each player will not change the fact that the game is potential (with the potential function concave) and thus, that it posses a pure NE. It remains to understand the computational complexity of maximizing the potential function (and thus, computing an NE).

4.4 Integer Programming Games

[5]

Motivation. Mixed integer programming has been extensively studied. Along with its success modeling decision problems, we have seen a remarkable rise in the power of solvers to tackle them; recall Section 2.2. Many state-of-the-art game theory tools are confined to finite games and "well-behaved" continuous games[6]; see Section 2.3.2. Our aim is to investigate the continuous game class, where the players' sets of strategies mix finite and uncountable sets; to this end, the players' best reactions are described through mixed integer programming problems. We call problems in this class integer programming games; see Figure 2.3.1.

In the previous sections, real-world interactions were modeled as simultaneous IPGs, highlighting the importance of exploring them. Note that for these games, enumerating all players' feasible strategies (as in finite games) can be impractical, and the players' objectives (best reactions) may lead to non-concave problems. Thus, the standard approaches for finite games and "well-behaved" continuous games are not directly applicable to IPGs.

In what follows, we study IPGs where each player's utility function is quadratic in her variables. Recalling the notation to define IPG in Section 2.3, each player p's utility function is

$$\Pi^p(x^p, x^{-p}) = c^p x^p + \sum_{k \in M} (x^k)^\intercal Q^p_k x^p, \tag{4.4.1}$$

where $c^p \in \mathbb{R}^{n_p}$ and Q^p_k is an n_k-by-n_p real matrix. Note that in the games previously described in this chapter, the players' utilities have the form (4.4.1). We make this assumption on the players' utilities for sake of clarity, although the algorithm that will be presented for the computation of equilibria is also applicable to games with more general utility functions.

Our Contributions and Organization of the Section. We aim at investigating Nash equilibria to simultaneous IPGs. In Section 4.4.1, it is proven that the existence of NE to an IPG is Σ^p_2-complete and sufficient conditions for equilibria existence are

[5]The results of this chapter appear in:
M. Carvalho, A. Lodi, J. P. Pedroso. Computing Nash equilibria: integer programming games, working paper.

[6]In "well-behaved" continuous games, players' best reaction problems (2.3.4) (maximization problems) must satisfy certain differentiability and concavity requirements.

derived. Section 4.4.2 starts by showing the challenge of extending integer programming methods to computing NE and formalizes a novel algorithm to compute an NE for IPGs. In Section 4.4.3, implementation details to our algorithm are described and validated, through computational results for a generalization of the coordination knapsack game (4.1.1) and the competitive uncapacitated lot-sizing game (4.3.1). Finally, we conclude in Section 4.4.4.

4.4.1 NE Complexity and Existence

It can be argued that players' computational power is bounded and thus, since the space of pure strategies is simpler and contained in the space of mixed strategies – *i.e.*, the space of Borel probability measures – pure equilibria are more plausible outcomes for games with large sets of pure strategies. In this way, it is important to understand the complexity of determining a pure equilibrium to an IPG.

According with Nash famous Theorem 2.3.11, all purely integer bounded IPGs have a Nash equilibrium. However, some IPGs do not possess a pure equilibrium, as illustrated in the following example.

Example 4.4.1. *Consider a simultaneous two-player game with $M = \{A, B\}$. Player A solves*

$$\begin{aligned} \max_{x^A} \quad & 18x^A x^B - 9x^A \\ s.\ t. \quad & x^A \in \{0,1\} \end{aligned}$$

and player B

$$\begin{aligned} \max_{x^B} \quad & -18x^A x^B + 9x^B \\ s.\ t. \quad & x^B \in \{0,1\}. \end{aligned}$$

Let us show that none of the pure profiles of strategies is an equilibrium. Under the profile $(x^A, x^B) = (0,0)$, player B has incentive to deviate to $x^B = 1$; for the profile $(x^A, x^B) = (1,0)$, player A has incentive to deviate to $x^A = 0$; for the profile $(x^A, x^B) = (0,1)$, player A has incentive to deviate to $x^A = 1$; for the profile $(x^A, x^B) = (1,1)$ player B has incentive to deviate to $x^B = 0$. Thus, there is no pure NE.

In Section 4.4.1.1, we classify both the computational complexity of deciding if there is a pure and a mixed NE for an IPG. It will be shown that even with linear utilities and two players, these problems are Σ_2^p-complete (defined in Section 2.1). Then, in Section 4.4.1.2, we state sufficient conditions for the game to have (finitely supported) Nash equilibria.

4.4.1.1 Complexity of the NE Existence

Theorem 4.4.2. *The problem of deciding if an IPG has a pure NE is Σ_2^p-complete.*

Proof. The problem of deciding if an IPG has a pure NE is in Σ_2^p, since we have to decide if there is a solution in the space of pure strategies such that for any unilateral deviation of a player, her utility is not improved (and evaluating the utility value for a profile of strategies can be done in polynomial time, because we consider them to be in the form (4.4.1)).

It remains to prove Σ_2^p-hardness; we will reduce DNeg to it. Recall from Section 3.1.3 that the input of DNeg are non-negative integers $a_1, \ldots, a_n, b_1, \ldots, b_n, A$ and B; from the proof of Theorem 3.2.1, the decision version of DNeg asks whether there is a leader strategy that makes her objective value less or equal to $B-1$. Our reduction starts from an instance of DNeg. We construct the following instance of IPG.

- The game has two players, $M = \{Z, W\}$.
- Player Z controls a binary decision vector z of dimension $2n+1$; her set of feasible strategies is
$$\begin{aligned} \sum_{i=1}^{n} a_i z_i &\leq A \\ z_i + z_{i+n} \leq &\ 1 \qquad i = 1, \ldots, n \\ z_{2n+1} + z_{i+n} \leq &\ 1 \qquad i = 1, \ldots, n. \end{aligned}$$
- Player W controls a binary decision vector w of dimension $n+1$; her set of feasible strategies is
$$B w_{n+1} + \sum_{i=1}^{n} b_i w_i \leq B. \tag{4.4.5}$$
- Player Z's utility is $(B-1)w_{n+1}z_{2n+1} + \sum_{i=1}^{n} b_i w_i z_{i+n}$.
- Player W's utility is $(B-1)w_{n+1} + \sum_{i=1}^{n} b_i w_i - \sum_{i=1}^{n} b_i w_i z_i - \sum_{i=1}^{n} b_i w_i z_{i+n}$.

We claim that in the constructed instance of IPG there is an equilibrium if and only if the DNeg instance has answer YES.

(Proof of if). Assume that the DNeg instance has answer YES. Then, there is x satisfying $\sum_{i=1}^{n} a_i x_i \leq A$ such that $\sum_{i=1}^{n} b_i y_i \leq B-1$ for any y satisfying constraints (3.1.3c) and (3.1.3d). Choose as strategy for player Z, $\widehat{z} = (x, \overbrace{0, \ldots, 0}^{n}, 1)$ and for player W $\widehat{w} = (\overbrace{0, \ldots, 0}^{n}, 1)$. We will prove that $(\widehat{z}, \widehat{w})$ is an equilibrium. First, note that these strategies are guaranteed to be feasible for both players. Second, note that none of the players has incentive to deviate from $(\widehat{z}, \widehat{w})$:

- Player Z's utility is $B-1$, and $B-1 \geq \sum_{i=1}^{n} b_i w_i$ holds for all the remaining feasible strategies w of player W.
- Player W's has utility $B-1$ which is the maximum possible given $\widehat{z}$.

(Proof of only if). Now assume that the IPG instance has answer YES. Then, there is a pure equilibrium $(\widehat{z}, \widehat{w})$.

If $\widehat{w}_{n+1} = 1$ then, by (4.4.5), $\widehat{w} = (\overbrace{0, \dots, 0}^{n}, 1)$. In this way, since player Z maximizes her utility in an equilibrium, $\widehat{z}_{2n+1} = 1$, forcing $\widehat{z}_{i+n} = 0$ for $i = 1, \dots, n$. The equilibrium inequalities (2.3.14) applied to player W, imply that for any of her feasible strategies w with $w_{n+1} = 0$:

$$B - 1 \geq \sum_{i=1}^{n} b_i w_i (1 - \widehat{z}_i),$$

which shows that DNeg is a YES instance with the leader selecting $x_i = \widehat{z}_i$ for $i = 1, \dots, n$.

If $\widehat{w}_{n+1} = 0$, under the equilibrium strategies, player Z's utility term $(B-1)\widehat{w}_{n+1} z_{2n+1}$ is zero. Thus, since in an equilibrium player Z maximizes her utility, it holds that $\widehat{z}_{i+n} = 1$ for all $i = 1, \dots, n$ with $\widehat{w}_i = 1$. However, this implies that player W's utility is non-positive given the profile $(\widehat{z}, \widehat{w})$. In this way, player W would strictly improve her utility by unilaterally deviating to $w = (\overbrace{0, \dots, 0}, 1)$. In conclusion, w_{n+1} is never zero in a pure equilibrium of the constructed game instance. □

Extending the existence property to mixed equilibria would increase the chance of an IPG to have an NE, and thus, a solution. The next theorem shows that the problem remains Σ_2^p-complete.

Theorem 4.4.3. *The problem of deciding if an IPG has an NE is Σ_2^p-complete.*

Proof. Analogously to the previous proof, the problem belongs to Σ_2^p.

It remains to show the Σ_2^p-hardness; we will reduce DeRi to it. Recall from Section 3.1.1 that the input of DeRi are non-negative integers a_1, ..., a_n, b_1, ..., b_n, A, C and C'; from the proof of Theorem 3.2.1, the decision version of DeRi asks whether there is a leader strategy that makes her objective value greater or equal to 1. Our reduction starts from an instance of DeRi. We construct the following instance of IPG.

- The game has two players, $M = \{Z, W\}$.
- Player Z controls a non-negative variable z and a binary decision vector $(z_1, \dots, z_{n+1})$;

her set of feasible strategies is

$$\begin{aligned}
\sum_{i=1}^{n} b_i z_i \leq\ & z \\
z_i + z_{n+1} \leq 1, & \qquad\qquad i = 1, \ldots, n \\
z \leq & C'(1 - z_{n+1}) \\
z \geq & C(1 - z_{n+1}).
\end{aligned}$$

- Player W controls a non-negative variable w and binary decision vector $(w_1, \ldots, w_n)$.
- Player Z's utility is $Az + \sum_{i=1}^{n} a_i z_i w_i + z_{n+1}$.
- Player W's utility is $z_{n+1} w + \sum_{i=1}^{n} b_i w_i z_i$.

We claim that in the constructed instance of IPG there is an equilibrium if and only if the DeRi instance has answer YES.

(Proof of if). Assume that the DeRi instance has answer YES. Then, there is x such that $C \leq x \leq C'$ and $Ax + \sum_{i=1}^{n} a_i y_i \geq 1$ for a y satisfying $\sum_{i=1}^{n} b_i y_i \leq x$. As strategy for player Z choose $\widehat{z} = C'$ and $(\widehat{z}_1, \ldots, \widehat{z}_n, \widehat{z}_{n+1}) = (y_1, \ldots, y_n, 0)$; for player W choose $\widehat{w} = 0$ and $(\widehat{w}_1, \ldots, \widehat{w}_n) = (y_1, \ldots, y_n)$. We prove that $(\widehat{z}, \widehat{w})$ is an equilibrium. First, note that these strategies are guaranteed to be feasible for both players. Second, note that none of the players has incentive to deviate from $(\widehat{z}, \widehat{w})$:

- Player Z's utility cannot be increased, since it is equal or greater than 1 and for $i = 1, \ldots, n$ such that $\widehat{z}_i = 0$ the utility coefficients are zero.
- Analogously, player W's utility cannot be increased, since for $i = 1, \ldots, n$ such that $\widehat{w}_i = 0$ the utility coefficients are zero and the utility coefficient of $\widehat{w}\widehat{z}_{n+1}$, is also zero.

(Proof of only if). Assume that DeRi is a NO instance. Then, for any x in $[C, C']$ the leader is not able to guarantee a utility of 1. This means that in the associated IPG, player Z has incentive to choose $z = 0$ and $(z_1, \ldots, z_n, z_{n+1}) = (0, \ldots, 0, 1)$. However, this player Z's strategy leads to a player W's unbounded utility. In conclusion, there is no equilibrium. $\square$

Remark. In the proof of Theorem 4.4.3, it is not used the existence of a mixed equilibrium to the constructed IPG instance. Therefore, it implies Theorem 4.4.2. The reason for presenting these two theorems is because in Theorem 4.4.2, the reduction is a game where the players have finite sets of strategies, while in Theorem 4.4.3, in the reduction, a player has an infinite set of strategies.

4.4.1.2 Existence of NE

Recall the theoretical results of Stein et al. [120], presented previously, in Section 2.3.2. The simultaneous IPGs that we study in this thesis have quadratic utility functions given by function (4.4.1); therefore, it is easy to see that each player p's utility can be written in the form of function (2.3.22), as required for separable games. Thus, we conclude:

Lemma 4.4.4. *If the set of strategies' profiles X for an IPG is bounded and the players' utilities have the form* (4.4.1)*, then IPG is a separable game.*

Lemma 4.4.4 and Corollary 2.3.15 give sufficient conditions for the existence of equilibria, and, moreover, the existence of finitely supported equilibria.

