FACULTEIT ECONOMIE EN
BEDRIJFSWETENSCHAPPEN

KATHOLIEKE
UNIVERSITEIT
LEUVEN

ALGORITHMS FOR SELECTION AND GRAPH-COLORING PROBLEMS WITH APPLICATIONS IN MARKETING AND MICRO-ECONOMICS

Proefschrift voorgedragen
tot het behalen van de graad
van Doctor in de Toegepaste
Economische Wetenschappen

door

Fabrice TALLA NOBIBON

Number 340 2010

Committee

Prof. Dr. Roel Leus (Advisor)	*Katholieke Universiteit Leuven*
Prof. Dr. Frits C. R. Spieksma (Advisor)	*Katholieke Universiteit Leuven*

Prof. Dr. Alessandro Agnetis	*Università degli Studi di Siena*
Prof. Dr. Laurens Cherchye	*Katholieke Universiteit Leuven*
Prof. Dr. Willy Gochet	*Katholieke Universiteit Leuven*
Prof. Dr. Cor Hurkens	*Eindhoven University of Technology*

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisors Roel Leus and Frits Spieksma for their continuous support during my PhD. I appreciate all your contributions of time, ideas, advices, and funding to make my stay productive and stimulating. The joy, the enthusiasm and the professionalism you have for your research were contagious and motivational for me, even during tough times. I learnt at lot from you and I will always remember some of your expressions. Roel regularly asks the question: "what is the practical application of the problem that you are solving?" and if I do not propose an answer, he will continue with "what is the difference between what you are doing and pure mathematics?". It is true that most of the time, I did not know the difference, I just wanted to solve a nice problem. Frits, I keep in mind this part of our conversations: "do you understand the problem you want to solve?". This question is really tricky because a positive answer opens the way to the next one: "could you explain it to me?" while a negative answer opens the door of his office (for me to go out and read again).

I would also like to express my gratitude to: Alessandro Agnetis, Laurens Cherchye, Cor Hurkens and Willy Gochet for kindly accepting to be a member of my doctoral committee. Your valuable and constructive remarks have significantly improved the quality of this dissertation. In particular I thank Alessandro Agnetis for pointing out some interesting extensions for future research. I thank Laurens Cherchye for introducing and explaining CARP to me and Cor Hurkens for pointing out some necessary adaptations on the topic of acyclic 2-coloring problem. Finally, I thank Willy Gochet for his careful reading of my thesis and for countless suggestions.

Next, I thank all my fantastic colleagues from the fourth and the fifth floor for the friendly environment and some animated discussions. I have certainly offended many of you during my stay, for which I would like to apologize and let you know that it was not deliberate. I have never tried to convince you, but rather to merely give my

opinion (which admittedly was sometimes provocative). I have especially appreciated contradictory debates with Dries, even if we rarely reached an agreement. I have contributed to Lotte's decision to leave our office, I am sorry for that. My former office mate, Johan, has helped me a lot with administrative details at my arrival but he has also surprised me with his punctuality and his ability of keeping our office quiet. I thank Filip for his interventions in helping me sort out some IT-issues. I have been impressed by the "no-care" theory of Stefan, although I will never implement such a theory. I have appreciated the help of Sofie with the translation of some of my administrative letters. I do not forget Sarah, Bart, Nicolas, Kris, Fabrizio, Brecht, Jeroen, Stijn, who graduated before me, and Kris (Coolen), Vikram, Leen, Wenchao, Koen, Nicolas (Depraetere), Jonathan, Marjolein, Vishva, Kukatharmini, Pieter, Yannick, Mieke, Wendi, Guoxuan, for their help and support.

A word of thanks goes out to Rahul Deb for some private communication and to Jeroen Sabbe and Bram De Rock for all the efforts they have spent explaining CARP to me. A similar acknowledgement goes to Jade Herbots for a remarkable collaboration.

I would further also like to thank all my friends. To Tchiengue and his girlfriend, thank you for making the "African dinner" a reality. I thank Patrick and Olivia Kamazi for the nice time we have had together. I will not forget Prosper Atangana, Mendana, Makomo and Pengou Fongang; thank you for all. A special mention is reserved to my former classmates all over the world, in particular Bibai, Diekouam, Kokomo, Kouamo, Mahiane, Mayag, Menoukeu, Ndeffo, Njacheun, Tabet, Takam, Tchoualag, Tchoukouegno.

A final word of gratitude goes out to my family: my parents, Nobibon and Mboukam, who have always guided my decisions. Although we sometimes have different opinions, you are fantastic. Thank you so much. I thank my sisters Matchie, Makam, Makougoum and Made and my brothers Takougang and Mbeukou. With you, I really enjoy the meaning of the word "family". I am proud of you, you do not have to follow my way because I am convinced you will find a better way.

Fabrice Talla Nobibon Leuven, September 2010.

Table of contents

Chapter 1

Introduction

The last half-century has seen a drastic advance in the application of operations research for solving real-life problems. Similarly, the development of operations-research techniques has followed. Nowadays, very efficient algorithms are available for solving basic problems; including simplex algorithms (Bertsimas and Tsitsiklis, 1997; Chvátal, 1983) and interior-point methods (Roos et al., 2006; Wright, 1997) for linear optimization problems; branch-and-bound, branch-and-cut, column-generation and dynamic programming algorithms for mixed-integer programming problems (Wolsey, 1998); backtracking algorithms for constraint satisfaction problems (Alsuwaiyel, 1999), depth-first search and breadth-first search algorithms for problems defined on graphs (Cormen et al., 2001), to name but just a few. Based on these algorithms, researchers are daily building new algorithms to deal with more difficult and complex problems arising in real-life.

One key feature of operations-research techniques is their multi-disciplinary nature. Hence, they are successfully applied in several fields including: computer sciences (Deo, 1974), electrical engineering (Mazer, 2007; Zhu, 2009), industrial engineering (Chan and Ao, 2008; Pochet and Wolsey, 2006), finance (Cornuejols and Tutuncu, 2007), marketing (Baker, 2003), health care (Brandeau et al., 2004) and biology (Aluru, 2006).

This thesis applies operations-research techniques in direct marketing, in production management, and in micro-economics. In addition to the modelling aspect of the encountered problems, we study in detail the mathematical programs obtained; thereby adding our (modest) contribution to the field of operations research. This contribution includes some complexity and non-approximability results, the extension of some well-known algorithms, and the development of new algorithms.

This thesis is organized into two parts. The first part deals with two selection problems, namely the direct marketing problem of selecting clients for a promotion campaign and a generalization of the order acceptance and scheduling problem in a single-machine environment encountered in production management. Variants of the former problem have been studied by Cohen (2004) for application in a retail bank and by De Reyck and Degraeve (2003) for application in an advertisement company. Two special cases of the latter problem have been intensively studied in the literature; these are a scheduling problem in a single-machine environment (Potts and Van Wassenhove, 1985; Tanaka et al., 2009) and an order acceptance problem (Rom and Slotnick, 2009; Slotnick and Morton, 2007). The second part of this thesis investigates the use of graph-coloring models to solve the micro-economic problem of testing the Collective Axiom of Revealed Preference. The latter is a testable, nonparametric, necessary and sufficient condition for a collective rationalization of the consumption data of multiple-member households (Cherchye et al., 2007). We first provide a graph interpretation of the Collective Axiom of Revealed Preference, which allows to formulate a sufficient condition as an acyclic graph-coloring problem. Next, we devise heuristics and exact algorithms for solving the obtained graph-coloring problem. Subsequently, we prove that the problem of testing the Collective Axiom of Revealed Preference is an NP-complete problem. Each chapter of this thesis has been made as self-contained as possible, in an attempt to make the reading independent of the other chapters.

The title of this thesis combines five keywords, which are: *algorithm*, *selection*, *graph-coloring*, *marketing* and *micro-economic*. The first keyword, algorithm, is defined as a sequence of computational steps that transform the input (some value or set of values) of a well-specified computational problem into the output (some value or set of values); see Cormen et al. (2001). Defined as such, several algorithms are proposed throughout this thesis. They are either exact or heuristic. In the former case, they either always find an optimal solution to the considered optimization problem or always output the correct answer to the decision problem at hand. Heuristics, on the other hand, are derived either to find good-quality feasible solutions to an optimization problem or to output an answer which is usually (but not always) a correct answer to a decision problem. The second keyword, selection, refers to the set of people or things that have been selected from a group (Collins, 1987). In Chapter 2, a set of clients is known and we wish to select for each product (within a finite set) a subset (possibly empty) of clients who should receive an offer of that product. In Chapter 3, our goal

is to select a subset of jobs (among a given finite set of jobs) that are to be executed on a single machine.

The third keyword, graph-coloring, is defined as an assignment of colors to elements of a graph subject to certain constraints. These elements might be vertices, edges/arcs or faces. In literature, graph coloring usually refers the a *proper* coloring of vertices of undirected graphs (Golumbic, 2004); that is a coloring of vertices of a given undirected graph such that no two adjacent vertices have the same color. In this thesis, we consider a graph-coloring problem defined on directed graphs where the constraint is that all the vertices of a cycle cannot have the same color. We investigate heuristics for solving this problem in Chapter 4 while exact algorithms are developed in Chapter 5. The fourth keyword, marketing, is defined as "the management process responsible for identifying, anticipating, and satisfying customer requirements profitably" (Brassington and Pettitt, 2003). The promotion campaign problems studied in Chapter 2 of this thesis are marketing problems. The last keyword, micro-economic, is defined as "the branch of economics that analyzes the market behavior of individual consumers and firms in an attempt to understand the decision-making process of firms and households" (see Downes and Goodman, 1995). The presence of this word in the title is related to the problem of testing the Collective Axiom of Revealed Preference studied in more details in Chapter 4 and Chapter 6. Broadly speaking, this test is used to check whether the consumption data of a multiple-members household is consistent with the collective consumption model.

The rest of this chapter unfolds as follows. Section 1.1 formally introduces the two selection problems; the promotion campaign problem is described in more details in Section 1.1.1 followed by a generalization of the order acceptance and scheduling problem on a single-machine environment in Section 1.1.2. Section 1.2 introduces the micro-economic problem of testing the Collective Axiom of Revealed Preference and the associated acyclic graph-coloring problem obtained as a sufficient condition. In Section 1.2.1, we describe the rules that define the Collective Axiom of Revealed Preference and we present the acyclic graph-coloring problem in Section 1.2.2.

1.1 Selection problems with applications in marketing and production management

In this section, we describe in more detail the two selection problems studied in the first part of this thesis. We first present direct marketing promotion campaign problems that financial institutions are facing. Subsequently, we present a generalization of the

order acceptance and scheduling problem in a single-machine environment encountered in production management.

1.1.1 Promotion campaign problems

We describe the promotion campaign problems encountered by financial institutions and studied in more details in Chapter 2 of this thesis. These problems, also called *optimal product targeting* problems, examine "which products should be targeted to which customers to maximize profits, under the constraints that only a limited number can be targeted to each customer, and each product has a minimum sales target" (see Knott et al., 2002). The objective is to find a way to achieve a maximum profit by offering n different products to m customers in the context of retention while taking into account various business constraints. We incorporate the following restrictions: the return on investment hurdle rate must be met for the campaign, the budget allocated to each product is limited, an upper bound is imposed on the number of products that can be offered to each client and there is also a minimum number of offers to be made for each product considered for the campaign. Let p_{ij} be the expected return to the firm when client i accepts the offer of product j and c_{ij} a variable cost associated with the offer of product j to client i. Further, let M_i be the maximum number of offers that can be made to a client i, B_j be the budget allocated to product j, f_j a fixed cost needed if product j is used for the campaign, O_j the minimum number of clients who must receive an offer of product j if used for the campaign, and finally R be the corporate hurdle rate. Using the decision variables $y_j \in \{0,1\}$, equal to 1 if product j is used during the campaign, 0 otherwise, and $x_{ij} \in \{0,1\}$, which is equal to 1 if product j is offered to client i and 0 otherwise, a mathematical formulation for the promotion campaign problem is:

$$\text{maximize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n}\left(p_{ij}-c_{ij}\right)x_{ij} - \sum_{j=1}^{n} f_j y_j \tag{1.1}$$

$$\text{subject to} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} \geq (1+R)\left[\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n} f_j y_j\right] \tag{1.2}$$

$$\sum_{i=1}^{m} c_{ij}x_{ij} \leq B_j \qquad j = 1,\ldots,n, \tag{1.3}$$

$$\sum_{j=1}^{n} x_{ij} \leq M_i \qquad i = 1, \ldots, m, \tag{1.4}$$

$$\sum_{i=1}^{m} x_{ij} \leq m y_j \qquad j = 1, \ldots, n, \tag{1.5}$$

$$\sum_{i=1}^{m} x_{ij} \geq O_j y_j \qquad j = 1, \ldots, n, \tag{1.6}$$

$$y_j, x_{ij} \in \{0, 1\} \qquad i = 1 \ldots, m,\ j = 1, \ldots, n. \tag{1.7}$$

The objective function (1.1) is the maximization of the total net benefit received from the offer of products to clients minus the fixed cost of using the products for the campaign. Constraint (1.2) makes sure that the campaign's return on investment is at least R. The set of constraints (1.3) enforces that we should not exceed the budget B_j allocated to the product j. The set of constraints (1.4) states that we cannot propose more than M_i products to client i; the sets of constraints (1.5) and (1.6) specify that when a product j is not part of the campaign, no clients will receive an offer, while if product j takes part in the campaign then at least $O_j > 0$ clients receive an offer, and finally the last set of constraints (1.7) is the integrality constraint.

In Chapter 2, we prove that this problem is strongly NP-hard and that it is unlikely to derive a polynomial-time algorithm which always achieves a certain proportion of the optimal profit. By applying the Dantzig-Wolfe decomposition to the formulation (1.1)-(1.7), we derive a new formulation called the *set-covering formulation*. We solve the latter formulation using a branch-and-price algorithm; a special cardinality constrained knapsack problem is obtained as a pricing problem, which we solve by adapting well-known results concerning the knapsack problem. We also develop efficient heuristics and a tabu-search algorithm that outperform methods used in practice on a benchmark of randomly generated instances.

1.1.2 Order acceptance and scheduling problem in a single-machine environment

We describe a generalization of the order acceptance and scheduling problem in a single-machine environment studied in Chapter 3 of this thesis. Given is a pool $N = \{1, 2, \ldots, n\}$ of jobs consisting of two disjoint subsets F and $\bar{F}$, for which F comprises the firm-planned orders and its complement $\bar{F}$ contains the "optional" orders (the ones that can still be rejected by the firm). Each order i is characterized by a known

processing time p_i, delivery date d_i, revenue Q_i and a weight w_i representing a penalty per unit-time delay beyond the delivery date. Our objective is to maximize total net profit, that is, the sum of the revenues of the selected jobs minus applicable total-weighted-tardiness penalties incurred for those jobs.

A solution to this problem requires two decisions to be made: which jobs in $\bar{F}$ to accept, and in which order to process the selected subset. We call M the set of jobs selected for processing, so $F \subseteq M \subseteq N$. If we let m be the cardinality of M, the sequencing decisions can be represented by a bijection $\pi : \{1, 2, \ldots, m\} \mapsto M$, where $\pi(t)$ is the job in position t in the sequence. The objective is thus

$$\max_{M,\pi} \quad \sum_{t=1}^{m} Q_{\pi(t)} - w_{\pi(t)}(C_{\pi(t)} - d_{\pi(t)})^+, \tag{1.8}$$

where $C_i = \sum_{t=1}^{\pi^{-1}(i)} p_{\pi(t)}$ is the completion time of job i and $s^+ = \max\{s, 0\}$.

When $N = F$, all jobs are mandatory and the problem reduces to the single-machine total-weighted-tardiness scheduling problem $1||\sum w_i T_i$ (Potts and Van Wassenhove, 1985; Tanaka et al., 2009). When $N = \bar{F}$, on the other hand, the problem is equivalent to the order acceptance problem with weighted-tardiness penalties discussed in (Rom and Slotnick, 2009; Slotnick and Morton, 2007).

In Chapter 3, we prove that it is unlikely to derive a polynomial-time algorithm which always achieves a certain proportion of the optimal value. Further, we propose two mixed-integer linear formulations, which can be solved using any mixed-integer-programming solver. We also develop two exact branch-and-bound algorithms that are able to solve instances of considerable size.

1.2 Graph-coloring problems with applications in micro-economics

This section investigates a solution to the micro-economic problem of testing the Collective Axiom of Revealed Preference by means of graph coloring. We first describe the rules defining this axiom and subsequently, we present the acyclic graph-coloring problem obtained as sufficient condition for the test of the Collective Axiom of Revealed Preference.

1.2.1 Collective Axiom of Revealed Preference

We define the Collective Axiom of Revealed Preference (CARP) studied in Chapter 4 and in Chapter 6 of this thesis. CARP provides a testable nonparametric necessary and sufficient condition for a collective rationalization of the consumption data set of two-member household. We refer to Cherchye et al. (2007) and references therein for detailed discussions on CARP.

More concretely, consider a two-member household that operates in an economy with N goods. At times $t = 1, 2, \ldots, T$, the household purchases a certain quantity of each of the goods $q_t \in \mathbb{R}^N_+$ (also known as a *bundle*), at corresponding prices $p_t \in \mathbb{R}^N_{++}$. We refer to a pair of N-vectors (p_t, q_t) as an *observation*, and we call the set of observations $S = \big\{(p_t, q_t) : \ t \in \mathbb{T} \equiv \{1 \ldots, T\}\big\}$ the *data set*.

CARP imposes empirical restrictions on hypothetical member-specific preference relations H^m_0 and H^m, $m \in \{1, 2\}$. In this case, the hypothetical relations H^m_0 and H^m represent feasible specifications of the true individual preference relations which are consistent with the information which is revealed by the set of observations S. First, $q_s H^m_0 q_t$ means that we "hypothesize" that member m (directly) prefers the quantities q_s over the quantities q_t, $m = 1, 2$. Next, $q_s H^m q_t$ represents the transitive closure, that is $q_s H^m q_t$ means that there exists a (possibly empty) sequence $u, \ldots, z \in T$ with $q_s H^m_0 q_u$, $q_u H^m_0 q_v$, ..., and $q_z H^m_0 q_t$. Thus given H^m_0 for $m \in \{1, 2\}$, the transitive closures H^m follow. Given this notion of hypothetical preference relations, CARP is defined as follows.

Definition 1.1 (CARP)**.** Given is a data set $S = \{(p_t, q_t) : \ t \in \mathbb{T}\}$. S satisfies CARP if there exist hypothetical relations H^1_0 and H^2_0 that satisfy for all $s, t, t_1, t_2 \in \mathbb{T}$:

Rule 1**:** if $p_s q_s \geq p_s q_t$ then either $(q_s, q_t) \in H^1_0$ or $(q_s, q_t) \in H^2_0$;

Rule 2**:** if $p_s q_s \geq p_s q_t$ and $(q_t, q_s) \in H^m$ then $(q_s, q_t) \in H^\ell_0$ with $\ell \neq m$;

Rule 3**:** if $p_s q_s \geq p_s(q_{t_1} + q_{t_2})$ and $(q_{t_1}, q_s) \in H^m$ then $(q_s, q_{t_2}) \in H^\ell_0$ with $\ell \neq m$;

Rule 4**:** if $p_s q_s > p_s q_t$ then either $(q_t, q_s) \notin H^1$ or $(q_t, q_s) \notin H^2$;

Rule 5**:** if $p_s q_s > p_s(q_{t_1} + q_{t_2})$ then either $(q_{t_1}, q_s) \notin H^1$ or $(q_{t_2}, q_s) \notin H^2$;

where $p_s q_s$ is the scalar product and represents the expenditure of observation (p_s, q_s).

The problem of testing whether a given data set S satisfies CARP can be phrased as the following decision problem.

INSTANCE: A data set $S = \{(p_t, q_t) : \ t \in \mathbb{T}\}$.
QUESTION: Does the data set satisfy CARP? In other words, do there exist H_0^1 and H_0^2 such that *Rules* 1-5 are satisfied?

In Chapter 4, we translate the rules defining CARP (*Rules* 1-5) into a directed graph setting; and based on the graph obtained, we derive necessary and sufficient conditions. We prove that the necessary condition can be verified in polynomial time while the problem of verifying the sufficient condition is an NP-complete problem. We propose and implement heuristics that quickly test our sufficient condition; these heuristics are applied both to real-life instances and to random instances. Chapter 6 deals with the complexity aspect of testing CARP. We prove that this problem is NP-complete via a reduction from the Not-All-Equal-3Sat problem.

1.2.2 Acyclic 2-coloring problem

We briefly define the acyclic 2-coloring problem studied in Chapter 4 and in Chapter 5 of this thesis. The problem is the following. We are given a finite, directed graph $G = (V, A)$, where V is the set of vertices and A the set of arcs. The goal is to partition the vertices of G into two subsets such that each subset induces an acyclic subgraph. In Chapter 4, we investigate the solution to a special case of this problem, obtained as sufficient condition for CARP, by means of heuristics. In Chapter 5, we investigate exact algorithms for solving the acyclic 2-coloring problem. We show that it is unlikely to find a constant factor approximation algorithm for solving an optimization formulation which maximizes the number of vertices that can be colored using two colors while avoiding monochromatic cycles. We identify classes of graphs for which this problem is easy. Further, we develop and implement three exact algorithms. These algorithms are tested both on graphs obtained in Chapter 4 and on randomly generated graphs.

Part I

Selection problems with applications in marketing and production management

Chapter 2

Promotion campaign problems

In this chapter, we study the optimization model defined in Section 1.1.1 for the selection of sets of clients that will receive an offer for one or more products during a promotion campaign. The complexity of the problem makes it very difficult to produce optimal solutions using standard optimization methods. We propose an alternative set-covering formulation and develop a branch-and-price algorithm to solve it. We also describe seven heuristics and a tabu-search algorithm to approximate an optimal solution. Two of the heuristics are algorithms based on restricted versions of the basic formulation, the third is a successive exact k-item knapsack procedure. A heuristic inspired by the Next-Product-To-Buy model and a depth-first branch-and-price heuristic are presented. We also explore the possibilities of truncation and linear programming rounding in the mixed-integer-programming solver. We perform extensive computational experiments for the two formulations as well as for the seven heuristics and the tabu-search algorithm.

2.1 Introduction

Promotion campaigns are fundamental direct marketing tools for improving the economic profit of a firm, either by acquiring new customers or by generating additional

This chapter is the result of a collaboration with Roel Leus and Frits Spieksma. A preliminary version is available as: F. Talla Nobibon, R. Leus and F.C.R. Spieksma, 2008. Models for the optimization of promotion campaigns: exact and heuristic algorithms. Research report KBI_0814, Department of Quantitative Methods and Information Management, Faculty of Business and Economics, KULeuven (Leuven, Belgium).

revenue from existing customers (Kotler and Armstrong, 2006). The former action is called "acquisition" while the latter is "retention" (Reinartz et al., 2005). In this chapter we are concerned with the latter case: campaigns that generate additional revenue by offering new products to existing customers. This study is justified by at least two practical facts. Reinartz et al. (2005) point out that "when firms trade off between expenditures for acquisition and those for retention, a suboptimal allocation of retention expenditures will have a greater impact on long-term customer profitability than will suboptimal acquisition expenditures". Moreover, models and methods used for data analysis are more suited for retention (Knott et al., 2002) since more information is available. Retention boosts the customer lifetime value, which is defined by Kumar et al. (2004) to be "the sum of cumulated cash flows – discounted using the weighted average cost of capital – of a customer over his or her entire lifetime with the firm". Customer lifetime value usually serves as a metric for a ranking or segmentation of the firm's customers (Ryals, 2005). During the last decades, the advances in data analysis coupled with the availability of customer data have pushed firms to develop a more customer-oriented strategy. Nowadays, such a strategy is globally accepted, but its practical implementation is far from being accomplished. This implementation delay is observed both in business-to-business and business-to-consumer settings, and is particularly pronounced in financial institutions such as large banks and insurance companies, which often have a large number of customers with full data available but may lack sophisticated tools that efficiently take into account these advantages in decision making (Dwyer, 1997).

In literature, promotion campaign models are also frequently referred to as optimal product targeting models (Knott et al., 2002). The latter examine "which products should be targeted to which customers to maximize profits, under the constraints that only a limited number can be targeted to each customer, and each product has a minimum sales target", which is mentioned by Knott et al. (2002) as an interesting issue for future research. A promotion campaign problem is essentially characterized by two steps, which are "data analysis" and "problem formulation and solution". The first step, which is mainly statistical, has received increasing attention with the advances in data analysis. Recently, numerous models that carry the name "response models" have been developed and are currently being used in practice (Cohen, 2004; Knott et al., 2002). Although this step is necessary for an application in financial institutions (as its outputs are used as inputs for the second step), its use can be less important for an application

in other areas. De Reyck and Degraeve (2003), for example, have developed a model mainly based on the second step for an advertisement company. Recently, Bhaskar et al. (2009) have proposed a fuzzy mathematical programming approach where fuzzy numbers are used to represent the output of the first step, allowing to take into account uncertainty. Their model, however, incorporates only a budget constraint and volume target constraints.

In this chapter, we investigate the development of optimization models for promotion campaigns based on integer programming. Motivation for studying this problem comes from a case occurring at FORTIS (Hellinckx, 2004), one of the former leading banks in Belgium. We aim to maximize the profit (return on investment) subject to business constraints such as the campaign's return on investment hurdle rate that must be met (the hurdle rate is the minimum acceptable rate of return that management will accept for the campaign), a limitation on the funding available for each product, a restriction on the maximum number of possible offers to a client, and a minimum quantity commitment (MQC) on the number of units of a product to be offered in order for that product to be part of the campaign. This constraint has been briefly mentioned by Cohen (2004) as a technical issue for an application in a bank. However, he did not explicitly incorporate it into his model. Our model takes into account this constraint, making it an extension of the model used by Cohen. In our formulation, we also impose a more general version of the MQC constraint, allowing the fixed minimum quantity to depend on the product, which distinguishes the constraint from comparable MQC restrictions studied in the analysis of transportation problems (Lim et al., 2006), bottleneck problems (Lim and Xu, 2006), and assignment problems (Lim et al., 2004).

We present a basic integer-programming formulation for the optimization of promotion campaigns. We show the non-approximability of the problem, which makes the existence of an algorithm that will always provide a feasible solution and guarantee a specified proportion of optimal profit in polynomial time, highly unlikely. We next derive a set-covering formulation and develop a branch-and-price algorithm for solving it. A dynamic programming algorithm and a 2-approximation algorithm are presented for solving the pricing problem, which is closely related to the k-item knapsack problem. The size of instances that can be solved optimally using this algorithm allows its efficient use for business-to-business promotion campaigns (which have moderate size and high variable and fixed costs) and for sampling approaches in financial institutions (Cohen, 2004). We then present seven heuristics and a tabu-search algorithm

to approximate an optimal solution, which can be used for large instances and hence for business-to-consumer promotion campaigns. The heuristics are either variants of the algorithms used in practice for application in a bank (Hellinckx, 2004; Knott et al., 2002) or developed based on the structure of the problem.

This chapter is organized as follows. Section 2.2 describes the basic integer-programming formulation for deciding on the composition of promotion campaigns. The complexity of the problem makes it very difficult to produce optimal solutions by straightforward use of a MIP-solver. We propose an alternative formulation called the set-covering formulation in Section 2.3 and develop a branch-and-price algorithm to solve it. In Section 2.4, we describe seven heuristics to approximate an optimal solution. The first two are algorithms based on restricted versions of the basic formulation, while the third is a successive exact k-item knapsack procedure, the fourth is a heuristic inspired by the Next-Product-To-Buy model used by Knott et al. (2002) and the fifth heuristic is a depth-first branch-and-price heuristic. The sixth heuristic is a truncated call to the MIP-solver CPLEX and the last one is an LP rounding heuristic. In Section 2.5, we develop a tabu-search algorithm for finding a nearly optimal solution to the promotion campaign problem. The experimental results for the two formulations (basic formulation and set-covering formulation) as well as for the seven heuristics and the tabu search are presented in Section 2.6. Finally, two extensions of the studied model are presented in Section 2.7, followed by some conclusions in Section 2.8.

2.2 Basic formulation

The objective of a promotion campaign is to find a way to achieve a maximum profit by offering n different products to m customers while taking into account various business constraints. We incorporate the following restrictions: the return on investment hurdle rate must be met for the campaign, the budget allocated to each product is limited, an upper bound is imposed on the number of products that can be offered to each client and there is also an MQC constraint for each product. We define the parameter r_{ij} as the probability that client i reacts positively to an offer of product j (or the probability that product j is the next product bought by client i (Knott et al., 2002)). This parameter is computed using a four steps procedure where the first step deals with compiling customers data, the second step selects a statistical model, the third step is the estimation and the evaluation of the model, and the last step scores and targets

customers; see (Knott et al., 2002) for more details. We also use the parameter DFV_{ij} to denote the return to the firm when client i responds positively to the offer of product j. This latter parameter is termed the Delta Financial Value by FORTIS (Hellinckx, 2004). These two parameters are the basis for the computation of customer lifetime value (Ryals, 2005). Practically, these parameters are estimated using response models based on historical data (Cohen, 2004; Knott et al., 2002; Prinzie and Van Den Poel, 2007) and are assumed to be available within the firm. We denote by p_{ij} the expected return to the firm (revenue) when client i is offered product j, so $p_{ij} = r_{ij}DFV_{ij}$. In the remainder of this chapter, we will mostly use p_{ij}. Further, there is a variable cost c_{ij} associated with the offer of product j to client i, the upper bound M_i of offers that can be made to a client i (this quantity is related to the status of the client), the minimum quantity commitment bound O_j associated with product j, the budget B_j allocated to the product j, a fixed cost f_j needed if product j is used for the campaign and finally the corporate hurdle rate R. The value of R is dependent on the firm and the riskiness of the investment. In practice, most firms use their weighted average cost of capital (WACC) as an estimation of R (Bruner et al., 1998). We define the decision variables $y_j \in \{0,1\}$, equal to 1 if product j is used during the campaign, 0 otherwise, and $x_{ij} \in \{0,1\}$, which is equal to 1 if product j is offered to client i and 0 otherwise. A basic formulation for the promotion campaign problem can be expressed as:

$$\text{(M1)} \quad \max \quad \sum_{i=1}^{m}\sum_{j=1}^{n}(p_{ij}-c_{ij})\,x_{ij} - \sum_{j=1}^{n} f_j y_j \tag{2.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} \geq (1+R)\left[\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n} f_j y_j\right] \tag{2.2}$$

$$\sum_{i=1}^{m} c_{ij}x_{ij} \leq B_j \qquad j = 1,\ldots,n, \tag{2.3}$$

$$\sum_{j=1}^{n} x_{ij} \leq M_i \qquad i = 1,\ldots,m, \tag{2.4}$$

$$\sum_{i=1}^{m} x_{ij} \leq m y_j \qquad j = 1,\ldots,n, \tag{2.5}$$

$$\sum_{i=1}^{m} x_{ij} \geq O_j y_j \qquad j = 1,\ldots,n, \tag{2.6}$$

$$y_j, x_{ij} \in \{0,1\} \qquad i = 1\ldots,m,\ j = 1,\ldots,n. \tag{2.7}$$

The objective function (2.1) is the maximization of the total net benefit received from the offer of products to clients minus the fixed cost of using the products for the campaign. The first constraint (2.2) is the corporate hurdle rate constraint, which makes sure that the campaign's return on investment is at least R, and which was first suggested by Cohen (2004) for an application in a bank. The set of constraints (2.3) enforces that we should not exceed the budget B_j allocated to the product j. Here, the product dependency of the budget reflects the situation in large firms where an individual business unit is responsible for the production and the sale of a product. Hence, each business unit has its own budget. The set of constraints (2.4) states that we cannot propose more than a certain number M_i of products to client i; the sets of constraints (2.5) and (2.6) constitute the MQC constraint, which specifies that when a product is not part of the campaign, no clients will receive an offer, while if product j takes part in the campaign then at least $O_j > 0$ clients receive an offer, and finally the last set of constraints (2.7) is the integrality constraint.