Corollary 4.4.5. *If the set of strategies' profiles X for an IPG is bounded, then it is a continuous game and it has a Nash equilibrium. In addition, if the utilities have the form* (4.4.1)*, for any Nash equilibrium σ there is a Nash equilibrium τ such that each player p mixes among at most $1+n_p+\frac{n_p(n_p-1)}{2}$ pure strategies and $\Pi^p(\sigma)=\Pi^p(\tau)$.*

Proof. In order to write player p's utility (4.4.1) in the form (2.3.22), there must be a function $f^p_{j_p}(x^p)$ for 1, x^p_1, …, $x^p_{n_p}$, $x^p_1x^p_1$, $x^p_1x^p_2$, …, $x^p_1x^p_{n_p}$, $x^p_2x^p_2$, …, $x^p_{n_p}x^p_{n_p}$; thus, $k_p=1+n_p+\frac{n_p(n_p-1)}{2}$ in Corollary 2.3.15. □

It is realistic to assume that the set of strategies X for an IPG is bounded. In other words, the players' strategies are likely to be bounded due to limitations in the players' resources, which guarantees that an IPG has an equilibrium. Moreover, by Corollary 4.4.5, this condition implies that the set of NE can be restricted to the finitely supported NE, which simplifies the equilibria computation.

4.4.2 Algorithm to Compute an NE

In this section an algorithm for the computation of an NE is proposed. But first, let us classify the complexity of finding an equibrium for a separable IPG.

When the definition of IPG was introduced, we remarked in Section 2.3 (see Figure 2.3.1), that any finite game can be transformed (in polynomial time) in an IPG. Furthermore, we mentioned Chen *et al.* [23]'s result, stating that solving a finite game (even with only two players) is PPAD-complete. Therefore:

Corollary 4.4.6. *The problem of computing an equilibrium to a separable IPG is PPAD-hard.*

In what follows, we assume that the IPGs in hand have X bounded, *i.e.*, they are separable games, and thus, their set of NE can be characterized by finitely-supported equilibria (Corollary 4.4.5). Next, in Section 4.4.2.1, we show that the standard idea in mathematical programming of relaxing integrality requirements does not provide IPGs with interesting information. Thus, the problem must be tackled from another perspective. The algorithm designed in Section 4.4.2.2 will approximate an IPG iteratively, incorporating the Porter-Nudelman-Shoham method (PNS) [107] (described in Section 2.3.2) and mathematical programming solvers (see Section 2.2). The basic algorithm is modified in Section 4.4.2.3 in an attempt to improve its performance.

4.4.2.1 Game Relaxation

A typical procedure to solve optimization problems consists in relaxing constraints that are hard to handle and 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 an IPG and the associated RIPG.

Example 4.4.7 (RIPG with 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

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

It is easy to see that the IPG has an unique equilibrium: $(x^A, x^B) = ((1,0),(1,0))$. This equilibrium also holds for the corresponding RIPG. However, RIPG possesses at least one more equilibrium: $(x^A, x^B) = ((0, \frac{2}{3}), (0, \frac{2}{3}))$.

Example 4.4.8 (RIPG with fewer 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

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

It is easy to see that there are at least 2 equilibria: $(x^A, x^B) = ((0,0),(0,0))$ *and* $(x^A, x^B) = ((0,1),(0,1))$*; however, from this set of equilibria to the IPG, none is an equilibrium of the associated RIPG. In fact, in 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' utilities in an NE can be extracted from the relaxation of an IPG. Moreover, to the best of our knowledge, the methods to compute equilibria of RIPGs are restricted to pure equilibria.

4.4.2.2 Algorithm Formalization

Recall the Nash equilibria conditions (2.3.14): find $(\sigma^1, \ldots, \sigma^m)$ such that

$$\sigma^p \in \Delta^p \qquad \forall p \in M \tag{4.4.7a}$$

$$\Pi^p(\sigma^p, \sigma^{-p}) \geq \Pi^p(x^p, \sigma^{-p}) \qquad \forall p \in M, \qquad \forall x^p \in X^p, \tag{4.4.7b}$$

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 uncountable or, in case the variables are all required to be integer and bounded, to be exponential. Thus, in general, the equilibria inequalities (4.4.7b) are unsuitable to be written for each pure strategy in X^p. We call *sample game* of an IPG to the finite game that results from restricting the players to a finite subset of feasible strategies of X.

Following the motivation idea of column generation [57] and cutting plane [60] approaches, not all variables and constraints in problem (4.4.7) may be needed to find a solution. In this context, the natural idea to find a solution to the constrained programming problem (4.4.7) 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 sample 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 *sample generation method* (SGM). Figure 4.4.1 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, converge 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, as the following example illustrates, SGM can only guarantee the computation of

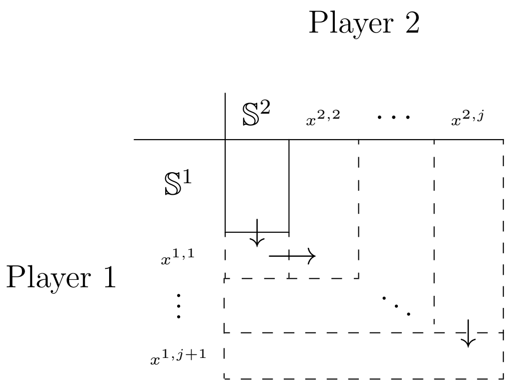


Figure 4.4.1: SGM: Sample generation method for $m = 2$. The notation $x^{p,k}$ represents the player p's strategy added at iteration k. A vertical (horizontal) arrow represents player 1 (player 2) incentive to unilaterally deviate from the previous sample game's equilibrium to a new strategy of her.

an ϵ-*equilibrium*, that is, a profile of strategies $\sigma \in \Delta$ such that for each player $p \in M$ the following inequalities hold

$$\Pi^p(\sigma) + \epsilon \geq \Pi^p(x^p, \sigma^{-p}) \quad \forall x^p \in X^p. \tag{4.4.8}$$

A 0-equilibrium is a NE. In this way, $\epsilon > 0$ becomes an input of SGM and this method stopping criteria is the following: if there is no player able to unilaterally increase more than ϵ her utility at the equilibrium σ of the current sample game, return σ.

Before providing the SGM a proof of correctness, in an attempt to clarify the method, we apply it to an instance of IPGs.

Example 4.4.9 (Computing an equilibrium with SGM)**.** *Consider an IPG with two players, $M = \{1, 2\}$. Player i wishes to maximize the utility function $\max_{x^i \geq 0} -(x^i)^2 + x^1x^2$. The player i's best reaction is given by $x^i(x^{-i}) = \frac{1}{2}x^{-i}$. The only equilibrium is $(x^1, x^2) = (0, 0)$. Initialize SGM with the sample 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 4.4.2. Thus, SGM converges to the equilibrium $(0, 0)$. If in the input of SGM, $\epsilon = 10^{-6}$ then, after 14 iterations, SGM would return an ϵ-equilibrium of the game.*

Theorem 4.4.10. *If X is 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 utility function is Lipschitz continuous in X^p.*

Proof. Since X is bounded, by Corollary 4.4.5, there is a finitely supported NE.

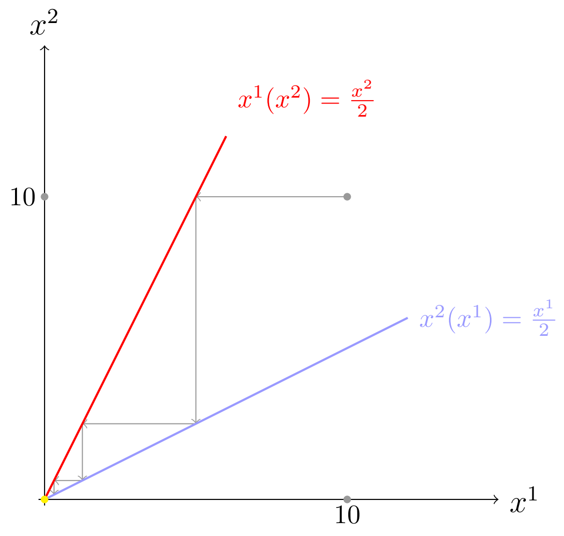


Figure 4.4.2: Players' best reaction functions.

SGM stops once an equilibrium of the sample 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 sample game will coincide with IPG, *i.e.*, $\mathbb{S} = X$. This means that an NE of the sample game is an NE of the IPG.

Case 2: Each player p utility 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 $\parallel x^p - \hat{x}^p \parallel \leq \frac{\epsilon}{L^p}$. Let σ be an NE of the sample game. Then

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

The first step follows from the fact that σ is an NE of the sample 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 utility is Lipschitz continuous in X^p. In this way σ is an ϵ-equilibrium of the IPG.

$\square$

Remark: As pointed out by Stein *et al.* [120] for a specific separable game, it seems that there must be some bound on the speed of variation of the utilities in order to guarantee that an algorithm computes an equilibrium in finite time; the Lipschitz condition ensures this bound. A utility function which is linear in that player's variables is Lipschitz continuous; a quadratic utility function when restricted to a bounded set satisfies the Lipschitz condition as will be the case of the competitive uncapacitated lot-sizing game.

A relevant fact about computing equilibria for a sample 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 (Nash's Theorem 2.3.11 states that any finite game has a Nash equilibrium). Given the good results achieved by PNS [107] for the computation of an NE in normal-form games (finite games), this is the method that our algorithm will apply to solve the sample games (additional advantages for adopting PNS will be given in the end of this section). Recall from Section 2.3.2 that PNS solves the constrained program (4.4.7) associated with a sample game (*i.e.*, $X = \mathbb{S}$ in (4.4.7)) through the resolution of simpler subproblems (note that in constraints (4.4.7b) the expected utilities $\Pi^p(\sigma^p, \sigma^{-p})$ 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 solving the associated Feasibility Problem (2.3.21). PNS reduces the set of candidates for support enumeration by making use of conditionally dominated strategies, since such strategies will never be selected with positive probability in an equilibrium. In addition, in the support enumeration of our implemention of PNS we require the equilibrium to satisfy the property given by Corollary 4.4.5: each player p has a support size of at most $1 + n_p + \frac{n_p(n_p-1)}{2}$.

We conclude SGM description by highlighting an additional advantage of PNS, besides being in practice the fastest algorithm. The authors' implementation of PNS [107] 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. 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 [82] 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 sample games in iterations 1 to $k-1$ is an NE of the current sample 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 sample game, these polytopes change deeply.

4.4.2.3 Modified SGM

Finally, through the tools described, we can slightly change SGM in an attempt to speed up the computation of an equilibrium. Its new version will be a depth-first search: while in SGM the size of the sample game strictly increases from one iteration to the next one, in its depth-first search version it will be possible to backtrack to previous sample games, with the aim of decreasing the size of the sample game. In each iteration of the improved SGM, we search for an equilibrium in the support the last strategy added to the sample game; in case such equilibrium does not exist, the method backtracks, and computes a new equilibrium to the previous sample 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 sample game small and decrease 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 4.4.11. *Let $\mathbb{S} = \mathbb{S}^1 \times \mathbb{S}^2 \times \ldots \times \mathbb{S}^m$ represent a sample 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 $x^p \in X^p$ (that is not in $\mathbb{S}^p$). By contradiction, assume that there is an equilibrium τ in $\mathbb{S}'$ such that x^p

ALGORITHMS	DESCRIPTION
$Initialization(IPG)$	Returns sample game of the IPG with one feasible strategy for each player.
$PlayerOrder(\mathbb{S}_{dev_0}, \dots, \mathbb{S}_{dev_k})$	Returns a list of the players order that takes into account the algorithm history.
$DeviationReaction(p, \sigma_k^{-p}, \Pi^p(\sigma_k), \epsilon, IPG)$	If there is $x^p \in X^p$ such that $\Pi^p(x^p, \sigma_k^{-p}) > \Pi^p(\sigma_k) + \epsilon$, returns x^p; otherwise, returns any player p's feasible strategy.
$SortSizes(\sigma_0, \dots, \sigma_{k-1})$	Returns an order for the support sizes enumeration that takes into account the algorithm history.
$SortStrategies(\mathbb{S}, \sigma_0, \dots, \sigma_{k-1})$	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 sample 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 4.2: Specialized algorithms.

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 sample 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 sample games there is more than one NE and thus, an equilibrium with the new added strategy in the support may fail to exist (Theorem 4.4.11 does not apply). Therefore, backtracking is introduced so that a previously processed sample 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. This algorithm is described in pseudo-code 4.4.2.1. The algorithms called by it are described in Table 4.2 and can be defined independently. We will propose an implementation of them in Section 4.4.3.2.

Let us explain in more detail our method for which Figure 4.4.3 can be illustrative. Fundamentally, whenever m-SGM 4.4.2.1 moves *forward*, Step 3, a new strategy $x(k+1)$ is added to the sample game k that is expected to be in the support of the equilibrium of that game (due to Theorem 4.4.11). For the sample 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, "if" part of Step 4, the algorithm *backtracks*: *revisits* the sample 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 4.4.2.1 that it starts from a sample game of the IPG with an unique equilibrium; to this end, the initialization determines one feasible solution for each player. See example B.0.21 in the Appendix B to clarify the application of the m-SGM 4.4.2.1.

Algorithm 4.4.2.1 Modified SGM

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

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

```
Step 1 Initialization:
    $\mathbb{S} = \prod_{p \in M} \mathbb{S}^p \leftarrow$ Initialization(IPG)
    $k \leftarrow 0$
    set $\mathbb{S}_{dev_k}, \mathbb{S}_{dev_{k+1}}$ and $\mathbb{S}_{dev_{k+2}}$ to be $\prod_{p \in M} \emptyset$
    $\sigma_k \leftarrow (1, \dots, 1)$ is Nash equilibrium of the current sampled game $\mathbb{S}$
    $list \leftarrow PlayerOrder(\mathbb{S}_{dev_0}, \dots, \mathbb{S}_{dev_k})$
Step 2 Termination:
    while list non empty do
        $p \leftarrow list.pop()$
        $x(k+1) \leftarrow DeviationReaction(p, \sigma_k^{-p}, \Pi^p(\sigma_k), \epsilon, IPG)$
        if $\Pi^p(\sigma_k) + \epsilon < \Pi^p(x(k+1), \sigma_k^{-p})$ then
            go to Step 3
    return $\sigma_k$, $\mathbb{S}$, $k$
Step 3 Generation of next sampled game:
    $k \leftarrow k+1$
    $\mathbb{S}^p_{dev_k} \leftarrow \mathbb{S}^p_{dev_k} \cup \{x(k)\}$
    $\mathbb{S}^p \leftarrow \mathbb{S}^p \cup \{x(k)\}$
    $\mathbb{S}_{dev_{k+2}} \leftarrow \prod_{p \in M} \emptyset$
Step 4 Solution of sampled game $k$
    $Sizes_{ord} \leftarrow SortSizes(\sigma_0, \dots, \sigma_{k-1})$
    $Strategies_{ord} \leftarrow SortStrategies(\mathbb{S}, \sigma_0, \dots, \sigma_{k-1})$
    $\sigma_k \leftarrow \text{PNS}_{adaptation}(\mathbb{S}, x(k), \mathbb{S}_{dev_{k+1}}, Sizes_{ord}, Strategies_{ord})$
    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 $\sigma_{k-1}$ and $\mathbb{S}_{k+2}$
        $k \leftarrow k-1$
        go to Step 4
    else
        $list \leftarrow PlayerOrder(\mathbb{S}_{dev_0}, \dots, \mathbb{S}_{dev_k})$
        go to Step 2
```

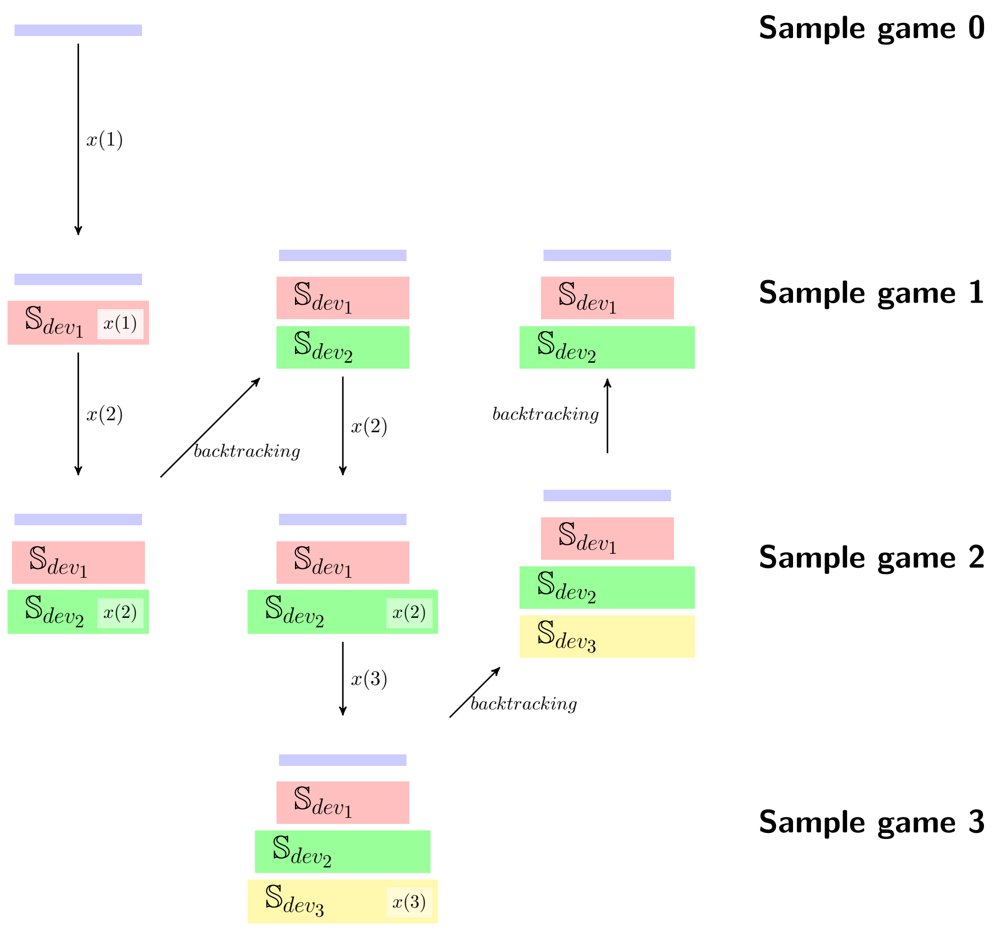


Figure 4.4.3: Sample games generated by m-SGM.

Next, modified SGM 4.4.2.1 correctness will be proven.

Lemma 4.4.12. *In the m-SGM 4.4.2.1, the sample game* 0 *is never revisited.*

Proof. If the sample game 0 is revisited, it would be because the algorithm backtracks. Suppose that at some sample game $k > 0$, the algorithm consecutively backtracks up to the sample game 0. Consider the first sample 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, sample game $j+1$, had multiple equilibria. By Theorem 4.4.11, when the algorithm moves forward from this sample 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, sample 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 sample game $j+1$ or *(case 2)* proceeds to the sample game $x(j+3)$. 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 supports and, by construction, the sample game $j+1$ has multiple equilibria. Under case 2, the same reasoning holds: the algorithm will backtrack to the sample game $j+2$ or move forward to the sample game $j+3$. In this way, because of the multiple equilibria in the successors of sample game j, the algorithm will never be able to return to the sample game j and thus, to the sample game 0. □

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

Lemma 4.4.13. *There is a strict increase in the size of the sample game k when the m-SGM 4.4.2.1 revisits it.*

Corollary 4.4.14. *If X is bounded, then in a finite number of steps, modified SGM 4.4.2.1 computes*

1. *an equilibrium if all players' decision variables are integer;*
2. *an ϵ-equilibrium if each player p's utility 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 4.4.12 the algorithm does not revisit sample game 0, it does not run into an error. Moreover, if the algorithm is moving forward to a sample game k, there is a strict increase in the size from the sample game $k-1$ to k. If the algorithm is revisiting a sample game k, by Lemma 4.4.13, there is also a strict increase of it in comparison with the previous sample game k. Therefore, applying the reasoning of Theorem 4.4.10 proof, the m-SGM will compute an equilibrium (case 1) or ϵ-equilibrium (case 2) in a finite number of steps. □

Remark. The m-SGM 4.4.2.1 is initialized with a sample 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 sample game with an unique equilibrium is valid. Furthermore, the m-SGM 4.4.2.1 might be easily adapted in order to be initialized with a sample game containing more than one NE. In the adaptation, backtracking to the sample game 0 can occur and thus, the PNS support enumeration must be total, this is, all NE of the sample game 0 must be feasible. The fundamental reasoning is similar to the one of the proof of Lemma 4.4.12: if there is backtracking up to the initial sample game 0, it is because it must contain an NE not previously computed, otherwise the successors would have successfully computed one.

4.4.3 Computational Investigation

Section 4.4.3.1 presents the two (separable) simultaneous IPGs in which m-SGM 4.4.2.1 and SGM will be tested. In Section 4.4.3.2, our implementations of the specific components in Table 4.2 are described, which have practical influence in the algorithms' performance. Our algorithms are validated in Section 4.4.3.3 by computational results on instances of two IPGs, the knapsack game and the competitive uncapacitated lot-sizing game.

4.4.3.1 Games: case studies

Next, the two games in which we test our algorithms are described: the knapsack game is the simplest purely integer programming game that one could devise, and the competitive uncapacitate lot-sizing game has practical applicability. The two-player kidney exchange game of Section 4.2 can be successfully solved in polynomial time, and, for that reason, we do not run m-SGM and SGM on its instances.

Let us label the set of players according with $M = \{1, 2, \dots, m\}$.

Knapsack game. This game is very similar to the two-player coordination knapsack game of Section 4.1. One of the most simple and natural IPGs would be one with each player's utility function linear. This is our main motivation to analyze the knapsack game. Under this setting, each player p aims to solve

$$\max_{x^p \in \{0,1\}^n} \sum_{i=1}^{n} v_i^p x_i^p + \sum_{k=1, k\neq p}^{m} \sum_{i=1}^{n} c_{k,i}^p x_i^p x_i^k \tag{4.4.10a}$$

$$\text{s. t.} \quad \sum_{i=1}^{n} w_i^p x_i^p \leq W^p. \tag{4.4.10b}$$

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 (4.4.10b) is satisfied and the associated utilities are maximized (4.4.10a).

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 4.4.5, it suffices to study finitely supported equilibria.

Since utilities are linear, through the proof of Corollary 4.4.5, we deduce that the bound on the equilibria supports for each player is $n+1$. We can sightly improve this bound using the basic polyhedral theory revised in Section 2.2. 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 utilities 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 [12], any point of this facet can be written as a convex combination of n strategies of X^p.

Lemma 4.4.15. *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 $p = 1, \ldots, m$.*

Competitive Uncapacitated Lot-Sizing Game. The Competitive Uncapacitated Lot-Sizing Game (ULSG) was studied in Section 4.3. Each player p's utility function (4.3.1a) is quadratic due to the term $\sum_{t=1}^{t} -b_t(q_t^p)^2$. Next we show that it satisfies the Lipschitz condition in order to guarantee 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

$$\begin{aligned}
|\sum_{t=1}^{T} b_t(q_t^p)^2 - \sum_{t=1}^{T} b_t(\hat{q}_t^p)^2| &= |\sum_{t=1}^{T} b_t\left((q_t^p)^2 - (\hat{q}_t^p)^2\right)| \\
&= |\sum_{t=1}^{T} b_t\left((q_t^p) + (\hat{q}_t^p)\right)\left((q_t^p) - (\hat{q}_t^p)\right)| \\
&\leq \sqrt{\sum_{t=1}^{T} b_t^2\left((q_t^p) + (\hat{q}_t^p)\right)^2}\sqrt{\sum_{t=1}^{T}\left((q_t^p) - (\hat{q}_t^p)\right)^2} \\
&\leq \sqrt{\sum_{t=1}^{T} b_t^2\left(\frac{2a_t}{b_t}\right)^2} \cdot \parallel q^p - \hat{q}^p \parallel \\
&\leq \sqrt{\sum_{t=1}^{T} 4a_t^2} \cdot \parallel q^p - \hat{q}^p \parallel .
\end{aligned}$$

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.

Proposition 4.3.6 states that ULSG is a potential game and a maximum of its potential function is a (pure) equilibrium. This is an additional motivation to analyze our algorithms in this problem: it can be compared with the maximization of the associated potential.

4.4.3.2 Implementation Details

Both our implementations, the m-SGM 4.4.2.1 and SGM, use the following specialized functions.

Initialization(IPG)**.** Algorithm 4.4.2.1 stops once an equilibrium is computed. Therefore, the equilibrium computed when applied to a game with more than one NE will depend on its initialization as the following examples illustrates.

Example 4.4.16. *Consider an instance of the two-player ULSG* (4.3.1) *with the following parameters:* $T = 1$, $a_1 = 15$, $b_1 = 1$, $C_1^1 = C^2 = 0$, $F_1^1 = F_1^2 = 15$. *It is a single-period game, therefore the quantities produced are equal to the quantities placed in the market (this is,* $x_1^1 = q_1^1$ *and* $x_1^2 = q_1^2$*). Given the simplicity of the players' optimization programs* (4.3.1)*, we can analytically compute the players' best reactions that are depicted in Figure 4.4.4.*

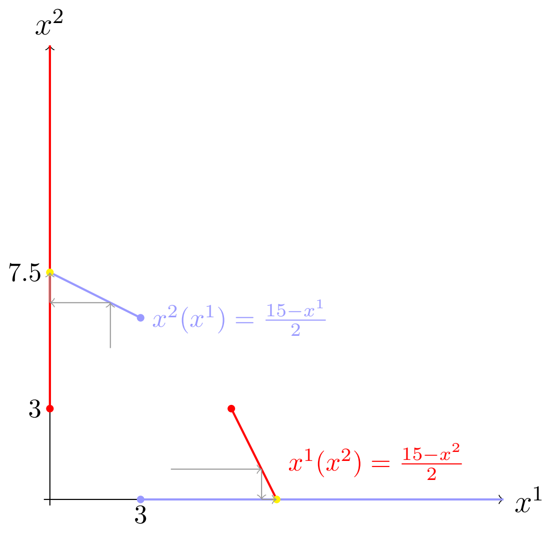


Figure 4.4.4: Players' best reaction functions.