The graph depicted in Figure 2.1 represents an artificial instance of the promotion campaign problem with two products and three clients. The first client can receive at most one product, the second at most two products and the third client at most one. The first product (VISA) has an allocated budget of 4 and should be offered to at least two clients if used for the campaign. On the other hand, the second product, MasterCard, has a budget of five and should be offered to at least two clients. Notice that in this example, both products have no fixed cost. An optimal solution to this artificial example will offer the first product (VISA) to client 1 and client 2 to achieve the optimal objective value of 1.

Example 2.1. *An integer program corresponding to the example described by Figure 2.1 is given by:*

$$
\begin{aligned}
\max \quad & -2x_{11} + 3x_{21} + 3x_{31} + x_{12} - 2x_{22} + 2x_{32} \\
s.t. \quad & -\frac{8}{3}x_{11} + \frac{8}{3}x_{21} + \frac{5}{3}x_{31} - \frac{1}{3}x_{12} - \frac{8}{3}x_{22} + \frac{4}{3}x_{32} \geq 0 \\
& 2x_{11} + x_{21} + 4x_{31} \leq 4, \qquad 4x_{12} + 2x_{22} + 2x_{32} \leq 5 \\
& x_{11} + x_{12} \leq 1, \qquad x_{21} + x_{22} \leq 2, \qquad x_{31} + x_{32} \leq 1 \\
& x_{11} + x_{21} + x_{31} \geq 2y_1, \qquad x_{12} + x_{22} + x_{32} \geq 2y_2 \\
& x_{11} + x_{21} + x_{31} \leq 3y_1, \qquad x_{12} + x_{22} + x_{32} \leq 3y_2 \\
& y_j, x_{ij} \in \{0,1\} \qquad\qquad i = 1,2,3, \quad j = 1,2.
\end{aligned}
$$

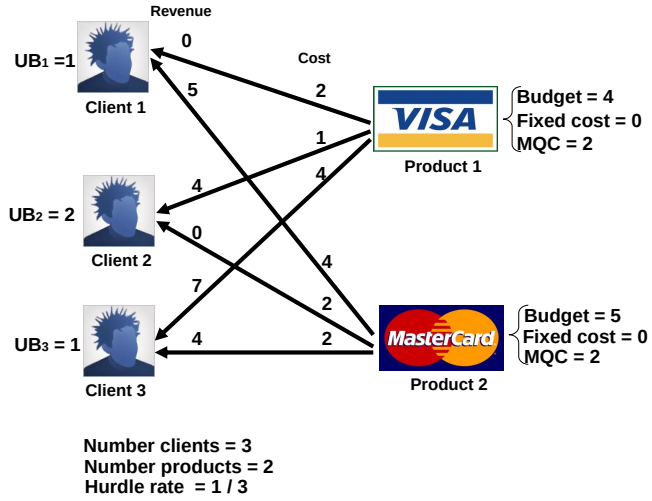


Figure 2.1: *An example of a promotion campaign with two products and three clients.*

An optimal solution offers Product 1 *to client* 1 *and client* 2. *So, we have* $x_{11} = x_{21} = y_1 = 1$, $x_{31} = 0$ *and* $x_{12} = x_{22} = x_{32} = y_2 = 0$ *for an optimal objective value of* 1. *On the other hand, an optimal solution to the LP relaxation of this example is* $x_{21} = 1$, $x_{31} = \frac{3}{4}$, $x_{12} = 1$, $x_{32} = \frac{1}{4}$, $y_1 = 0.583$, $y_2 = 0.417$ *with optimal objective value* 6.75.

Definition 2.2. A **non-trivial feasible solution** to the basic formulation (M1) is a feasible solution which achieves a non-zero objective value.

The following result shows that there is little hope for finding a polynomial-time algorithm for solving (M1).

Proposition 2.3. *The promotion campaign problem defined by the formulation (M1) is strongly NP-hard, even for* $O_j = 1$ *for all* j.

Proof. This result follows directly from the fact that the basic formulation (M1) has the generalized assignment problem (GAP) (Martello and Toth, 1990; Savelsbergh, 1997) as a special case for $R = 0$, $M_i = 1$, $\forall i$, and $O_j = 1$, $f_j = 0$, $\forall j$ and $p_{ij} \geq c_{ij}$ for all i and j. The latter problem is known to be strongly NP-hard (Savelsbergh, 1997). □

Moreover, the basic formulation (M1) is difficult to solve even approximately. We

prove this non-approximability result by showing that it is NP-hard to find a non-trivial feasible solution to (M1).

Proposition 2.4. *Finding a non-trivial feasible solution to the basic formulation (M1) is NP-hard.*

Proof. The proof uses the following variant of Partition (Garey and Johnson, 1979); by adding $|A|$ dummy elements of size 0 to an instance of the usual Partition problem, one easily sees that this variant of Partition is as hard as the original problem:

INSTANCE: A finite set $A = \{1, 2, \ldots, 2q\}$ (where q is an integer greater than 0) with size $s(i) \in \mathbb{Z}^+$ for each $i \in A$ and $K = \frac{1}{2}\sum_{i \in A} s(i)$.
QUESTION: Does there exist a subset $A' \subset A$ with $|A'| = q$ and $\sum_{i \in A'} s(i) = K$?

For a given arbitrary instance of Partition, consider the following polynomial reduction to an instance of (M1). Each $i \in A$ indicates a client with a unit upper bound, so $m = 2q$ and $M_i = 1$ for all $i = 1, 2, \ldots, 2q$. Suppose there is one product, $n = 1$. Let $\delta = 2K + 1$. For each client $i = 1, 2, \ldots, 2q$, the cost $c_{i1} = \delta + s(i)$, the revenue $p_{i1} = \delta s(i)$. Suppose also that for our product, product 1, the lower bound $O_1 = q$, the budget $B_1 = q\delta + K$, and the fixed cost $f_1 = 0$. Finally, we set the hurdle rate $R = \frac{(K-q)\delta - K}{q\delta + K}$. Now we prove that a non-trivial feasible solution to (M1) exists if and only if there is a solution A' to Partition.

On the one hand, if Partition has a solution A', we offer the product to the clients in A'. This is a non-trivial feasible solution to the instance of (M1) constructed since:

(i) the lower bound $O_1 = q$ is met,

(ii) the budget is met ($q\delta + \sum_{i \in A'} s(i) = q\delta + K = B_1$) and

(iii) the hurdle rate is achieved: $\sum_{i \in A'} p_{i1} = \delta \sum_{i \in A'} s(i) = \delta K = \frac{\delta K}{q\delta + K}(q\delta + K) = \left(1 + \frac{(K-q)\delta - K}{q\delta + K}\right)(q\delta + K) = (1 + R)(q\delta + K)$.

On the other hand, if we have a non-trivial feasible solution, then consider the set of clients A' receiving the product. We have $|A'| = q$ and $\sum_{i \in A'} c_{i1} = \sum_{i \in A'} \delta + s_i \leq B_1 = q\delta + K$. Suppose that $\sum_{i \in A'} c_{i1} < B_1$, then $\sum_{i \in A'} c_{i1} = q\delta + K_1$ with $K_1 < K$. Thus

$$\begin{aligned} \sum_{i \in A'} p_{i1} &= K_1\delta = K\delta\frac{K_1}{K} < K\delta\frac{q\delta + K_1}{q\delta + K} = (q\delta + K_1)\frac{K\delta}{q\delta + K} \\ &= \left(1 + \frac{(K-q)\delta - K}{q\delta + K}\right)(q\delta + K_1) = (1 + R)(q\delta + K_1), \end{aligned}$$

contradicting the fact that we have a non-trivial feasible solution because the corporate hurdle rate constraint is not satisfied. Therefore $|A'| = q$ and $\sum_{i \in A'} c_{i1} = q\delta + K$, which implies that A' is a solution to the variant of Partition defined above. □

The results of Proposition 2.3 and Proposition 2.4 justify the intensive use of heuristics in practice (Cohen, 2004; De Reyck and Degraeve, 2003; Knott et al., 2002).

The basic formulation (M1) can be strengthened by using the following disaggregate version of constraints (2.5):

$$x_{ij} \leq y_j \quad i = 1, \ldots, m,\ j = 1, \ldots, n. \tag{2.8}$$

The following result allows the relaxation of the integrality constraint $y_j \in \{0, 1\}$ to $0 \leq y_j \leq 1$ for all j.

Proposition 2.5. *A feasible solution to the integer program (M1) is also feasible for the set of constraints (2.2), (2.3), (2.4), (2.6), (2.8) and* $0 \leq y_j \leq 1$, $x_{ij} \in \{0, 1\}$ *for all* i, j *and vice versa.*

Proof. Clearly, any feasible solution to (M1) satisfies (2.2), (2.3), (2.4), (2.6), (2.8) and $0 \leq y_j \leq 1$, $x_{ij} \in \{0, 1\}$. Let (x^0, y^0) be feasible for the set of constraints (2.2), (2.3), (2.4), (2.6), (2.8) and $0 \leq y_j^0 \leq 1$, $x_{ij}^0 \in \{0, 1\}$ for all i, j. Then (x^0, y^0) satisfies (2.2), (2.3), (2.4), (2.6) and $x_{ij} \in \{0, 1\}$. Next, (x^0, y^0) satisfies (2.8) implies that (x^0, y^0) satisfies (2.5). Furthermore $y_j^0 \in \{0, 1\}$ for all j because if there was a product j_0 such that $0 < y_{j_0}^0 < 1$, then $x_{ij_0}^0 = 0$ for all i and constraints (2.6) would be violated. We then have $y_j^0 \in \{0, 1\}$ for all j and therefore (x^0, y^0) is a feasible solution to (M1). □

2.3 Branch-and-price

This section is devoted to the application of a branch-and-price algorithm for solving the promotion campaign problem. In the first subsection, we present an appropriate formulation, called a set-covering formulation. The next subsection studies the pricing problem and the last subsection presents a branching strategy.

2.3.1 A set-covering formulation

Proposition 2.5 and the inequalities (2.8) lead to the following strengthened formulation of our problem:

$$\text{(M2)} \quad \max \quad \sum_{i=1}^{m}\sum_{j=1}^{n}(p_{ij}-c_{ij})\,x_{ij} - \sum_{j=1}^{n} f_j y_j \tag{2.9}$$

$$\text{s.t.} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} \geq (1+R)\left[\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n} f_j y_j\right] \tag{2.10}$$

$$\sum_{j=1}^{n} x_{ij} \leq M_i \qquad i=1,\ldots,m, \tag{2.11}$$

$$\sum_{i=1}^{m} c_{ij}x_{ij} \leq B_j \qquad j=1,\ldots,n, \tag{2.12}$$

$$\sum_{i=1}^{m} x_{ij} \geq O_j y_j \qquad j=1,\ldots,n, \tag{2.13}$$

$$x_{ij} \leq y_j \leq 1 \qquad i=1,\ldots,m,\ j=1,\ldots,n, \tag{2.14}$$

$$y_j \geq 0,\ x_{ij} \in \{0,1\} \quad i=1,\ldots,m,\ j=1,\ldots,n. \tag{2.15}$$

We investigate a relaxation of (M2), in which we consider (2.10) and (2.11) to be *coupling constraints* and for each product $j = 1,\ldots,n$, we have the *special constraints* given by (2.12), (2.13), (2.14) and (2.15). For $j = 1,\ldots,n$, we define $A_j = \{(x_{.j}, y_j);\ \sum_{i=1}^{m} c_{ij}x_{ij} \leq B_j,\ \sum_{i=1}^{m} x_{ij} \geq O_j y_j,\ x_{ij} \leq y_j,\ i = 1,\ldots,m,\ y_j \leq 1,\ y_j \geq 0,\ x_{.j} \in \{0,1\}^m\}$, where $x_{.j}$ is a vector with m entries x_{ij}. Because $(0_{.j}, 0) \in A_j$, A_j is nonempty. Furthermore, A_j is bounded as $0 \leq y_j, x_{ij} \leq 1$. For every product j, $A_j = \{(x^0_{.j}, y^0_j), (x^1_{.j}, y^1_j), (x^2_{.j}, y^2_j), \ldots, (x^{k_j}_{.j}, y^{k_j}_j)\}$. Explicitly, A_j consists of $(0_{.j}, 0)$ and all the $(x^0_{.j}, 1)$ with $\sum_{i=1}^{m} c_{ij}x_{ij} \leq B_j$, $\sum_{i=1}^{m} x_{ij} \geq O_j$, $x_{ij} \in \{0,1\}$, that is the empty set of clients coupled with 0 and any subset of clients of cardinality greater than or equal to O_j and total cost less than or equal to B_j, coupled with 1. We assume that $(x^0_{.j}, y^0_j) = (0_{.j}, 0)$ so that $A_j = \{(0_{.j}, 0), (x^1_{.j}, 1), (x^2_{.j}, 1), \ldots, (x^{k_j}_{.j}, 1)\}$.

We relax (M2) by considering $conv(A_j)$ for each product j, where $conv(A_j)$ is the convex hull of A_j. Any element $(x_{.j}, y_j) \in conv(A_j)$ is a convex combination of its extreme vertices and hence can be represented in the form $(x_{.j}, y_j) = \sum_{p=0}^{k_j} z_{pj}(x^p_{.j}, y^p_j)$ where the coefficients z_{pj} are nonnegative and satisfy $\sum_{p=0}^{k_j} z_{pj} = 1$, $j = 1,\ldots,n$. A Dantzig-Wolfe decomposition of the relaxation of (M2) is then given by the master

problem (MP).

$$\text{(MP)} \quad \max \quad \sum_{j=1}^{n}\left[\sum_{p=0}^{k_j}\left[\left(\sum_{i=1}^{m}(p_{ij}-c_{ij})x_{ij}^{p}\right)-f_j y_j^p\right]z_{pj}\right] \tag{2.16}$$

$$\text{s.t.} \quad \sum_{j=1}^{n}\left[\sum_{p=0}^{k_j}\left[\left(\sum_{i=1}^{m}(p_{ij}-(1+R)c_{ij})x_{ij}^{p}\right)-(1+R)f_j y_j^p\right]z_{pj}\right] \geq 0 \tag{2.17}$$

$$\sum_{j=1}^{n}\sum_{p=0}^{k_j} x_{ij}^{p} z_{pj} \leq M_i \qquad i=1,\ldots,m, \tag{2.18}$$

$$\sum_{p=0}^{k_j} z_{pj} = 1 \qquad j=1,\ldots,n, \tag{2.19}$$

$$z_{pj} \geq 0 \qquad j=1,\ldots,n,\ p\in\{0,1,\ldots,k_j\}. \tag{2.20}$$

In what follows, we will consider the subset of extreme points without $(0_{\cdot j},0)$, subsequently denoted as $\bar{A}_j=\{(x_{\cdot j}^1,1),(x_{\cdot j}^2,1),\ldots,(x_{\cdot j}^{k_j},1)\}$. This change requires that each constraint in (2.19) becomes an inequality in order for the optimal value to remain unaffected.

For every product j, we map each extreme point $(x_{\cdot j}^p, y_j^p)\in\bar{A}_j$ to S_{pj} where $S_{pj}=\{i\in\{1,\ldots,m\}\,|\,x_{ij}^p=1\}$. Using the equalities $\sum_{i=1}^m p_{ij}x_{ij}^p=\sum_{i\in S_{pj}}p_{ij}$ and $\sum_{i=1}^m c_{ij}x_{ij}^p=\sum_{i\in S_{pj}}c_{ij}$, an integer programming version of (MP) is

$$\text{(M3)} \quad \max \quad \sum_{j=1}^{n}\left[\sum_{p=1}^{k_j}\left(\sum_{i\in S_{pj}}(p_{ij}-c_{ij})-f_j\right)z_{pj}\right] \tag{2.21}$$

$$\text{s.t.} \quad \sum_{j=1}^{n}\left[\sum_{p=1}^{k_j}\left(\sum_{i\in S_{pj}}(p_{ij}-(1+R)c_{ij})-(1+R)f_j\right)z_{pj}\right] \geq 0 \tag{2.22}$$

$$\sum_{j=1}^{n}\sum_{p:i\in S_{pj}} z_{pj} \leq M_i \qquad i=1,\ldots,m, \tag{2.23}$$

$$\sum_{p=1}^{k_j} z_{pj} \leq 1 \qquad j=1,\ldots,n, \tag{2.24}$$

$$z_{pj}\in\{0,1\} \qquad j=1,\ldots,n,\ p\in\{1,\ldots,k_j\}. \tag{2.25}$$

In the formulation (M3), the binary variable z_{pj} indicates whether product j is offered to the set of clients S_{pj} ($z_{pj}=1$) or not ($z_{pj}=0$). The first constraint (2.22) enforces that

the campaign's return on investment must be at least R, the set of constraints (2.23) ensures that at most M_i products are offered to client i and the set of constraints (2.24) states that at most one nonempty set of clients is selected for each product. We call the formulation (M3) a *set-covering formulation* to reflect the fact that its solution can be viewed as a cover for the set of clients. The set-covering formulation (M3) is then essentially obtained by applying Dantzig-Wolfe decomposition to a relaxation of the basic formulation (M2). As a consequence, the value of the bound provided by the LP relaxation of (M3) is equal to the value of the Lagrangian dual obtained by dualizing the constraints (2.2) and (2.4) (Nemhauser and Wolsey, 1988). Furthermore, the example described by Figure 2.1 illustrates that the LP relaxation of (M3) is stronger than the LP relaxation of (M1). We denote $S_{11} = \{1, 2\}$, $S_{12} = \{2, 3\}$ and z_{11}, z_{12} the corresponding variables. The associated LP relaxation of (M3) is

$$\begin{array}{lll} \max & 1z_{11} + 0z_{12} & \\ \text{s.t.} & 0z_{11} - \dfrac{4}{3}z_{12} \geq 0 & \\ & 0 \leq z_{1j} \leq 1 & j = 1, 2. \end{array}$$

Remark that the constraints (2.23) and (2.24) are obviously satisfied. The optimal solution is $(1, 0)$ with an optimal value of 1, which is also the optimal objective function value of the integer program. The result highlighted in the above example has been observed for the generalized assignment problem by Savelsbergh (1997).

Let z be any feasible solution to the LP relaxation of (M3) and let $x^*_{ij} = \sum_{p:i\in S_{pj}} z_{pj}$, $y^*_j = \sum_{p=1}^{k_j} z_{pj}$, then (x^*, y^*) is a feasible solution to the LP relaxation of (M2) which achieves the same objective value as z. Furthermore, we have the following result, which is useful for the branching strategies.

Proposition 2.6. *Given a feasible solution z to the LP relaxation of (M3), if, for a given product j, z_{pj} is fractional, then there must be an i such that $x^*_{ij} = \sum_{p:i\in S_{pj}} z_{pj}$ is fractional.*

Proof. Suppose that z_{pj} is fractional and there is no client i such that x^*_{ij} is fractional. Let $F_j = \{p \in \{1, \ldots, k_j\} \,|\, 0 < z_{pj} < 1\}$ be the set of fractional variables associated with product j. We may assume that $|F_j| \geq 2$, otherwise the hypothesis (there is no client i such that x^*_{ij} is fractional) will be violated. We have $\sum_{p\in F_j} z_{pj} = y^*_j \leq 1$ for $j = 1, \ldots, n$ and $\sum_{p\in F_j : i\in S_{pj}} z_{pj} = x^*_{ij} \in \{0, 1\}$, $i = 1, \ldots, n$. In particular for $i \in \cup_{p\in F_j} S_{pj}$, we have $\sum_{p\in F_j : i\in S_{pj}} z_{pj} = 1$. But $\sum_{p\in F_j : i\in S_{pj}} z_{pj} = \sum_{p\in F_j} \mathbf{1}_{S_{pj}}(i)\, z_{pj} = 1$ and $\sum_{p\in F_j} z_{pj} = 1$;

where $\mathbf{1}_{S_{pj}}$ is the indicator function of S_{pj}. Hence, $\mathbf{1}_{S_{pj}}(i) = 1$, $\forall p \in F_j$, meaning that for all $p_1, p_2 \in F_j$, we have $S_{p_1} = S_{p_2}$, contradicting $|F_j| \geq 2$. □

Proposition 2.6 implies that the bound provided by the LP relaxation of the set-covering formulation is at least as strong as that obtained by the LP relaxation of the basic formulation; in fact, in Section 2.6 we show that for most instances the former bound is stronger than the latter.

2.3.2 The pricing problem

We consider the LP relaxation (LPM3) of (M3) obtained by replacing constraints (2.25) by

$$z_{pj} \geq 0 \qquad\qquad j = 1, \ldots, n,\ p \in \{1, \ldots, k_j\}. \tag{2.26}$$

The formulation (LPM3) has an exponential number of variables, which makes it difficult to solve even instances of average size. Instead of solving the master problem (LPM3), we consider a restricted problem that includes only a subset of variables (columns) and can be solved directly. Additional columns for the restricted problem can be generated by looking at the dual of (LPM3) given by:

$$\begin{aligned}
\text{(DLPM3)} \quad \min \quad & \sum_{i=1}^{m} M_i u_i + \sum_{j=1}^{n} v_j \\
\text{s.t.} \quad & \sum_{i \in S_{pj}} [(p_{ij} - (1+R)c_{ij})d + u_i] - (1+R)f_j d + v_j \\
& \qquad\qquad \geq \sum_{i \in S_{pj}} (p_{ij} - c_{ij}) - f_j,\ \forall p, j, \qquad (2.27) \\
& u_i, v_j \geq 0,\ d \leq 0 \qquad i = 1, \ldots, m,\ j = 1, \ldots, n, \qquad (2.28)
\end{aligned}$$

where we use the dual variables d corresponding to (2.22), u_i corresponding to the set of constraints (2.23) and v_j corresponding to (2.24). The optimal solution found for the restricted problem is not optimal for the master problem if the associated dual variables u, v and d violate one of the constraints (2.27). Note that the set of constraints (2.28) is automatically satisfied by the dual variables. Since it is not computationally viable to compute and check the inequality (2.27) for all couples (p, j) not included in the restricted problem, we propose to proceed as follows. We drop the index j and assume

that the product is given. We solve the following question called the *pricing problem*:

$$\exists S_p \text{ with } \sum_{i \in S_p} [p_i(d-1) + c_i(1-(1+R)d) + u_i] + v + f(1-(1+R)d) < 0\,? \quad (2.29)$$

Remark that for a given product j, the left hand side of the inequality is exactly the reduced cost of the variable z_{pj}, so that the pricing problem (2.29) checks the primal optimality condition.

Solving the pricing problem

A solution to the pricing problem (2.29) for a fixed product j can be obtained by solving the following variant of the *k-item knapsack problem*. The k-item knapsack problem, also known as cardinality constrained knapsack problem (Kellerer et al., 2004), is a knapsack problem with an extra constraint enforcing a lower or upper bound on the number of items that a feasible solution must or may contain. For ease of exposition, we define $w_i = p_i(d-1) + c_i(1-(1+R)d) + u_i$ for all i.

$$\begin{aligned}
\text{(Price)} \quad \min \quad & \sum_{i=1}^{m} w_i x_i && (2.30)\\
\text{s.t.} \quad & \sum_{i=1}^{m} c_i x_i \leq B && (2.31)\\
& \sum_{i=1}^{m} x_i \geq O && (2.32)\\
& x_i \in \{0,1\} \qquad i = 1,\ldots,m. && (2.33)
\end{aligned}$$

We refer to the problem corresponding to (2.30)−(2.33) as the problem kKP. We take the w_i's as the weights and the c_i's as the costs. Notice that in general the problem kKP is weakly NP-hard. Furthermore in our case, the w_i need not always be positive nor are they always integers. These last observations make the use of dynamic programming by weight (Kellerer et al., 2004) for solving kKP inefficient. Kellerer et al. (2004) derive a dynamic programming algorithm for the case where the cardinality constraint (2.32) is replaced by $\sum_{i=1}^{m} x_i \leq O$. We show here that this algorithm is easily modified to deal with (2.30)−(2.33).

Exact algorithm

We propose a dynamic programming algorithm based on the algorithm described in (Kellerer et al., 2004) for solving kKP in pseudopolynomial time. Let k be the maximum number of items that can be used by a feasible solution to kKP. Clearly, since $c_i \geq 0$, k can be computed by taking the items with smallest cost c_i until the budget constraint (2.31) is violated. If $k < O$ then the problem kKP is infeasible. Assuming that $k \geq O$, we define the two-dimensional dynamic programming function $Y_i(c, \ell)$ for $i = 0, 1, \ldots, m$; $\ell = 0, 1, \ldots, k$; $c = 0, 1, \ldots, B$, as the optimal solution value of the following problem:

$$Y_i(c, \ell) = \min \left\{ \sum_{j=1}^{i} w_j x_j \mid \sum_{j=1}^{i} c_j x_j = c, \sum_{j=1}^{i} x_j = \ell, x_j \in \{0, 1\} \right\}.$$

An entry $Y_i(c, \ell) = q$ means that among the clients $1, 2, \ldots, i$, the minimum weight subset of clients with total cost c and cardinality ℓ has weight q.

The initialization is given by $Y_0(c, \ell) = +\infty$ for $\ell = 0, 1, \ldots, k$; $c = 0, 1, \ldots, B$, with $Y_0(0, 0) = 0$. Then, for $i = 1, \ldots, m$, the entries Y_i can be computed from those of Y_{i-1} by the following recursion

$$Y_i(c, \ell) = \begin{cases} Y_{i-1}(c, \ell) & \text{if } c_i > c \\ \min \{Y_{i-1}(c, \ell), Y_{i-1}(c - c_i, \ell - 1) + w_i\} & \text{if } \ell > 0,\ c_i \leq c. \end{cases} \tag{2.34}$$

After computing Y_1 to Y_m, the optimal objective function value of kKP is given by

$$\min \{Y_m(c, \ell) \mid \ell \geq O,\ \ c = 0, 1, \ldots, B\}.$$

We observe that only entries in $Y_{i-1}(c, \ell)$ need to be stored in order to derive $Y_i(c, \ell)$. Caprara et al. (2000) propose an implementation based on pointers that achieves a space complexity of $O(k^2 B)$ and a time complexity of $O(mkB)$.

Approximation algorithm

An approximation algorithm for kKP is based on the LP relaxation of the problem, which is obtained by replacing constraints (2.33) by

$$0 \leq x_i \leq 1 \quad i = 1, \ldots, m. \tag{2.35}$$

Denoting the optimal objective function value of the LP relaxation of kKP by z^{LP}, the following result proved in (Caprara et al., 2000; Kellerer et al., 2004) holds.

Lemma 2.7. *An optimal basic solution x^* of the LP relaxation (2.30), (2.31), (2.32) and (2.35), has at most two fractional components. Let $J_1 := \{l \,|\, x^*_\ell = 1\}$. If the basic solution has two fractional components x^*_i and x^*_j, supposing without loss of generality $c_j \leq c_i$, then $w_i + \sum_{\ell \in J_1} w_\ell \leq z^{LP}$ and the solution defined by $J_1 \cup \{j\}$ is feasible for the kKP.*

Algorithm 2.1 Pseudocode of the approximation algorithm for kKP

Let x^{LP} be an optimal solution of the LP relaxation of kKP
$J_1 := \{\ell \,|\, x^{LP}_\ell = 1\}, \quad F := \{\ell \,|\, 0 < x^{LP}_\ell < 1\}$ // fractional variables
if $F = \emptyset$ **then**
  $z^A := z^{LP}$
end if
if $F = \{i\}$ **then**
  $z^A := \min\{\sum_{\ell \in J_1} w_\ell, \sum_{\ell \in \{i\} \cup R_i} w_\ell\}$ where R_i is the set of the $k-1$ items with the smallest weight in $\{1, \ldots, m\} \setminus \{i\}$
end if
if $F = \{i, j\}$ with $c_j \leq c_i$ **then**
  $z^A := \min\{\sum_{\ell \in J_1 \cup \{j\}} w_\ell, \sum_{\ell \in \{i\} \cup R_i} w_\ell\}$ where R_i is the set of the $k-1$ items with the smallest weight in $\{1, \ldots, m\} \setminus \{i\}$
end if

Proposition 2.8. *The approximation algorithm for kKP described by Algorithm 2.1 is a 2-approximation algorithm and runs in $O(m)$ time.*

Proof. This follows from an easy modification of a proof in (Caprara et al., 2000; Kellerer et al., 2004). □

Remark that the pricing problem is solved for each product $j = 1, \ldots, n$. Therefore, up to n columns can be added to the master problem at each iteration of the column-generation procedure.

Using the column-generation procedure outlined above, we can solve the master problem (LPM3) in reasonable time. There is no guarantee, however, that the solution found will be integral; if this is not the case we will proceed with a branch-and-bound algorithm.

2.3.3 Branch-and-bound

Branching is needed when the optimal solution to (LPM3) turns out not to be integral. However, as Savelsbergh points out in (Savelsbergh, 1997), a naive branching strategy may sometimes lead to a conflict between the variable used for branching and the column generated. This occurs when the column generated by the pricing problem contains a client forbidden by previous branching decisions. Hence, it is worth looking for a branching strategy compatible with the pricing problem.

Proposition 2.6 allows the use of a hybrid branching policy (Barnhart et al., 1998; Savelsbergh, 1997); that is to perform branching using the basic formulation while working with the set-covering formulation. We will then fix a single variable (variable dichotomy) (Savelsbergh, 1997; Wolsey, 1998). This variable dichotomy branching strategy is exactly the branching scheme proposed by Ryan and Foster for set partitioning master problems (Barnhart et al., 1998).

In the basic formulation, fixing x_{ij} to zero forbids product j to be offered to client i, and fixing x_{ij} to one requires product j to be offered to client i. In the set-covering formulation, this is done by adding (not explicitly) one extra constraint. Fixing x_{ij} to zero leads to $\sum_{p:i\in S_{pj}} z_{pj} = 0$ and fixing to one leads to $\sum_{p:i\in S_{pj}} z_{pj} = 1$. Hence, at the node u of the branching tree, let $H(u) \subseteq \{1,\ldots,n\}$ be the subset of products j for which there exists a non-empty set of clients $R_j^u \subseteq \{1,\ldots,m\}$ who must receive an offer of product j. Similarly, let $L(u) \subseteq \{1,\ldots,n\}$ be the set of products j for which there exists a non-empty set of clients $N_j^u \subseteq \{1,\ldots,m\}$ who cannot receive an offer of product j. The LP problem (LP_u) to be solved at the node u is the combination of the LP relaxation (LPM3) of (M3) and the following two constraints.

$$\sum_{p:R_j^u\subseteq S_{pj}} z_{pj} = 1 \qquad j \in H(u), \tag{2.36}$$

$$\sum_{p:N_j^u\cap S_{pj}\neq\emptyset} z_{pj} = 0 \qquad j \in L(u). \tag{2.37}$$

The branching scheme resulting from the variable dichotomy branching strategy is compatible with the pricing problem since it does not render the pricing problem more difficult. To meet the set of constraints (2.37), we set $x_i = 0$, $\forall i \in N_j^u$ when solving the pricing problem associated with the product $j \in L(u)$. Similarly, for the set of constraints (2.36), we set $x_i = 1$, $\forall i \in R_j^u$ when solving the pricing problem corresponding to the product $j \in H(u)$. The constraints (2.31) and (2.32) are updated and

the remaining problem is still a kKP, of reduced size. Moreover, the inequality (2.24) of the restricted master will be changed into equality for the product $j \in H(u)$ to make sure that the solution to (LP_u) effectively offers product j to clients in R_j^u.