The game possesses two (pure) equilibria: $(\hat{x}^1, \hat{y}^1, \hat{x}^2, \hat{y}^2) = (0, 0, 7.5, 1)$ *and* $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2) = (7.5, 1, 0, 0)$. *Thus, the equilibrium that m-SGM 4.4.2.1 determines depend on its initialization: Figure 4.4.4 depicts the convergence to* $(\hat{x}^1, \hat{y}^1, \hat{x}^2, \hat{y}^2)$ *when the initial sample game is* $\mathbb{S} = \{(2, 1)\} \times \{(5, 1)\}$ *and to* $(\tilde{x}^1, \tilde{y}^1, \tilde{x}^2, \tilde{y}^2)$ *when the initial sample game is* $\mathbb{S} = \{(4, 1)\} \times \{(1, 1)\}$.

In an attempt to keep as small as possible the size of the sample games (*i.e.*, number of strategies explicitly enumerated), the initialization implemented computes a single strategy for each player. We experimented initializing the algorithm with the social optimal strategies (strategies that maximize the total players' utilities), pure equilibrium for the potential part of the game [7] 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 this 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 uses the players' strategies if they were alone in the game, given that these optimizations must be simpler.

$PlayerOrder(\mathbb{S}_{dev_0}, \ldots, \mathbb{S}_{dev_k})$. The equilibrium returned by our algorithm depends on the players' order when we check their incentives to deviate: for the equilibrium σ_k of

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

the sample 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 increasing order of number of previous iterations without receiving a new strategy.

$DeviationReaction(p, \sigma_k^{-p}, \Pi^p(\sigma_k), \epsilon, IPG)$**.** When checking if a player p has incentive to deviate, it suffices to determine whether she has a strategy that strictly increases her utility when she unilaterally deviates to it. Nowadays, there are powerful software tools to tackle mixed integer quadratic programming problems[8]. Thus, our implementation solves the player p's best reaction problem (4.3.1) to σ_k^{-p}. We use Gurobi 5.6.3 to solve these reaction problems.

$SortSizes(\sigma_0, \dots, \sigma_{k-1})$**.** The authors of PNS [107] recommend that the support strategies' enumeration starts with support sizes ordered, first, by total size, 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 sample game to its successor or predecessor, the sample 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 sample games are expected to be close). Therefore, our criteria 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 plus 1 distance to the one of the previously computed equilibrium 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 plus 1 distance to the one of the previously computed equilibrium, third, sum of the players' support sizes and, fourth, balance.

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

[8]In the knapsack game, a player's best reaction problem is an integer linear programming problem. In the ULSG, a player's best reaction problem is a concave mixed integer quadratic programming problem (the proof that it is a concave MIQP is analogous to the one in Appendix A).

$SortStrategies(\mathbb{S}, \sigma_0, \ldots, \sigma_{k-1})$**.** Following the previous reasoning, the strategies of the current sample 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 order words, $x(k)$ and $\mathbb{S}_{dev_{k+1}}$ are not in the input of the PNS.

4.4.3.3 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 uncapacitated lot-sizing games in order to validate the importance 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 [51]. All algorithms have been coded in Python 2.7.2. Since for both the knapsack and competitive uncapacitated lot-sizing games the Feasibility Problems 2.3.3 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.

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 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 4.3 and 4.4 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 sample 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 sample 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 sample 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 as 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 sample games is usually equal for both algorithms, the advantage is in the support enumeration: m-SGM 4.4.2.1 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 the section, we discuss the reasons why backtracking is unlikely to occur.

In Table 4.3, we can observe that for instance 4 with $n = 100$, the m-SGM performed more iterations than SGM. The reason for 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 sample games, and thus, terminating with different outputs. This event is be 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 4.4.15) did not contribute to prune the search space of PNS support enumeration, since the algorithm terminates with sample games of smaller size.

Competitive Uncapacitated Lot-Sizing Game. 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.

A player p's best reaction (4.3.1) for a fixed $(y^{-p}, x^{-p}, q^{-p}, h^{-p})$ can be computed in polynomial time (Lemma 4.3.3), however, for easiness of implementation and fair comparison with the computation of the potential function optimum, we do not use the dynamic programming procedure, but Gurobi 5.6.3.

As previously proved, Proposition 4.3.6, the ULSG is potential, which implies the existence of a pure equilibrium. In particular, each sample game of the competitive lot-sizing game is potential and thus has a pure equilibrium. In fact, our algorithms will return a pure equilibrium: both m-SGM and SGM start with a sample 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

		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]		
		avg.					numb.		avg.					numb.	
		time	iter	$\lvert S_1 \rvert$	$\lvert S_2 \rvert$		pNE	mNE	time	iter	$\lvert S_1 \rvert$	$\lvert S_2 \rvert$		pNE	mNE
		0.21	8.10	4.90	4.20		2	8	0.27	8.10	4.90	4.20		2	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]		
		avg.					numb.		avg.					numb.	
		time	iter	$\lvert S_1 \rvert$	$\lvert S_2 \rvert$		pNE	mNE	time	iter	$\lvert S_1 \rvert$	$\lvert S_2 \rvert$		pNE	mNE
		0.37	10.40	5.80	5.60		0	10	0.45	10.40	5.80	5.60		0	10
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]		
		avg.					numb.		avg.					numb.	
		time	iter	$\lvert S_1 \rvert$	$\lvert S_2 \rvert$		pNE	mNE	time	iter	$\lvert S_1 \rvert$	$\lvert S_2 \rvert$		pNE	mNE
		52.31	18.70	10.30	9.40		0	10	98.62	18.70	10.30	9.40		0	10
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]		
		avg.					numb.		avg.					numb.	
		time	iter	$\lvert S_1 \rvert$	$\lvert S_2 \rvert$		pNE	mNE	time	iter	$\lvert S_1 \rvert$	$\lvert S_2 \rvert$		pNE	mNE
		739.09	21.67	11.67	11.00		0	3	1024.44	21.33	11.33	11.00		0	3

Table 4.3: 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]		
		avg.					numb.		avg.					numb.	
		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
		0.29	7.90	3.70	3.20	3.00	2	8	0.39	7.90	3.70	3.20	3.00	2	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]		
		avg.					numb.		avg.					numb.	
		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
		12.06	13.10	5.50	5.00	4.60	0	10	25.21	13.10	5.50	5.00	4.60	0	10
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]		
		avg.					numb.		avg.					numb.	
		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
		502.11	22.29	8.14	6.71	9.43	0	7	870.35	22.29	8.14	6.71	9.43	0	7

Table 4.4: Computational results for the **knapsack game** with $m = 3$.

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 the former deeply depend on the initialization (which in our implementation does not take into account the players' interaction).

Table 4.5 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 sample games ("avg. iter"), the number of strategies in the last sample 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 sample 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, allow larger exploration of the set of equilibria.

Algorithm *PlayerOrder* has a crucial impact in the number of sample 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.

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 in Example B.0.21). This is the reason why both m-SGM and SGM, in general, coincide in the number of sample games generated: it is in the support enumeration for each sample 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 sample 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 sample games had to be generated in order to find an equilibrium, meaning that the challenge is to make the computation of equilibria for sample 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 4.5: 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 4.6 and 4.7, 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 have been enumerated.

4.4.4 Summary

Literature in computational equilibria lacks the study of games with diverse sets of strategies with practical interest. This work presents the first attempt to address the computation of equilibria to integer programming games.

We started by classifying the game's complexity in terms of existence of pure and mixed equilibria. For both cases, it was proved that the problems are Σ_2^p-complete. However, if the players' set of strategies is bounded, the game is guaranteed to have an equilibrium. Even when there are equilibria, the computation of one is a PPAD-complete problem, which is likely to be a class of hard problems.

Under our game model, each player's goal is described through a mathematical programming model. Therefore, we mixed 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 of the original game; in each iteration, by solving the player's best reactions to an equilibrium of the previous sample 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 sample 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

			m-SGM							direct PNS					
n	m	INS	time	iter	pNE	mNE	$\prod_{p=1}^{m} \|\mathbb{S}^p\|$	numb. back		time	pNE	mNE	$\prod_{p=1}^{m} \|\mathbb{S}^p\|$		
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	$\|S_1\|$	$\|S_2\|$		pNE	mNE	time	$\|S_1\|$	$\|S_2\|$		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	$\prod_{p=1}^{m} \|\mathbb{S}^p\|$	numb. back		time	pNE	mNE	$\prod_{p=1}^{m} \|\mathbb{S}^p\|$		
	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	$\|S_1\|$	$\|S_2\|$	$\|S_3\|$	pNE	mNE	time	$\|S_1\|$	$\|S_2\|$	$\|S_3\|$	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	$\prod_{p=1}^{m} \|\mathbb{S}^p\|$	numb. back		time	pNE	mNE	$\prod_{p=1}^{m} \|\mathbb{S}^p\|$		
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	$\|S_1\|$	$\|S_2\|$		pNE	mNE	time	$\|S_1\|$	$\|S_2\|$		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	$\prod_{p=1}^{m} \|\mathbb{S}^p\|$	numb. back		time	pNE	mNE	$\prod_{p=1}^{m} \|\mathbb{S}^p\|$		
	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	$\|S_1\|$	$\|S_2\|$	$\|S_3\|$	pNE	mNE	time	$\|S_1\|$	$\|S_2\|$	$\|S_3\|$	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 4.6: 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	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$	numb. back		time	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$		
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	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$	numb. back		time	pNE	mNE	$\prod_{p=1}^{m} \lvert\mathbb{S}^p\rvert$		
	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 4.7: Computational results for the m-SGM and PNS to the knapsack game with $n = 10$.

SGM. Our algorithms were experimentally validated through two particular games: the knapsack and the competitive uncapacitated 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 utilities. For the competitive uncapacitated 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 uncountable (ULSG) 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 sample games) are usually "far" from describing (even partially) a player p's polytope conv(X^p) and thus, at this point, this information is not used in PNS to speed up its computations. For instance, Corollary 4.4.5 and Lemma 4.4.15 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.

Chapter 5

Conclusions and Open Questions

In this thesis, we discuss integer programming games, a class of games characterized by a novel representation of the players' sets of strategies. To this end, mathematical programming formulations are used to model each player's optimization problem, in a context where each player's decision affects the opponents utility (objective function). Therefore, IPG is a two direction generalization; on the one hand, finite, infinite or even exponential sets of strategies fit in the IPG framework, generalizing important games in the literature (such as finite and "well-behaved" continuous games); on the other hand, it generalizes mathematical programming problems with a single decision-maker. With regard to the game's dynamics, we focused on Stackelberg competitions (bilevel programming) and simultaneous games. In both cases, as motivated in Chapter 1, our goal was threefold: real-world applicability, the study of games' computational complexity and the development of algorithms to compute solutions.

Applications. The contributions of this thesis do not reduce to new mathematical results. Our achievements are enriched by the fact that the games modeled are a step forward in the direction of successfully approximate real-world problems.

The bilevel knapsack problems presented in Chapter 3 are games played sequentially by two players (the leader and the follower); these games share in common the follower's optimization problem, which is a knapsack. Given these problems' simplicity, they are likely to be sub-problems of mixed integer bilevel programming models of real-world applications. In particular, the bilevel knapsack with interdiction constraints is of high importance due to its min-max structure and to the interdiction constraints, which establish the connection with *robust optimization* (Ben-Tal *et al.* [9]) and security planning problems (Smith [118], Smith and Lim [119]). Robust optimization focuses on mathematical programming problems with uncertainty, where instead of assuming the underlying probability distribution for the uncertainty parameters, it considers the worst-case scenario; in specific, a min-max bilevel programming problem is formulated, where the upper level models the mathematical programming problem with uncertainty parameters, which are controlled by the lower level. In security planning problems, bilevel programming is used to model situations where the goal is to minimize the maximum damage that an attack can lead to in a network by enforcing the security in parts of it, which are interdicted to the attacker. Therefore, since the bilevel knapsack with interdiction constraints is a particular case of these two larger classes of problems, the developed work provides insights to approach more general problems in these settings.

Portfolio management is suitable to be modeled through a knapsack game: each player has a limited budget (knapsack constraint) and aims at maximizing the profit associated with her investments; these profits depend on other investors' decisions. Thus, the two-player coordination knapsack game (Section 4.1) and its generalization in Section 4.4 are the simplest discrete portfolio models that one can devise. Our study shows that these models are likely to have many equilibria, and thus, it is problematic to predict the most probable/rational outcome. The addition of more complex constraints to the knapsack game, allowing to model more complex portfolio management problems, would decrease the number of players' feasible strategies, which could potentially decrease the number of equilibria. Thus, formulating portfolio management situations under IPG settings deserves further research.

Multi-player kidney exchange programs are procedures recently proposed in order to potentially increase the number of kidney transplants to patients in need. In this thesis, for the first time in the literature, the kidney exchange program is investigated from a non-cooperative game theory point of view: the players are the entities owning a pool of incompatible (with respect to kidney transplantation) patient-donor pairs, and each player goal is to maximize the number of patients in her pool receiving a kidney. The competitive two-player kidney exchange game devised in Section 4.2 yields a market design with optimal social outcomes. The success in solving the proposed game results from the generalization of a polynomialy-solvable single decision-maker optimization problem (the maximum matching problem) to many decision makers (players). Moreover, our preliminary analysis of the game beyond two players and pairwise exchanges, indicates a promising line of research.

The competitive uncapacitated lot-sizing game of Section 4.3 is suitable to approximate the real challenge that firms face when planning their production. In this game, the players are firms producing a homogeneous good, where each firm's utility reflects her production costs and revenues, which depend on the opponent firms' strategies. We were able to prove the existence of an equilibrium (solution), by proving the existence of a potential function and of its maximum. If more complex constraints are included on it – *e.g.*, production capacities, initial and final inventory quantities, backlogging, lower bounds on production, to name a few – the game remains potential and, thus, has a pure equilibrium. The existence of a solution (equilibrium) to this game under these more general constraints keeps the interest of understanding it in deep.