An upper bound at node u is provided by the optimal solution to the master problem (LP_u) or estimated by dualizing some constraints; details on these computations are available in (Talla Nobibon et al., 2008).

2.4 Heuristics

In this section, we present seven heuristics for the promotion campaign problem. These heuristics are either variants of the algorithms used in practice for application in a bank or specifically developed based on the structure of the problem. The first is a variant of the algorithm developed by FORTIS (Hellinckx, 2004). It assumes that the variable cost and the profitability of the different clients are identical and that the campaign involves all products. The second heuristic is also based on these simplifying assumptions, but allows for choosing the products to be used for the campaign. The third is a procedure that successively solves a number of *Exact k-item knapsack problems* (E-kKP) (which are kKP's with an equality for the cardinality constraint (2.32)). It uses an approximation algorithm for solving the (E-kKP) to identify the best product to be offered as well as the selected set of clients at each iteration. The fourth heuristic is also an iterative algorithm, inspired by the Next-Product-To-Buy model used by Knott et al. (2002) for an application in a retail bank. The fifth heuristic is a depth-first branch-and-price heuristic. The sixth heuristic is a truncated call to a MIP-solver and the last one is a LP rounding heuristic. In this section, we will say that client i is *active* if it has not yet received M_i offers.

2.4.1 Heuristic 1

Heuristic 1 is a variant of the algorithm developed by FORTIS. It uses the average cost and the average revenue for each product. These quantities are defined by $C_j := \frac{1}{m}\sum_{i=1}^{m} c_{ij}$ and $P_j := \frac{1}{m}\sum_{i=1}^{m} p_{ij}$. This heuristic ignores the selection of products for the campaign and simply imposes the minimum quantity O_j on the number of offers of each product j. Using a new decision variable $u_j :=$ number of clients that receive an

offer for the product j, the simplified formulation used by FORTIS is the following.

$$
\begin{aligned}
\text{(M4)} \quad \max \quad & \sum_{j=1}^{n} \left[(P_j - C_j) u_j - f_j \right] \\
\text{s.t.} \quad & \sum_{j=1}^{n} \left[P_j - (1+R) C_j \right] u_j \geq (1+R) \left(\sum_{j=1}^{n} f_j \right) \\
& C_j u_j \leq B_j, \quad O_j \leq u_j \leq m \quad j = 1, \ldots, n, \\
& u_j \in \mathbb{N} \qquad\qquad\qquad\quad j = 1, \ldots, n.
\end{aligned}
$$

We remark that the problem formulated here is still NP-hard as it includes an integer knapsack problem (Kellerer et al., 2004) as a special case. On the other hand, (M4) does not take into account the third constraint (2.4) of the basic formulation (M1); that is it does not enforce an upper bound on the number of products that can be offered to a client.

Algorithm 2.2 Pseudocode of Heuristic 1

1: for each product j, compute the average revenue P_j and the average cost C_j
2: solve the resulting integer-programming formulation (M4)
3: sort products such that for two products j and k, $j < k$ if and only if $P_j u_j \geq P_k u_k$
4: for each product j, sort clients such that for two clients i and ℓ, $i < \ell$ if and only if $p_{ij} \geq p_{\ell j}$
5: consider the products following the order obtained in line 3; offer each product j to the first u_j active clients in the sequence given by line 4
6: keep this solution if it is feasible and if its total profit is greater than 0; otherwise, output the trivial solution

2.4.2 Heuristic 2

Heuristic 2 is an improvement of Heuristic 1; here the average revenue and the average cost per product are still used, but the choice of products to be offered during the campaign is taken into account. Using an additional binary variable y_j that equals 1 if product j takes part in the campaign and 0 otherwise, the integer programming

problem to be solved is

$$
\begin{aligned}
\text{(M5)} \quad \max \quad & \sum_{j=1}^{n} [(P_j - C_j)u_j - f_j y_j] \\
\text{s.t.} \quad & \sum_{j=1}^{n} [(P_j - (1+R)C_j)\, u_j - (1+R) f_j y_j] \geq 0 \\
& C_j u_j \leq B_j \qquad\qquad j = 1, \ldots, n, \\
& O_j y_j \leq u_j \leq m y_j \qquad j = 1, \ldots, n, \\
& y_j \in \{0,1\}, \quad u_j \in \mathbb{N} \qquad j = 1, \ldots, n.
\end{aligned}
$$

Notice that (M5) generalizes (M4), since (M4) arises when we set $y_j = 1$ for all j. This formulation also does not take into account the upper bound on the number of products that can be offered to a client. However, unlike formulation (M4), (M5) does incorporate the choice of products to be offered during the campaign.

Algorithm 2.3 Pseudocode of Heuristic 2

1: for each product j, compute the average revenue P_j and the average cost C_j
2: solve the resulting integer-programming formulation (M5)
3: sort products such that for two products j and k, $j < k$ if and only if $P_j u_j \geq P_k u_k$
4: for each product j, sort clients such that for two clients i and ℓ, $i < \ell$ if and only if $p_{ij} \geq p_{\ell j}$
5: consider the products following the order obtained in line 3; offer each product j to the first u_j active clients in the sequence given by line 4
6: keep this solution if it is feasible and if its total profit is greater than 0; otherwise, output the trivial solution

Notice that if the last line (line 6) is not included in the description of Heuristic 2 then its outcome may be an infeasible solution; the same is true for Heuristic 1. As an example, consider an instance with one product and three clients, where $c_{11} = 5$, $c_{21} = 2$ and $c_{31} = 2$. The revenues are $p_{11} = 10$, $p_{21} = 8$ and $p_{31} = 3$. The hurdle rate $R = 50\%$, the budget $B_1 = 6$, the lower bound $O_1 = 2$ and the fixed cost $f_1 = 0$. The application of either Heuristic 1 or Heuristic 2 will offer the product to the first and second client. However, this solution is not feasible as the total cost is 7, greater than the budget.

2.4.3 Heuristic 3

Heuristic 3 iteratively solves a number of successive Exact k-item knapsack problem (E-kKP); an E-kKP being a kKP with an equality for the cardinality constraint (2.32). For a given product j, we define the following E-kKP:

$$\begin{aligned}
\text{(E-kKPj)} \quad \max \quad & \sum_{i=1}^{m} \left[p_{ij} - (1+R)c_{ij}\right] x_{ij} - (1+R)f_j \\
\text{s.t.} \quad & \sum_{i=1}^{m} c_{ij} x_{ij} \leq B_j \\
& \sum_{i=1}^{m} x_{ij} = O_j \\
& x_{ij} \in \{0,1\} \qquad i = 1, \ldots, m.
\end{aligned}$$

We denote the associated k-item knapsack problem by kKPj. Remark that the LP

Algorithm 2.4 Pseudocode of Heuristic 3

$val := 0, \quad Rate := 0, \quad V := \{1, \ldots, m\}$ //val is the objective value
for each $j = 1 \ldots, n$, solve (E-kKPj) and compute val_j^E
while there exists a product j with $val_j^E > 0$ and $Rate + z_j^E \geq 0$ **do**
  select such product j^* with the highest profit $val_{j^*}^E$
  $y_{j^*} := 1, \quad val := val + val_{j^*}^E, \quad Rate := Rate + z_{j^*}^E$
  for $i \in J_{j^*}^E$ **do**
    $x_{ij^*} := 1, \quad M_i := M_i - 1$
    if $M_i = 0$ **then**
      $V := V \setminus \{i\}$
    end if
    forbid any further offer of the product j^* to client i //by setting c_{ij^*} greater than B_{j^*}
  end for
  $O_{j^*} := 1, \quad f_{j^*} := 0$
  update B_{j^*}
  for each $j = 1 \ldots, n$ satisfying $O_j \leq |V|$, solve (E-kKPj) and compute val_j^E
end while
$val := \max\{val; \max\{val_j^A : \ j = 1 \ldots, n, z_j^A \geq 0\}\}$

relaxation of (E-kKPj) has a solution with either 0 or 2 fractional components. The approximation algorithm for kKP is used as a 2-approximation algorithm for (E-kKPj).

The pseudocode of Heuristic 3 (Algorithm 2.4) describes an iterative procedure for finding a solution to the promotion campaign problem using a 2-approximate solution for (E-kKPj). We denote the objective value found by the approximation algorithm for the problem (E-kKPj) (respectively kKPj) by z_j^E (respectively z_j^A) and the set of solution components equal to 1 by J_j^E (respectively J_j^A). The quantity $val_j^E = \sum_{i \in J_j^E} [p_{ij} - c_{ij}] - f_j$ (respectively $val_j^A = \sum_{i \in J_j^A} [p_{ij} - c_{ij}] - f_j$) is the profit collected by offering product j to the clients selected by J_j^E (respectively J_j^A). Heuristic 3 works as follows: it first selects the product j^* with the highest positive profit $val_{j^*}^E$ such that the hurdle rate constraint is not violated. Then, this product is offered to the set of clients $J_{j^*}^E$, the problem is updated and the procedure is repeated until no more product can be offered to clients; the quantity val represents the objective value of the solution obtained by the algorithm at each stage of its execution. Notice that after each iteration, the reconstructed problem is still a promotion campaign problem. The last line (line 17) chooses the best solution between the obtained solution and the solutions achievable by solving kKPj for each product j.

Algorithm 2.5 Pseudocode of Heuristic 4

```
1:  for j = 1, ..., n do
2:    sort clients such that i < ℓ if and only if r_ij ≥ r_ℓj
3:    for i = 1, ..., m do
4:      if i is active and there is enough budget then
5:        x_ij := 1,    M_i := M_i − 1
6:        if M_i = 0 then
7:          i becomes inactive
8:        end if
9:      else
10:       x_ij := 0
11:     end if
12:   end for
13: end for
14: keep this solution if it is feasible and if its total profit is greater than 0; otherwise,
    output the trivial solution
```

2.4.4 Heuristic 4

This heuristic is inspired by the Next-Product-To-Buy model proposed by Knott et al. (2002) for an application in a retail bank. This heuristic is based on the probability r_{ij} that product j is the next product bought by client i. The pseudocode of this heuristic is given by Algorithm 2.5

2.4.5 Heuristic 5

This is a depth-first heuristic based on the branch-and-price approach. The goal of this heuristic is to find a feasible and high-quality solution as quickly as possible. To achieve this, Heuristic 5 performs a partial traversal of the nodes corresponding to a particular branching decision within our branch-and-price approach. In the heuristic, we branch on a variable with value closest to 1 and we have preference for following the branch where that variable is set to 1.

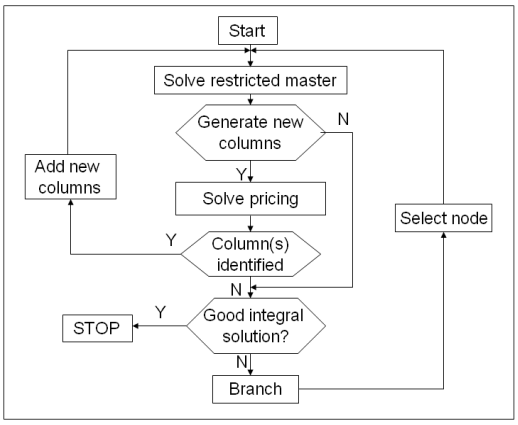


Figure 2.2: *Flow chart of the branch-and-price heuristic (Heuristic 5)*

Figure 2.2 describes the precise steps to follow. To start the algorithm, we need an initial restricted master problem with a feasible LP relaxation. Therefore, for each product, we solve the kKP problem (kKPj) with the 2-approximation algorithm. If a feasible solution is obtained, it is added to the master problem. We speed up our heuristic using ideas proposed by Vance et al. (1997). At the root node, the restricted

master problem is solved only once (so without using column generation). Next, we branch. Since the goal of this heuristic is to find a good feasible solution and not necessarily an optimal one, we do not have to branch in such a way that the solution space is divided evenly. Thus, using the hybrid branching strategy, we branch on the fractional variable x_{ij} closest to 1 and set $x_{ij} = 1$ in one branch and $x_{ij} = 0$ in the other branch. If the solution at the root node is integral we branch on a variable $x_{ij} = 1$. In case of a tie, we select the variable with the highest revenue p_{ij}. With this branching rule, we are more likely to find a good solution in the node with $x_{ij} = 1$, and we investigate that node first. At each node an upper bound on the number of calls to the pricing problem is used to stop the column generation; our implementation uses a bound of $20 \times n$, which was chosen based on some preliminary experiments. Moreover, a target value equal to 110% of the value obtained at the root node is used at each node to stop the column-generation procedure. That is, if at a given node the value of the objective function becomes larger than the target value, the column generation is stopped. We stop the algorithm when we find a feasible solution with an objective value greater than or equal to the target value. In other words, the algorithm continues when it finds a feasible solution with a value less than the target value. Similar heuristics have been used successfully in a variety of applications, including raw materials logistics planning (Luo et al., 2007) and the airline crew pairing problem (Barnhart et al., 1998; Vance et al., 1997).

2.4.6 Heuristic 6

This is a truncated call to the MIP-solver CPLEX: the solver is used to solve (M2) and is interrupted when a time limit is reached. For our implementation, we use one hour as time limit. This limit is set according to the average time spent by the other heuristics on large instances. Consequently, after at most one hour, CPLEX stops and either has already found a feasible solution, in which case the best feasible solution obtained so far is output by the heuristic, or CPLEX has not yet found a feasible solution and the heuristic outputs the trivial solution.

2.4.7 Heuristic 7

This heuristic is based on the LP relaxation of (M2). Given a solution x_{ij}, $i = 1, \ldots, m$, $j = 1, \ldots, n$ to the LP relaxation of (M2), there are two possibilities: either that solution is integer, in which case it is the output of the heuristic, or it is

fractional and we round it up to integer values as follows. The product j is used if $y_j \geq \frac{1}{2}$ and we set x_{ij} to 1 if it is greater than or equal to $\frac{1}{2}$. On the other hand, if $y_j < \frac{1}{2}$ then we set $y_j = 0$ and all corresponding x-variables also receive the value 0. If this solution is infeasible or has a non positive total profit, then the heuristic outputs the trivial solution. A time limit of one hour is imposed for solving the LP relaxation. When the limit is reached, we use the best solution found as solution of the LP.

2.5 Tabu search

In this section, we present a tabu-search algorithm for solving the promotion campaign problem. A rationale behind the choice of tabu search is the fact that it has been successfully applied to solve the generalized assignment problem (Laguna et al., 1991), which is a special case of the problem studied in this chapter.

The tabu-search algorithm is described by Algorithm 2.6. In the description, we use the function $g(x, y)$ to represent the objective function (2.1) of our problem and $\mathcal{N}_k(x, y)$ to identify a set of solutions in the neighborhood of a given solution (x, y). At any time, (x^*, y^*) is the best feasible solution obtained so far.

Algorithm 2.6 Tabu-search algorithm

1: choose an initial solution (x, y) and set $x^* = x$, $y^* = y$ and $k = 0$
2: set $k = k+1$ and consider the first solution $(x', y') \in \mathcal{N}_k(x, y)$ with $g(x^*, y^*) < g(x', y')$ which is non-tabu
3: if (x', y') exists, then set $x^* = x = x'$, $y^* = y = y'$
4: else choose (x', y') to be the best non-tabu solution in $\mathcal{N}_k(x, y)$ and set $x = x'$, $y = y'$
5: update the tabu list
6: if a stopping condition is met, then stop; else goto 2

When comparing Algorithm 2.6 to a general description of the tabu search (Aarts and Lenstra, 1997), we should point out two main differences. First, in line 2 we do not necessarily investigate the entire neighborhood $\mathcal{N}_k(x, y)$; in fact when we find a solution with a better objective value than the current best solution, we do not investigate remaining solutions in $\mathcal{N}_k(x, y)$ but rather update our solution immediately. The main reason is the fact that $\mathcal{N}_k(x, y)$ is very large. The second difference is the absence of aspiration conditions in Algorithm 2.6 (aspiration conditions are conditions that allow to consider an attractive tabu solution as our current solution).

In the rest of this section, we describe in more details each step of our tabu-search algorithm.

Algorithm 2.7 Initial solution procedure

1: $val := 0$, $exp := 0$, $V := \{1, \ldots, m\}$, $S := \emptyset$
2: for each $j = 1 \ldots, n$ and for each $i \in V$ compute $NPP_{ij} := \frac{p_{ij} - c_{ij}}{c_{ij}}$
3: **for** all product $j \notin S$ **do**
4: sort the clients in V in non-increasing order of their $NPP_{.j}$
5: $C_j :=$ the cost of offers to the first O_j clients in the sorted list
6: $P_j :=$ the revenue of offers to the first O_j clients in the sorted list
7: $PR_j := P_j - C_j - F_j$
8: **end for**
9: select $j^* \notin S$ with the highest PR_{j^*} and satisfying $C_{j^*} \leq B_{j^*}$ and $val + P_{j^*} \geq (1 + R)\big(exp + C_{j^*} + F_{j^*}\big)$
10: **if** j^* exists and $PR_{j^*} > 0$ **then**
11: $S := S \cup \{j^*\}$, $y_{j^*} := 1$, $val := val + P_{j^*}$, $exp := exp + C_{j^*} + F_{j^*}$
12: **for** each client i amongst the first O_{j^*} clients in the sorted list **do**
13: $x_{ij^*} := 1, \quad M_i := M_i - 1$
14: **if** $M_i = 0$ **then**
15: $V := V \setminus \{i\}$
16: **end if**
17: **end for**
18: goto 3
19: **end if**
20: **for** each active client i **do**
21: **for** each $j \in S$ **do**
22: if $p_{ij} > c_{ij}$ and the offer of product j to client i leads to a feasible solution, make that offer and update the current solution
23: **end for**
24: **end for**

2.5.1 Initial solution

The initial solution used by the tabu search is obtained using the procedure proposed by Van Praag (2010). This procedure is given by Algorithm 2.7.

Notice that Algorithm 2.7 runs in polynomial time; and from Proposition 2.4, there is no guarantee that it will always produce a non-trivial feasible solution. For the benchmark instances used in this chapter, however, this procedure turns out to be quite effective, as it produces a non-trivial feasible solution to all the instances.

2.5.2 Neighborhood $\mathcal{N}_k(x, y)$ and selection of (x', y')

Let (x, y) be the current solution at iteration k. The neighborhood $\mathcal{N}_k(x, y) = \mathcal{N}_k^1(x, y) \cup \mathcal{N}_k^2(x, y) \cup \mathcal{N}_k^3(x, y)$ of (x, y) is the union of three sets. The set $\mathcal{N}_k^1(x, y)$ contains feasible solutions (x', y') obtained from (x, y) by considering two clients i_1 and i_2, and a product j satisfying $y_j = 1$, $x_{i_1 j} = 1$ and $x_{i_2 j} = 0$; then we set $x'_{i_1 j} = 0$ and $x'_{i_2 j} = 1$. Notice that this solution, (x', y'), is in $\mathcal{N}_k^1(x, y)$ only if it is feasible. The second set, $\mathcal{N}_k^2(x, y)$, contains feasible solutions (x', y') obtained from (x, y) by considering two clients i_1 and i_2, and two products j and ℓ satisfying $y_j = y_\ell = 1$, $x_{i_1 j} = 1$, $x_{i_2 j} = 0$, $x_{i_1 \ell} = 0$ and $x_{i_2 \ell} = 1$; then we set $x'_{i_1 j} = 0$, $x'_{i_2 j} = 1$, $x'_{i_1 \ell} = 1$ and $x'_{i_2 \ell} = 0$. Once again, the solution (x', y') is in $\mathcal{N}_k^2(x, y)$ only if it is feasible. The last set, $\mathcal{N}_k^3(x, y)$, contains feasible solutions (x', y') obtained from (x, y) by permuting two clients i_1 and i_2; that is for all products j, we set $x'_{i_1 j} = x_{i_2 j}$ and $x'_{i_2 j} = x_{i_1 j}$. Here also, $(x', y') \in \mathcal{N}_k^3(x, y)$ only if it is feasible.

The three sets $\mathcal{N}_k^1(x, y)$, $\mathcal{N}_k^2(x, y)$ and $\mathcal{N}_k^3(x, y)$ are investigated successively in this order. Once we find a non-tabu solution (x', y') with better objective than the current best solution, we stop the investigation and that solution is considered as our new solution. On the other hand, if this is not the case, at the end of the investigation, we have found the best non-tabu solution in $\mathcal{N}_k(x, y)$. That solution is considered as our new solution. In the implementation, when this happens five times consecutively (finding a non-improving solution), we apply a diversification procedure to the current solution. The number five is chosen based on some preliminary experiments. This diversification procedure searches a feasible solution (x', y') obtained from the current solution (x, y) by replacing a product j currently in the campaign ($y_j = 1$) by another product ℓ not used in the current solution ($y_\ell = 0$); that is $y'_j = 0$ and $y'_\ell = 1$. Of course, from the set of clients $S = \{i \,|\, x_{ij} = 1 \text{ or } x_{ij} = 0 \text{ and } i \text{ is still active}\}$, we should be able to find at least O_ℓ clients who should receive an offer of product ℓ.

2.5.3 Tabu list

From iteration k to the next iteration $k+1$, we move from a solution (x, y) to a solution (x', y'). As mentioned above, this new solution can be obtained in different ways; it is either in $\mathcal{N}_k^i(x, y)$ $(i = 1, 2, 3)$ or is obtained after a diversification procedure. In any case, the reverse modification needed to go from (x', y') to (x, y) is considered as tabu for the next 20 similar iterations. More concretely, suppose that $(x', y') \in \mathcal{N}_k^2(x, y)$ is obtained from (x, y) by giving product j to client i_2 and product ℓ to client i_1. This move is then considered as tabu during the next 20 similar iterations; that means during the next 20 iterations we will not consider any solution obtained from the current solution by the reverse operation (offering product j to client i_1 and product ℓ to client i_2). Again, the number 20 is chosen based on some preliminary experiments.

2.5.4 Stopping conditions

In our implementation, we combine two stopping conditions. We stop the tabu search when either a time limit of one hour is reached or the diversification procedure was not able to find any other non-tabu feasible solution.

2.6 Computational experiments

All algorithms have been coded in C using Visual Studio C++ 2005; all the experiments were run on a Dell Optiplex $GX620$ personal computer with Pentium R processor with 2.8 GHz clock speed and 1.49 GB RAM, equipped with Windows XP. CPLEX 10.2 was used for solving the linear programs. A number of issues related to the implementation of the algorithms is explained in more detail in (Talla Nobibon et al., 2008). Below, we first provide some details on the generation of the data sets and subsequently, we discuss the computational results.

2.6.1 Generating test instances

The instances used for the experiments are randomly generated, with cost c_{ij} randomly generated from the set $\{1, 2, 3\}$ and the return to the firm $p_{ij} = r_{ij}DFV_{ij}$ is an integer randomly generated between 0 and 16. The choice of these figures is guided by examining the real-life data used by Cohen (2004). The corporate hurdle rate R belongs to the set $\{5\%, 10\%, 15\%\}$. There are six different values for the number of clients m:

these are 100, 200 and 300 clients (small size; S1, S2 and S3), 1 000 and 2 000 clients (medium size; M1 and M2) and 10 000 clients for instances of large size (L). For each number of clients, we have three different numbers of products n; these are 5, 10 and 15 products. In total, we have 54 groups of instances, as described in Table 2.1.

Table 2.1: *Size of the generated inputs*

Group	rate R	number of clients (m)	number of products (n)
S1	5%, 10% or 15%	100	5, 10 or 15
S2	5%, 10% or 15%	200	5, 10 or 15
S3	5%, 10% or 15%	300	5, 10 or 15
M1	5%, 10% or 15%	1 000	5, 10 or 15
M2	5%, 10% or 15%	2 000	5, 10 or 15
L	5%, 10% or 15%	10 000	5, 10 or 15

For each group, we generate a minimum quantity commitment bound O_j as a random integer selected between $\lceil \frac{\sum_i M_i}{n} \rceil$ and $\lceil \frac{2\sum_i M_i}{n} \rceil$ ($\lceil a \rceil$ is the ceiling of a). We consider three values for the budget B_j, namely a random integer chosen between $O_j \frac{\sum_i c_{ij}}{m}$ and $2\frac{\sum_i c_{ij}}{n}$ and the two extreme budgets which are $\lfloor O_j \frac{\sum_i c_{ij}}{m} \rfloor$ ($\lfloor a \rfloor$ is the floor of a) and $\lceil 2\frac{\sum_i M_i}{n} \frac{\sum_i c_{ij}}{m} \rceil$. The fixed cost f_j is a random integer between $\frac{O_j}{2m(1+R)} \sum_i [p_{ij} - (1+R)c_{ij}]$ and $\frac{O_j}{m(1+R)} \sum_i [p_{ij} - (1+R)c_{ij}]$. These bounds have been chosen in such a way that the instances become feasible and consistent. We generate one instance with a small upper bound M_i for each client, selected between 1 and $\frac{n}{5}$, (denoted by s) and another one with a large random upper bound M_i for each client, selected between $\lceil \frac{n}{3} \rceil$ and $\lceil \frac{2n}{3} \rceil$ (denoted by ℓ). In conclusion, we have in total $3 \times 2 \times 54 = 324$ test instances.

Since Heuristic 4 requires as input r_{ij} (and not p_{ij}), we have decided to generate for each of the 324 test instances, three different sets of values for r_{ij}. The first set is generated randomly from $]0, 1[$, while the second is also randomly generated in $]0, 1[$ but proportional to c_{ij}, and the last is inversely proportional to p_{ij}. Note that by choosing r_{ij} in $]0, 1[$, we discard certainty. These $3 \times 324 = 972$ test instances are solved using Heuristic 4. All instances can be found at http://www.econ.kuleuven.be/public/NDBAC96/promotion_campaigns.htm

2.6.2 Computational results

In this section, we report on different implementations of the column-generation procedure (Table 2.2), the performance of the LP relaxation (Table 2.3), different implementations of the branch-and-price (Table 2.4), a comparison of the basic model and the branch-and-price algorithms (Table 2.5) and a comparison of heuristics and the tabu search (Table 2.6 and Table 2.7). Throughout this section, the computation time (Time) is expressed in seconds.

Table 2.2: *Comparison of column-generation procedures for solving the LP relaxation of the set-covering formulation*

Group	n	M	Approx.		Exact		CPLEX	
			nr.	Time	nr.	Time	nr.	Time
Single column								
S1	5	s	2643	2.05	7	0.03	597	7.42
		ℓ	881	0.76	22	0.74	233	1.20
	10	s	2902	2.37	37	0.18	412	6.47
		ℓ	1325	1.08	24	0.70	179	2.17
	15	s	2896	2.30	126	0.42	319	5.83
		ℓ	1054	0.96	72	2.36	108	1.24
Multiple columns								
S1	5	s	11788	3.17	7	0.04	545	10.98
		ℓ	5056	1.41	35	1.22	266	3.69
	10	s	16020	4.28	140	0.81	455	16.14
		ℓ	6880	1.72	19	0.61	185	7.02
	15	s	14601	3.96	226	0.87	337	12.02
		ℓ	5449	1.61	68	2.36	167	3.86

Table 2.2 compares two implementations of the column-generation procedure for solving the LP relaxation of the set-covering formulation. In this table, each cell is the average of nine values. The first implementation (Single column) adds a single column per product to the master problem. For each product, the approximation algorithm is first used to solve the pricing problem; if a column is identified it is added to the master problem. On the other hand, if no column is found for any product, we successively use the exact algorithm for solving the pricing problem for each product. Once a column is identified, we stop and do not use the exact algorithm for the remaining products. The LP problem obtained is then solved using CPLEX. The second implementation (Multiple columns) can add up to five different columns per product, but only in the

case where the first column is identified using the approximation algorithm. For each product, the approximation algorithm is first used to solve the pricing problem; if a column is identified we generate at most four additional columns as follows. We randomly select a single client i_1 with $x_{i_1} = 1$ and we fix $x_{i_1} = 0$ and the pricing problem is re-solved using the approximation algorithm. If a feasible solution with negative reduced cost is obtained, the corresponding column is added and a new client i_2 with $x_{i_2} = 1$ in the above column is randomly chosen and we set $x_{i_1} = x_{i_2} = 0$. The pricing problem is re-solved and this procedure is repeated until either five columns are added or the pricing problem (with these additional constraints) is either infeasible or the solution has a non-negative reduced cost. In Table 2.2, we report for the approximation algorithm (Approx.), the exact algorithm (Exact) and CPLEX (CPLEX) the number of calls (nr.) and the total time spent (Time). We observe that the average time spent is $0.00081s$ (respectively $0.01539s$) per call of the approximation algorithm (respectively the exact algorithm) for the "Single column" implementation which is greater than the corresponding average time for the "Multiple columns" implementation, $0.00027s$ and $0.0114s$ respectively. However, the increasing number of columns and the increasing average time of using CPLEX for solving intermediary LP problems ($0.02748s$ for the second implementation compared to $0.01317s$ for the first implementation) make the overall time of the "Multiple columns" implementation larger than the overall time of the first implementation.

We have implemented a local-search algorithm to generate multiple columns per product, (a similar technique has been used by Van Den Akker et al. (2006) for application in parallel machine scheduling), but the results were not better than the results reported above. In the sequel, the LP relaxation of the set covering is solved using the first implementation.

Table 2.3 shows a comparison between the LP relaxation of the basic formulation and of the set-covering formulation. Here also, each cell is the average of nine values. The results of the group S3 are not reported in this table because there are instances in that group for which we do not know the optimal value, while for group S1 and group S2 we have obtained an optimal solution using either the MIP-solver CPLEX for solving (M2) or the branch-and-price algorithms (see Table 2.5). In Table 2.3, LP-Gap is a percentage $\frac{z_{LP}-z_{OP}}{z_{OP}} \times 100\%$, where z_{OP} is the optimal objective value and z_{LP} is the objective value of the LP relaxation.

Table 2.3 confirms the theoretical result obtained in Section 2.3.1 that the set-

Table 2.3: *LP relaxation of the basic formulation and the set-covering formulation*

			Basic formulation				Set-covering formulation			
Group	n	M	LP-Gap	Time			LP-Gap	Time		
				min	mean	max		min	mean	max
S1	5	s	8.33	0.06	0.16	0.31	5.05	0.44	9.56	24.47
		ℓ	6.51	0.08	0.30	0.55	0.03	1.05	2.20	3.86
	10	s	3.11	0.05	0.13	0.23	0.62	6.05	9.02	14.83
		ℓ	3.26	0.08	0.26	0.50	0.18	1.81	3.65	7.76
	15	s	2.96	0.05	0.14	0.27	0.45	5.51	8.59	10.66
		ℓ	1.28	0.06	0.25	0.53	0.07	1.69	4.68	11.36
S2	5	s	2.51	0.11	0.31	0.67	0.78	152.18	378.81	852.84
		ℓ	12.80	0.19	0.93	2.31	5.47	0.47	36.39	108.04
	10	s	3.37	0.11	0.46	0.91	0.21	105.60	220.75	375.74
		ℓ	10.51	0.20	1.06	2.28	0.80	0.45	92.01	176.33
	15	s	2.47	0.11	0.43	0.83	0.15	80.75	190.46	271.88
		ℓ	5.00	0.16	1.12	2.23	0.00	4.73	49.48	119.54

covering formulation is at least as strong as the the basic formulation: that is, it gives a solution very close to the optimal solution compared to the solution obtained by the basic formulation. This difference is reflected by LP-Gap for the set-covering formulation which is less than 1% for many instances. However, this quality of the LP relaxation of the set-covering formulation comes with a relatively high computation time. Notice that this computation time increases with the size of the problem (number of clients). Table 2.3 explicitly displays the trade-off between the solution quality and the computation time. Notice also that the computation time reported in Table 2.3 for the set-covering formulation is not necessarily the computation time spent by the branch-and-price algorithm at the root node as we stop solving the LP at each node of the branching tree (and therefore at the root node) when we go through 30 iterations without an improvement of the objective value.