Computational Complexity. As broadly accepted in the game theory community, an equilibrium is a solution to a game; that is what we propose to compute. Thereby, it matters to determine under which conditions equilibria exist and if equilibria can be computed efficiently.

The bilevel knapsack problems considered here have always an optimal solution (equilibrium), but we proved that it is Σ_2^p-complete to compute it. Thus, assuming $P \neq NP$, there is no efficient algorithm that can find a solution in polynomial time. For simultaneous IPGs, even deciding the existence of equilibria was proven to be Σ_2^p-complete. For the three particular simultaneous IPGs of Chapter 4, we showed the existence of (pure) equilibria, and thus, the challenge is in computing equilibria and, in case of multiple equilibria, to decide the players' preferred ones.

The two-player coordination knapsack game has several equilibria, which motivated us to focus on its Pareto efficient pure equilibria; their computation was proven to amount to solve NP-complete problems. For the competitive two-player kidney exchange game, we were able to characterize players' best reactions and to efficiently compute an equilibrium to which we argue that the players converge. Concerning the competitive uncapacitated lot-sizing game, for a special type of instances, algorithms to compute an equilibrium in polynomial time were presented, while the complexity of the general case remains open.

For general simultaneous integer programming games we were able to determine sufficient conditions for the existence of equilibria: the players' sets of strategies must be bounded. For these games, computing an equilibrium is PPAD-complete, which implies that it is unlikely to exist an algorithm able to do it in polynomial time. Furthermore, the PPAD class does not seem to provide the proper landscape for classifying the computational complexity of computing an equilibrium in simultaneous IPGs. In fact, PPAD class has its root in finite games that are an easier class of games, in comparison with general IPGs. Note that for IPGs, verifying if a profile of strategies is an equilibrium implies solving each player's best response optimization, which is an NP-complete problem, while for finite games this computation can be done efficiently. In this context, it would be interesting to explore the definition of a "second level PPAD" class, that is, a class of problems for which a solution could be verified in polynomial time if there was access to an NP oracle.

Algorithms. The main goal of this work was to develop algorithms to compute as efficiently as possible equilibria, and thus, to serve as a decision support tool.

For the bilevel knapsack problem with interdiction constraints, a novel algorithmic approach solves medium-sized instances in reasonable time. Its good performance (in practice) is due to two different types of cuts, which are determined during the algorithm execution, enabling to reduce the search space for the optimal solution. Hence, the algorithm adaptation to robust optimization and security planning models is of great interest. In Ralphs *et al.* presentation [110], it was conjectured that the NG_3 cuts (Section 3.3.2) are *Benders cuts*, *i.e.*, they lead to the exact function representing the optimal value for the follower's problem. Therefore, applying Benders' Principle [10] to a follower's problem, we would get a generalization of the NG_3-cuts to any bilevel programming problem, which is a key idea in our algorithm.

For the two-player coordination knapsack game (CKG), the competitive two-player kidney exchange game (2–KEG) and the competitive uncapacitated lot-sizing game (ULSG), we where able to propose algorithms to compute an equilibrium. For the CKG, we can determine the set of Pareto efficient equilibria by solving a two-objective integer programming problem. For 2–KEG, an algorithm capable of determining efficiently an equilibrium has been devised, and it is argued that it would lead to the rational outcome for the players. For the ULSG, it is proven that it is a potential game, and, consequently, a tâtonnement process allows to converge to an equilibrium. However, to determine whether convergence is obtained in polynomial time is an open problem. In the design of these algorithms, we exploited the problems' specific structure in order to devise methods as effective as possible. It may be observed that the algorithmic ideas associated with each of these three simultaneous IPGs are very distinct, which results from exploring each game specific structure.

The last part of the thesis, Section 4.4, concerns a method, the modified sample generation method, capable of approximating an equilibrium in finite time for more general simultaneous IPGs. Moreover, given a specific IPG, our algorithm can be enhanced with specialized methods, as detailed in Section 4.4.3.2. The open questions that follow from this work are: How to characterize the set of all equilibria? How to decide among equilibria? Is the equilibria definition still suitable as a solution concept of simultaneous IPGs? Simultaneous IPGs are quite hard to understand from the perspective of determining equilibria. Therefore, assuming that the games outcomes are equilibria might be an unreasonable assumption. This leads to two types of research lines. One is to explore game design, *i.e.*, study policies for the games in order to have a unique equilibrium. Another is to study other definitions of solution, *e.g.*, approximated equilibria or robust strategies (to assume that the rivals choose the strategy that minimizes a player utility).

IPG is a new framework encompassing well-known game models, as well as more general situations, making it an interesting tool for better formulation of real-world applications. This thesis deepens the mathematical knowledge of this class of games, unveiling new algorithmic approaches but, at the same time, putting in evidence the intrinsic complexity carried in IPGs. Therefore, undoubtedly, the additional study of IPGs is an appealing subject for further research.

References

[1] D. J. Abraham, A. Blum, and T. Sandholm, *Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges*, Proceedings of the 8th ACM conference on Electronic commerce (New York, NY, USA), EC '07, ACM, 2007, pp. 295–304. (Cited on page 104.)

[2] C. Àlvarez, J. Gabarró, and M. Serna, *Pure Nash equilibria in games with a large number of actions*, Mathematical Foundations of Computer Science 2005 (Joanna Jedrzejowicz and Andrzej Szepietowski, eds.), Lecture Notes in Computer Science, vol. 3618, Springer Berlin Heidelberg, 2005, pp. 95–106 (English). (Cited on page 50.)

[3] H. Anton and C. Rorres, *Elementary linear algebra: Applications version*, John Wiley & Sons, 2005. (Cited on page 201.)

[4] I. Ashlagi, F. Fischer, I. A. Kash, and A. D. Procaccia, *Mix and match: A strategyproof mechanism for multi-hospital kidney exchange*, Games and Economic Behavior **91** (2015), 284–296. (Cited on page 104.)

[5] I. Ashlagi and A. Roth, *Individual rationality and participation in large scale, multi-hospital kidney exchange*, Proceedings of the 12th ACM conference on Electronic commerce (New York, NY, USA), EC '11, ACM, 2011, pp. 321–322. (Cited on page 104.)

[6] ———, *Individual rationality and participation in large scale, multi-hospital kidney exchange*, Working paper, `http://web.mit.edu/iashlagi/www/papers/LargeScaleKidneyExchange_1_13.pdf`, 2011. (Cited on page 104.)

[7] E. Balas and R. Jeroslow, *Canonical cuts on the unit hypercube*, SIAM Journal on Applied Mathematics **23** (1972), 61–69. (Cited on page 78.)

[8] O. Ben-Ayed, *Bilevel linear programming*, Computers & Operations Research **20** (1993), no. 5, 485–501. (Cited on page 43.)

[9] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust optimization*, Princeton University Press, 2009. (Cited on page 181.)

[10] J. F. Benders, *Partitioning procedures for solving mixed-variables programming problems*, Numerische Mathematik **4** (1962), no. 1, 238–252 (English). (Cited on page 184.)

[11] C. Berge, *Two theorems in graph theory*, Proceedings of the National Academy of Sciences **43** (1957), no. 9, 842–844. (Cited on page 30.)

[12] D. P. Bertsekas, A. E. Ozdaglar, and A. Nedić, *Convex analysis and optimization*, Athena scientific optimization and computation series, Athena Scientific, Belmont (Mass.), 2003. (Cited on page 167.)

[13] J. A. Bondy and U. S. R. Murty, *Graph theory with applications*, Elsevier Science Publishing Co., Inc., 1976. (Cited on page 30.)

[14] E. Borel, *La théorie du jeu et les équations intégrales à noyau symétrique*, Compt. Rend. Acad. Sci. **173** (1921), 1304–1308, Translated by Leonard J. Savage in Econometrica, Vol. 21, No. 21, No. 1, 1953, pp. 97–100. (Cited on page 37.)

[15] ———, *Sur les jeus où interviennent le hasard et l'habileté des joueurs*, Théorie des probabilités (1924), 204–224, Translated by Leonard J. Savage in Econometrica, Vol. 21, No. 21, No. 1, 1953, pp. 101–105. (Cited on page 37.)

[16] ———, *Sur les systèmes de formes linéaires à déterminant symétrique gauche et la théorie générale du jeu*, Algèbre et calcul des probabilités **184** (1927), 52–53, Translated by Leonard J. Savage in Econometrica, Vol. 21, No. 21, No. 1, 1953, pp. 116–117. (Cited on page 37.)

[17] L. Brotcorne, S. Hanafi, and R. Mansi, *A dynamic programming algorithm for the bilevel knapsack problem*, Operations Research Letters **37** (2009), no. 3, 215–218. (Cited on pages 55 and 61.)

[18] ———, *One-level reformulation of the bilevel knapsack problem using dynamic programming*, Discrete Optimization **10** (2013), 1–10. (Cited on pages 55, 71, 72, 87, and 94.)

[19] I. Caragiannis, A. Filos-Ratsikas, and A. D. Procaccia, *An improved 2-agent kidney exchange mechanism*, Internet and Network Economics (Ning Chen, Edith Elkind, and Elias Koutsoupias, eds.), Lecture Notes in Computer Science, vol. 7090, Springer Berlin Heidelberg, 2011, pp. 37–48. (Cited on page 104.)

[20] Cbc, `https://projects.coin-or.org/Cbc`. (Cited on page 28.)

[21] K. Cechlárová, T. Fleiner, and D. Manlove, *The kidney exchange game.*, Proc. of the 8th International Symposium on Operational Research, SOR 5, 2005, pp. 77–83. (Cited on page 104.)

[22] L. Chen and G. Zhang, *Approximation algorithms for a bi-level knapsack problem*, Combinatorial Optimization and Applications (W. Wang, X. Zhu, and D.-Z. Du, eds.), Lecture Notes in Computer Science, vol. 6831, Springer Berlin Heidelberg, 2011, pp. 399–410. (Cited on page 72.)

[23] X. Chen and X. Deng, *Settling the complexity of two-player Nash equilibrium*, Foundations of Computer Science, 2006. FOCS '06. 47th Annual IEEE Symposium on, Oct 2006, pp. 261–272. (Cited on pages 47 and 155.)

[24] A. Cobham, *The intrinsic computational difficulty of functions*, Proc. 1964 International Congress for Logic Methodology and Philosophy of Science (1964), 24–30. (Cited on page 22.)

[25] B. Colson, P. Marcotte, and G. Savard, *Bilevel programming: A survey*, 4OR **3** (2005), no. 2, 87–107 (English). (Cited on page 44.)

[26] M. Constantino, X. Klimentova, A. Viana, and A. Rais, *New insights on integer-programming models for the kidney exchange problem*, European Journal of Operational Research **231** (2013), no. 1, 57–68. (Cited on pages 103 and 104.)

[27] S. A. Cook, *The complexity of theorem-proving procedures*, In Proc. 3rd Ann. ACM Symp. on Theory of Computing,, ACM, 1971, pp. 151–158. (Cited on page 22.)

[28] G. Cornuéjols, *Valid inequalities for mixed integer linear programs*, Mathematical Programming **112** (2008), no. 1, 3–44 (English). (Cited on page 28.)

[29] A. A. Cournot, *Researches into the mathematical principles of the theory of wealth*, Nathaniel T. Bacon, Trans. Macmillan, New York, 1927. (Cited on page 45.)

[30] 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. (Cited on page 50.)

[31] C. D'Ambrosio, A. Frangioni, L. Liberti, and A. Lodi, *On interval-subgradient and no-good cuts*, Operations Research Letters **38** (2010), 341–345. (Cited on page 78.)

[32] G. B. Dantzig, *Linear programming and extensions*, Rand Corporation Research Study, Princeton Univ. Press, Princeton, NJ, 1963. (Cited on page 27.)

[33] G.B. Dantzig, *Discrete-variable extremum problems*, Operations Research **5** (1957), 266–277 (English). (Cited on page 31.)

[34] ———, *Linear programming and extensions*, Princeton landmarks in mathematics and physics, Princeton University Press, 1963. (Cited on pages 25 and 37.)

[35] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, *The complexity of computing a Nash equilibrium*, Proceedings of the thirty-eighth annual ACM symposium on Theory of computing (New York, NY, USA), STOC '06, ACM, 2006, pp. 71–78. (Cited on page 47.)

[36] M. de Klerk, K. M. Keizer, F. H. Claas, B. J. J. M. Haase-Kromwijk, and W. Weimar, *The Dutch national living donor kidney exchange program*, American Journal of Transplantation **5** (2005), 2302–2305. (Cited on page 103.)

[37] C. Delort and O. Spanjaard, *Using bound sets in multiobjective optimization: Application to the biobjective binary knapsack problem*, Experimental Algorithms (Paola Festa, ed.), Lecture Notes in Computer Science, vol. 6049, Springer Berlin Heidelberg, 2010, pp. 253–265 (English). (Cited on page 101.)

[38] S. Dempe, *Foundations of bilevel programming*, Kluwer Academic Publishers, Dordrecht, The Nerthelands, 2002. (Cited on page 43.)

[39] ———, *Annotated bibliography on bilevel programming and mathematical programs with equilibrium constraints*, Optimization **52** (2003), no. 3, 333–359. (Cited on page 43.)