Table 2.4 compares three different strategies of traversing the branching tree. These are breadth first, depth first, and best first. The main difference between these three strategies is observed after the branching, when selecting the child to investigate first (Savelsbergh, 1997).

Table 2.4: *Comparison of different tree traversal strategies for the branch-and-price algorithm*

			breadth first		depth first		best first	
	n	M	Time	Nr Nodes	Time	Nr Nodes	Time	Nr Nodes
S1	5	s	10131.57	178	7365.78	74	2031.13	26
		ℓ	64.68	8	38.75	6	72.85	2
	10	s	5057.54	936	1141.08	210	3035.86	410
		ℓ	9967.78	754	20172.64	1395	8587.53	208

The comparison is made for 36 instances from Group S1. We clearly see from Table 2.4 that the breadth-first branch-and-price algorithm is dominated either by the depth-first branch-and-price algorithm or by the best-first branch-and-price algorithm. Remark that the number of nodes might be important here as more nodes is likely to imply more columns generated. The generation of these columns is time consuming, and too many columns may sometimes lead to a memory problem for our branch-and-price algorithm. Observe that for $n = 10$ and $M = s$, the best-first strategy explores more nodes than the depth-first strategy; this is due to a single instance for which the best-first strategy has to investigate a very hight number of nodes in order to find a guaranteed optimal solution.

Table 2.5 displays the results of the exact algorithms for the promotion campaign problem. The basic formulation (M2) is solved using the MIP-solver of CPLEX 10.2. In Table 2.5, the minimum, the mean (average), the maximum computation time as well as the number of instances solved (Solved) are recorded. Notice that each cell is based on nine instances. For Group S1, CPLEX was able to solve each instance. However, the computation time increases when we move to Group S2. For that group, CPLEX could not solve four instances, due to memory problems. Amongst these four instances, three have a small upper bound per client M_i and only one was an instance with large upper bound. For Group S3, up to ten instances were not solved by CPLEX. In that group, all unsolved instances have a small upper bound M_i. The above observations coupled with an analysis of the computation time given in Table 2.5 for the basic formulation lead to the following conclusions. Computation time seems to increase exponentially with the size of the instance (number of clients). Moreover, instances with small upper bound M_i seem to be harder to solve than instances with large upper bound.

Table 2.5 also shows the computation times obtained by the branch-and-price algo-

Table 2.5: *Basic formulation and branch-and-price algorithms*

			Basic formulation				depth first				best first			
Group	n	M	Solved	Time			Solved	Time			Solved	Time		
				min	mean	max		min	mean	max		min	mean	max
S1	5	s	9	1.72	57.39	190.13	9	103.69	7365.78	21135.56	9	279.97	2031.13	4999.85
		ℓ	9	0.11	3.91	12.28	9	11.95	38.75	75.78	9	11.20	72.85	168.86
	10	s	9	1.33	27.90	114.84	9	24.98	1141.08	11052.67	9	578.39	3035.86	5493.34
		ℓ	9	0.17	6.05	35.95	9	61.72	20172.64	35798.08	9	74.16	8587.53	16683.14
	15	s	9	1.50	22.57	74.73	8	17259.00	39845.53	62432.07	8	3038.83	19521.5	36004.17
		ℓ	9	0.11	3.22	14.03	9	23.81	31117.1	62210.36	9	23.56	18014.18	36004.79
S2	5	s	9	0.23	253.47	1036.68	9	2862.80	36725.30	62143.55	7	2862.80	36382.61	51143.56
		ℓ	8	0.44	138.18	1094.44	9	358.72	12125.45	36017.63	9	251.53	15699.19	36101.53
	10	s	8	3.58	2246.54	16249.26	9	518.71	36138.39	57844.56	8	422.7	36696.35	57842.13
		ℓ	9	0.33	20.40	124.16	9	336.17	12010.04	36021.12	9	631.33	18334.06	36036.80
	15	s	7	3.59	1811.47	14259.19	9	1575.14	12545.51	36061.38	7	444.75	36209.22	64422.08
		ℓ	9	0.30	9.47	53.83	9	458.99	5742.13	11052.67	9	179.95	18183.22	36186.49
S3	5	s	8	0.48	241.79	1311.32	–	×	×	×	–	×	×	×
		ℓ	9	2.63	103.94	596.19	–	×	×	×	–	×	×	×
	10	s	4	12.81	823.01	3213.22	–	×	×	×	–	×	×	×
		ℓ	9	0.75	809.26	5646.58	–	×	×	×	–	×	×	×
	15	s	5	1.53	127.03	370.22	–	×	×	×	–	×	×	×
		ℓ	9	0.66	172.78	1490.54	–	×	×	×	–	×	×	×

rithms. We concentrate only on depth-first and best-first branch-and-price algorithms since Table 2.4 has shown that these two dominate the breadth-first algorithm. Using the depth-first branch-and-price algorithm, we are able to solve all the instances in the group S1 and the group S2 except for one instance in S1 while the best-first branch-and-price algorithm fails to find an optimal solution to six instances, one in the group S1 and five in S2. It is clear that using the MIP-solver of CPLEX 10.2 for solving the basic formulation (M2) (enhanced with the default cutting planes) is much faster than the branch-and-price approaches. Although the pricing problem is solved fast and LP-Gap is smaller at the root node as shown in Table 2.2 and Table 2.3, the number of nodes in the branching tree can be excessive, as witnessed in Table 2.5. We do not apply the branch-and-price algorithms for solving the instances in S3.

Table 2.6: *Comparison of heuristics for S*3

		S3		
n		5	10	15
Heuristic 3	Gap	8.46	8.50	9.17
	Time	2.80	9.63	1.66
	Feas	100	100	100
Heuristic 5	Gap	8.15	8.17	8.30
	Time	1498.87	2455.66	2301.48
	Feas	100	100	100
Heuristic 6	Gap	6.91	4.11	4.59
	Time	320.54	1452.93	1653.53
	Feas	100	100	100
Heuristic 7	Gap	0.54	0.00	0.00
	Time	0.47	1.20	3.13
	Feas	11.11	5.56	22.22
Initial solution	Gap	19.20	17.85	18.34
	Time	0.00	0.00	0.00
	Feas	100	100	100
Tabu search	Gap	6.86	6.52	7.76
	Time	1.17	0.60	0.79
	Feas	100	100	100

In Table 2.6, the outcomes of the heuristics and the tabu search are exhibited for the set of instances S3. The groups S1 and S2 are not considered because we have obtained optimal solutions to all instances in these groups. Here, Gap is a percentage

$\frac{z_{UB}-z_{AP}}{z_{UB}} \times 100\%$, where z_{UB} is either the optimal objective value, found by solving (M2) with the MIP-solver CPLEX, or an upper bound of that value obtained by solving the LP relaxation of the problem and z_{AP} is the objective value obtained either by the heuristic or by the tabu search. This Gap is computed only for instances for which a non-trivial feasible solution is found; that is when the heuristic or the tabu search finds a feasible solution with $z_{AP} > 0$. Feas is the percentage of feasible solutions with positive objective value. Unlike the previous tables, here each cell is the average of 18 values, described by three values for R, three values for the budget B and the two values of M. The experimental results of Heuristic 1, Heuristic 2 and Heuristic 4 are not displayed in Table 2.6 because they could not produce any non-trivial feasible solution. A time limit of one hour is imposed for each heuristic and for the tabu search.

The results of Table 2.6 confirm the existence of a trade-off between computation time and solution quality. Heuristic 3 gives a feasible solution with strictly positive objective value for all the test instances. Moreover, the Gap reported for each instance is less than 10% and the computation time is at most 10s. The solutions provided by Heuristic 5 are of good quality with Gap less than 9%. However, unlike Heuristic 3, Heuristic 5 requires much more time. Heuristic 6 provides feasible solutions with strictly positive objective value for each instance in S3. Although it takes more time than Heuristic 3 and the tabu search, the Gap is the smallest compared to other heuristics able to solve all the instances. The last heuristic (Heuristic 7) has a competitive computation time (less than 4s) but provides very few feasible solutions with strictly positive objective value (less than 23%). On the other hand, the tabu-search algorithm gives a non-trivial feasible solution to all the test instances. Moreover, the Gap reported for each instance is less than 7.77% and the computation time is at most 1.5s. Notice that for each instance, the initial solution is feasible and is obtained almost instantaneously (computation time is 0s). The average smallest gap is 17.85% for instances with $n = 10$ while the final output of the tabu search has a gap of 6.52%; clearly indicating an improvement of more than 11% over the initial solution.

To summarize the comparison between the four heuristics and the tabu search reported in Table 2.6, we formulate the following advice. If the solution quality is the only criterion to be taken into account, the use of Heuristic 3, Heuristic 5, Heuristic 6 or the tabu search is advised if an efficient exact algorithm is not available. However, if both the computation time and the solution quality are relevant, we strongly recommend the use of the tabu search or of the Heuristic 3. Finally, we advise not to use Heuristic 7

Table 2.7: *Comparison of heuristics for medium and large size instances*

		M1			M2			L		
n		5	10	15	5	10	15	5	10	15
H2	Gap	–	81.61	–	97.77	62.21	–	–	–	62.81
	Time	0.01	0.01	0.01	0.01	0.01	0.01	0.03	0.04	0.04
	Feas	0.00	2.00	0.00	2.25	2.25	0.00	0.00	0.04	1.50
H3	Gap	11.20	13.23	14.46	13.10	13.15	12.31	34.04	33.52	37.80
	Time	146.60	394.54	304.32	2125.62	2231.14	1195.45	3573.92	3143.41	3353.81
	Feas	100	100	100	100	100	100	100	100	100
H5	Gap	8.72	13.04	14.43	12.40	12.84	12.01	13.60	24.54	34.26
	Time	2441.74	3318.04	3274.59	1978.80	3565.28	3384.92	3730.05	3711.15	3634.33
	Feas	100	100	100	100	100	100	100	100	100
H6	Gap	4.44	2.97	23.11	2.77	17.74	23.20	14.19	13.11	–
	Time	1845.20	2414.50	3018.14	2048.79	3132.60	3523.35	3667.33	3763.12	3803.61
	Feas	100	100	77.78	100	83.33	77.78	61.11	5.56	0.00
H7	Gap	0.00	0.00	0.00	0.00	0.39	0.27	0.00	2.75	51.10
	Time	8.63	28.30	64.26	17.63	124.36	298.37	558.66	2974.50	3558.28
	Feas	11.11	11.11	16.67	11.11	22.22	16.67	22.22	5.56	11.11
Init.	Gap	17.90	16.20	18.36	18.92	19.23	17.84	26.22	24.86	25.65
	Time	0.01	0.01	0.01	0.02	0.03	0.03	0.28	0.32	0.36
	Feas	100	100	100	100	100	100	100	100	100
TBS	Gap	7.22	8.54	7.60	9.75	9.58	9.11	10.86	11.04	10.23
	Time	16.10	11.62	15.36	56.58	50.33	67.24	1268.30	1347.23	1149.28
	Feas	100	100	100	100	100	100	100	100	100

unless no other choice is available as it does not guarantee a feasible solution.

In Table 2.7, Feas and Gap have the same meaning as in Table 2.6. For some instances, the LP relaxation was not solved to optimality, and we impose a time limit of one hour and consider the value obtained at that time as z_{UB}; an upper bound of the optimal objective value since we use the dual simplex algorithm of CPLEX. For each heuristic and the tabu search, a time limit of one hour is imposed.

Table 2.7 confirms the difficulty experienced by Heuristic 2 (H2) in finding a feasible solution with strictly positive objective value. Moreover, when it finds a feasible solution, the Gap is very large (more than 60%). The only good news comes from the computation time, which increases very slowly. Heuristic 3 (H3), by contrast, behaves very well for large instances: all the instances are solved with Gap less than 15% for Group M1 and Group M2, and less that 38% for the instances with 10 000 clients. This good quality of the solutions is coupled with an increased running time. Heuristic 5 (H5) finds non-trivial feasible solutions to all instances, with a smaller Gap than

Heuristic 3. Compared to Heuristic 3, the running time is higher, however. The solutions proposed by Heuristic 6 (H6) are usually close to the optimal solution (judging from the Gap), but for large instances, this heuristic is quite limited in terms of the number of non-trivial feasible solutions it can find. We clearly see in Table 2.7 that for instances in the set L, Heuristic 6 provides at most 62% of feasible solutions when $n = 5$ and at most 6% for $n = 10$. When $n = 15$, this heuristic fails to find any non-trivial feasible solution. The last heuristic, Heuristic 7 (H7), provides good solutions from time to time, but is very limited by the number of feasible solutions it is able to find (at most 23%). As for the tabu search (TBS), it has an excellent behavior. In fact, all the instances are solved with Gap less than 10% for Group M1 and Group M2 in about one minute, and with Gap less that 12% for the instances with 10 000 clients with average time less than 23 minutes. As observed in Table 2.6, the initial solution procedure is very fast and always provides a good feasible solution. Further, there is a difference of at least 7% between the gap of the initial solution and the gap of the tabu search.

To conclude this comparison, although Heuristic 6 and Heuristic 7 provide good solutions, they are strongly limited by the number of non-trivial solutions they output for large instances. Therefore, we advice the use of either Heuristic 3, Heuristic 5 or the tabu-search algorithm with a clear preference for the latter.

In practice, there may be instances with millions of clients (Cohen, 2004). We believe that the use of the tabu search with more running time may provide a good feasible solution. In practice nowadays, when dealing with such instances, one often-used technique is to cluster the clients into groups (these techniques are also called *aggregate techniques* (Cohen, 2004)). Next, clients within a group are treated as identical. We point out here that a variant of (M1) provides a model for this situation where x_{ij} is then defined as the number of clients of group i that receive the product j and the parameters c_{ij} and p_{ij} are redefined accordingly.

2.7 Extensions

In this section, we deal with two additional constraints regularly encountered in financial institutions (Cohen, 2004; Knott et al., 2002). The first constraint is a single time window constraint. So far, our formulations take into account only the probability that client i reacts positively to an offer of product j, missing the question of *when* client i

is likely to accept an offer of j. This condition can be incorporated in our formulations by taking for a given time window T the probability r_{ij}^T that client i reacts positively to an offer of product j over the next period T. These quantities are computed using well known statistical and probability tools (Knott et al., 2002). The extension with sequentially ordered products over multiple time windows (Li et al., 2005; Prinzie and Van Den Poel, 2006) is an avenue for future work.

The second constraint is related to the maximum number of offers that can be made for a given product during the campaign. This constraint is easily incorporated into the basic formulation by adding a constraint of the form $\sum_i x_{ij} \leq N_j y_j$ for each product j, where N_j represents the maximum number of offers allowed for the product j. The set-covering formulation (M3) remains valid under a slight modification in the definition of S_{pj} such that it contains at least O_j clients and at most N_j clients. The major effect appears in the pricing problem: instead of solving the k-item knapsack problem kKP ((2.30)-(2.33)), we have to solve a $k_1|k_2$-item knapsack problem, which is a k-item knapsack problem in which the cardinality constraint (2.32) is replaced by

$$k_1 \leq \sum_{i=1}^{m} x_i \leq k_2.$$

The dynamic programming recursion applied for the k-item knapsack problem is also valid for the $k_1|k_2$-item knapsack problem if we replace k by $\min\{k, k_2\}$, while the 2-approximation algorithm is valid without any modification. This is due to the fact that a solution to the $k_1|k_2$-item knapsack problem has at most two fractional components. This second constraint is also easily incorporated in all the heuristics defined above after some minor modifications and can be taken into account by the tabu-search algorithm.

2.8 Conclusions and future work

This chapter explicitly models the promotion campaign problem taking into account both business constraints and customer preferences and specificities. The problem is strongly NP-hard and it is unlikely that a constant factor approximation algorithm can be proposed for solving this problem. We have also presented a set-covering formulation for the promotion campaign problem in which each product is associated with a subset of clients (which can be empty) in the optimal solution and developed a branch-and-price algorithm for solving it. We have shown that this last formulation is stronger than the basic formulation. These two formulations are used to produce

optimal solutions to the promotion campaign problem. Experimental results confirm that these two formulations are limited by the size of instances that they can efficiently solve, which makes their application more suited for business-to-business applications in which variable (and fixed) costs are more important and the number of clients is not very large (around 1000 clients (Air Transport Action Group, 2007)).

To extend the application to a business-to-consumer environment with considerably more customers, we have presented seven heuristics and a tabu-search algorithm. Some of the heuristics are currently used in practice (Heuristic 1, 2, 4), while others are new (Heuristic 3, 5, 6, 7). Based on extensive experimental results, we provide a comparison and comments on the efficiency and quality of the results obtained using the different formulations, the heuristics and the tabu-search algorithm. These results clearly show a trade-off between computation time and solution quality and suggest the use of optimal algorithms for small and medium size instances, while heuristics and the tabu search are preferable for large size instances (tabu search and Heuristic 5) and when time is an important factor (tabu search).

An important immediate further research direction that might be pursued is an extension of this work to multichannel offers where the products can be offered through many different channels and each channel has its own constraints (e.g. minimum and maximum number of offers through this channel (Cohen, 2004)). Further, an exploration of "high-multiplicity" approach where similar clients are put together in one class and the promotion campaign problem is run with these classes might be investigated for solving very large instances. For the long term, we may consider some extensions of the problems. One such extension concerns the case where the revenue obtained by offering a given product to a client is a S-shaped or a r-shaped function of the (effort) cost put in the promotion. Another extension of this work may take into account different related products that are offered over multiple time windows (Li et al., 2005; Prinzie and Van Den Poel, 2006). Further, since the computation of some input data, especially the probability that product j is the next product bought by client i (r_{ij}), are certainly affected by uncertainty, a robust approach of the problem that explicitly deals with such uncertainty is an important direction for future work.

Chapter 3

Order acceptance and scheduling problem in a single-machine environment

In this chapter, we study a generalization of the order acceptance and scheduling problem in a single-machine environment where a pool consisting of firm-planned orders as well as potential orders is available from which an over-demanded company can select. The capacity available for processing the accepted orders is limited and each order is characterized by a known processing time, delivery date, revenue and a weight representing a penalty per unit-time delay beyond the delivery date. We prove that the existence of a constant-factor approximation algorithm for this problem is unlikely. We propose two linear formulations that are solved using a MIP-solver and we devise two exact branch-and-bound procedures able to solve instances with up to 50 jobs within reasonable CPU times. We compare the efficiency and quality of the results obtained using the different solution approaches.

This chapter is the result of a collaboration with Roel Leus and is available as: F. Talla Nobibon and R. Leus. Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment. *Computer & Operations Research*, 38:367–378, 2011.

3.1 Introduction

Many organizations give no formal consideration to either order acceptance or rejection. Instead, an order-entry process is operated that tacitly accepts *all* orders. In today's competitive manufacturing environment, an organization must respect order deadlines agreed to with customers, but order acceptance often takes place without consideration of the effect on the planning of the other jobs in the order portfolio. Herbots et al. (2007) observe that "this is often the consequence of the functional separation between the order acceptance decision, which is made by the sales department, and the capacity planning, which usually lies in the hands of the production department". The sales department tends to accept as many jobs as possible while the production department attempts to meet the promised delivery dates. In order to avoid conflict of interest between these two departments, which may result in considerable delays, violated due dates and/or excessive use of highly expensive non-regular capacity, an integration of job selection and planning is needed (Guerrero and Kern, 1998; Herbots et al., 2007; Zijm, 2000).

Over the past decade, order acceptance has gained increasing attention both from academia as well as from practitioners communities. As clearly described by Rom and Slotnick (2009), "this decision is intricate because it should strike a balance between the revenue obtained from an accepted order on the one hand, and the (opportunity) costs of capacity as well as potential tardiness penalties on the other hand". In this chapter, we study a generalization of the order acceptance and sequencing problem. Order acceptance refers to the selection decision an over-demanded company has to make, while sequencing determines the order in which accepted jobs are executed. More specifically, the focus of this chapter is on the order acceptance and sequencing decisions of an organization that has a pool consisting of firm-planned orders as well as potential orders to choose from, while orders have known processing times, delivery dates and revenues. The capacity available for processing the accepted orders is limited. In addition, the urgency of individual orders may be emphasized by the importance of the client and/or the contract details.

We examine a generalization of the order acceptance and planning problem on a specialized scarce resource, which is represented as a single machine and which constitutes the bottleneck of the manufacturing environment. We propose two linear formulations and devise two exact branch-and-bound algorithms to solve the problem of deciding which potential orders to retain and which to reject for profit maximization, and we

determine the processing order of the accepted jobs. By means of computational experiments on a number of benchmark data sets, we solve instances with up to 50 jobs.

The problem studied can be seen as a generalization of two existing problems. The first problem is the order acceptance and sequencing problem with weighted-tardiness penalties studied by Slotnick and Morton (2007). Our problem reduces to the latter one when the pool of firm-planned orders is empty and all jobs are eligible for rejection. The second problem is the total-weighted-tardiness scheduling problem studied e.g. by Potts and Van Wassenhove (1985). It corresponds with the problem studied in this chapter when all orders are firm-planned orders.

The remainder of this chapter is structured as follows. First, we survey the existing literature in Section 3.2. In Section 3.3, we provide a formal description of the problem that we wish to solve. Section 3.4 contains the proof of the non-approximability result and the two linear formulations. Section 3.5 is devoted to the development of two branch-and-bound algorithms. We comment the results of the computational experiments in Section 3.6 and conclude in Section 3.7.

3.2 Literature review

Excellent literature surveys on the topic of order acceptance and scheduling are provided by Guerrero and Kern (1998), Keskinocak and Tayur (2004), Rom and Slotnick (2009) and Roundy et al. (2005). The objective of this section is therefore not to provide an exhaustive listing of the existing literature, but rather to survey the different perspectives that have been developed on the topic by different researchers, with a particular focus on the work in single-machine environments. Although the total-weighted-tardiness problem is strongly NP-hard (Lawler, 1977; Lenstra et al., 1977), Potts and Van Wassenhove (1985) have developed a branch-and-bound algorithm that efficiently solves problems with up to 50 jobs; recently, Pan and Shi (2007) has derived a new bound which allows to solve instances with up to 100 jobs and Tanaka et al. (2009) has presented a Successive Sublimation Dynamic Programming method able to solve instances with up to 300 jobs. This review will therefore only discuss the most closely related articles on the subject of order acceptance and scheduling. First, we discuss the order acceptance problem with weighted-tardiness penalties; subsequently, we briefly consider alternative objective functions and on-line decision making.

Slotnick and Morton study the single-machine job selection and sequencing problem

with deterministic job processing times and job rewards; their objective is to maximize the rewards in case of lateness (Slotnick and Morton, 1996) and tardiness (Slotnick and Morton, 2007) penalties. A pseudo-polynomial-time algorithm to solve the former problem was developed by Ghosh (1997). The problem was extended in (Lewis and Slotnick, 2002) to multiple periods for the case where rejecting a job will result in the loss of all future jobs from that customer. An exact approach for solving the order acceptance problem with weighted-tardiness penalties was developed in (Slotnick and Morton, 2007). Since the proposed algorithm can only deal with very moderately sized problem instances (at most ten jobs), suboptimal algorithms were also presented. Other heuristics include genetic algorithms (Rom and Slotnick, 2009) and greedy algorithms (Alidaee et al., 2001). Yang and Geunes (2007) consider an extension of the problem where job processing times are reducible at a cost and every job has a release time. In their paper, Yang and Geunes develop an optimal algorithm for maximizing schedule profit for a given sequence of jobs, along with heuristics to solve the entire problem. Sengupta (2003) studies a similar problem with *maximum* lateness and tardiness objectives, and proposes pseudo-polynomial time algorithms and approximations schemes. Both a mixed-integer programming (MIP) formulation as well as a branch-and-bound algorithm for solving the order acceptance problem with weighted-tardiness penalties are mentioned in the very brief article of Yugma (2005); he reports (without details) results "in reasonable time" via MIP for up to 15 jobs, and up to 30 using branch-and-bound. Finally, Bilginturk et al. (2007) look into a generalization of the order acceptance problem with weighted-tardiness penalties that includes release times, deadlines and sequence-dependent setup times; they conclude that a MIP-solution is not attainable for problem sizes exceeding ten jobs and resort to simulated annealing.

Other objective functions for the selection and sequencing problem have been considered in literature. Engels et al. (2003) seek to minimize the sum of the weighted completion times of the scheduled jobs and the total rejection penalty of the rejected jobs. A related objective function is used in (Lu et al., 2008) for the unbounded parallel-batch-machine scheduling problem with release dates. A parallel batch machine can process a number of jobs simultaneously, so that the makespan is the same for all jobs in a batch. A different but related problem is the job-interval selection problem (JISP) (Spieksma, 1999), where a job is determined by a set of intervals. In (Chuzhoy et al., 2006), some special cases of the JISP are considered. The paper develops algorithms that aim to maximize the number of jobs scheduled between their release

dates and deadlines. Another objective function was examined by Gupta et al. (1992), who develop an efficient polynomial-time dynamic-programming method that solves the project selection and sequencing problem (a fixed number of projects is selected from a set) while maximizing the net present value of the total return. De et al. (1991) study the sequencing problem and minimize the weighted number of tardy jobs. In (De et al., 1991), a project-dependent cost is charged when starting the execution of a project, leading to a selection problem. It is also assumed that a revenue is collected at the completion of a project. The goal is to maximize the expected rewards of a selection of jobs with random processing times and random deadlines. Meng-Gérard et al. (2009) study a related problem with unit durations inspired by a practical case of a satellite launcher, and describe polynomial-time dynamic-programming recursions for some special cases.

The on-line problem, in which projects arrive dynamically over time and need to be selected or rejected upon arrival, has been studied by several authors for a broad range of objective functions. For dynamic arrivals with a single resource constraint, Kleywegt and Papastavrou (2001) study a dynamic and stochastic knapsack problem, where the size of the knapsack represents the available resource quantity. Each arrival demands some amount of the resource, and a reward (unknown prior to the job arrival) is received upon acceptance. They provide an optimal acceptance policy that maximizes expected profits. For the batch process industry, Ivănescu et al. (2006) develop policies that focus on delivery reliability while maintaining high utilization rates. Epstein et al. (2002) consider a single-machine on-line job selection and scheduling problem with job-dependent rejection penalties. Their algorithm aims to minimize the total completion time of accepted jobs plus job rejection penalties.

3.3 Problem statement

A set of jobs $N = \{1, 2, \ldots, n\}$ with durations p_i ($i \in N$) is to be scheduled on a single machine; all jobs are available for processing at the beginning of the planning period. Each job i has a due date d_i and a revenue Q_i; the weight w_i represents a penalty per unit-time delay beyond d_i in the delivery to the customer. The pool N of jobs consists of two disjoint subsets F and $\bar{F}$ ($N = F \cup \bar{F}$ and $F \cap \bar{F} = \emptyset$), where F comprises the firm-planned orders and its complement $\bar{F}$ contains the "optional" jobs (the ones that can still be rejected). Our objective is to maximize total net profit, that is, the sum

of the revenues of the selected jobs minus applicable total-weighted-tardiness penalties incurred for those jobs.

There are two decisions to be made: which jobs in $\bar{F}$ to accept, and in which order to process the selected subset. We call M the set of jobs selected for processing, so $F \subseteq M \subseteq N$. If we let $m = |M|$ (m is the cardinality of M), the sequencing decisions can be represented by a bijection $\pi : \{1, 2, \ldots, m\} \mapsto M$, where $\pi(t)$ is the job in position t in the sequence. Each such bijection is in one-to-one correspondence with a total order on set M. The objective is thus

$$\max_{M,\pi} \quad \sum_{t=1}^{m} Q_{\pi(t)} - w_{\pi(t)}(C_{\pi(t)} - d_{\pi(t)})^+, \tag{3.1}$$

where C_i is the completion time of job i, i.e. $C_i = \sum_{t=1}^{\pi^{-1}(i)} p_{\pi(t)}$, and $s^+ = \max\{s, 0\}$. In the remainder of this chapter, we refer to (3.1) as either the objective function or the problem formulation; there will be little danger of confusion.

When $N = F$, all jobs are mandatory and the problem reduces to the single-machine total-weighted-tardiness scheduling problem $1||\sum w_i T_i$ (Pan and Shi, 2007; Potts and Van Wassenhove, 1985; Tanaka et al., 2009). When $N = \bar{F}$, on the other hand, the problem is equivalent to the order acceptance problem with weighted-tardiness penalties discussed in (Rom and Slotnick, 2009; Slotnick and Morton, 2007), which is akin to a number of other recently examined optimization problems, as discussed in the literature review. This problem contains the scheduling problem; it is therefore strongly NP-hard. The branch-and-bound procedure in (Slotnick and Morton, 2007) only solves relatively small problem instances (at most ten jobs) and is primarily used as a benchmark for evaluating the performance of heuristics.

We mention that although our problem generalizes the order acceptance problem with weighted-tardiness penalties problem ($F = \emptyset$), the opposite also holds. In fact, given an instance of problem (3.1) we simply assign sufficiently large revenues to jobs in F to make them "non-removable" (see page 63 for the definition).

3.4 Non-approximability and linear formulations

In this section, we first show that it is unlikely that a constant-factor approximation algorithm can be developed for solving the problem (3.1); subsequently, we present two mixed-integer linear formulations of (3.1), which can be solved using a MIP-solver.

3.4.1 Non-approximability

The next result shows that (3.1) is difficult to solve even approximately. We prove this non-approximability result by showing that any polynomial-time constant-factor approximation algorithm for solving (3.1) can be used to solve the following variant of Partition (see also page 18):

INSTANCE: A finite set $A = \{1, 2, \ldots, 2q\}$ (where q is an integer greater than 0) with size $s(i) \in \mathbb{Z}^+$ for each $i \in A$, and $K = \frac{1}{2}\sum_{i \in A} s(i)$.
QUESTION: Does there exist a subset $A' \subset A$ with $|A'| = q$ and $\sum_{i \in A'} s(i) = K$?

Proposition 3.1. *Unless $P = NP$, there is no polynomial-time algorithm that guarantees a constant-factor approximation for solving the problem (3.1).*

Proof. Suppose that there is a polynomial-time constant-factor approximation algorithm for solving the problem (3.1). We build from every instance of Partition an instance of (3.1) in polynomial time and we prove that by applying the approximation algorithm to this instance of (3.1), if, on the one hand, the objective value is equal to 0 then the instance of Partition is a YES instance. If the value is not zero, on the other hand, we conclude that the instance of Partition is a NO instance.

For a given arbitrary instance of Partition, consider the following polynomial reduction to an instance of (3.1) with $n = 2q + 2$ jobs. The set of firm-planned orders $F = \{2q + 1, 2q + 2\}$ and we let $\delta = 2K + 1$. The properties of each job are the following: for job $i = 1, \ldots, 2q$ we have a revenue $Q_i = \delta s(i)$, the processing time $p_i = \delta + s(i)$, the due date $d_i = q\delta + K + 2$ and the weight $w_i = s(i)(\delta + 1)$. For the last two jobs, we have $Q_{2q+1} = Q_{2q+2} = \delta$, $p_{2q+1} = p_{2q+2} = 1$, $d_{2q+1} = d_{2q+2} = 1$ and $w_{2q+1} = w_{2q+2} = \delta(K + 2)$.

If the instance of Partition is a YES instance then an appropriate set A' exists. We consider the solution of our problem that selects the jobs in A' (jobs built from the elements in A') plus those in F and executes them as follows. Job $2q + 1$ is first, job $2q + 2$ second and the jobs in A' in any order thereafter. For this solution, the only job delivered after its due date is job $2q + 2$. The total revenue obtained from selected jobs is $K\delta$ for jobs built from the elements in A' plus 2δ received for job $2q + 1$ and job $2q + 2$. The latter job, however, is tardy by one time unit, for which we incur a cost $\delta(K + 2)$. Together, the objective value of this solution is $K\delta + 2\delta - \delta(K + 2) = 0$, which can be seen to be the highest value possible for any feasible solution. Therefore,

a constant-factor approximation algorithm will provide an optimal solution, from which we can easily infer the elements of A'.

Conversely, if a constant-factor approximation algorithm leads to an objective value different from 0, we conclude that the instance of Partition is a NO instance. □

Proposition 3.1 implies that while heuristics can be devised to produce solutions to large instances of our problem, there is no guarantee that they will always attain at least a specific proportion of the optimal objective value.

3.4.2 Linear formulations

We present an assignment formulation and a time-indexed formulation of our problem, which can be solved using a MIP-solver.

Assignment formulation

This formulation uses a binary decision variable $y_i \in \{0,1\}$ $(i \in N)$, which takes the value 1 if job i is accepted and 0 otherwise. Notice that each job in F must be accepted. This constraint translates into:

$$y_i = 1, \quad \forall i \in F. \tag{3.2}$$

A second set of binary variables $x_{it} \in \{0,1\}$ $(i \in N,\, t \in \{1,\ldots,n\})$ is used to identify the position of the accepted jobs in the sequence of execution. The variable x_{it} is equal to 1 if job i is accepted and is the t^{th} job processed, and to 0 otherwise. The following set of equality constraints is imposed, stating that an accepted job is scheduled at exactly one position, while a rejected job is not put into any position:

$$y_i = \sum_{t=1}^{n} x_{it}, \quad i = 1,\ldots,n. \tag{3.3}$$

The set of constraints (3.3) allows the integrality constraints on the variable y_i to be replaced by $0 \leq y_i \leq 1$, $i = 1,\ldots,n$. The capacity constraints entail that a given position can be attributed to at most one job; this is enforced by adding the following set of constraints.

$$\sum_{i=1}^{n} x_{it} \leq 1, \quad t = 1,\ldots,n. \tag{3.4}$$

To linearize the objective function (3.1), we introduce the binary variable $z_{ji} \in \{0,1\}$ $(i,j \in N,\ i \neq j)$, which is equal to 1 if both jobs i and j are accepted and job j is executed before job i, otherwise z_{ji} takes the value 0. Since both jobs i and j must be accepted when $z_{ji} = 1$, we have

$$z_{ji} \leq y_i, \quad z_{ji} \leq y_j, \quad i,j = 1,\ldots,n,\ i \neq j. \tag{3.5}$$

We add the following set of constraints to enforce the fact that if job i is accepted and job j is processed before job i (meaning that job j is also accepted) then $z_{ji} = 1$:

$$\sum_{q<t} x_{jq} + \sum_{q \geq t} x_{iq} \leq 1 + z_{ji}, \quad i,j,t = 1,\ldots,n,\ i \neq j,\ t \neq 1. \tag{3.6}$$

To complete our formulation, we need a real variable $T_i \geq 0$ representing the tardiness of job $i \in N$, satisfying

$$T_i \geq \sum_{j=1}^{n} p_j z_{ji} + p_i y_i - d_i, \quad i = 1,\ldots,n. \tag{3.7}$$

The objective is

$$\text{maximize} \qquad \sum_{i=1}^{n} \left(Q_i y_i - w_i T_i\right). \tag{3.8}$$

We refer to the mixed-integer linear formulation given by the objective function (3.8) and the constraints (3.2)–(3.7) as ASF. The following result shows that ASF is equivalent to the problem as stated in (3.1).

Proposition 3.2. *From any solution to ASF, we can infer a solution to the problem (3.1) with the same objective value and vice versa.*

Proof. $\Rightarrow$) Suppose that there is a solution y_i, x_{it}, z_{ji} and T_i to ASF. Consider $M^* = \{i \in N : y_i = 1\}$; it holds that $F \subseteq M^*$. Each job $i \in M^*$ has a unique value t that represents its position, namely t for which $x_{it} = 1$. The function π^* that orders the jobs in M^* according to increasing position is a bijection. The pair (M^*, π^*) forms a feasible solution to (3.1). Moreover, this solution has the same objective value as the initial solution to ASF because it selects the same jobs and follows the same execution sequence.

$\Leftarrow$) On the other hand, suppose that we have a solution (M, π) to (3.1), then take $x_{it} = 1$ if $i \in M$ and $\pi(t) = i$, otherwise set $x_{it} = 0$. We can easily infer y_i, z_{ji} and T_i. This solution is feasible and has the same objective value as (M, π). □

The formulation ASF can be strengthened by adding the following inequalities:

$$\text{(Cuts)} \qquad \sum_{i=1}^{n} x_{i,t+1} \leq \sum_{i=1}^{n} x_{it}, \quad t = 1, \ldots, n-1.$$

These inequalities remove empty positions between two consecutive jobs. We refer to these inequalities as "cuts", although a number of integer solutions to ASF are also eliminated.

The following result states that we can relax the integrality constraints on the variables z_{ji} without harm. Let ASF$'$ be the formulation ASF in which the domain $\{0,1\}$ of the variables z_{ji} is replaced by $[0,1]$.

Proposition 3.3. *In any optimal solution to ASF$'$ with non-zero processing times and non-zero weights, each $z_{ji} \in \{0,1\}$.*

Proof. Consider a given optimal solution y_i, x_{it}, z_{ji} and T_i to ASF$'$ and suppose that there exists a variable $z_{j^*i^*}$ with $0 < z_{j^*i^*} < 1$. Due to constraints (3.5), both job i^* and job j^* are accepted ($y_{i^*} = y_{j^*} = 1$) and constraints (3.3) imply that they are executed at different positions. The constraints (3.6) and inequalities $0 < z_{j^*i^*} < 1$ imply that job i^* is executed before job j^* and thus $z_{i^*j^*} = 1$. Finally, constraints (3.7) allow to see that the above solution, which executes job i^* before job j^* and has $0 < z_{j^*i^*} < 1$, is dominated by the solution obtained by setting $z_{j^*i^*} = 0$ and therefore is not optimal. □

Time-indexed formulation

This formulation is based on a discretization of the time horizon $[0, H]$ into H one unit-time buckets, where $H = \sum_{i \in N} p_i$ and the bucket t is the time interval $[t-1, t]$. It has been applied to various single-machine scheduling problems; in particular with the weighted-completion-time objective in (Van Den Akker et al., 2000), and to the total-weighted-tardiness problem in (Bigras et al., 2008; Tanaka et al., 2009).

In our setting, the decision variables are the binary quantities $x_{jt} \in \{0,1\}$ with $j = 1, \ldots, n$ and $t = 1, \ldots, H - p_j + 1$, which equal 1 if job j is selected and its execution starts in bucket t, and 0 otherwise. We let $c_{jt} = \max\{t + p_j - d_j - 1; 0\}$, the tardiness of job j associated with the decision x_{jt}. The time-indexed formulation

is given by:

$$\text{(TIF)} \qquad \max \quad \sum_{j=1}^{n} \sum_{t=1}^{H-p_j+1} x_{jt} \left(Q_j - w_j c_{jt}\right) \tag{3.9}$$

$$\text{s.t.} \quad \sum_{t=1}^{H-p_j+1} x_{jt} = 1 \qquad \forall j \in F, \tag{3.10}$$

$$\sum_{t=1}^{H-p_j+1} x_{jt} \leq 1 \qquad \forall j \in \bar{F}, \tag{3.11}$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1 \qquad t = 1, \ldots, H, \tag{3.12}$$

$$x_{jt} \in \{0, 1\} \qquad j = 1, \ldots, n,\ t = 1, \ldots, H - p_j + 1. \tag{3.13}$$

The set of constraints (3.10) specifies that each job $j \in F$ is processed exactly once while the constraints (3.11) state that each job $j \in \bar{F}$ should be executed at most once. The set of constraints (3.12) reflects the capacity constraints: it avoids the processing of more than one job at the same time.

3.5 Implicit-enumeration algorithms

In this section, we present two branch-and-bound (B&B) algorithms for solving the problem (3.1). These algorithms are inspired by the work of Slotnick and Morton (1996, 2007). The first B&B algorithm is hierarchical, in that it performs selection and scheduling separately. At each node of the branching tree, the B&B algorithm developed by Potts and Van Wassenhove (1985) is used to schedule a set of selected jobs. Notice that the recent Successive Sublimation Dynamic Programming (SSDP) algorithm proposed by Tanaka et al. (2009) represents an attractive alternative. The second B&B algorithm performs both selection and scheduling simultaneously. In what follows, these two B&B algorithms are called *two-phase* and *direct*, respectively.

In the related literature, a two-phase algorithm is used only for the case of *lateness* penalties as studied by Slotnick and Morton (1996) (where the scheduling problem is easy; in fact after the selection process, the weighted shortest processing time (WSPT) algorithm (Slotnick and Morton, 1996) is used to optimally schedule the accepted jobs), although our algorithm gradually adds jobs while jobs are stepwise *removed* in (Slotnick and Morton, 1996) while going down the search tree. Slotnick and Morton (2007) do

refer to a *pilot study* in which they "consider all permutations of a given sequence", but this approach turns out to be impractical for all but a very small number of jobs. In our implementations, the two-phase approach will prove to be at least as efficient as the direct one.

A direct algorithm for solving the special case $F = \emptyset$ is proposed by Slotnick and Morton (2007) in combination with an assignment-based upper bound. They do not specify how the assignment problems are solved, however (reference is made only to an "assignment algorithm"), and they also implement an approximate solution to the assignment problems (Vogel's approximation). In our implementations, we examine also a lateness-based bound and an LP relaxation based bound next to an assignment bound, and the latter is computed using a very efficient cost-scaling algorithm. Additionally, our direct algorithm is augmented with a number of dominance rules borrowed from the scheduling literature.

Optimal solutions are reported by Slotnick and Morton (2007) only for instances with $n = 10$; the instances solved are unavailable from the authors because they are generated "on the fly". Additionally, it is underlined in (Slotnick and Morton, 2007) that finding optimal solutions to 20-job problems is not "practical". In our computational results reported in Section 3.6, we produce optimal solutions for instances with up to 50 jobs. While the average reported computation time for solving ten-job instances to optimality is 6154 seconds in (Slotnick and Morton, 2007), we solve similar-size instances with $F = \emptyset$ in less than one second. These differences seem large enough to justify the assertion that the computational improvements are not only due to a different computer infrastructure.

We will use the same problem instance to illustrate the working of the two algorithms. This instance has four jobs, one of which (job 2) has already been accepted, so $F = \{2\}$. The properties of each job are given in Table 3.1. In the rest of this chapter, we refer to this instance as Example 1. An optimal solution to Example 1 selects $M = \{2, 3, 4\}$ and executes job 4 first, followed by job 2 and finally job 3, yielding an optimal objective value of 10. Note that a solution to the corresponding instance with $F = \emptyset$ will select the jobs 1, 3, 4 and achieve the value of 14 by executing job 4 first, then job 1 and subsequently job 3.

3.5.1 Two-phase B&B algorithm

In this section, we describe in detail each step of the two-phase B&B algorithm.

Table 3.1: *Job properties for Example* 1.

Job	1	2	3	4
Q_i	5	4	5	7
w_i	3	2	1	4
p_i	2	3	2	2
d_i	4	4	3	2

Removable set

We wish to distinguish a set $\bar{R}$ of jobs for which we are certain that they are part of each optimal solution, we call these jobs *non-removable*. By symbol R, we denote the set of *removable* jobs: $R = N \setminus \bar{R}$. Slotnick and Morton (2007) propose the following procedure to identify R for the order acceptance problem with $F = \emptyset$: given an optimal sequence of all the jobs and given a job j, if removing job j from the sequence decreases the net profit then $j \in \bar{R}$, otherwise $j \in R$. The next result generalizes this procedure to our problem.

Proposition 3.4. *For a given instance of problem (3.1) and a removable set A for the corresponding instance with $F = \emptyset$, the set R of removable jobs of our problem is given by $R = A \setminus F$.*

Proof. This follows from the reasoning given by Slotnick and Morton (2007) for the total-weighted-lateness penalties and the fact that jobs in F must be selected. □

Let us assume from now on that the jobs in R are labelled $J_1,\ldots,J_{|R|}$, where $|R|$ is the cardinality of R and $J_i < J_{i+1}$ for $i = 1,\ldots,|R|-1$. For Example 1, we obtain $R = \{1,3\}$ and $\bar{R} = \{2,4\}$.

Branching strategy

Figure 3.1 depicts the branching tree explored by the two-phase B&B algorithm for solving Example 1. The top node (node 1) in Figure 3.1 corresponds to the decision where all the jobs in $\bar{R} = \{2,4\}$ are accepted and the other ones rejected. Solving the total-weighted-tardiness scheduling problem with the selected jobs (the optimal solution schedules job 4 first and job 2 second), leads to a tardiness cost of 2. This value is subtracted from the revenue of $Q_2 + Q_4 = 11$ collected, resulting in a lower

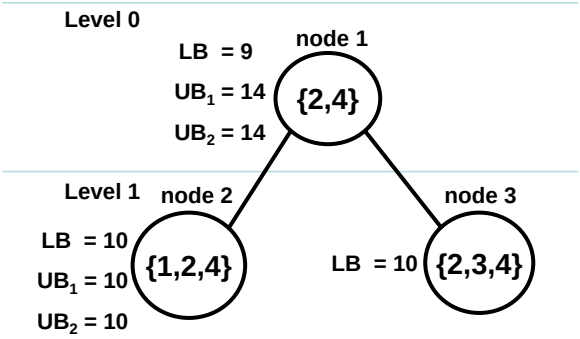


Figure 3.1: *Illustration of the two-phase B&B algorithm for solving Example* 1*. Optimal solutions are obtained at node* 2 *and node* 3*. Inside each node* u*, the set* M_u *of selected jobs is described. The assignment bound and lateness bound are both 13 at the root node.*

bound (LB) of 9 at node 1. Three upper-bound (UB) procedures are implemented. The first procedure obtains an upper bound UB_1 by solving an assignment problem, the second upper bound UB_2 is the optimal objective value of an order acceptance problem with weighted *lateness* penalties while the last upper bound is the objective value of the LP relaxation of the time-indexed formulation (TIF). More details on the upper bounds are provided later on page 65.

Starting from level 0, each node u at each subsequent level of the search tree represents a new set of selected jobs obtained by adding an extra job from R to the set of jobs selected at the parent node. For example, at node 2 in Figure 3.1 where three jobs are selected, job 1 is added to the set of jobs $\{2, 4\}$ selected at the parent node (node 1). Consequently, at level k a node corresponds to a set of $|\bar{R}| + k$ jobs selected. To avoid repetition (having two nodes in the branching tree with the same set of selected jobs), a job J_j in R is added at a child node if and only if $J_j > J_u$, with J_u the job added at the parent node. To illustrate this in Figure 3.1, consider node 3 where job 3 is added. One might expect the tree to include a child node of node 3 with selection $\{1, 2, 3, 4\}$, but because $1 < 3$ this child node is not considered. Notice that the use of the above criterion may yield a strongly unbalanced branching tree and that we do not compute UBs for nodes that are certainly leaves of the tree. In Figure 3.1, node 3 is a leaf node

because it selects job 3, the job with the highest index in R. Node 2 is a leaf node because the LB of 10 is already equal to the UBs, and the potential child node with selection $\{1, 2, 3, 4\}$ is therefore not entered.

Node selection

We explore the branching tree in a *best-first search* (BFS) manner. During the search, a list of unfathomed nodes is kept in non-increasing order of their UB. The first node in the list is the next node to investigate. When a new node is created, the list is updated by inserting that node at the appropriate position such that the non-increasing-UB order is preserved.

Scheduling algorithm and lower bound

In each node u, we have a set M_u of selected jobs, which contains the firm-planned orders, that is $F \subseteq M_u$. Therefore, each set M_u together with any execution sequence for the jobs in M_u constitutes a feasible solution to our problem. We use the B&B algorithm developed by Potts and Van Wassenhove (1985) to optimally schedule the jobs in M_u to minimize the total-weighted tardiness. The objective value of the solution obtained at node u is the total revenue $\sum_{i \in M_u} Q_i$ from the accepted jobs minus the tardiness costs provided by Potts and Van Wassenhove's algorithm. This value is a LB of the optimal objective value of our problem.

Upper bound

At node u, an UB is computed only if node u is not a leaf node. We have implemented three UBs. The first one is obtained by solving an assignment problem; this bound generalizes a similar bound used by Slotnick and Morton (2007). The second bound is the optimal objective value of an order acceptance and scheduling problem with weighted-*lateness* penalties and the last bound is the objective value of the LP relaxation of TIF.

Assignment bound In the computation of this UB, we relax the implicit non-preemption constraint on the jobs. Each job is divided into joblets with a duration of one time unit, and these joblets are then scheduled independently of each other. The resulting scheduling problem is solved by an assignment problem that assigns joblets to unit-duration time buckets. At node u, we include dummy joblets and

dummy positions to allow for the rejection of joblets stemming from jobs not in M_u.

Consider a node u at level ℓ of the search tree and let J_z be the last job added; it holds that $z \geq \ell$. The set $S_u = N \setminus (\{J_1, \ldots, J_z\} \setminus M_u)$ contains the jobs that remain eligible for selection in the descendant nodes of node u (and some of them have already been selected with certainty); let $K = \sum_{i \in S_u} p_i$. We associate with the allocation of any joblet of a job in M_u to a time instant larger than K a very large negative cost in the assignment problem, while the assignment of any joblet of jobs not in M_u to a time instant larger than K and the assignment of any dummy joblet corresponds with zero cost. A non-dummy joblet assigned to a time bucket with index lower than or equal to K corresponds with acceptance of that joblet; the contribution of such a combination to the objective function is determined based on the per-joblet reward, weight and due date. The per-joblet reward and weight are obtained by dividing the original job's reward and weight by the job's processing time; the due date of a joblet is a function of the position of the joblet (within the job) and the job's due date (Rom and Slotnick, 2009; Slotnick and Morton, 2007). The assignment problem is solved using a cost-scaling algorithm (Ahuja et al., 1993; Goldberg and Kennedy, 1995) and the optimal value constitutes an UB.

Lateness bound In an arbitrary node u of the search tree, the total net profit is

$$\max_{M_u \subseteq M \subseteq N_u, \pi} \quad \sum_{t=1}^{|M|} Q_{\pi(t)} - w_{\pi(t)}(C_{\pi(t)} - d_{\pi(t)})^+,$$

where $N_u = N \setminus \{j_i \in R : J_i < J_u \text{ and } J_i \notin M_u\}$, with J_u the last job accepted. We replace the tardiness penalties by lateness penalties; jobs can then also receive a reward for being early. We obtain the following problem:

$$\max_{M_u \subseteq M \subseteq N_u, \pi} \quad \sum_{t=1}^{|M|} Q_{\pi(t)} - w_{\pi(t)}(C_{\pi(t)} - d_{\pi(t)}).$$

This latter problem is a generalization of the job selection and sequencing problem with weighted-lateness penalties (Ghosh, 1997; Slotnick and Morton, 1996). The generalization resides in the fact that a specific subset of jobs (namely M_u) is necessarily selected. The pseudo-polynomial-time dynamic-programming algorithm proposed by Ghosh (1997) is easily modified to solve this problem. The optimal objective value provides an UB for our problem.

LP relaxation bound This bound is the optimal value of the LP relaxation of TIF where for each job $j \in M_u \setminus F$, the sense of the corresponding constraint (3.11) is changed into equality.

It turns out, from empirical observations, that the assignment bound is virtually always at least as strong as the lateness bound, but also computationally more expensive. Similarly, the LP relaxation bound is at least as strong as the assignment bound, but takes more time. In the working paper (Talla Nobibon et al., 2009), we investigate two combinations of the first two UBs. The first combination computes an assignment bound at the root node, while at all subsequent nodes a lateness bound is computed first. If the lateness value is less than or equal to α times the UB of its parent node ($\alpha \in]0, 1[$) then an assignment bound is computed, in an attempt to prune the node. The second combination is based on similar principles but α varies during the search, by choosing its value as the number of jobs accepted at the parent node divided by the number of jobs accepted at the child node. The use of these two combinations in the implementation of the B&B, however, did not improve the CPU time achieved by the initial upper bounds, and we will not report the computational results of these implementations in Section 3.6.

3.5.2 Direct B&B algorithm

The main steps of the direct B&B algorithm are described in this section.

Branching strategy

Figure 3.2 depicts the search tree explored by the direct B&B algorithm when applied to Example 1. Optimal solutions are found in node 9 and node 10. The root node (node 1) represents the situation where no job has been accepted yet. The LB of 4 corresponds with the value achieved by the solution that selects all the jobs in F and executes them in increasing order of job index (in this case, it is only job 2). The UB reported in Figure 3.2 is an assignment bound; for more details, see page 69. At level 1 of the search tree, n new nodes are created representing n (possibly partial) solutions, each generated by selecting one job and scheduling it at the first position. In Figure 3.2, these are node 2, which selects job 1 (whence the entry '1' inside the node), node 3, which selects job 2, and similarly up to node 5. Given a parent node at level $k-1$ ($k \geq 1$), we create $n-k+1$ child nodes each with k jobs selected with

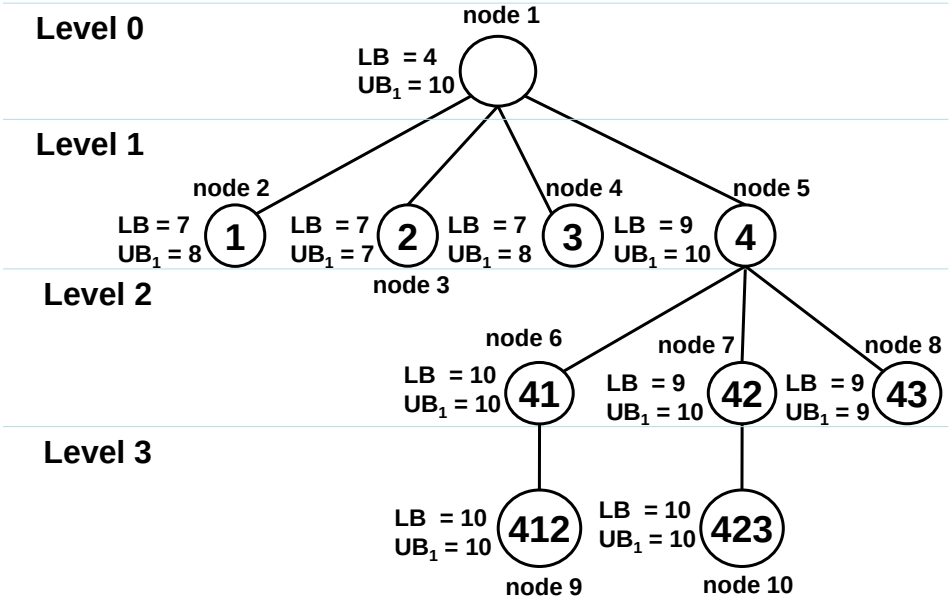


Figure 3.2: *Illustration of the direct B&B algorithm with lateness bound UB_2 for solving Example* 1. *Inside each node, the corresponding (possibly partial) solution is represented as a sequence of job indices.*

known position: the job added at level k is the k^{th} job processed. Node 5, for instance, has three child nodes (nodes 6, 7 and 8), each of which selects one additional job to be executed immediately after job 4. At node 6, for example, job 1 is selected and the text '41' in the node means that job 4 comes first, followed by job 1. In the same way as in the two-phase algorithm, the branching tree is explored in a BFS manner.

Lower bound

At a given node u of the branching tree, let $M_u \subseteq N$ be the set of selected jobs and V_u the return (net profit) obtained from the selection of jobs in M_u and the sequencing decisions inherent in node u. Two cases can occur. Either

(1) $F \subseteq M_u$, in which case V_u is a LB, or

(2) $F \nsubseteq M_u$, and then V_u is not a LB at node u.

In the latter case, our LB is the objective value of the solution that selects the jobs in $F \cup M_u$ and executes them following the sequence corresponding with node u for M_u, followed by the jobs in $F \setminus M_u$ in increasing order of job index.

Upper bound

Both the assignment, the lateness bound as well as the LP relaxation bound presented on page 65 for the two-phase algorithm only need minor modifications to be applicable in the direct algorithm. We denote by $\bar{M}_u$ the complement $N \setminus M_u$. At node u, the contribution to the objective function of the jobs in M_u is known exactly; therefore, only the contribution of potential jobs in $\bar{M}_u$ needs to be bounded. Because the $|M_u|$ selected jobs are scheduled at the first $|M_u|$ positions, the due date of each job in $\bar{M}_u$ is updated accordingly by subtracting $\sum_{i \in M_u} p_i$, we refer to these new values as $\bar{d}_i$ for $i \in \bar{M}_u$.

Assignment bound This UB is similar to the one presented in (Slotnick and Morton, 2007), apart from the updated due dates. The sum of V_u and the optimal objective value of the assignment problem yields an UB.

Lateness bound At node u, the unscheduled jobs can achieve a net profit of

$$\max_{M \subseteq \bar{M}_u, \pi} \quad \sum_{t=1}^{|M|} Q_{\pi(t)} - w_{\pi(t)}(C_{\pi(t)} - \bar{d}_{\pi(t)})^+ \leq$$

$$\max_{M \subseteq \bar{M}_u, \pi} \quad \sum_{t \in M} Q_{\pi(t)} - w_{\pi(t)}(C_{\pi(t)} - \bar{d}_{\pi(t)}) = \Delta,$$

where $F \setminus M_u \subseteq M$. The value $\Delta + V_u$ is an UB.

LP relaxation bound At node u, the LP relaxation of the restricted TIF formulation including only jobs in $\bar{M}_u$ with updated due date is solved. The optimal value added to V_u constitute an upper bound.

In addition to the two UB-combinations mentioned for the two-phase B&B, two more combinations have been investigated in (Talla Nobibon et al., 2009) for the direct B&B algorithm. One computes the assignment bound at the root node, while a lateness bound is computed first at subsequent nodes. If the value of the LB at that node is less

than or equal to $\alpha \in]0,1[$ times the LB of its parent node then an assignment bound is also computed, in an attempt to prune the node. The second combination follows the same idea but puts α equal to the number of jobs accepted at the parent node divided by the number of jobs accepted at the current node. More details on these combinations and the resulting computational results can be found in (Talla Nobibon et al., 2009).

Dominance rules

Below, we outline some global and local dominance rules that are used as pruning devices for the direct B&B algorithm.

Global dominance rules The global dominance rules presented below generalize Emmons' rules (Bigras et al., 2008; Emmons, 1969; Rinnooy Kan et al., 1975). The goal of these rules is to identify for each job j a set B_j containing jobs that, if selected together with job j, can be processed before job j, and A_j, a set of jobs that, if selected together with job j, can be executed after job j. Let B_j^F be the set of jobs in F that must be processed before job j, and A_j^F the set of jobs in F that must be processed after job j. Both B_j^F and A_j^F are initially empty, and the rules below are applied iteratively.

Lemma 3.5. *If job i and job j are selected, then there is an optimal sequence in which job i is processed before job j, if one of the following conditions holds:*
Rule 1: $p_i \leq p_j$, $w_i \geq w_j$ *and* $d_i \leq \max\{d_j, p_j + \sum_{h \in B_j^F} p_h\}$.
Rule 2: $w_i \geq w_j$, $d_i \leq d_j$ *and* $d_j + p_j \geq \sum_{h \in N \setminus A_i^F} p_h$.
Rule 3: $d_j \geq \sum_{h \in N \setminus A_i^F} p_h$.

Proof. The proof of these three rules follows from the proof proposed by Rinnooy Kan et al. (1975). In fact, if job i and job j are accepted then the set of selected jobs contains $F \cup \{i, j\}$. For the scheduling problem associated with that set, Rule 1 corresponds to Corollary 1 in (Rinnooy Kan et al., 1975), Rule 2 corresponds to Corollary 2 and Rule 3 to Corollary 3. □

When $F = N$, these three rules are exactly Emmons' rules (Rinnooy Kan et al., 1975) for the total-weighted-tardiness scheduling problem; Lemma 3.5 proposes similar rules for the order acceptance problem. As for the scheduling problem,

these three rules are used to construct a predecessor graph. Unlike the scheduling problem, however, where each arc in the graph is unconditionally respected, here a precedence relation imposing the execution of job i before job j is respected only in the branches where both job i and job j are accepted.

Once a number of precedence-related activity pairs have been distinguished, additional pairs can be transitively inferred. Our implementation of this transitivity is slightly different from Rinnooy Kan et al. (1975) because, again, we need to select jobs as well as schedule them. We distinguish between jobs in F and those in $N \setminus F$ and proceed as follows. Whenever we identify a new dominant decision of the form 'job j precedes job k', we consider two possible cases:
Case 1: If job $j \in F$ then we ensure that job k and each successor of k is processed after job j and any of its predecessors.
Case 2: If job $k \in F$ then we enforce that job j and each predecessor of j is executed before job k and any of its successors.

Local dominance rules At each node of the branching tree, we use two local dominance rules as pruning devices. The following lemma presents the selection criterion. At node u, let J_u be the last job selected and scheduled at position $|M_u|$ and t_u the start time of the execution of job J_u.

Lemma 3.6. *At a given node u of the branching tree, consider $j \in \bar{M}_u \setminus F$. If $Q_j - w_j(t_u + p_{J_u} + p_j - d_j) \leq 0$ then the child node of u in which job j is scheduled at position $|M_u| + 1$ can be pruned without losing all optimal solutions.*

Proof. In the best case, selecting job j can lead to a net profit equal to the optimal net profit obtained when job j is not selected. □

The rule defined by Lemma 3.6 is an adaptation of the adjacent-job interchange rule. Consider a child node of node u where a job j is appended to the schedule. If it is possible to execute j before J_u and still accept (and execute) J_u thereafter, and if the profit obtained by putting j before J_u is larger than that obtained when J_u is executed before j, then that child node is not considered.

3.6 Computational experiments

All algorithms have been coded in C using Visual Studio C++ 2005; all the experiments were run on a Dell Optiplex $GX620$ personal computer with Pentium R processor with 2.8 GHz clock speed and 1.49 GB RAM, equipped with Windows XP. CPLEX 10.2 was used for solving the linear formulations. Below, we first provide some details on the generation of the data sets, followed by a discussion of the computational results.