[40] S. Dempe and K. Richter, *Bilevel programming with knapsack constraints*, CEJOR Centr. Eur. J. Oper. Res. (2000), no. 8, 93–107. (Cited on pages 41, 53, 54, 55, and 61.)

[41] S. DeNegre, *Interdiction and discrete bilevel linear programming*, Ph.D. thesis, Lehigh University, 2011. (Cited on pages 43, 53, 56, 70, 71, 72, 89, 95, and 215.)

[42] S. DeNegre and T. K. Ralphs, *A branch-and-cut algorithm for integer bilevel linear programs*, Operations Research and Cyber-Infrastructure (J.W. Chinneck, B. Kristjansson, and M.J. Saltzman, eds.), Operations Research/Computer Science Interfaces, vol. 47, Springer US, 2009, pp. 65–78. (Cited on pages 89, 95, and 215.)

[43] X. Deng, *Complexity issues in bilevel linear programming*, Multilevel Optimization: Algorithms and Applications (Athanasios Migdalas, Panos M. Pardalos, and Peter V ärbrand, eds.), Nonconvex Optimization and Its Applications, vol. 20, Springer US, 1998, pp. 149–164 (English). (Cited on page 43.)

[44] J. P. Dickerson, A. D. Procaccia, and T. Sandholm, *Price of fairness in kidney exchange*, Proceedings of the 2014 International Conference on Autonomous Agents

and Multi-agent Systems (Richland, SC), AAMAS '14, International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 1013–1020. (Cited on page 129.)

[45] A. G. Doig and B. H. Land, *An automatic method for solving discrete programming problems*, Econometrica (1960), 497–520. (Cited on page 28.)

[46] T. Dudás, B. Klinz, and G. Woeginger, *The computational complexity of multi-level bottleneck programming problems*, Multilevel Optimization: Algorithms and Applications (Athanasios Migdalas, PanosM. Pardalos, and Peter V ärbrand, eds.), Nonconvex Optimization and Its Applications, vol. 20, Springer US, 1998, pp. 165–179 (English). (Cited on page 43.)

[47] J. Edmonds, *Paths, trees, and flowers*, Canadian Journal of Mathematics **17** (1965), 449–467. (Cited on pages 22 and 30.)

[48] C. E. J. Eggermont and G. J. Woeginger, *Motion planning with pulley, rope, and baskets*, Theory of Computing Systems **53** (2013), no. 4, 569–582 (English). (Cited on pages 23, 57, and 60.)

[49] P. Erdös and P. Turán, *On a problem of Sidon in additive number theory, and on some related problems.*, Journal of the London Mathematical Society (1941), no. 16, 212–215. (Cited on page 62.)

[50] A. Federgruen and J. Meissner, *Competition under time-varying demands and dynamic lot sizing costs*, Naval Research Logistics (NRL) **56** (2009), no. 1, 57–73. (Cited on pages 133 and 134.)

[51] Python Software Foundation, *Python v2.7.3 documentation*, `http://docs.python.org/library/random.html`, 2012. (Cited on pages 89 and 171.)

[52] D.H. Fremlin, *Measure theory*, Measure Theory, no. vol. 4, Torres Fremlin, 2000. (Cited on page 36.)

[53] D. Fudenberg and J. Tirole, *Game theory*, MIT Press, Cambridge, MA, 1991, Translated into Chinesse by Renin University Press, Bejing: China. (Cited on page 34.)

[54] S. A. Gabriel, S. A. Siddiqui, A. J. Conejo, and C. Ruiz, *Solving discretely-constrained Nash-cournot games with an application to power markets*, Networks and Spatial Economics **13** (2013), no. 3, 307–326 (English). (Cited on page 50.)

[55] D. Gale, H. W. Kuhn, and A. W. Tucker, *Linear programming and the theory of games*, Activity Analysis of Production and Allocation **1** (1951), no. 3, 371–329. (Cited on page 26.)

[56] M.l R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman & Co., New York, NY, USA, 1979. (Cited on pages 21, 23, 31, 62, and 143.)

[57] P. C. Gilmore and R. E. Gomory, *A linear programming approach to the cutting-stock problem*, Operations Research **9** (1961), no. 6, 849–859. (Cited on page 157.)

[58] I. L. Glicksberg, *A Further Generalization of the Kakutani Fixed Point Theorem, with Application to Nash Equilibrium Points*, Proceedings of the American Mathematical Society **3** (1952), no. 1, 170–174. (Cited on page 49.)

[59] A. V. Goldberg and R. E. Tarjan, *Finding minimum-cost circulations by canceling negative cycles*, Journal of the ACM **36** (1989), no. 4, 873–886. (Cited on page 146.)

[60] R. E. Gomory, *Outline of an algorithm for integer solutions to linear programs*, Bull. Amer. Math. Soc. **64** (1958), no. 5, 275–278. (Cited on pages 28 and 157.)

[61] S. Govindan and R. Wilson, *A global Newton method to compute Nash equilibria*, Journal of Economic Theory **110** (2003), no. 1, 65–86. (Cited on page 52.)

[62] ______, *Computing Nash equilibria by iterated polymatrix approximation*, Journal of Economic Dynamics and Control **28** (2004), no. 7, 1229–1241. (Cited on page 52.)

[63] Inc. Gurobi Optimization, *Gurobi optimizer reference manual*, `http://www.gurobi.com`, 2015. (Cited on page 28.)

[64] C. Hajaj, J. P. Dickerson, A. Hassidim, T. Sandholm, and D. Sarne, *Strategy-proof and efficient kidney exchange using a credit mechanism*, Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA., 2015, pp. 921–928. (Cited on page 132.)

[65] M. Hemmati, J. C. Smith, and M. T. Thai, *A cutting-plane algorithm for solving a weighted influence interdiction problem*, Computational Optimization and Applications **57** (2014), no. 1, 71–104. (Cited on pages 43 and 72.)

[66] W. Heuvel, P. Borm, and H. Hamers, *Economic lot-sizing games*, European Journal of Operational Research **176** (2007), no. 2, 1117–1130. (Cited on page 133.)

[67] Ilog-Cplex, *IBM ILOG CPLEX Optimizer*, `www.ilog.com/products/cplex`, 2015. (Cited on page 28.)

[68] E. Israel, *System interdiction and defense*, Ph.D. thesis, Naval Postgraduate School, 1999. (Cited on page 40.)

[69] R. G. Jeroslow, *The polynomial hierarchy and a simple model for competitive analysis*, Mathematical Programming **32** (1985), no. 2, 146–164 (English). (Cited on page 42.)

[70] B. Johannes, *New classes of complete problems for the second level of the polynomial hierarchy*, Ph.D. thesis, Mathematik und Naturwissenschaften der Technischen Universität Berlin, 2011. (Cited on page 23.)

[71] D. S. Johnson, *A brief history of NP-completeness, 1954-2012*, Optimization Stories, Special Volume of Documenta Mathematica (2012), 359–376. (Cited on page 21.)

[72] M. Jünger, T. Liebling, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi, and L.A. Wolsey (eds.), *50 years of integer programming 1958-2008: From the early years to the state-of-the-art*, Springer, Heidelberg, 2010. (Cited on page 28.)

[73] W. Karush, *Minima of Functions of Several Variables with Inequalities as Side Constraints*, Master's thesis, Dept. of Mathematics, Univ. of Chicago, 1939. (Cited on page 41.)

[74] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack problems*, Springer, 2004. (Cited on page 30.)

[75] L. G. Khachiyan, *Polynomial algorithms in linear programming*, Doklady Akademiia Nauk SSSR **244** (1979), 1093 – 1096, (English translation: Soviet Mathematics Doklady, 20(1):191 - 194, 1979). (Cited on page 27.)

[76] K. I. Ko and C. L. Lin, *On the complexity of min-max optimization problems and their approximation*, Minimax and Applications (Ding-Zhu Du and PanosM. Pardalos, eds.), Nonconvex Optimization and Its Applications, vol. 4, Springer US, 1995, pp. 219–239 (English). (Cited on pages 57 and 64.)

[77] M. Köppe, C. T. Ryan, and M. Queyranne, *Rational generating functions and integer programming games*, Oper. Res. **59** (2011), no. 6, 1445–1460. (Cited on pages 38 and 51.)

[78] M. M. Kostreva, *Combinatorial optimization in Nash games*, Computers & Mathematics with Applications **25** (1993), no. 10 - 11, 27– 34. (Cited on page 50.)

[79] H. W. Kuhn and A. W. Tucker, *Nonlinear programming*, Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability (Berkeley, Calif.), University of California Press, 1951, pp. 481–492. (Cited on page 41.)

[80] E. L. Lawler, *Fast approximation algorithms for knapsack problems*, Mathematics of Operations Research **4** (1979), no. 4, 339–356. (Cited on pages 64 and 68.)

[81] K. H. 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** (2003), no. 4, 1573–1580. (Cited on page 51.)

[82] C. E. Lemke and J. T. Howson, *Equilibrium points of bimatrix games*, Journal of the Society for Industrial and Applied Mathematics **12** (1964), no. 2, pp. 413–423. (Cited on pages 52 and 161.)

[83] H. Li and J. Meissner, *Competition under capacitated dynamic lot-sizing with capacity acquisition*, International Journal of Production Economics **131** (2011), no. 2, 535–544. (Cited on pages 133 and 134.)

[84] D. F. Manlove and G. O'Malley, *Paired and altruistic kidney donation in the UK: algorithms and experimentation*, Experimental Algorithms (Ralf Klasing, ed.), Lecture Notes in Computer Science, vol. 7276, Springer Berlin Heidelberg, 2012, pp. 271–282. (Cited on page 103.)

[85] R. Mansi, C. Alves, J. M. Valério de Carvalho, and S. Hanafi, *An exact algorithm for bilevel 0-1 knapsack problems*, Mathematical Problems in Engineering **2012** (2012), 23, Article ID 504713. (Cited on pages 53 and 55.)

[86] S. Martello, D. Pisinger, and P. Toth, *Dynamic programming and strong bounds for the 0-1 knapsack problem*, Management Science **45** (1999), 414–424. (Cited on pages 71 and 89.)

[87] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*, John Wiley & Sons, Inc., New York, NY, USA, 1990. (Cited on pages 30 and 60.)

[88] Eric Maskin and Jean Tirole, *A theory of dynamic oligopoly, i: Overview and quantity competition with large fixed costs*, Econometrica **56** (1988), no. 3, 549–569. (Cited on page 133.)

[89] G. P. McCormick, *Computability of global solutions to factorable nonconvex programs: Part I - convex underestimating problems*, Mathematical Programming **10** (1976), 147–175. (Cited on page 75.)

[90] R. D. Mckelvey, A. M. Mclennan, and T. L. Turocy, *Gambit: Software Tools for Game Theory*, Tech. report, Version 15.0.0, 2014. (Cited on page 52.)

[91] A. R. Meyer and L. J. Stockmeyer, *The equivalence problem for regular expressions with squaring requires exponential space*, Switching and Automata Theory, 1972., IEEE Conference Record of 13th Annual Symposium on, Oct 1972, pp. 125–129. (Cited on page 23.)

[92] D. Monderer and L. S. Shapley, *Potential games*, Games and Economic Behavior **14** (1996), no. 1, 124–143. (Cited on pages 45 and 145.)

[93] J. T. Moore and J. F. Bard, *The mixed integer linear bilevel programming problem*, Operations Research **38** (1990), 911–921 (English). (Cited on pages 42 and 43.)

[94] J. Nash, *Non-cooperative games*, Annals of Mathematics **54** (1951), no. 2, 286–295. (Cited on page 47.)

[95] G. L. Nemhauser and L. A. Wolsey, *Integer and combinatorial optimization*, Wiley-Interscience, New York, NY, USA, 1988. (Cited on page 25.)

[96] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*, Cambridge University Press, New York, NY, USA, 2007. (Cited on pages 47 and 111.)

[97] E. Nudelman, J. Wortman, Y. Shoham, and K. Leyton-Brown, *Run the GAMUT: a comprehensive approach to evaluating game-theoretic algorithms*, Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on, July 2004, pp. 880–887. (Cited on page 52.)

[98] K. O'Bryant, *A complete annotated bibliography of work related to Sidon sequences.*, The Electronic Journal of Combinatorics [electronic only] **DS11** (2004), 39 p., electronic only–39 p., electronic only (eng). (Cited on page 62.)

[99] G. Owen, *Game theory*, Emerald Group Publishing Limited; 3rd edition, 1995. (Cited on page 34.)

[100] M. Padberg and G. Rinaldi, *Optimization of a 532-city symmetric traveling salesman problem by branch and cut*, Oper. Res. Lett. **6** (1987), no. 1, 1–7. (Cited on page 28.)

[101] C. H. Papadimitriou, *Computational complexity*, Addison-Wesley, 1994. (Cited on page 21.)

[102] ———, *On the complexity of the parity argument and other inefficient proofs of existence*, Journal of Computer and System Sciences **48** (1994), no. 3, 498–532. (Cited on page 24.)

[103] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982. (Cited on pages 109, 111, and 125.)

[104] J. P. Pedroso and Y. Smeers, *Equilibria on a game with discrete variables*, preprint, `http://arxiv.org/abs/1407.8394`, 2014. (Cited on pages 133 and 134.)