3.6.1 Data generation

The algorithms are tested on randomly generated instances with n jobs, for $n = 10$, 20, 30, 40 and 50. For each job i, an integer processing time p_i and an integer weight w_i are drawn from the discrete uniform distribution on $[1, 10]$.

In the choice of the job parameters, we follow Potts and Van Wassenhove (1985) and consider multiple values for the relative range r of the due dates and for the average tardiness factor T. The values chosen for r are 0.3, 0.6 and 0.9; the same values apply for T. For given values of r and T and with $P = \sum_{i=1}^{n} p_i$, for each job i we draw an integer due date d_i from the uniform distribution with support the set of integers in the interval $[\max\{P(1 - T - \frac{r}{2}), p_i\}, \max\{P(1 - T + \frac{r}{2}) - 1, p_i\}]$. In this way, we avoid the creation of instances in which jobs might be processed last without being late, if all jobs are executed. Each job revenue Q_i is an observation of a lognormal distribution with an underlying normal distribution with mean 0 and standard deviation 1, rounded to the nearest integer. This reflects a situation in which job characteristics are fairly similar but where the potential revenue of a job may vary widely (Slotnick and Morton, 1996, 2007).

For every value of n and for each of the nine pairs of values of r and T, one instance is generated. These instances are subsequently modified by randomly choosing the firm-planned orders out of the n jobs, as follows. Two extreme cases considered are $|F| = 0$ and $|F| = n$; intermediate choices for $|F|$ are $0.2n$, $0.4n$, $0.6n$ and $0.8n$. For each of the latter values for $|F|$, two different sets F are set up, and we ensure that each smaller set F is embedded in the larger F of another instance. This means, for example, that for a given instance with a set F_1 of firm-planned orders with $|F_1| = 0.2n$, there exists an instance with a set of firm-planned orders F_2 with $|F_2| = 0.4n$ such that $F_1 \subset F_2$. For a given value of n, we have $9 \times (2 + 2 \times 4) = 90$ test instances, yielding $5 \times 90 = 450$ instances in total.

In order to study the effect of the size of the processing times on algorithmic performance, we have also generated a set of 90 instances with $n = 10$ jobs each, following the methodology described above but with larger processing times: the p_i are generated from the discrete uniform distribution on $[10, 100]$.

3.6.2 Computational results

In this section, computation time is referred to as *Time* and is expressed in seconds; *Nodes* is the number of nodes explored in the search tree of the considered algorithm. Each cell in the tables appearing in this section is (unless mentioned otherwise) the average of either nine values (corresponding to the cells with $|F|$ equal to $100\%n$ and $0\%n$) or 18 values (corresponding to the cells with $80\%n$, $60\%n$, $40\%n$ and $20\%n$). Some tables contain rows entitled *Unsolved*, which indicate the number of instances of each group that remained unsolved when the time limit was reached.

Size $n = 10$

Table 3.2 displays the results of the mixed-integer linear formulations for the data set with $n = 10$ and processing times between 1 and 10. The table shows that using the time-indexed formulation (TIF), CPLEX solves all instances within a fraction of a second (at most 0.08 seconds). Most solutions are obtained at the root node of the LP-based branching tree, which can also be seen by the entry '0' in the cells corresponding to 100%, 80%, 40% and 0% in the row entitled 'Nodes'. TIF dominates the other linear formulations: it reports both the lowest average CPU time and explores the lowest number of nodes. As for the other formulations, we observe that ASF$'$ dominates

Table 3.2: *Linear formulations for $n = 10$ with small processing times.*

Firm-planned orders		100%	80%	60%	40%	20%	0%
ASF+Cuts	Nodes	96680 (2)	120075	97140	48417	59467	51417
	Time	505.58	657.61	548.21	342.52	401.73	378.15
ASF$'$	Nodes	17870	17920	21137	14935	15256	19118
	Time	123.46	91.28	97.30	70.97	78.40	118.30
ASF$'$+Cuts	Nodes	30556	10058	16110	7339	6315	6989
	Time	194.94	99.89	107.67	61.06	52.42	54.39
TIF	Nodes	0	0	7	0	1	0
	Time	0.05	0.07	0.07	0.08	0.07	0.07

ASF+Cuts, which is the formulation ASF with the addition of cuts. On average, CPLEX solves ASF′ faster than ASF′+Cuts for instances with at least 60% of firm-planned orders, but the reverse is true when $|F| \leq 40\%$: the benefits of the cuts are more pronounced when the cardinality of F is small. With the addition of cuts to ASF (ASF+Cuts), there are two instances with 100% of firm-planned orders that CPLEX is not able to solve within the time limit of one hour, which is indicated by the number 2 between brackets after the number of nodes explored.

Table 3.3: *B&B algorithms for $n = 10$ with small processing times.*

Firm-planned orders			100%	80%	60%	40%	20%	0%
two-phase	assign.	Nodes	0	1	4	12	20	39
		Time	0.00	0.02	0.05	0.13	0.21	0.41
	lateness	Nodes	0	2	10	34	125	442
		Time	0.00	0.00	0.00	0.00	0.01	0.04
	LP-TIF	Nodes	0	1	4	10	17	26
		Time	0.03	0.04	0.04	0.05	0.07	0.12
direct	assign.	Nodes	61	33	31	38	45	84
		Time	0.23	0.16	0.21	0.32	0.39	0.59
	lateness	Nodes	58	56	82	138	267	579
		Time	0.00	0.00	0.01	0.02	0.05	0.13
	LP-TIF	Nodes	36	24	34	33	43	56
		Time	0.09	0.09	0.12	0.14	0.19	0.31

Table 3.3 reports on the B&B algorithms applied to the data set with $n = 10$ and small processing times, including the three UBs. The assignment bound is identified by *assign.*, the lateness bound by *lateness* while *LP-TIF* refers to the implementation using the LP relaxation bound. All settings lead to a solution in less than one second on average by each of the B&B algorithms, with a maximum of 0.59 seconds reported for the direct B&B with assignment bound on instances with $F = \emptyset$. Both B&B algorithms are faster when implemented with the lateness bound than when used with either the assignment bound or the LP relaxation bound – this will no longer be the case for larger values of n. Although both the assignment bound and LP relaxation bound are tighter than the lateness bound and reduce the average number of nodes investigated by the B&B algorithm, for $n = 10$ this does usually not lead to a lower running time. For $n = 10$ and small processing times, the two-phase algorithm with lateness bound is the fastest of the six B&B algorithms, and it is also faster on average than the linear

Table 3.4: *Linear formulations for $n = 10$ with large processing times.*

Firm-planned orders		100%	80%	60%	40%	20%	0%
ASF′	Unsolved	–	–	–	–	–	–
	Nodes	23875	20054	31021	36362	58677	48539
	Time	214.50	88.88	105.12	141.57	202.97	179.38
ASF′+Cuts	Unsolved	–	–	–	–	–	–
	Nodes	39134	9699	9192	11663	7064	13548
	Time	284.14	78.53	78.13	75.12	66.41	124.38
TIF	Unsolved	–	1	2	3	3	1
	Nodes	0	2585	1033	3202	1348	1395
	Time	7.08	228.04	54.03	38.06	15.74	27.38

formulation TIF. On the other hand, the B&B algorithm with the LP relaxation bound is usually faster than the same algorithm implemented with the assignment bound.

Next, we study the effect of the processing times on the algorithms. Table 3.4 contains the main results of the two linear formulations when processing times range from 10 to 100. In this table, we do not include the formulation ASF+Cuts since this setting is dominated by the formulation ASF′ (see Table 3.2). The number of variables of the formulation TIF is strongly dependent on the size of the processing times, and this obviously impacts the formulation's computational performance: the average CPU time rises from at most 0.08 seconds for small processing times to up to 228 seconds for large durations. Additionally, there are ten instances that CPLEX is unable to solve within the time limit of one hour, and an exploration of only the root node of the search tree is usually not sufficient anymore, except when $F = N$. The formulations ASF′ and ASF′+Cuts, on the other hand, solve all instances within a timespan of less than three times the average time reported for the instances with small job durations.

Table 3.5 displays the results of the B&B algorithms for $n = 10$ with large processing times. The algorithms with lateness bound are only little affected by the size of the processing times, while the assignment bound and the LP relaxation bound turn out to be very sensitive. This observation is intuitive: both the number of objects to be matched in the assignment instances and the size of TIF are proportional to the order of magnitude of the durations. The maximum CPU time is now some 40 (resp. 5) seconds for the two-phase B&B with assignment (resp. LP relaxation) bound and around 41 (resp. 17) seconds for the direct B&B with assignment (resp. LP relaxation) bound, compared to 0.41 (resp. 0.13) and 0.59 (resp. 0.31) seconds when $p_i \in [1, 10]$. The two-

Table 3.5: *B&B algorithms for $n = 10$ with large processing times.*

Firm-planned orders			100%	80%	60%	40%	20%	0%
two-phase	assign.	Nodes	0	1	3	9	25	53
		Time	0.00	1.54	3.93	8.13	18.79	40.59
	lateness	Nodes	0	2	10	44	178	722
		Time	0.00	0.01	0.01	0.02	0.03	0.08
	LP-TIF	Nodes	0	1	2	4	19	31
		Time	0.14	0.84	0.95	1.67	3.90	5.31
direct	assign.	Nodes	67	31	25	34	52	93
		Time	13.87	8.88	10.45	15.46	24.57	41.17
	lateness	Nodes	53	55	87	155	259	536
		Time	0.00	0.01	0.01	0.03	0.07	0.20
	LP-TIF	Nodes	24	30	64	75	43	79
		Time	8.32	10.20	12.42	15.71	11.94	17.27

phase B&B with lateness bound reports the smallest average CPU times in Table 3.5, and these times are also smaller than those for the linear formulations, making the two-phase B&B with lateness bound the most efficient exact algorithm for solving any ten-job instance.

The range of the job durations tested by Slotnick and Morton (2007) is unclear (they are "adjusted by a constant"). Although our results do not pertain to the same instances nor to the same computer as Slotnick and Morton, the differences in running times seem large enough to justify the assertion that the computational improvements are not only due to a different computer infrastructure: while the average reported computation time for solving ten-job instances to optimality is 6154 seconds in (Slotnick and Morton, 2007), we solve similar-size instances with $F = \emptyset$ in less than one second.

Size $n = 20$

Table 3.6 displays the results of the time-indexed formulation and the B&B algorithms for the instances with $n = 20$. Using TIF, CPLEX is unable to solve four instances (two for 80% and two for 60%) within the time limit of two hours, and solutions are no longer found at the root node (contrary to Table 3.2). The two-phase B&B solves all instances, regardless of the UB used. The best running times are obtained by the lateness bound when the proportion of firm-planned orders is greater than or equal to 40%, and by the assignment bound otherwise. This observation differs from our analysis for $n = 10$, where the lateness bound was unconditionally recommended.

Table 3.6: *Results for $n = 20$ with exact algorithms.*

Firm-planned orders			100%	80%	60%	40%	20%	0%
TIF		Unsolved	–	2	2	–	–	–
		Nodes	14	9255	19288	192019	4129	1677
		Time	0.54	19.45	27.11	703.54	6.25	4.21
two-phase	assign.	Unsolved	–	–	–	–	–	–
		Nodes	0	5	20	70	140	315
		Time	0.00	0.23	0.90	3.11	6.39	13.32
	lateness	Unsolved	–	–	–	–	–	–
		Nodes	0	13	162	2461	35565	471566
		Time	0.00	0.02	0.16	1.39	12.87	208.35
	LP-TIF	Unsolved	–	–	–	–	–	–
		Nodes	0	4	17	67	137	265
		Time	0.04	0.09	0.30	3.16	36.59	78.48
direct	assign.	Unsolved	–	–	–	–	–	–
		Nodes	182	412	249	621	1446	2948
		Time	2.60	4.52	6.12	15.52	34.63	65.66
	lateness	Unsolved	–	–	–	–	2	5
		Nodes	1138	3412	11028	64553	416139	904924
		Time	0.01	0.43	2.98	40.42	1346.03	2332.84
	LP-TIF	Unsolved	–	–	–	–	–	–
		Nodes	151	327	321	508	1159	2586
		Time	5.05	23.36	57.13	208.04	1157.91	1875.67

This might be explained by the fact that for $n = 20$, the potential size of the branching tree increases when the cardinality of F decreases and that therefore, the time spent in computing the assignment bound is compensated by the extra nodes pruned. On the other hand, the use of the LP relaxation bound usually prunes more nodes but the overall running time is larger than the overall running time of the algorithm with the assignment bound for instances with proportion of firm-planned orders greater than or equal to 40%. This implies that although the LP relaxation bound is the tightest bound, it is substantially more expensive to compute than the other bounds.

The direct B&B algorithm solves all instances with both the assignment bound and the LP relaxation bound, and all but seven of the 90 instances with the lateness bound. The best performance for large F is again associated with the lateness bound, while the assignment bound is preferable for a low number of firm-planned orders. In both cases, the two-phase algorithm is more efficient and is therefore recommended as the fastest

Table 3.7: *Results for $n = 30$ with exact algorithms.*

Firm-planned orders			100%	80%	60%	40%	20%	0%
TIF		Unsolved	–	2	2	2	–	1
		Node	5	1334	4486	3223	4855	25063
		Time	2.44	14.99	26.43	15.90	25.21	115.33
two-phase	assign.	Unsolved	–	–	–	–	–	–
		Nodes	0	7	61	174	765	1385
		Time	0.30	1.06	5.40	16.66	72.83	128.60
	lateness	Unsolved	–	–	–	–	8	7
		Nodes	0	33	1742	98302	416965	0
		Time	0.30	2.43	21.60	585.66	560.51	0.00
	LP-TIF	Unsolved	–	–	–	–	4	5
		Nodes	0	8	124	392	1015	1651
		Time	0.45	1.51	13.51	195.20	375.29	315.22
direct	assign.	Unsolved	–	–	–	3	3	2
		Nodes	14279	9807	9907	5441	13441	6146
		Time	408.47	192.93	166.30	149.44	336.47	300.42
	lateness	Unsolved	–	–	3	11	15	7
		Nodes	15088	89888	192703	516127	267993	257571
		Time	0.50	132.66	293.13	1207.88	619.58	1055.19
	LP-TIF	Unsolved	–	–	–	4	8	6
		Nodes	1585	7926	9392	6191	19582	5795
		Time	180.99	222.73	184.76	208.15	590.36	839.97

exact algorithm for solving 20-job instances, although TIF is competitive, especially for the case where all jobs can still be rejected.

Size $n = 30, 40$ and 50

The results of the exact algorithms for $n = 30$ are reported in Table 3.7. The time limit is again two hours. The number of unsolved instances by TIF increases from four in Table 3.6 to seven. With the assignment bound, the two-phase B&B solves all the instances, while the direct B&B is not able to solve eight instances within the time limit. With the lateness bound, on the other hand, the two-phase B&B cannot solve 15 instances while the direct B&B fails to solve 36 instances within the time limit. Implemented with LP relaxation bound, the two-phase B&B algorithm is unable to solve nine instances while the direct B&B algorithm fails to solve 17 instances within the time limit. We conjecture that for $n = 30$, the branching tree becomes large enough

to fully warrant a time-consuming but strong bound. Although the LP relaxation bound is at least as strong as the assignment bound, it usually take more time to be computed.

Overall, taking into account both the running times and the unsolved instances, we conclude that the two-phase B&B with assignment bound is the best exact algorithm for $n = 30$.

Table 3.8: *Results for* $n = 40$ *and* $n = 50$.

Firm-planned orders		100%	80%	60%	40%	20%	0%
40 jobs							
TIF	Unsolved	–	6	4	6	5	1
	Time	7.73	169.24	222.80	122.98	320.44	182.19
two-phase	Unsolved	–	–	–	–	–	–
	Time	8.96	105.01	107.87	178.76	290.79	533.76
50 jobs							
two-phase	Unsolved	–	–	–	–	1	–
	Time	24.51	345.38	379.71	878.53	1334.96	2091.59

In Table 3.8, the results for the two-phase B&B algorithm with the assignment bound and the TIF formulation are reported (for the latter only for $n = 40$). Again, a two-hour time limit is applied. For $n = 40$, the two-phase B&B solves all instances, while the linear formulation TIF leaves up to six out of 18 unsolved for some settings. When $n = 50$, the two-phase B&B algorithm solves all but one of the instances to guaranteed optimality within two hours.

3.7 Summary and conclusions

In this chapter, we have studied a generalization of the order acceptance problem with weighted-tardiness penalties, considering both firm-planned orders as well as potential orders, where the latter are orders that can still be rejected. We show that it is unlikely that a constant-factor approximation algorithm can be developed for this problem. We have presented two mixed-integer linear formulations, the first of which is rather intuitive, the second is a time-indexed formulation. Our results indicate that the MIP solver of CPLEX solves the latter formulation faster than the former one, even with the addition of cuts, when the processing time of each job is relatively small. In case of larger processing times, the two formulations appear to be competitive with each other, although the time-indexed model is frequently still the fastest.

We have also developed two B&B algorithms for finding optimal solutions. The first algorithm (a 'two-phase' algorithm) is hierarchical and performs selection and scheduling separately, while the second one (a 'direct' algorithm) integrates these two decisions. Three upper-bound procedures have been implemented, one based on an assignment problem, one that uses a job selection problem with weighted-lateness penalties and one based on the LP relaxation of the time-indexed formulation. Our experimental results demonstrate the efficiency of these two algorithms; the two-phase algorithm comes out best overall. For small instances (with a low number of jobs), the lateness bound leads to the fastest implementation; the full benefit of the computationally more expensive bound is achieved for instances with more than 20 jobs. For large instances, the two-phase B&B algorithm with assignment bound generally dominates the other exact algorithms and produces optimal solutions for instances with up to 50 jobs in less than two hours.

An important research direction that might be pursued in the future is an extension of this work to on-line scheduling, where not all jobs are available at the beginning of the planning horizon but arrive dynamically throughout time. A second obvious extension that deserves attention is the case where the manufacturing capacity is inadequately modeled as a single machine, so that multiple parallel machines or a more general scheduling environment should be considered. The extension of the Successive Sublimation Dynamic Programming method to solve our problem is also an interesting issue to pursue. For the long term, we may study a robust approach of our problem where the processing time of each job is not known exactly but rather takes its value either in a finite discrete set or in an interval. Further, we may investigate the existence of pseudo-polynomial time algorithms for a special case of our problem where there are deadlines instead of due dates.

Part II

Graph-coloring problems with applications in micro-economics

Chapter 4

A Graph-based interpretation of the Collective Axiom of Revealed Preference

In this chapter, we propose a directed graph for testing whether observed data of household consumption behavior satisfy the Collective Axiom of Revealed Preferences (CARP). More precisely, the data sets satisfy CARP if the graph allows a vertex-partitioning into two induced subgraphs that are acyclic. We prove that the problem of checking whether the obtained graph can be partitioned into two acyclic subgraphs is NP-complete. Next, we derive a necessary condition for CARP that can be verified in polynomial time, and we present an example to show that our necessary and sufficient conditions do not coincide. We also propose and implement fast heuristics for testing the sufficient condition. These heuristics can be used to check reasonably large data sets for CARP, and can be of particular interest when used prior to computationally demanding approaches. Finally, based on computational results both on real-life data and on synthetic data, we conclude that our heuristics are effective in testing CARP.

This chapter is the result of a collaboration with Laurens Cherchye, Bram De Rock, Jeroen Sabbe, and Frits Spieksma and is available as: F. Talla Nobibon et al. Heuristics for deciding collectively rational consumption behavior. *Computational Economics*, 2010, DOI 10.1007/s10614-010-9228-9.

4.1 Introduction

The economics literature has paid notable attention to modeling household consumption behavior. In this respect, Chiappori's 1988; 1992 collective model of household consumption has become increasingly popular in recent years. The model explicitly recognizes that a household consists of multiple individuals (household members and/or decision makers) with their own rational preferences. In this sense, this collective approach contrasts with the conventional unitary approach, which models households as if they were single decision makers.

The distinguishing feature of the collective model is that it only assumes Pareto efficiency of the collective household decisions, i.e. the intra-household allocation process yields consumption outcomes such that no household member can be made better off without making another member worse off. The use of Pareto efficiency as the sole assumption is in sharp contrast with usual cooperative models of household consumption behavior, which typically combine multiple bargaining assumptions (see Lundberg and Pollak, 2007, for a recent survey).

In this chapter, we concentrate on a general collective consumption model, which accounts for consumption externalities and public consumption within the household Browning and Chiappori (1998); Donni (2008) provides a neat overview of alternative collective consumption models. In the present context, public consumption of a certain good, which must be distinguished from private consumption, means that consumption of this good by one household member does not affect the supply available for another household member, and no individual can be excluded from consuming it (at least if one wants to maintain the household). Of course, some commodities may be partly publicly consumed (e.g. car use for a family trip) and partly privately consumed (e.g. car use for work). Next, consumption externalities refer to the fact that one household member gets utility from another member's consumption (e.g. the wife enjoys her husband's nice clothes).

The general collective model provides a useful starting point for testing Pareto efficiency of household collective consumption decisions: a rejection of the corresponding empirical restrictions can be interpreted as a rejection of the efficiency assumption. Moreover, given that all cooperative models use Pareto efficiency as a basic assumption and since it is also a natural benchmark in most non-cooperative settings, this test can also serve as basic input for these models.

Cherchye, De Rock, and Vermeulen (2007, 2009b) propose a testable nonparamet-

ric revealed preference condition for data consistency with the collective consumption model. Throughout, nonparametric analysis stands for revealed preference analysis in the tradition of, among others, Afriat (1967), Diewert (1973) and Varian (1982). Essentially, nonparametric (revealed preference) conditions allow for testing data consistency with a particular consumption model starting from a finite set of consumption observations, while avoiding the use of a (parametric) functional structure for the consumption decision process. The condition proposed by Cherchye, De Rock, and Vermeulen (2007, 2009b) (see Section 4.2) is known as the Collective Axiom of Revealed Preferences (CARP). CARP is a necessary and sufficient condition, i.e., observed household consumption behavior is consistent with the collective consumption model if and only if observed household consumption behavior satisfies CARP (Cherchye et al., 2009b). Because it uses minimal prior structure, checking CARP consistency implies a "pure" test of Pareto efficiency. Such a test can provide a most convincing case for the goodness of, in general, the Pareto efficiency assumption and, in particular, the collective consumption model.

Essentially, CARP provides the collective counterpart of the unitary concept GARP (Generalized Axiom of Revealed Preferences; see Section 4.2). The issue of testing data consistency with GARP has attracted considerable attention in the literature on the unitary model of household consumption; (see Cherchye et al., 2009a; Varian, 2006, for a recent survey). This chapter complements this rich literature by focusing on testing CARP.

More specifically, we consider in this chapter the computational issues involved in checking CARP and we propose a graph-theoretical representation of the CARP conditions. Following this approach, we derive a sufficient condition for CARP consistency, of which verification is shown to be an NP-complete problem, and a necessary condition for CARP consistency, which can be verified in polynomial time. Moreover, our graph-theoretical approach allows us to propose and implement heuristics that quickly test our sufficient condition for CARP. A consequence of attempting to test CARP quickly, is that the outcome of a heuristic may be inconclusive, i.e., it is possible that after running the heuristic it is still not clear whether the data satisfy CARP. By performing computational experiments, however, we show that a vast majority of real-life instances is susceptible to our approach. This leads us to conclude that heuristics are relevant for testing CARP, particularly for large data sets; see Cherchye et al. (2008a) and Deb (2008b) for recent discussions of the relevance of testing CARP for large instances.

Our graph-theoretical approach complements recent work of Cherchye et al. (2008a), who formulate the computational problem of verifying CARP as an Integer Programming (IP) problem. These authors show practical usefulness of this IP test for empirically evaluating the collective model. Using the MIP-solver of CPLEX, they perform their test on real-life data sets that are of reasonably large size when compared to existing nonparametric studies. It is well-known, however, that solving IP problems with exact implicit enumeration methods is computationally demanding. The approach presented in this chapter is particularly useful for (large) instances where Cherchye et al.'s IP approach takes a long time, or is unable to decide whether or not consumption behavior is collectively rational (see Section 4.5). In fact, even if the proposed heuristics fail to directly decide CARP, their outputs can be used to reduced the size of the IP to be solved, by fixing some variables.

In another study, Deb (2008b) proposes a different heuristic for testing the collective model. This heuristic pertains to a condition for collective rationality which he shows to be NP-complete. Yet, Deb's collective rationality condition is sufficient but not necessary for CARP; that is, data satisfying this condition satisfy CARP, but not necessarily vice versa. We will show that our sufficient condition extends the condition proposed by Deb. Specifically, all data sets that pass Deb's test also pass our test, but the opposite conclusion does not hold.

At a more general level, our analysis demonstrates the usefulness of operations research techniques to implement nonparametric (revealed preference) conditions for economic decision behavior. In fact, our insights on testing CARP consistency can also be instrumental for designing operational tests in alternative settings. For instance, they readily extend to the general case of multi-person group consumption. See Chiappori and Ekeland (2006, 2009) for discussion on the relevance of the collective model within the context of group consumption. To ease our exposition, the theoretical discussion in the following sections focuses on two-person households. Generalizations for M-member groups ($M \geq 2$) are fairly easy and can be obtained along the lines of Cherchye, De Rock, and Vermeulen (2007, supplemental material). The sufficient condition as well as the heuristics derived in this chapter can easily be extended to deal with the general case of M-member groups. Next, the nonparametric approach for analyzing collective consumption behavior is closely related to the literature on testable nonparametric conditions of general equilibrium models, which deals with formally similar issues. See, for example, Brown and Matzkin (1996), Brown and Shannon (2000) and,

for more recent developments, Carvajal, Ray, and Snyder (2004). Lastly, our results for the collective consumption model can also be relevant for nonparametric production analysis. See Cherchye, De Rock, and Vermeulen (2008b), who adopt a formally similar collective model for analyzing economies of scope in the context of multi-output production.

The rest of this chapter unfolds as follows. Section 4.2 defines collective rationality, and the corresponding CARP condition. Section 4.3 introduces the graph formulation, establishes a sufficient and a necessary condition for CARP. It also discusses the complexity of testing these conditions. Section 4.4 presents the heuristics. Section 4.5 discusses the computational results. Section 4.6 concludes.

4.2 Rationality conditions

Household consumption behavior that is consistent with the collective consumption model is said to be collectively rational. The empirical condition for collective rationality requires that a collective rationalization is possible, i.e. the data can be made consistent with the collective consumption model. As indicated above, a collective rationalization of the data is possible if and only if the data are consistent with CARP. This section provides formal definitions of the different concepts of rationalization. We first present a preliminary discussion of unitary rationality and the corresponding Generalized Axiom of Revealed Preference (GARP). This will set the stage for our subsequent discussion of the collective model.

4.2.1 Unitary rationality and GARP

Suppose we observe T individual choices of N-valued bundles. For each observation t the vector $q_t \in \mathbb{R}^N_+$ (with non-negative components) records the chosen quantities under the prices $p_t \in \mathbb{R}^N_{++}$ (with strictly positive components). We let $S = \{(p_t, q_t); t \in \mathbb{T} \equiv \{1, \ldots, T\}\}$ be the corresponding set of T observations, also referred to as the data. For ease of exposition, the scalar product $p_t \cdot q_t$ is written as $p_t q_t$.

We recall that the unitary model assumes that the household behaves as a single decision maker, i.e. the observed household quantities maximize a well-behaved household utility function U subject to the household budget constraint. In the context of the unitary model, well-behavedness means that the utility functions are locally non-satiated; see, for example, Varian (1982). In the context of the collective model, it means that

utility functions satisfy local collective non-satiation; this is the collective consumption analogue of the standard local non-satiation concept for the unitary model (for more discussion, see Cherchye et al., 2009b). We obtain the following condition for household behavior to be consistent with the unitary model.

Definition 4.1 (unitary rationalization)**.** Let $S = \{(p_t, q_t); t \in \mathbb{T}\}$ be a set of observations. A utility function U provides a unitary rationalization of S if for each observation t we have $U(q_t) \geq U(z)$ for all quantities z with $p_t q_t \geq p_t z$.

Varian (1982) (based on Afriat (1967)) demonstrates that a data rationalizing utility function exists if and only if the observed set S is consistent with the Generalized Axiom of Revealed Preference (GARP). Essentially, GARP imposes empirical restrictions on revealed preference relations R_0 and R. First, $q_s R_0 q_t$ means that the decision maker (directly) reveals her or his preference for the quantities q_s over the quantities q_t. Next, $q_s R q_t$ represents the transitive closure, that is $q_s R q_t$ means that there exists a (possibly empty) sequence $u, \ldots, w \in T$ with $q_s R_0 q_u$, $q_u R_0 q_v$,..., and $q_w R_0 q_t$. We can now define the GARP condition that applies to the unitary model.

Definition 4.2 (GARP)**.** Let $S = \{(p_t, q_t); t \in \mathbb{T}\}$ be a set of observations. The set S satisfies GARP if there exist relations R_0, R that meet for all $s, t \in \mathbb{T}$:

Rule 1**:** if $p_s q_s \geq p_s q_t$, then $q_s R_0 q_t$;

Rule 2**:** if $p_t q_t > p_t q_s$, then $\neg(q_s R q_t)$.

Testing GARP proceeds in two steps. First, one recovers the relations R_0 and (the transitive closure) R on the basis of Rule 1. Subsequently, one checks Rule 2, which requires $p_t q_t \leq p_t q_s$ for $q_s R q_t$. Varian (1982, p. 949) presents an efficient testing algorithm. Dobell (1965), Chung-Piaw and Vohra (2003) and Varian (1982) discus graph-theoretical representations of GARP. Our graph-theoretical formulation extends these studies by considering formally similar representations for the collective consumption model.

4.2.2 Collective rationality

Once again, we consider a household that purchases the (non-zero) N-vector of quantities $q \in \mathbb{R}^N_+$ with corresponding prices $p \in \mathbb{R}^N_{++}$, and we start from a set of observations S. As indicated in the introduction, we focus on a two-member household to keep our

exposition simple. All goods can be consumed privately (e.g. member 1 uses the car alone), publicly (e.g. member 1 and member 2 use the car together), or both. Generally, we assume that the empirical analyst has no information on the decomposition of the observed q. That is, we need to consider all allocations $q = q^1 + q^2 + q^h$ for q the (observed) aggregate quantities, q^1 and q^2 the (unobserved) private quantities of each household member, and q^h the (unobserved) public quantities.

The collective model explicitly recognizes the individual preferences of the household members. Because we account for consumption externalities, these preferences may depend not only on the own private and public quantities, but also on the other individual's private quantities. Formally, this means that the preferences of each household member m ($m = 1, 2$) can be represented by a well-behaved utility function of the form U^m that is defined in the arguments q^1, q^2 and q^h. Note that we do not demand that these utility functions are concave. Indeed, it has been argued that in the presence of externalities (i.e. the utility of one member depends on the private consumption of the other member) this assumption of concave utility functions (or, alternatively, convex preferences) is problematic (see, for example Starr, 1969; Starret, 1972).

For aggregate quantities q, we define feasible personalized quantities $\widehat{q}$ as

$$\widehat{q} = \left(\mathfrak{q}^1, \mathfrak{q}^2, \mathfrak{q}^h\right) \text{ with } \mathfrak{q}^1, \mathfrak{q}^2, \mathfrak{q}^h \in \mathbb{R}^n_+ \text{ and } \mathfrak{q}^1 + \mathfrak{q}^2 + \mathfrak{q}^h = q.$$

Each $\widehat{q}$ captures a feasible decomposition of the aggregate quantities q into private quantities (q^1 and q^2) and public quantities (q^h). This reflects that our model allows for general preferences that depend on private and public consumption. In the following, we consider feasible personalized quantities because we assume the minimalistic prior that only the aggregate quantity bundle q and not the "true" personalized quantities are observed. Throughout, we will use that each $\widehat{q}$ defines a unique q.

Given this, a collective rationalization of S requires the existence of utility functions U^1 and U^2 such that each observed consumption bundle can be characterized as Pareto efficient. Thus, we get the following definition, which has an analogous structure as Definition 4.1.

Definition 4.3 (collective rationalization)**.** Let $S = \{(p_t, q_t); t \in \mathbb{T}\}$ be a set of observations. A pair of utility functions U^1 and U^2 provides a collective rationalization of S if for each observation t there exist feasible personalized quantities $\widehat{q}_t$ such that $U^m(\widehat{z}) > U^m(\widehat{q}_t)$ implies $U^\ell(\widehat{z}) < U^\ell(\widehat{q}_t)$ ($m \neq \ell$) for all feasible personalized quantities $\widehat{z}$ with $p_t q_t \geq p_t z$.

4.2.3 Collective Axiom of Revealed Preference (CARP)

This section presents CARP, which provides a testable nonparametric necessary and sufficient condition for a collective rationalization of the data as described in the previous section. We refer to Cherchye, De Rock, and Vermeulen (2007, 2009b) for detailed discussions on CARP and the equivalent results.

Essentially, CARP imposes empirical restrictions on hypothetical member-specific preference relations H_0^m and H^m, which have a similar interpretation as the revealed preference relations R_0 and R for the unitary model. In this case, the hypothetical relations H_0^m and H^m represent feasible specifications of the true individual preference relations that are consistent with the information revealed by the set of observations S. First, $q_s H_0^m q_t$ means that we "hypothesize" that member m (directly) prefers the quantities q_s over the quantities q_t, $m = 1, 2$. Next, $q_s H^m q_t$ represents the transitive closure, that is $q_s H^m q_t$ means that there exists a (possibly empty) sequence $u, \ldots, w \in T$ with $q_s H_0^m q_u$, $q_u H_0^m q_v$,. . ., and $q_w H_0^m q_t$. Notice that, while the "true" preferences are -of course- expressed in terms of the feasible personalized quantities $\widehat{q}$ (i.e. member m prefers q_s over q_t only if $U^m(\widehat{q}_s) \geq U^m(\widehat{q}_t)$), the hypothetical preferences only use observable information (captured by the observed aggregate prices p and quantities q in the set S). This naturally complies with the assumption that in the general model we have no information concerning the feasible personalized quantities.

Given this notion of hypothetical preference relations, we can define CARP. The next definition, which reformulates Definition 6 of Cherchye, De Rock, and Vermeulen (2009b), gives us a condition that can be empirically tested on aggregate price-quantity information. Moreover, these authors show that there exists a collective rationalization of the data in terms of Definition 4.3 if and only if the data set is consistent with CARP. As such, we obtain the desired test of Pareto efficiency.

Definition 4.4 (CARP)**.** Let $S = \{(p_t, q_t); t \in \mathbb{T}\}$ be a set of observations. S satisfies CARP if there exist hypothetical relations H_0^m, H^m for each member $m \in \{1, 2\}$ that meet for all $s, t, t_1, t_2 \in \mathbb{T}$:

Rule 1**:** if $p_s q_s \geq p_s q_t$ then either $q_s H_0^1 q_t$ or $q_s H_0^2 q_t$;

Rule 2**:** if $p_s q_s \geq p_s q_t$ and $q_t H^m q_s$ then $q_s H_0^\ell q_t$ with $\ell \neq m$;

Rule 3**:** if $p_s q_s \geq p_s (q_{t_1} + q_{t_2})$ and $q_{t_1} H^m q_s$ then $q_s H_0^\ell q_{t_2}$ with $\ell \neq m$;

Rule 4**:** if $p_s q_s > p_s q_t$ then either $\neg(q_t H^1 q_s)$ or $\neg(q_t H^2 q_s)$;

Rule 5**:** if $p_s q_s > p_s(q_{t_1} + q_{t_2})$ then either $\neg(q_{t_1} H^1 q_s)$ or $\neg(q_{t_2} H^2 q_s)$.

This CARP condition has a similar structure as the unitary GARP condition in Definition 4.2. Specifically, GARP states (in casu unitary) rationality conditions in terms of the preference information that is revealed by the observed price and quantity data. CARP does the same, but now the revealed preference information is understood in terms of the collective model of household consumption and, thus, pertains to the individual household members. It can be verified that a data set S satisfies CARP if it satisfies GARP, but not vice versa. In other words, CARP consistency is necessary but not sufficient for GARP consistency.

Interestingly, CARP has a direct interpretation in terms of the Pareto efficiency requirement that underlies collective rationality. Rule 1 states that, if the quantities q_s were chosen while the quantities q_t were equally attainable (under the prices p_s), then it must be that at least one member prefers the quantities q_s over the quantities q_t (i.e. $q_s H_0^1 q_t$ or $q_s H_0^2 q_t$). Rule 2 can also be interpreted in terms of Pareto efficiency as follows: if member m prefers q_t over q_s for the bundle q_t not more expensive than q_s (i.e. $p_s q_s \geq p_s q_t$), then the choice of q_s can be rationalized only if the other member, member ℓ, prefers q_s over q_t. Indeed, if this last condition were not satisfied, then the bundle q_t (under the given prices p_s and outlay $p_s q_s$) would imply a Pareto improvement over the chosen bundle q_s. Analogously, Rule 3 states that, if the summed bundle $q_{t_1} + q_{t_2}$ is attainable and member m prefers q_{t_1} over q_s, then Pareto efficiency requires that the other member (member ℓ) prefers q_s over q_{t_2}. Finally, Rule 4 complements Rule 2 and Rule 5 complements Rule 3; and their interpretation in terms of Pareto efficiency is the following. Rule 4 states that, if q_t was attainable when q_s was chosen, then it cannot be that both members prefer q_t over q_s; otherwise Pareto improvements would have been possible (under the given prices p_s and outlay $p_s q_s$), which conflicts with collective rationality. Similarly, Rule 5 states that, if $q_{t_1} + q_{t_2}$ was attainable when q_s was chosen, then it cannot be that member m prefers q_{t_1} over q_s while, at the same time, member ℓ prefers q_{t_2} over q_s. In the rest of this chapter, we will refer to the inequality $p_s q_s \geq p_s(q_{t_1} + q_{t_2})$ as double-sum inequality. The following example, which we also use in the next section, illustrates CARP.

Example 4.5. *Consider a situation with three goods ($N = 3$) and two household members ($M = 2$) with data given in Figure 4.1. The observed price-quantity combinations are:*

1: $q_1 = (8 \quad 2 \quad 2)'$ *and* $p_1 = (6 \quad 2 \quad 2)'$

Two-member household

time	i = 1 Monday	i = 2 Tuesday	i = 3 Wednesday
good	(price, quantity)	(price, quantity)	(price, quantity)
water	(6, 8)	(2, 1)	(2, 1)
bread	(2, 2)	(6, 8)	(3, 2)
vegetable	(2, 2)	(1, 3)	(5, 8)

$p_1q_1 = 56 > 28 = p_1q_2$ $p_2q_2 = 53 > 30 = p_2q_1$ $p_3q_3 = 48 > 32 = p_3q_1$

$p_1q_1 = 56 > 26 = p_1q_3$ $p_2q_2 = 53 > 22 = p_2q_3$ $p_3q_3 = 48 > 41 = p_3q_2$

$p_1q_1 > p_1q_2 + p_1q_3$ $p_2q_2 > p_2q_1 + p_2q_3$

Figure 4.1: *Example of data set.*

2: $q_2 = (1 \quad 8 \quad 3)'$ *and* $p_2 = (2 \quad 6 \quad 1)'$

3: $q_3 = (1 \quad 2 \quad 8)'$ *and* $p_3 = (2 \quad 3 \quad 5)'$

and the inequalities are:

I_1: $p_sq_s \geq p_sq_t$ *for each pair* $s, t \in \{1, 2, 3\}$

I_2: $p_1q_1 > p_1(q_2 + q_3)$

I_3: $p_2q_2 > p_2(q_1 + q_3)$.

Consider H_0^1 *and* H_0^2 *defined as follows. For each observation* $s \in \{1, 2, 3\}$, *we have* $q_sH_0^mq_s$ *for* $m = 1, 2$. *Moreover,* $q_1H_0^1q_2$, $q_1H_0^1q_3$ *and* $q_3H_0^1q_2$ *while* $q_2H_0^2q_1$, $q_3H_0^2q_1$

and $q_2 H_0^2 q_3$*. One can verify that this specification of* H_0^1 *and* H_0^2 *satisfies Rules* 1-5 *in Definition 4.4. As such, we conclude that the data satisfies CARP.*

One concluding remark pertains to the fact that CARP only uses information on the (observed) aggregate quantities q, and not on the (unobserved) private quantities q^1 and q^2 and (unobserved) public quantities q^h. Because CARP provides a necessary and sufficient condition for a collective rationalization of the data, this actually means that the distinction between private consumption (with or without externalities) and public consumption is irrelevant in view of empirical tests of the collective consumption model. See also Cherchye, De Rock, and Vermeulen (2009b) for a detailed treatment of this issue.

In Chapter 6, we prove via a reduction from the Not-All-Equal-3Sat problem, that the problem of testing CARP is an NP-complete problem.

4.3 A graph-theoretic formulation

In this section, we show how to build from the data set S a directed graph $G(S) = (V(S), A(S))$ and use it to derive the following sufficient condition for CARP. If the vertices of $V(S)$ can be partitioned into two subsets such that each induced subgraph is acyclic, then the data satisfy CARP. By *induced subgraph*, we mean a subset of vertices of G together with any arcs whose both endpoints are in that subset; an *acyclic subgraph* is a subgraph which does not contain a cycle. Subsequently, we present a necessary condition for CARP. We also provide an example which shows that there exist instances of data set S for which neither the sufficient condition nor the necessary condition presented here are satisfied, while there exist hypothetical relations H_0^1, H_0^2 satisfying Rules 1-5 in Definition 4.4. Finally, we prove that while checking the necessary condition can be achieved in polynomial time, deciding whether a partition of vertices into two acyclic subgraphs exists for the graph $G(S)$ is NP-complete. For reasons of notational convenience, we will simply write G, V, and A instead of $G(S)$, $V(S)$, and $A(S)$ respectively.

4.3.1 Construction of the graph

Given a set of observations $S = \{(p_t, q_t); t \in \mathbb{T}\}$, each pair of distinct observations (s, t) with $s, t \in \mathbb{T}$ represents a vertex in V if $p_s q_s \geq p_s q_t$. Hence, the vertices (s, t) and (t, s) (if they exist) are different. No other vertices exist in V. A vertex (s, t) is said

to be involved in a double-sum inequality if there exists a vertex $(s,u) \in V$ such that $p_s q_s \geq p_s(q_t + q_u)$. Such vertices (s,t) and (s,u) will be called *double-sum vertices*. The set of arcs A is defined in two stages:

a: First of all, we draw an arc from a vertex (s,t) to a vertex (u,v) whenever $t = u$. The resulting graph is denoted by $G' = (V, A')$ and is a *line graph* (Gross and Yellen, 2004).

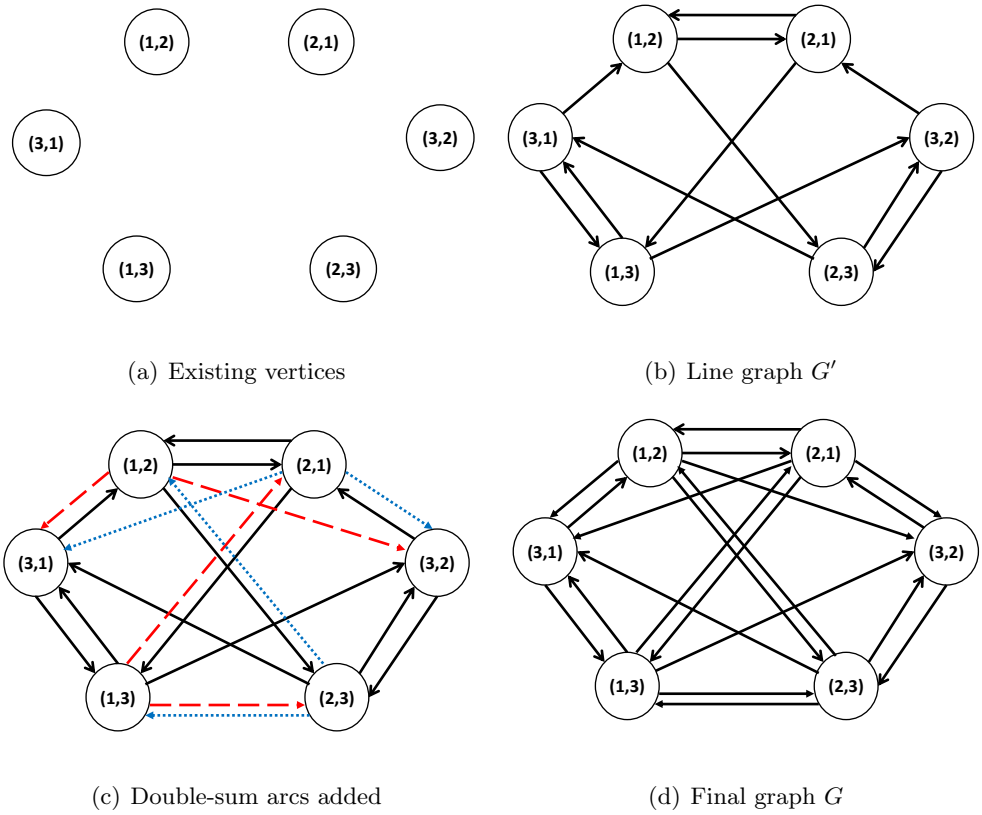
b: Second, for any given three distinct observations $s, t_1, t_2 \in \mathbb{T}$, verify whether $p_s q_s \geq p_s(q_{t_1} + q_{t_2})$ and whether there exist two observations $u, v \in \mathbb{T}$ (respectively $u', v' \in \mathbb{T}$) such that $(t_1, u), (v, s) \in V$ (respectively $(t_2, u'), (v', s) \in V$). If so, we distinguish two different cases:

- $(t_1, u) \neq (v, s)$ (respectively $(t_2, u') \neq (v', s)$)
  - If there is a path[1] in G' from (t_1, u) to (v, s) (respectively from (t_2, u') to (v', s)), then we draw an arc from (s, t_2) to (t_1, u) (respectively from (s, t_1) to (t_2, u')). Notice that the vertices (s, t_1) and (s, t_2) exist in V.
- $(t_1, u) = (v, s)$ (respectively $(t_2, u') = (v', s)$)
  - We draw an arc from (s, t_2) to (t_1, u) (respectively from (s, t_1) to (t_2, u')).

The directed graph $G = (V, A)$ is then defined by the set of vertices V described above and the set of arcs A described by a) and b). The arcs defined in b) will be called *double-sum arcs*. Notice that if there is no extra arc defined in b), then $G = G'$. Observe that in the construction of G, we associate a vertex with a pair of observations. This allows us to take into account relationships between three observations as formulated in Rule 3 and Rule 5. The following example illustrates the above construction.

Example 4.6. *We consider the data of Example 4.5. The first set of inequalities (I_1) implies the existence of all possible vertices in the graph. Figure 4.2(a) represents the vertices of graph G. The first step in the construction of the set of arcs leads to the line graph given by Figure 4.2(b). Next, the double-sum arcs are added to the line graph. In Figure 4.2(c), the dashed arcs $- - -\triangleright$ correspond with the double-sum inequality I_2 (i.e. $p_1 q_1 > p_1(q_2 + q_3)$) while the dashed arcs $\cdots\cdots\triangleright$ correspond with I_3 (i.e. $p_1 q_1 > p_1(q_2 + q_3)$). Finally the graph G is depicted in Figure 4.2(d).*

[1] A sequence of vertices $[v_0, v_1, \ldots, v_\ell]$ is called a *path* from v_0 to v_ℓ if there exists an arc from v_{i-1} to v_i for $i = 1, \ldots, \ell$.



(a) Existing vertices

(b) Line graph G'

(c) Double-sum arcs added

(d) Final graph G

Figure 4.2: *Illustration of the construction of G.*

Notice that the construction of G is done in polynomial time. In fact, an algorithm that finds the set V of vertices and determines double-sum vertices is implemented to run in time $O(T^3)$. Having the set of vertices, the line graph is immediate. To build the double-sum arcs, we proceed as follows. For a given vertex (s,t) involved in a double-sum inequality $p_s q_s \geq p_s(q_t + q_u)$, we use Dijkstra's algorithm (Ahuja et al., 1993) to find all the vertices which are such that there is a path in G' from (s,t) to those vertices. Among those vertices, we identify the vertices ending with s (these are vertices $(.,s)$) and draw an arc from (s,u) to the vertex $(t,.)$ appearing in each path. In the rest of this chapter, graphs built from a set S of observations following the procedure described in this section constitute the class of *CARP-graphs*.

4.3.2 A sufficient condition for CARP

In this section, we show that if the graph G can be vertex-partitioned into two acyclic subgraphs, then the set S of observations satisfies CARP; that is there exist H_0^1 and H_0^2 satisfying Rules 1-5 in Definition 4.4. In other words, when we can color each vertex of the directed graph G with one of the two colors red or blue, such that $V = V_B \cup V_R$, $V_B \cap V_R = \emptyset$, where V_B (respectively V_R) is the set of vertices colored blue (respectively red); and the induced subgraphs $G_B = (V_B, A_B)$, $G_R = (V_R, A_R)$ are each acyclic, then there exists hypothetical relations H_0^1 and H_0^2 that satisfy Rules 1-5 in Definition 4.4.

An equivalent way of phrasing this sufficient condition is as follows: can we color each vertex of G red or blue such that no monochromatic[2] cycle exists? For an arbitrary directed graph G, the problem of vertex-partitioning the graph into two acyclic induced subgraphs is proven to be NP-complete by Deb (2008a). Results for undirected graphs can be found in Chen (2000) (who gives an efficient algorithm to minimize the number of acyclic subgraphs), and more recently by Chang, Chen, and Chen (2004) (who study the complexity of the problem for specific graph classes).

Theorem 4.7. *If a CARP-graph G can be vertex-partitioned into two acyclic subgraphs then the corresponding set S of observations satisfies CARP.*

Proof. Suppose that G can be partitioned into two acyclic subgraphs $G_B = (V_B, A_B)$ and $G_R = (V_R, A_R)$. From this partition we infer H_0^1 and H_0^2 as follows.

H_0^1: (i) $q_s H_0^1 q_t$ if and only if $(s,t) \in V_B$ and (ii) $q_s H_0^1 q_s$ for all $s \in \mathbb{T}$.

H_0^2: (i) $q_s H_0^2 q_t$ if and only if $(s,t) \in V_R$ and (ii) $q_s H_0^2 q_s$ for all $s \in \mathbb{T}$.

In other words, for each observation $s \in \mathbb{T}$, $q_s H_0^1 q_s$ and for each vertex (s,t) which is colored blue, we have $q_s H_0^1 q_t$. For each vertex (s,t) which is colored red, we have $q_s H_0^2 q_t$ and for each observation $s \in \mathbb{T}$, $q_s H_0^2 q_s$. We are now going to check that Rules 1-5 hold.

Rule 1: Let $s, t \in \mathbb{T}$ be two distinct observations such that $p_s q_s \geq p_s q_t$. Then $(s,t) \in V = V_B \cup V_R$, which implies that $(s,t) \in V_B$ or $(s,t) \in V_R$, and hence $q_s H_0^1 q_t$ or $q_s H_0^2 q_t$ by construction of H_0^1 and H_0^2. Moreover, for each observation $s \in \mathbb{T}$ $q_s H_0^i q_s$ $(i = 1, 2)$ by definition. Thus Rule 1 is satisfied.

[2] A monochromatic cycle is a cycle containing vertices of the same color.

Rule 2**:** Clearly, this rule is satisfied for a single observation s. Let $s, t \in \mathbb{T}$ be two distinct observations such that $p_s q_s \geq p_s q_t$ and $q_t H^1 q_s$. Now, $p_s q_s \geq p_s q_t$ implies that $(s,t) \in V$ and $q_t H^1 q_s$ implies that there exist observations u, u_0, u_1, $\ldots$, u_k, $v \in \mathbb{T}$ such that (t,u), (u,u_0), (u_0,u_1), $\ldots$, (u_{k-1},u_k), (u_k,v), $(v,s) \in V$. By construction of G, there is a cycle containing the vertices (s,t), (t,u), (u,u_0), $(u_0,u_1),\ldots,(u_{k-1},u_k)$, (u_k,v) and (v,s). Since $q_t H^1 q_s$, all the vertices $(t,u),(u,u_0),(u_0,u_1),\ldots,(u_{k-1},u_k),(u_k,v),(v,s)$ are in V_B. $G_B = (V_B, A_B)$ is an acyclic subgraph implies that $(s,t) \in V_R$ and hence $q_s H_0^2 q_t$. Notice that a similar reasoning is applied to show that if $p_s q_s \geq p_s q_t$ and $q_t H^2 q_s$ then $q_s H_0^1 q_t$ for any distinct observations s and t. This completes the proof that Rule 2 is satisfied.

Rule 3**:** Suppose $s, t_1, t_2 \in \mathbb{T}$. Notice that if $s = t_1$ or $s = t_2$ then $p_s q_s < p_s(q_{t_1} + q_{t_2})$ and if $t_1 = t_2$ checking this rule becomes equivalent to checking Rule 2. Hence, we assume that s, t_1, t_2 are three distinct observations such that $p_s q_s \geq p_s(q_{t_1} + q_{t_2})$ and $q_{t_1} H^1 q_s$. Now, $p_s q_s \geq p_s(q_{t_1} + q_{t_2})$ implies that (s,t_1) and (s,t_2) belong to V. Also, $q_{t_1} H^1 q_s$ implies that there exists $u, v \in \mathbb{T}$ such that (t_1,u), $(v,s) \in V$ and either $(t_1,u) \neq (v,s)$ and there is a path from (t_1,u) to (v,s) or $(t_1,u) = (v,s)$. By construction of G, there is a cycle containing the vertices (s,t_2) and (t_1,u). Remark that if $(t_1,u) = (v,s)$ then that cycle contains only two vertices which are (t_1,s) and (s,t_2). Moreover, $q_{t_1} H^1 q_s$ indicates that all the vertices of the path from (t_1,u) to (v,s) (included) are in V_B or $(t_1,s) \in V_B$ if $(t_1,u) = (v,s)$. Since $G_B = (V_B, A_B)$ is an acyclic subgraph, $(s,t_2) \in V_R$ and $q_s H_0^2 q_{t_2}$. As in the proof of Rule 2, the symmetry between H_0^1 and H_0^2 allows us to conclude that if $p_s q_s \geq p_s(q_{t_1} + q_{t_2})$ and $q_{t_1} H^2 q_s$ then $q_s H_0^1 q_{t_2}$ for any three distinct observations s, t_1, t_2. This completes the proof of Rule 3.

Rule 4**:** Since $V_B \cap V_R = \emptyset$ and $p_s q_s = p_s q_s$ for each $s \in \mathbb{T}$, this property holds.

Rule 5**:** Suppose that $s, t_1, t_2 \in \mathbb{T}$ are three distinct observations such that $p_s q_s > p_s(q_{t_1} + q_{t_2})$ and $q_{t_1} H^1 q_s$ and $q_{t_2} H^2 q_s$. Now, $p_s q_s > p_s(q_{t_1} + q_{t_2})$ implies that $(s,t_1) \in V = V_B \cup V_R$. From $q_{t_2} H^2 q_s$ and Rule 3, we know that $(s,t_1) \in V_B$. $q_{t_1} H^1 q_s$ implies that there exists $u, v \in \mathbb{T}$ such that (t_1,u), $(v,s) \in V$ and either $(t_1,u) \neq (v,s)$ and there is a path from (t_1,u) to (v,s) in $G_B = (V_B, A_B)$ or $(t_1,u) = (v,s)$ and $(t_1,s) \in V_B$. $(s,t_1) \in V_B$ implies that $G_B = (V_B, A_B)$ contains a cycle. This contradicts the fact that G_B is acyclic.

We have shown that if the graph G can be partitioned into two acyclic subgraphs, then from these subgraphs, we can infer H_0^1 and H_0^2 satisfying Rules 1-5. □

The next example illustrates our sufficient condition.

Example 4.8. *Consider the graph G of Figure 4.2(d) built from data of Example 4.5. A possible coloring of vertices into two acyclic subgraphs is to color the vertices* $(1,2)$*,* $(1,3)$ *and* $(3,2)$ *blue while the vertices* $(2,1)$*,* $(3,1)$ *and* $(2,3)$ *get the color red. It is not difficult to see that each subgraph induced by the color class is acyclic. Therefore, Theorem 4.7 implies that the set of observations of Example 4.5 satisfies CARP.*

We remark that Theorem 4.7, which gives a sufficient condition for CARP, is not a necessary condition. In fact, Example 4.12, given later in the chapter, proves this point. Consequently, there can exist data sets S, with corresponding graphs $G = (V, A)$, which are such that there is no partition of V into two acyclic subgraphs while S satisfies CARP. Or equivalently, there can exist data sets S, with corresponding graphs $G = (V, A)$, that satisfy CARP but such that for any partition of V into two subsets, at least one induced subgraph is cyclic.

Next, we want to note that Cherchye, De Rock, and Vermeulen (2007, supplemental material) also generalize the definition of CARP for dealing with households (or groups) of more than two members. Given our construction of the graph, one can in an analogous way extend the above theorem to deal with this general case. That is, if the graph G built from the data set S can be vertex-partitioned into at most M acyclic subgraphs, then there exist H_0^1, H_0^2,..., H_0^M satisfying the corresponding generalization of CARP. Inter alia, this allows us to test for the number of decision makers in the household.

We end this section by showing that the problem of partitioning the vertices of our graph $G = (V, A)$ into two subsets such that each induced subgraph is acyclic, is an NP-complete problem.

Theorem 4.9. *The problem of deciding whether a vertex-partitioning of a CARP-graph G into two acyclic subgraphs exists, is NP-complete.*

Proof. This proof is a refinement of Deb's proof (2008b) for arbitrary graphs to CARP-graphs. It uses a reduction from the Not-All-Equal-3Sat problem defined as follows.

INSTANCE: Set $X = \{x_1, \ldots, x_n\}$ of n variables, collection $C = \{C_1, \ldots, C_m\}$ of m

clauses over X such that each clause $C_\ell = x_i \vee x_j \vee x_k$ depends on exactly three distinct variables.

QUESTION: Is there a truth assignment for C such that each clause in C has at least one true literal and at least one false literal?

Garey and Johnson (1979) proved that the Not-All-Equal-3Sat problem is NP-complete.

For a given instance of the Not-All-Equal-3Sat problem, consider the following polynomial-time reduction to an instance of our graph partitioning problem. For each variable $x_i \in X$, we have a pair of observations, that gives rise to the existence of two vertices called $(x_i, \bar{x}_i)$ and $(\bar{x}_i, x_i)$. (Notice that the existence of these vertices has implications for the prices and the quantities of goods corresponding to those observations. Here, we will ignore this issue, and simply create vertices assuming that the prices and quantities satisfy the corresponding relationships.) Hence, since $|X| = n$, we have $2n$ such vertices called *variable vertices* as they come from variables. For each clause $C_\ell = x_i \vee x_j \vee x_k \in C$, we define 18 *clause vertices* as follows. There are three *initial vertices* (x_i^ℓ, x_j^ℓ), (x_j^ℓ, x_k^ℓ) and (x_k^ℓ, x_i^ℓ) and there are three *complement vertices* (x_j^ℓ, x_i^ℓ), (x_k^ℓ, x_j^ℓ) and (x_i^ℓ, x_k^ℓ). Moreover, for each initial vertex, we define four *path vertices* which are used to create a path from that initial vertex to a given variable vertex. We say that these four path vertices are *associated* to this initial vertex. Explicitly, for the first initial vertex (x_i^ℓ, x_j^ℓ), we have $(s^\ell, \bar{x}_i)$, $(\bar{x}_i, s^\ell)$, (s^ℓ, x_j^ℓ) and (x_j^ℓ, s^ℓ); we refer to these four path vertices as the first, the second, the third and the fourth path vertices. For the second initial vertex (x_j^ℓ, x_k^ℓ), we define $(t^\ell, \bar{x}_j)$, $(\bar{x}_j, t^\ell)$, (t^ℓ, x_k^ℓ) and (x_k^ℓ, t^ℓ). Finally, for the third initial vertex (x_k^ℓ, x_i^ℓ), are created the path vertices $(u^\ell, \bar{x}_k)$, $(\bar{x}_k, u^\ell)$, (u^ℓ, x_i^ℓ) and (x_i^ℓ, u^ℓ). For each initial vertex, we define the path containing the vertices from the first path vertex to the complement vertex via the initial vertex. For instance, for the initial vertex (x_i^ℓ, x_j^ℓ), we have the path $P(x_i^\ell, x_j^\ell) = \{(s^\ell, \bar{x}_i), (\bar{x}_i, s^\ell), (s^\ell, x_j^\ell), (x_j^\ell, s^\ell), (x_i^\ell, x_j^\ell), (x_j^\ell, x_i^\ell)\}$. We use P to denote such path. In total, we have $|V| = 2n + 18m$ vertices. To complete the definition of our graph $G = (V, A)$, we now specify the arcs. Clearly, as described in Section 4.3, there is an arc going from (u, v) to (v, t) whenever (u, v) and (v, t) are vertices in V. Also, we add specific double-sum arcs. These arcs are derived from specific double-sum inequalities. For a given clause $C_\ell = x_i \vee x_j \vee x_k \in C$, we consider 9 double-sum inequalities, three for each initial vertex. For the initial vertex (x_i^ℓ, x_j^ℓ), we have three inequalities:

1. $p_{x_j^\ell} q_{x_j^\ell} \geq p_{x_j^\ell}(q_{x_i^\ell} + q_{s^\ell})$. This inequality implies the existence of arcs from vertex

(x_j^ℓ, s^ℓ) to vertices $(x_i^\ell, .)$, and arcs from vertex (x_j^ℓ, x_i^ℓ) to vertices $(s^\ell, .)$. Notice that all these double-sum arcs are between clause vertices from the clause C_ℓ.

2. $p_{s^\ell} q_{s^\ell} \geq p_{s^\ell}(q_{x_j^\ell} + q_{\bar{x}_i})$. This inequality implies the existence of double-sum arcs from vertex $(s^\ell, \bar{x}_i)$ to vertices $(x_j^\ell, .)$, and from vertex (s^ℓ, x_j^ℓ) to vertices $(\bar{x}_i, .)$. Notice that there may be an arc between two vertices of different clauses; indeed, if x_i occurs in another clause C_r, then there is a double-sum arc from (s^ℓ, x_j^ℓ) to vertex $(\bar{x}_i, s^r)$.