[105] A. V. Plyasunov, *A two-level linear programming problem with a multi-variant knapsack at the lower level.*, Diskretn. Anal. Issled. Oper. Ser. [In Russian.] (2003), no. 10, 44–52. (Cited on page 55.)

[106] Y. Pochet and L. 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. (Cited on pages 33 and 149.)

[107] R. Porter, E. Nudelman, and Y. Shoham, *Simple search methods for finding a Nash equilibrium*, Games and Economic Behavior **63** (2008), no. 2, 642–662, Second World Congress of the Game Theory Society. (Cited on pages 48, 52, 156, 160, and 170.)

[108] K. Pruhs and G. J. Woeginger, *Approximation schemes for a class of subset selection problems*, Theoretical Computer Science **382** (2007), no. 2, 151–156, Latin American Theoretical Informatics. (Cited on pages 68 and 69.)

[109] T. Ralphs, *MibS: Mixed Integer Bilevel Solver*, `https://github.com/tkralphs/MibS`. (Cited on page 52.)

[110] T. Ralphs, S. Tahernajad, S. DeNegre, M. Güzelsoy, and A. Hassanzadeh, *Bilevel integer optimization: Theory and algorithms*, International Symposium on Mathematical Programming, 2015 `http://coral.ie.lehigh.edu/~ted/files/talks/BILEVEL-ISMP15.pdf`. (Cited on page 184.)

[111] R. W. Rosenthal, *A class of games possessing pure-strategy Nash equilibria*, International Journal of Game Theory **2** (1973), no. 1, 65–67 (English). (Cited on page 145.)

[112] A. Rubinstein, *Modeling bounded rationality*, MIT Press, Cambridge, Massachusetts, 1998. (Cited on page 139.)

[113] G. K. D. Saharidis, A. J. Conejo, and G. Kozanidis, *Exact solution methodologies for linear and (mixed) integer bilevel programming*, Metaheuristics for Bi-level Optimization (E.-G. Talbi and L. Brotcorne, eds.), Studies in Computational Intelligence, vol. 482, Springer Berlin Heidelberg, 2013, pp. 221–245. (Cited on page 44.)

[114] V. Scalzo, *Pareto efficient Nash equilibria in discontinuous games*, Economics Letters **107** (2010), no. 3, 364–365. (Cited on page 35.)

[115] G. R. Schoenebeck and S. Vadhan, *The computational complexity of Nash equilibria in concisely represented games*, ACM Trans. Comput. Theory **4** (2012), no. 2, 4:1–4:50. (Cited on page 50.)

[116] SCIP, *Solving Constraint Integer Programs*, `http://scip.zib.de/`. (Cited on page 28.)

[117] H. Simon, *Models of bounded rationality*, vol. 2, MIT Press, Cambridge, Massachusetts, 1982. (Cited on page 139.)

[118] J. C. Smith, *Basic interdiction models*, Wiley Encyclopedia of Operations Research and Management Science (In J. Cochran, ed.), Wiley, Hoboken, 2011, pp. 323–330. (Cited on pages 43 and 181.)

[119] J. Cole Smith and C. Lim, *Algorithms for network interdiction and fortification games*, Pareto Optimality, Game Theory And Equilibria (A. Chinchuluun, P. M. Pardalos, A. Migdalas, and L. Pitsoulis, eds.), Springer Optimization and Its Applications, vol. 17, Springer New York, 2008, pp. 609–644 (English). (Cited on pages 43 and 181.)

[120] N. D. Stein, A. Ozdaglar, and P. A. Parrilo, *Separable and low-rank continuous games*, International Journal of Game Theory **37** (2008), no. 4, 475–504 (English). (Cited on pages 49, 51, 155, and 160.)

[121] L. J. Stockmeyer, *The polynomial-time hierarchy*, Theoretical Computer Science **3** (1976), no. 1, 1–22. (Cited on page 21.)

[122] SYMPHONY, `https://projects.coin-or.org/SYMPHONY/`. (Cited on page 101.)

[123] T. Ui, *A shapley value representation of potential games*, Games and Economic Behavior **31** (2000), no. 1, 121–135. (Cited on pages 46 and 140.)

[124] C. Umans, *Hardness of approximating Σ_2^p minimization problems*, Foundations of Computer Science, 1999. 40th Annual Symposium on, 1999, pp. 465–474. (Cited on pages 57 and 64.)

[125] ______, *Optimization problems in the polynomial-time hierarchy*, Theory and Applications of Models of Computation (Jin-Yi Cai, S.Barry Cooper, and Angsheng Li, eds.), Lecture Notes in Computer Science, vol. 3959, Springer Berlin Heidelberg, 2006, pp. 345–355 (English). (Cited on page 57.)

[126] G. v. d. Laan, A. J. J. Talman, and L. v. d. Heyden, *Simplicial variable dimension algorithms for solving the nonlinear complementarity problem on a product of unit simplices using a general labelling*, Math. Oper. Res. **12** (1987), no. 3, 377–397. (Cited on page 52.)

[127] J. v. Neumann, *Zur theorie der gesellschaftsspiele*, Mathematische Annalen **100** (1928), no. 1, 295–320 (German), Translated by Sonya Bargmann in A. W. Trucker and R. D. Luce (eds.). Contributions to the Theory of Games. Vol. IV, Annals of Mathematics Study No, 40. Princeton University Press, 1959. pp. 13–42. (Cited on page 37.)

[128] ______, *Überein ökonomisches gleichungssystem undeine verallgemeinerung des brouwerschen fixpunktsatzes*, Ergebnisse eines mathematischen Kolloquiums (1937), no. 8, 73–83, Translated in Rev. Econ. Studies. Vol. 13, No. 1, 1945–46. (Cited on page 37.)

[129] ______, *On a maximization problem*, Institute for Advanced Study, Princeton, 1947 (Manuscript). (Cited on page 25.)

[130] J. v. Neumann and O. Morgenstern, *Theory of games and economic behavior*, Princeton Univ. Press, Princeton, NJ, 1944. (Cited on pages 36 and 37.)

[131] H. v. Stackelberg, *Marktform und gleichgewicht*, J. Springer, 1934. (Cited on page 39.)

[132] L. N. Vicente and P. H. Calamai, *Bilevel and multilevel programming: A bibliography review*, Journal of Global Optimization **5** (1994), no. 3, 291–306 (English). (Cited on page 43.)

[133] B. von Stengel (ed.), *Economic theory: Special issue of on computation of Nash equilibria in finite games*, vol. 42, Springer Berlin/Heidelberg, January 2010. (Cited on page 48.)

[134] Zhenbo Wang, Wenxun Xing, and Shu-Cherng Fang, *Two-group knapsack game*, Theoretical Computer Science **411** (2010), no. 7-9, 1094–1103. (Cited on page 98.)

[135] Xpress, `http://www.fico.com/en/products/fico-xpress-optimization-suite`. (Cited on page 28.)

[136] W. I. Zangwill and C .B. Garcia, *Pathways to solutions, fixed points, and equilibria*, Prentice-Hall series in computational mathematics, Prentice-Hall, 1981. (Cited on page 50.)

Appendix A

Potential Function Concavity

The canonical form in MIQP for the potential function 4.3.6 is:

$$\sum_{t=1}^{T}\sum_{p=1}^{m}[-F_t^p y_t^p - C_t^p x_t^p + a_t q_t^p] - \frac{1}{2}q^\intercal Q q$$

where:

$$Q = \begin{pmatrix}
2b_1 & b_1 & b_1 & \dots & b_1 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\
b_1 & 2b_1 & b_1 & \dots & b_1 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \dots & \vdots \\
b_1 & b_1 & b_1 & \dots & 2b_1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\
0 & 0 & 0 & \dots & 0 & 2b_2 & b_2 & b_2 & \dots & b_2 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\
0 & 0 & 0 & \dots & 0 & b_2 & 2b_2 & b_2 & \dots & b_2 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \dots & \vdots \\
0 & 0 & 0 & \dots & 0 & b_2 & b_2 & b_2 & \dots & 2b_2 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \dots & \vdots \\
0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 2b_T & b_T & b_T & \dots & b_T \\
0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & b_T & 2b_T & b_T & \dots & b_T \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \dots & \vdots \\
0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & b_T & b_T & b_T & \dots & 2b_T
\end{pmatrix}$$

and:

$$q = \begin{pmatrix} q_1^1 & q_1^2 & \dots & q_1^m & q_2^1 & q_2^2 & \dots & q_2^m & \dots & q_T^1 & q_T^2 & \dots & q_T^m \end{pmatrix}.$$

If the matrix Q is positive semi-definite, then the problem of maximizing the potential function 4.3.6 continuous relaxation over X becomes concave; as mentioned in Section 2.2, there are polynomial time algorithms to solve concave quadratic programming optimizations. If the eigenvalues of Q are all positive, then Q is positive definite (in particular, semi-definite). Matrix Q is a *block matrix*, thus the eigenvalues of Q are the eigenvalues of each of its blocks. See Anton and Rorres [3] for details in linear algebra. The eigenvalues for each of the diagonal blocks of Q are given in the following lemma.

Lemma A.0.17. *A matrix with the form:*

$$B = \begin{pmatrix} 2b & b & b & \dots & b \\ b & 2b & b & \dots & b \\ \vdots & \cdots & \ddots & \dots & \vdots \\ b & b & b & \dots & 2b \end{pmatrix}$$

has exactly two distinct eigenvalues: $(m+1)b$ and b.

Proof. Suppose that $(x_1, x_2, \dots, x_m)$ is an eigenvector for B corresponding to an eigenvalue λ. Then by definition:

$$\begin{pmatrix} 2b & b & b & \dots & b \\ b & 2b & b & \dots & b \\ \vdots & \cdots & \ddots & \dots & \vdots \\ b & b & b & \dots & 2b \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} = \lambda \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} \Leftrightarrow \begin{pmatrix} bx_1 + bx_2 + \dots bx_m \\ bx_1 + bx_2 + \dots bx_m \\ \vdots \\ bx_1 + bx_2 + \dots bx_m \end{pmatrix} = \begin{pmatrix} x_1(\lambda - b) \\ x_2(\lambda - b) \\ \vdots \\ x_m(\lambda - b) \end{pmatrix}.$$

One solution for the system above is the eigenspace associated with the eigenvalue b:

$$\mathcal{E}_b = \{(x_1, x_2, \dots, x_m) : x_1 + x_2 + \dots + x_m = 0\},$$

which has dimension $m-1$ (number of linear independent vectors). Another solution is the eigenspace associated with the eigenvalue $(m+1)b$:

$$\mathcal{E}_{(m+1)b} = \{(x_1, x_2, \dots, x_m) : x_1 = x_2 = \dots = x_m\},$$

which has dimension 1.

Note that $\mathcal{E}_b \cap \mathcal{E}_{(m+1)b} = \{(0, 0, \dots, 0)\}$, and thus the dimension of $\mathcal{E}_b \cup \mathcal{E}_{(m+1)b}$ is m, which cannot exceed the dimension of B. Therefore, all distinct eigenvalues were found and are $(m+1)b$ and b. $\square$

Corollary A.0.18. *For an ULSG with m players, the eigenvalues associated with Q are:*

$$\{(m+1)b_1, (m+1)b_2, \dots, (m+1)b_T, b_1, b_2, \dots, b_T\}.$$

Corollary A.0.19. *For an ULSG with m players, the associated Q is symmetric positive definite.*

Proof. All eigenvalues of Q are positive, since $b_t > 0$ for $t = 1, \dots, T$. $\square$

Corollary A.0.20. *Maximizing function 4.3.6 over the set of feasible strategies X is a concave MIQP.*

Appendix B

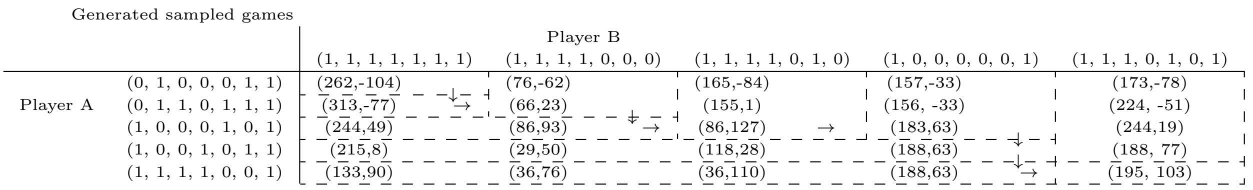
Applying modified SGM

Example B.0.21. *Consider the two-player game described by the following best reactions.*

$$
\begin{aligned}
\mathit{PlayerA}: \quad & \max_{x^A \in \{0,1\}^n} - 14x_1^A + 15x_2^A + 12x_3^A - 35x_4^A - 13x_5^A + 27x_6^A + 18x_7^A + \\
& \quad 95x_1^A x_1^B + 16x_2^A x_2^B - 9x_3^A x_3^B - 62x_4^A x_4^B + 61x_5^A x_5^B + 89x_6^A x_6^B + 97x_7^A x_7^B \\
& s.\ t. \quad 87x_1^A + 25x_2^A + 11x_3^A - 60x_4^A - 22x_5^A + 46x_6^A - 45x_7^A \leq 30.
\end{aligned}
$$

$$
\begin{aligned}
\mathit{PlayerB}: \quad & \max_{x^B \in \{0,1\}^n} 5x_1^B - 45x_2^B + 41x_3^B - 4x_4^B + 18x_5^B + 34x_6^B + 39x_7^B + \\
& \quad 96x_1^A x_1^B - 59x_2^A x_2^B + 85x_3^A x_3^B - 43x_4^A x_4^B - 58x_5^A x_5^B - 56x_6^A x_6^B - 77x_7^A x_7^B \\
& s.\ t. \quad - 28x_1^B - 71x_2^B + 39x_3^B - 32x_4^B + 32x_5^B + 10x_6^B - 47x_7^B \leq -70.
\end{aligned}
$$

Since all decision variables are binary, in the input of both SGM and modified SGM ϵ is zero. Figure B.0.1 summarizes the sampled games resultant from the application of SGM.


Generated sampled games

		Player B				
		(1, 1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 0, 0, 0)	(1, 1, 1, 1, 0, 1, 0)	(1, 0, 0, 0, 0, 0, 1)	(1, 1, 1, 0, 1, 0, 1)
	(0, 1, 0, 0, 0, 1, 1)	(262,-104) ↓	(76,-62)	(165,-84)	(157,-33)	(173,-78)
Player A	(0, 1, 1, 0, 1, 1, 1)	(313,-77) →	(66,23) ↓	(155,1)	(156, -33)	(224, -51)
	(1, 0, 0, 0, 1, 0, 1)	(244,49)	(86,93) →	(86,127) →	(183,63) ↓	(244,19)
	(1, 0, 0, 1, 0, 1, 1)	(215,8)	(29,50)	(118,28)	(188,63) ↓	(188, 77)
	(1, 1, 1, 1, 0, 0, 1)	(133,90)	(36,76)	(36,110)	(188,63) →	(195, 103)

Figure B.0.1: SGM applied to Example B.0.21.

The modified SGM 4.4.2.1 generates one strategy less than SGM, as we will see next. In what follows, each iteration of the modified SGM 4.4.2.1 is described, which is complemented with Figures B.0.2 and B.0.3.

Sampled game 0. *The NE is $\sigma_0 = (1; 1)$. Player A has incentive to deviate to $x(1) = (0, 1, 1, 0, 1, 1, 1)$.*

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

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

Sampled game 3. *The NE is* $\sigma = (0, 0, 1; 0, 1)$. *Player B has incentive to deviate to* $x(4) = (1, 1, 1, 1, 0, 1, 0)$.

Sampled game 4. *The NE is mixed with* $supp(\sigma_4^A) = \{(0, 1, 0, 0, 0, 1, 1), (1, 0, 0, 0, 1, 0, 1)\}$, $supp(\sigma_4^B) = \{(1, 1, 1, 1, 0, 0, 0), (1, 1, 1, 1, 0, 1, 0)\}$, *and* $\sigma_4 = (\frac{17}{28}, 0, \frac{11}{28}; 0, \frac{79}{89}, \frac{10}{89})$. *Player B has incentive to deviate to* $x(5) = (1, 0, 0, 0, 0, 0, 1)$.

Sampled game 5. *The NE is mixed with* $supp(\sigma_5^A) = \{(0, 1, 0, 0, 0, 1, 1), (1, 0, 0, 0, 1, 0, 1)\}$, $supp(\sigma_5^B) = \{(1, 1, 1, 1, 0, 1, 0), (1, 0, 0, 0, 0, 0, 1)\}$, *and* $\sigma_5 = (\frac{64}{115}, 0, \frac{51}{115}; 0, 0, \frac{26}{105}, \frac{79}{105})$. *Player A has incentive to deviate to* $x(6) = (1, 0, 0, 1, 0, 1, 1)$.

Sampled game 6. *The NE is mixed with* $supp(\sigma_6^A) = \{(1, 0, 0, 0, 1, 0, 1), (1, 0, 0, 1, 0, 1, 1)\}$, $supp(\sigma_6^B) = \{(1, 1, 1, 1, 0, 0, 0), (1, 0, 0, 0, 0, 0, 1)\}$, *and* $\sigma_6 = (0, 0, \frac{13}{43}, \frac{30}{43}; 0, \frac{5}{62}, 0, \frac{57}{62})$. *Player A has incentive to deviate to* $x(7) = (1, 1, 1, 1, 0, 0, 1)$.

Sampled game 7. *The NE is mixed with* $supp(\sigma_7^A) = \{(1, 0, 0, 1, 0, 1, 1), (1, 1, 1, 1, 0, 0, 1)\}$, $supp(\sigma_7^B) = \{(1, 0, 0, 0, 0, 0, 1)\}$, *and* $\sigma = (0, 0, 0, \frac{47}{82}, \frac{35}{82}; 0, 0, 0, 1)$. *Player B has incentive to deviate to* $x(8) = (1, 1, 1, 0, 1, 0, 1)$.

Sampled game 8. *There is no NE with* $x(8) = (1, 1, 1, 0, 1, 0, 1)$ *in the support of player B. Thus, initialize backtracking.*

Revisiting sampled game 7. *There is no NE with* $x(7) = (1, 1, 1, 1, 0, 0, 1)$ *in the support of player A. Thus, initialize backtracking.*

Revisiting Sampled game 6. *The NE is mixed with* $supp(\sigma_6^A) = \{(0, 1, 0, 0, 0, 1, 1),$ $(1, 0, 0, 0, 1, 0, 1)$, $(1, 0, 0, 1, 0, 1, 1)\}$, $supp(\sigma_6^B) = \{(1, 1, 1, 1, 0, 0, 0)$, $(1, 1, 1, 1, 0, 1, 0),$

$(1,0,0,0,0,0,1)\}$, *and* $\sigma_6 = (\frac{109}{448}, 0, \frac{11}{28}, \frac{163}{448}, 0;\ 0, \frac{409}{2314}, \frac{766}{3471}, \frac{47}{78})$. *This is an equilibrium of the original game.*

Sampled game 0

		Player B
		(1, 1, 1, 1, 1, 1, 1)
Player A	(0, 1, 0, 0, 0, 1, 1)	(262,-104)

Sampled game 1

		Player B
		(1, 1, 1, 1, 1, 1, 1)
Player A	(0, 1, 0, 0, 0, 1, 1)	(262,-104)
	(0, 1, 1, 0, 1, 1, 1)	(313,-77)

Sampled game 2

		Player B	
		(1, 1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 0, 0, 0)
Player A	(0, 1, 0, 0, 0, 1, 1)	(262,-104)	(76,-62)
	(0, 1, 1, 0, 1, 1, 1)	(313,-77)	(66,23)

Sampled game 3

		Player B	
		(1, 1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 0, 0, 0)
Player A	(0, 1, 0, 0, 0, 1, 1)	(262,-104)	(76,-62)
	(0, 1, 1, 0, 1, 1, 1)	(313,-77)	(66,23)
	(1, 0, 0, 0, 1, 0, 1)	(244,49)	(86,93)

Sampled game 4

		Player B		
		(1, 1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 0, 0, 0)	(1, 1, 1, 1, 0, 1, 0)
Player A	(0, 1, 0, 0, 0, 1, 1)	(262,-104)	(76,-62)	(165,-84)
	(0, 1, 1, 0, 1, 1, 1)	(313,-77)	(66,23)	(155,1)
	(1, 0, 0, 0, 1, 0, 1)	(244,49)	(86,93)	(86,127)

Sampled game 5

		Player B			
		(1, 1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 0, 0, 0)	(1, 1, 1, 1, 0, 1, 0)	(1, 0, 0, 0, 0, 0, 1)
Player A	(0, 1, 0, 0, 0, 1, 1)	(262,-104)	(76,-62)	(165,-84)	(157,-33)
	(0, 1, 1, 0, 1, 1, 1)	(313,-77)	(66,23)	(155,1)	(156, -33)
	(1, 0, 0, 0, 1, 0, 1)	(244,49)	(86,93)	(86,127)	(183,63)

Sampled game 6

		Player B			
		(1, 1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 0, 0, 0)	(1, 1, 1, 1, 0, 1, 0)	(1, 0, 0, 0, 0, 0, 1)
Player A	(0, 1, 0, 0, 0, 1, 1)	(262,-104)	(76,-62)	(165,-84)	(157,-33)
	(0, 1, 1, 0, 1, 1, 1)	(313,-77)	(66,23)	(155,1)	(156, -33)
	(1, 0, 0, 0, 1, 0, 1)	(244,49)	(86,93)	(86,127)	(183,63)
	(1, 0, 0, 1, 0, 1, 1)	(215,8)	(29,50)	(118,28)	(188,63)

Sampled game 7

		Player B			
		(1, 1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 0, 0, 0)	(1, 1, 1, 1, 0, 1, 0)	(1, 0, 0, 0, 0, 0, 1)
Player A	(0, 1, 0, 0, 0, 1, 1)	(262,-104)	(76,-62)	(165,-84)	(157,-33)
	(0, 1, 1, 0, 1, 1, 1)	(313,-77)	(66,23)	(155,1)	(156, -33)
	(1, 0, 0, 0, 1, 0, 1)	(244,49)	(86,93)	(86,127)	(183,63)
	(1, 0, 0, 1, 0, 1, 1)	(215,8)	(29,50)	(118,28)	(188,63)
	(1, 1, 1, 1, 0, 0, 1)	(133,90)	(36,76)	(36,110)	(188,63)

Sampled game 8

		Player B				
		(1, 1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 0, 0, 0)	(1, 1, 1, 1, 0, 1, 0)	(1, 0, 0, 0, 0, 0, 1)	(1, 1, 1, 0, 1, 0, 1)
Player A	(0, 1, 0, 0, 0, 1, 1)	(262,-104)	(76,-62)	(165,-84)	(157,-33)	(173,-78)
	(0, 1, 1, 0, 1, 1, 1)	(313,-77)	(66,23)	(155,1)	(156, -33)	(224, -51)
	(1, 0, 0, 0, 1, 0, 1)	(244,49)	(86,93)	(86,127)	(183,63)	(244,19)
	(1, 0, 0, 1, 0, 1, 1)	(215,8)	(29,50)	(118,28)	(188,63)	(188, 77)
	(1, 1, 1, 1, 0, 0, 1)	(133,90)	(36,76)	(36,110)	(188,63)	(195, 103)

Figure B.0.2: Modified SGM applied to Example B.0.21. The strategies in cyan are mandatory to be in the equilibrium support to be computed.

Revisiting sampled game 7

		Player B				
		(1, 1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 0, 0, 0)	(1, 1, 1, 1, 0, 1, 0)	(1, 0, 0, 0, 0, 0, 1)	(1, 1, 1, 0, 1, 0, 1)
	(0, 1, 0, 0, 0, 1, 1)	(262,-104)	(76,-62)	(165,-84)	(157,-33)	(173,-78)
Player A	(0, 1, 1, 0, 1, 1, 1)	(313,-77)	(66,23)	(155,1)	(156, -33)	(224, -51)
	(1, 0, 0, 0, 1, 0, 1)	(244,49)	(86,93)	(86,127)	(183,63)	(244,19)
	(1, 0, 0, 1, 0, 1, 1)	(215,8)	(29,50)	(118,28)	(188,63)	(188, 77)
	(1, 1, 1, 1, 0, 0, 1)	(133,90)	(36,76)	(36,110)	(188,63)	(195, 103)

Revisiting sampled game 6

		Player B			
		(1, 1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 0, 0, 0)	(1, 1, 1, 1, 0, 1, 0)	(1, 0, 0, 0, 0, 0, 1)
	(0, 1, 0, 0, 0, 1, 1)	(262,-104)	(76,-62)	(165,-84)	(157,-33)
Player A	(0, 1, 1, 0, 1, 1, 1)	(313,-77)	(66,23)	(155,1)	(156, -33)
	(1, 0, 0, 0, 1, 0, 1)	(244,49)	(86,93)	(86,127)	(183,63)
	(1, 0, 0, 1, 0, 1, 1)	(215,8)	(29,50)	(118,28)	(188,63)
	(1, 1, 1, 1, 0, 0, 1)	(133,90)	(36,76)	(36,110)	(188,63)

Figure B.0.3: Continuation of Figure B.0.2. The strategies in gray are not considered in the support enumeration.

Appendix C

List of Acronyms

BP	-	Bilevel programming
CKG	-	Two-player coordination knapsack game
DeRi	-	Dempe-Richter bilevel knapsack problem
DNeg	-	DeNegre bilevel knapsack problem
IA	-	Independent agent
IP	-	Integer programming problem
IPG	-	Integer programming game
KEP	-	Kidney exchange problem
KKT	-	Karush-Kuhn-Tucker conditions
KP	-	Knapsack problem
LP	-	Linear programming
LSP	-	Lot-sizing problem
MACH	-	Mansi-Alves-de-Carvalho-Hanaf bilevel knapsack problem
MIBP	-	Mixed integer bilevel programs
MIP	-	Mixed integer programming
MIQP	-	Mixed integer quadratic programming problem
MMG	-	Maximum matching in a graph
m-SGM	-	Modified sampled generation method
NE	-	Nash equilibrium
NG_0	-	Nogood constraint
NG_1	-	Strong maximal constraint
NG_2	-	Nogood constraint for the follower
NG_3	-	Cutting plane constraint
N–KEG	-	N-player kidney exchange game
PNS	-	Porter, Nudelman and Shoham's algorithm
QP	-	Quadratic programming

RIPG	-	Relaxed integer programming game
SGM	-	Sampled generation method
SWE	-	Social welfare equilibrium
ULSG	-	Competitive uncapacitated lot-sizing game
ULSG-sim	-	Modified ULSG
ULSP	-	Uncapacitated lot-sizing problem

List of Figures

List of Tables