3. $p_{\bar{x}_i} q_{\bar{x}_i} \geq p_{\bar{x}_i}(q_{x_i} + q_{s^\ell})$. This inequality implies the existence of arcs from vertex $(\bar{x}_i, s^\ell)$ to vertices $(x_i, .)$, and from vertex $(\bar{x}_i, x_i)$ to vertices $(s^\ell, .)$. Again, if $\bar{x}_i$ occurs in another clause C_r, then there is an arc from $(\bar{x}_i, s^\ell)$ to vertex (x_i, s^r).

For each of the two remaining initial vertices (x_j^ℓ, x_k^ℓ) and (x_k^ℓ, x_i^ℓ), the construction is similar. We simply list here the corresponding double-sum inequalities. For the initial vertex (x_j^ℓ, x_k^ℓ), we have the three inequalities

4. $p_{x_k^\ell} q_{x_k^\ell} \geq p_{x_k^\ell}(q_{x_j^\ell} + q_{t^\ell})$ 5. $p_{t^\ell} q_{t^\ell} \geq p_{t^\ell}(q_{x_k^\ell} + q_{\bar{x}_j})$ 6. $p_{\bar{x}_j} q_{\bar{x}_j} \geq p_{\bar{x}_j}(q_{x_j} + q_{t^\ell})$,

and for the initial vertex (x_k^ℓ, x_i^ℓ), the double-sum inequalities are:

7. $p_{x_i^\ell} q_{x_i^\ell} \geq p_{x_i^\ell}(q_{x_k^\ell} + q_{u^\ell})$ 8. $p_{u^\ell} q_{u^\ell} \geq p_{u^\ell}(q_{x_i^\ell} + q_{\bar{x}_k})$ 9. $p_{\bar{x}_k} q_{\bar{x}_k} \geq p_{\bar{x}_k}(q_{x_k} + q_{u_\ell})$.

This completes the definition of our graph. Clearly, the above reduction can be done in polynomial time. Notice that each consecutive pair of vertices in each path P induces a cycle.

To have an overview of the above reduction, let us consider the following example. $X = \{x, y, z\}$ and there are two clauses $C_1 = x \vee y \vee z$ and $C_2 = \neg x \vee y \vee \neg z$ (here, $\neg x$ is the negation of the variable x). Remark that the assignment $x = y = 1$ and $z = 0$ is a solution to this Not-All-Equal-3Sat problem. From our reduction, $V = \{(x, \neg x),\ (\neg x, x),\ (y, \neg y),\ (\neg y, y),\ (z, \neg z),\ (\neg z, z),\ (x^1, y^1),\ (y^1, x^1),\ (y^1, s^1),\ (s^1, y^1),\ (\neg x, s^1),\ (s^1, \neg x),\ (y^1, z^1),\ (z^1, y^1),\ (z^1, t^1),\ (t^1, z^1),\ (\neg y, t^1),\ (t^1, \neg y),\ (z^1, x^1),\ (x^1, z^1),\ (x^1, u^1),\ (u^1, x^1),\ (\neg z, u^1),\ (u^1, \neg z),\ (\neg x^2, y^2),\ (y^2, \neg x^2),\ (y^2, s^2),\ (s^2, y^2),\ (x, s^2),\ (s^2, x),\ (y^2, \neg z^2),\ (\neg z^2, y^2),\ (\neg z^2, t^2),\ (t^2, \neg z^2),\ (\neg y, t^2),\ (t^2, \neg y),\ (\neg z^2, \neg x^2),\ (\neg x^2, \neg z^2),\ (\neg x^2, u^2),\ (u^2, \neg x^2),\ (z, u^2),\ (u^2, z)\}$. The graph obtained is depicted in Figure 4.3. Notice that for reason of clarity, not all the double-sum arcs are present in that figure.

Now, we prove that the graph $G = (V, A)$ obtained by the reduction can be partitioned into two acyclic subgraphs if and only if the instance of the Not-All-Equal-3Sat problem is a Yes-instance.

On the one hand, if graph G can be partitioned into two acyclic subgraphs G_1 and

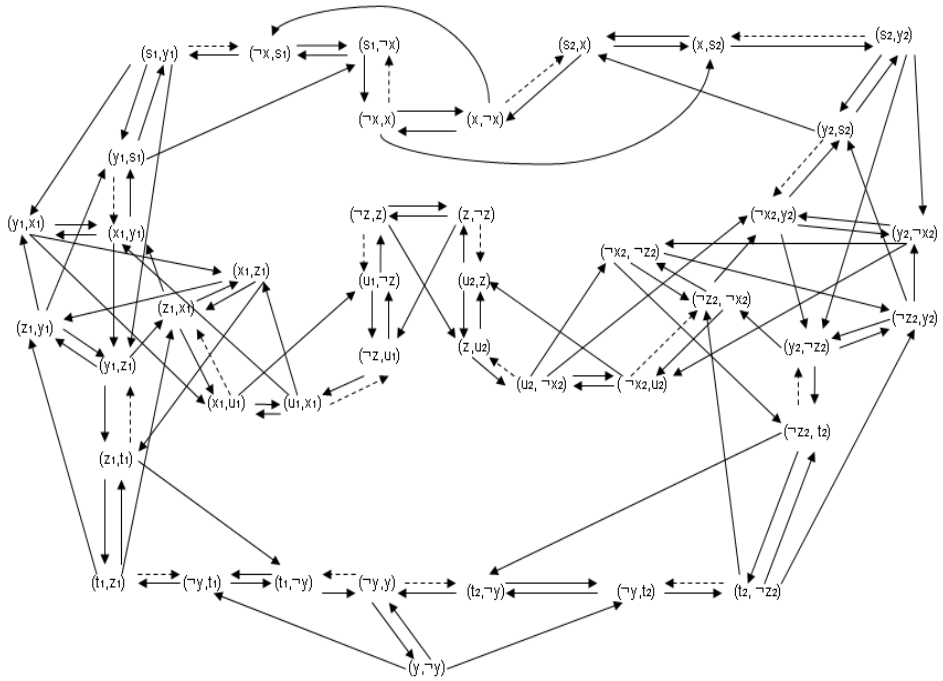


Figure 4.3: *Example of reduction*

G_2, then for each variable $x_i \in X$, if the vertex $(x_i, \bar{x}_i) \in G_1$, then we set the variable $x_i = 1$; else we set the variable $x_i = 0$. Let us prove that this assignment is a truth assignment for the set of clauses C. Let $C_\ell = x_i \vee x_j \vee x_k \in C$ be any clause $1 \leq \ell \leq m$. If $x_i = x_j = x_k = 1$ or $x_i = x_j = x_k = 0$, then the vertices (x_i^ℓ, x_j^ℓ), (x_j^ℓ, x_k^ℓ) and (x_k^ℓ, x_i^ℓ) are in the same partition and this will contradict the fact that each subgraph is acyclic.

On the other hand, if there is a truth assignment for C, then consider the following partition of G. For each $x_i \in X$, if $x_i = 1$ we color the variable vertex $(x_i, \bar{x}_i)$ red and $(\bar{x}_i, x_i)$ blue. Otherwise, if $x_i = 0$ we color the variable vertex $(x_i, \bar{x}_i)$ blue and $(\bar{x}_i, x_i)$ red. Moreover, we alternate the color of the nodes on the path P by coloring the first path vertex different from the corresponding variable vertex. This completes the coloring.

Clearly, the blue subgraph and the red subgraph define a partition of G. It remains to show that each subgraph is acyclic. We associate a *parity* to each vertex (except variable vertices) as follows: the first path vertex, the third path vertex, and the corresponding initial vertex are *odd* vertices, while the second path vertex, the fourth path vertex, and the complement vertex of the corresponding initial vertex are *even* vertices. Let us now argue that each cycle in G is not monochromatic. First, we consider cycles containing vertices from different clauses.

As described above, some double-sum arcs may link path vertices of different clauses. From the definition of our coloring, it turns out that such an arc links vertices of different colors. In fact, suppose that there is a double-sum arc from (s^ℓ, x_j^ℓ) to a path vertex $(\bar{x}_j, s^r)$ of another clause C_r. Then the coloring implies that $(\bar{x}_j, s^r)$ and $(\bar{x}_j, s^\ell)$ have the same color. Therefore, $(\bar{x}_j, s^r)$ and (s^ℓ, x_j^ℓ) have different colors. Thus, any cycle including vertices from different clauses linked using a double-sum arc, is not monochromatic. It follows that any monochromatic cycle containing vertices of different clauses necessarily contains a variable vertex. Moreover, since each arc leaving a variable vertex goes to a vertex with a different color, any cycle containing a variable vertex is not monochromatic. We conclude that cycles with clause vertices from different clauses are not monochromatic.

Second, we consider cycles within the subgraph defined by a single clause. Obviously, no monochromatic cycle can contain an arc between two consecutive vertices from path P. Thus each cycle in the subgraph consists of three arcs, linking three vertices of the three different paths that exist within each subgraph.

We claim that there do not exist arcs between vertices of different parity.

This claim implies that a monochromatic cycle would consist of three vertices of the same parity. However, the three initial vertices have the same parity, and the solution of the Not-All-Equal-3Sat problem implies that these vertices do not form a monochromatic cycle. The coloring then implies that any set of three vertices of the same parity do not form a monochromatic cycle. Hence, the validity of our claim implies the result.

To establish the claim, observe that each regular (i.e., non double-sum) arc between vertices of different paths is induced by a literal from the initial vertices, e.g. from $(., x_i^\ell)$ to $(x_i^\ell, .)$. Since this literal occurs in the three vertices once in the first position and once in second position, this implies that each regular arc links vertices of the same parity. In fact, it can be verified that this is also true for double-sum arcs. Hence, the

claim is valid. This completes the proof. □

4.3.3 A necessary condition for CARP

In this section, we present a necessary condition for CARP. Consider the data set $S = \{(p_t, q_t); t \in \mathbb{T}\}$. A subset S_1 of S containing at least four observations and at most six observations is called a *double-sum block subset* of S if S_1 contains three pairs of observations (t_1, t_2), (s_1, s_2) and (ℓ_1, ℓ_2) such that $S_1 = \{(p_t, q_t); t \in \{t_1, t_2, s_1, s_2, \ell_1, \ell_2\}\}$ and the following inequalities are met:

N_1: $p_{t_i} q_{t_i} > p_{t_i} q_{t_j}$, $p_{s_i} q_{s_i} > p_{s_i} q_{s_j}$ and $p_{\ell_i} q_{\ell_i} > p_{\ell_i} q_{\ell_j}$ with $i \neq j$ and $i, j \in \{1, 2\}$

N_2: $p_{t_1} q_{t_1} \geq p_{t_1}(q_{t_2} + q_{s_1})$ if $s_1 \neq t_2$

N_3: $p_{t_1} q_{t_1} \geq p_{t_1}(q_{t_2} + q_{\ell_1})$ if $\ell_1 \neq t_2$

N_4: $p_{s_1} q_{s_1} \geq p_{s_1}(q_{s_2} + q_{t_1})$ if $t_1 \neq s_2$

N_5: $p_{s_1} q_{s_1} \geq p_{s_1}(q_{s_2} + q_{\ell_1})$ if $\ell_1 \neq s_2$

N_6: $p_{\ell_1} q_{\ell_1} \geq p_{\ell_1}(q_{\ell_2} + q_{s_1})$ if $s_1 \neq \ell_2$

N_7: $p_{\ell_1} q_{\ell_1} \geq p_{\ell_1}(q_{\ell_2} + q_{t_1})$ if $t_1 \neq \ell_2$

The following result show that if the data set S contains a double-sum block subset, then S does not satisfy CARP.

Theorem 4.10. *If the data set S contains a double-sum block subset, then S does not satisfy CARP; that is there are no H_0^1 and H_0^2 satisfying Rules* 1-5*.*

Proof. Suppose that the data set S contains a double-sum block subset S_1. This subset has either four, five or six distinct observations.

Case 1: The double-sum block subset S_1 has four distinct observations. Without loss of generality, suppose that $s_1 = t_2 = \ell_2$. Let us assume that there exist hypothetical relations H_0^1 and H_0^2. Suppose without loss of generality that $q_{t_2} H_0^1 q_{s_2}$ and $q_{s_2} H_0^2 q_{t_2}$. The pair of observations (t_1, t_2) is such that either $q_{t_1} H_0^1 q_{t_2}$ or $q_{t_1} H_0^2 q_{t_2}$. If $q_{t_1} H_0^1 q_{t_2}$ then $q_{t_2} H_0^2 q_{t_1}$. The double-sum inequality $p_{t_2} q_{t_2} \geq p_{t_2}(q_{t_1} + q_{s_2})$ (inequality N_4) and the fact that $q_{t_1} H_0^1 q_{t_2}$ imply $q_{t_2} H_0^2 q_{s_2}$ from Rule 3. This leads to $q_{t_2} H_0^1 q_{s_2}$ and $q_{t_2} H_0^2 q_{s_2}$ and Rule 4 is violated. We conclude that $q_{t_1} H_0^2 q_{t_2}$ and thus $q_{t_2} H_0^1 q_{t_1}$. A similar reasoning allows to conclude that $q_{\ell_1} H_0^2 q_{t_2}$ and $q_{t_2} H_0^1 q_{\ell_1}$.

The double-sum inequality $p_{t_1}q_{t_1} \geq p_{t_1}(q_{t_2} + q_{\ell_1})$ from N_3 and the fact that $q_{t_2}H_0^1 q_{t_1}$ imply that $q_{t_1}H_0^2 q_{\ell_1}$ (using Rule 3). The double-sum inequality $p_{\ell_1}q_{\ell_1} \geq p_{\ell_1}(q_{t_1} + q_{t_2})$ from N_7 together with $q_{t_1}H_0^2 q_{\ell_1}$ imply, using Rule 3, that $q_{\ell_1}H_0^1 q_{t_2}$. We then have $q_{\ell_1}H_0^1 q_{t_2}$ and $q_{\ell_1}H_0^2 q_{t_2}$, contradicting Rule 4. This conclude that if the data set S contains a double-sum block subset with four distinct observations, then there is no H_0^1 and H_0^2 satisfying Rules 1-5.

Case 2: The double-sum block subset S_1 has five distinct observations. Without loss of generality, suppose that $s_1 = t_2$. Suppose also that the hypothetical relations H_0^1 and H_0^2 exist such that $q_{t_2}H_0^1 q_{s_2}$ and $q_{s_2}H_0^2 q_{t_2}$. We know from the reasoning of Case 1 that $q_{t_1}H_0^2 q_{t_2}$ and $q_{t_2}H_0^1 q_{t_1}$.

The double-sum inequality $p_{t_2}q_{t_2} \geq p_{t_2}(q_{s_2} + q_{\ell_1})$ from N_5 together with $q_{s_2}H_0^2 q_{t_2}$ imply, from Rule 3, that $q_{t_2}H_0^1 q_{\ell_1}$ and $q_{\ell_1}H_0^2 q_{t_2}$ comes from Rule 2. On the other hand, the double-sum inequality $p_{\ell_1}q_{\ell_1} \geq p_{\ell_1}(q_{\ell_2} + q_{t_2})$ identified by N_6 and $q_{t_2}H_0^1 q_{\ell_1}$ imply that $q_{\ell_1}H_0^2 q_{\ell_2}$ from Rule 3. Rule 2 allows to conclude $q_{\ell_2}H_0^1 q_{\ell_1}$. The double-sum inequality $p_{t_1}q_{t_1} \geq p_{t_1}(q_{t_2} + q_{\ell_1})$ from N_3 with $q_{t_2}H_0^1 q_{t_1}$ imply that $q_{t_1}H_0^2 q_{\ell_1}$ from Rule 3. The inequality $p_{\ell_1}q_{\ell_1} \geq p_{\ell_1}(q_{\ell_2} + q_{t_1})$ from N_7 and $q_{\ell_2}H_0^1 q_{\ell_1}$ imply that $q_{\ell_1}H_0^2 q_{t_1}$ from Rule 3. Therefore, from Rule 2 we have $q_{t_1}H_0^1 q_{\ell_1}$. This implies that $q_{t_1}H_0^1 q_{\ell_1}$ and $q_{t_1}H_0^2 q_{\ell_1}$, hence contradicting Rule 4. Therefore, there is no H_0^1 and H_0^2 satisfying Rules 1-5.

Case 3: The double-sum block subset S_1 has six distinct observations. A reasoning combining ideas of Case 1 and Case 2 allows to conclude that the existence of a double-sum block subset implies that the data set S does not satisfy CARP.

□

The following example describes a data set S containing a double-sum block subset.

Example 4.11. *Consider a situation with 4 goods ($N = 4$), two household members ($M = 2$) and the following four observed price-quantity combinations ($T = 4$):*

1*:* $q_1 = (8 \quad 2 \quad 2 \quad 0)'$ *and* $p_1 = (6 \quad 2 \quad 2 \quad 10)'$

2*:* $q_2 = (1 \quad 8 \quad 3 \quad 0)'$ *and* $p_2 = (2 \quad 6 \quad 1 \quad 10)'$

3*:* $q_3 = (1 \quad 2 \quad 8 \quad 0)'$ *and* $p_3 = (2 \quad 3 \quad 10 \quad 4)'$

4*:* $q_4 = (1 \quad 2 \quad 0 \quad 5)'$ *and* $p_4 = (1 \quad 1 \quad 1 \quad 1.7)'$

The inequalities satisfied are:

I_1*:* $p_s q_s \geq p_s q_t$ *for each pair* $s, t \in \{1, 2, 3\}$

I_2*:* $p_3 q_3 > p_3 q_4$ *and* $p_4 q_4 > p_4 q_3$

I_3*:* $p_1 q_1 > p_1(q_2 + q_3)$

I_4*:* $p_2 q_2 > p_2(q_1 + q_3)$

I_5*:* $p_3 q_3 > p_3(q_1 + q_4)$

I_6*:* $p_3 q_3 > p_3(q_2 + q_4)$

Consider the subset S_1 *of* S *containing the four observations, grouped in pair of observations as follows. The first pair is* $(1, 3)$*, the second pair is* $(2, 3)$ *and the third pair is* $(3, 4)$*. One can easily check that* S_1 *satisfies the inequalities* N_1*-*N_7*. Therefore, from Theorem 4.10, the data set* S *does not satisfy CARP.*

Notice that checking whether the data set S contains a double-sum block subset can be done in polynomial time; more precisely, an algorithm for identifying a double-sum block subset can be designed to run in time $O(T^3)$.

Finally, we provide an example of a data set that shows that our necessary and sufficient condition do not coincide. That is, S does not contain a double-sum block subset and the corresponding graph G cannot be vertex-partitioned into two acyclic subgraphs.

Example 4.12. *Consider a situation with* 4 *goods* ($N = 4$)*, two household members* ($M = 2$) *and the following four observed price-quantity combinations* ($T = 4$)*:*

1*:* $q_1 = (8 \quad 2 \quad 2 \quad 0)'$ *and* $p_1 = (6 \quad 2 \quad 2 \quad 10)'$

2*:* $q_2 = (1 \quad 8 \quad 3 \quad 0)'$ *and* $p_2 = (2 \quad 6 \quad 1 \quad 10)'$

3*:* $q_3 = (1 \quad 2 \quad 8 \quad 0)'$ *and* $p_3 = (2 \quad 3 \quad 10 \quad 4)'$

4*:* $q_4 = (1 \quad 2 \quad 0 \quad 5)'$ *and* $p_4 = (1 \quad 1 \quad 1 \quad 1)'$

The inequalities satisfied are:

I_1*:* $p_s q_s \geq p_s q_t$ *for each pair* $s, t \in \{1, 2, 3\}$

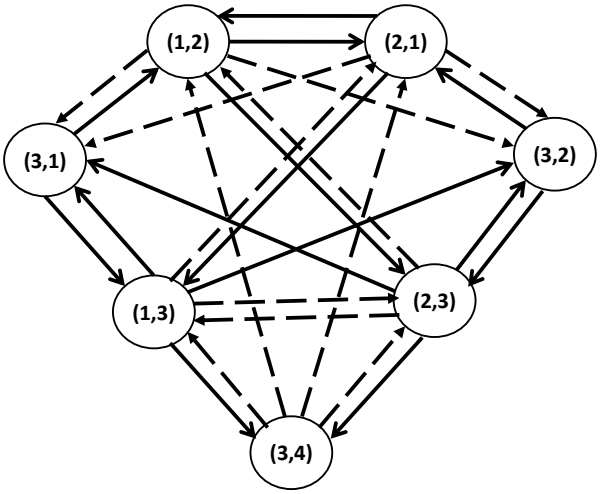


Figure 4.4: *The graph built from the data of Example 4.12.*

I_2: $p_3q_3 > p_3q_4$

I_3: $p_1q_1 > p_1(q_2 + q_3)$

I_4: $p_2q_2 > p_2(q_1 + q_3)$

I_5: $p_3q_3 > p_3(q_1 + q_4)$

I_6: $p_3q_3 > p_3(q_2 + q_4)$

The directed graph representation of this problem is given by Figure 4.4. We realize that it is not possible to color the vertices of the graph using only two colors in such a way that both subgraphs are acyclic. More explicitly, in any feasible coloring of this graph, one can deduce that vertices $(1, 3)$ *and* $(2, 3)$ *need to have a different color. It follows that* $(3, 4)$ *cannot be feasibly colored. Further, the data* S *of this example does not contain a double-sum block subset. However, it is not difficult to see that* H_0^1 *and* H_0^2 *defined as follows satisfy Rules* 1-5 *in Definition 4.4.*

H_0^1: $q_1H_0^1q_2$, $q_1H_0^1q_3$, $q_3H_0^1q_2$, $q_3H_0^1q_4$ *and* $q_sH_0^1q_s$ *for all* $s = 1, \ldots, 4$.

H_0^2: $q_2H_0^2q_1$, $q_2H_0^2q_3$, $q_3H_0^2q_1$, $q_3H_0^2q_4$ *and* $q_sH_0^2q_s$ *for all* $s = 1, \ldots, 4$.

Notice that the preference relations H_0^1 and H_0^2 obtained for the data of Example 4.12 have a non-trivial intersection; that is there exist two distinct observations s, t with $p_s q_s \geq p_s q_t$ such that $q_s H_0^1 q_t$ and $q_s H_0^2 q_t$. In the case of Example 4.12, we have $s = 3$ and $t = 4$. In fact, any pair of hypothetical relations H_0^1 and H_0^2 satisfying CARP for this data will have a non-trivial intersection. This non-trivial intersection is crucial to have data sets as in this example. Indeed, if there exists H_0^1 and H_0^2 with only a trivial intersection for a given data set, then the corresponding graph can be partitioned into two acyclic subgraphs and the converse of Theorem 4.7 will hold.

In this respect, we can also relate our proposal to the sufficient condition for CARP of Deb (2008b). Essentially, Deb's condition boils down to finding hypothetical relations with a trivial intersection. However, it is fairly easy to verify that Deb's approach, which focuses on a partitioning of some given data set, only considers a subset of all possible hypothetical relations with a trivial intersection. To give a specific illustration, the data in our Example 4.5 do not satisfy Deb's (2008b) condition for the collective model with two household members, while it passes our sufficient condition in Theorem 4.7. As such, Deb's sufficient condition is more stringent than ours and, in this sense, our condition extends Deb's condition.

4.4 Heuristics

This section is devoted to the development of simple heuristics for partitioning the directed graph $G = (V, A)$ built in Section 4.3 into two acyclic subgraphs. We first prove that the graph G can always be partitioned into two acyclic subgraphs when $G = G'$ is a line graph. We next present heuristics for solving the general case by combining a greedy rule for coloring the vertices of G with a specific sequence of the vertices. The main motivations to derive simple heuristics are twofold:

(i) Sophisticated and time consuming heuristics will not allow the rejection of CARP when they fail to color the vertices of G.

(ii) Heuristics are used prior to an exact and time consuming algorithm.

The heuristics developed in this section keep as a hard constraint the number of members in the household (two members). This is in accord with practical data where the rationality is tested for two members household data (Cherchye et al., 2008a; Deb, 2008b). Therefore, these heuristics try to color as many vertices as possible with two

colors, as opposed to the heuristic developed by Deb (2008b) which aims to color all the vertices using as few colors as possible. Although the heuristics described here focus on partitioning the vertices of the graph G built in Section 4.3, they can easily be adapted to deal with similar graph partitioning problem encountered by Deb (2008b). In what follows, we first prove that a line graph can always be colored using two colors, subsequently we describe heuristics for general case.

Lemma 4.13. *If $G = G'$ is a line graph, then G can be partitioned into two acyclic subgraphs.*

Proof. Since $G = G'$, we have no double-sum arcs. As such, the subgraph of G containing vertices (s, t) with $s < t$ is by construction acyclic and the same holds for the subgraph with vertices (s, t) where $s > t$. Clearly these two subgraphs form a partition of G. □

We remark that this special case with $G = G'$ can be quite relevant for empirical exercises; see for instance our own application in Section 4.5. For the general case where G is not a line graph, we develop heuristics by distinguishing coloring strategies on the one hand, and specifying vertex orderings, or sequences, on the other hand. More specifically, we present four coloring strategies for attempting to color a directed graph into two acyclic subgraphs and 13 sequences of vertices. A heuristic then is a combination of a coloring strategy and an ordering of the vertices.

4.4.1 Coloring strategies

CS1: Given a sequence of vertices, color iteratively each vertex red, unless this would create a red cycle. In case coloring the current vertex blue would create a blue cycle, we stop (and output: 0), else we color it blue, and continue.

CS2: Given a sequence of vertices, this coloring strategy colors iteratively each even (respectively odd) vertex red (respectively blue), unless this would create a red (respectively blue) cycle. In case coloring the current vertex blue (respectively red) would create a blue (respectively red) cycle, we stop (and output: 0), else we color it blue (respectively red), and continue. Notice that in this coloring strategy, a vertex is called "even" (respectively "odd") when its position in the sequence is even (respectively odd).

CS3: Given a sequence of vertices, this coloring strategy colors iteratively each vertex by a randomly generated color (from the set {blue, red}), unless this would create a monochromatic cycle. If coloring the current vertex red or blue would create a monochromatic cycle, we stop (and output: 0), else we color it with the remaining color, and continue.

CS4: Given a sequence of vertices, this coloring strategy colors iteratively each vertex with the same color as its predecessor, unless this would create a monochromatic cycle. If coloring the current vertex with the other color would also create a monochromatic cycle, we stop (and output: 0), else we color it with the other color, and continue.

Notice that in each coloring strategy we often need to check whether a (sub)graph is acyclic. We use the *topological ordering* algorithm to do so, see Ahuja et al. (1993) for more details. This algorithm labels the vertices of the graph ($order(i)$ to each vertex i) in such a way that every arc joins a lower-labelled vertex to a higher-labelled vertex. If for each connected pair of vertices i, j with an arc from i to j we have $order(i) < order(j)$, the graph is acyclic. Otherwise, it contains a cycle. The time complexity of the topological ordering algorithm is $O(m)$ where m is the number of arcs in the graph.

4.4.2 Ordering of the vertices

In the previous section, we assumed that a sequence of the vertices was given as input for each of the coloring strategies. Since there are $n!$ possible sequences for a graph G consisting of n vertices, it is not practical to try all of them. Therefore, we now describe specific sequences of vertices (often based on the structure of the graph) that will be used as input for the above coloring strategies.

Sq1: Sequence 1 is a natural sequence given by: $(0,1)$, $(0,2)$, $\ldots$, $(0,T)$, $(1,0)$, $(1,2)$, $\ldots$, $(1,T)$, $(2,0)$, $(2,1)$, $\ldots$, $(2,T)$, $\ldots$, $(T-1,1)$, $(T-1,2)$, $\ldots$, $(T-1,T)$ (recall that T is the number of observations). Of course, not all of these vertices need to exist, the non-existing vertices are simply removed from the list.

Sq2: Sequence 2 is the reverse of Sequence 1, hence it starts with $(T-1,T)$ and ends with $(0,1)$ (provided these vertices exist).

Sq3: Sequence 3 is found by placing each vertex (s,t) with $s < t$ before each vertex (s,t) with $s > t$; within each of these two sets of vertices we use the ordering implied by Sequence 1.

Sq4: Sequence 4 is the reverse of Sequence 3. Here, we follow the idea of Sequence 1, but we select vertex (s,t) with $s > t$ before vertex (s,t) with $s < t$.

Sq5: In this sequence, the position of a vertex is chosen randomly.

The next two sequences partition the vertices into those involved in a double-sum inequality, and those that are not. The idea is that vertices involved in a double-sum inequality might be more difficult to color than other vertices, and hence it might be worthwhile to place these vertices in the beginning of the sequence.

Sq6: Sequence 6 also uses the ordering of Sequence 1, but we place each double-sum vertex before each other vertex.

Sq7: Sequence 7 is the reverse of Sequence 6.

The following six sequences are based on the degree of a vertex. The *degree* of a vertex is the number of arcs it is incident to; the *indegree* is the number of arcs that enter a vertex while the *outdegree* of a vertex is the number of arcs that leave a vertex. Again, the rationale for using this measure is that the number of arcs a vertex is incident to is a measure of the difficulty of coloring that vertex.

Sq8: Sequence 8 is found by sorting the vertices with respect to their degree in increasing order; if there is a tie we use the ordering of Sequence 1.

Sq9: Sequence 9 is the reverse of Sequence 8.

Sq10: Sequence 10 is found by sorting the vertices in increasing order of their indegree; if there is a tie we use the ordering of Sequence 1.

Sq11: Sequence 11 is the reverse of Sequence 10.

Sq12: Sequence 12 is found by sorting the vertices in increasing order of their outdegree; if there is a tie we use the ordering of Sequence 1.

Sq13: Sequence 13 is the reverse of Sequence 12.

Notice that we have specified $13 \times 4 = 52$ heuristics since we can combine each of the four coloring strategies with each of the 13 sequences. We will apply all these heuristics on the given instances, and we comment on their efficiency in Section 4.5.2.

We mention that even if the heuristics fail to partition the vertices of G using two colors, its output can be used to reduced the size of the integer-programming problem to be solved. This could be done by fixing the variables of integer-programming model corresponding with the vertices of G colored before the heuristic stops.

4.5 Computational experiments

All algorithms have been coded in C using Visual Studio C++ 2005 and are available from the authors upon simple request. The experiments were run on a HP Pavilion dv6000 laptop with AMD Turion (tm) 64×2 Mobile Technology TL-56 processor with 1.80 GHz clock speed and 2047 MB RAM, equipped with Windows Vista. Below, we first provide some details on the data sets used and subsequently, we discuss the computational results.

4.5.1 Data

Our goal is to investigate the usefulness of the graph construction from Section 4.3, and to assess the quality and the speed of the heuristics proposed in Section 4.4. To do so, we apply the heuristics to two types of data sets drawn from Phase II of the Russian Longitudinal Monitoring Survey, which covers detailed consumption data from a nationally representative sample of Russian two-person households (or couples) during the time period between 1994 and 2003 (Rounds V-XII). When assuming homogeneity of the intra-household allocation process and individual preferences over time, such panel data enable us to treat each household as a time series in its own right. For each household, we focus on a rather detailed consumption bundle that consists of 21 nondurable goods. Only two-person households sharing certain characteristics are retained, which results in a basic sample consisting of 148 couples that are observed eight times. We refer to Cherchye et al. (2008a) for more details on the data.

Data I consists of the same real-life instances used by Cherchye et al. (2008a); as such this allows us to compare the integer-programming approach and the heuristics described here, see Section 4.5.2. In order to obtain bigger data sets that are still usefully interpretable from an economic point of view, these authors merged all households

of which males share the same birth year into one data set. In fact, this pertains to testing homogeneity of the intra-household allocation process and individual preferences for these couples. Next, to optimize the CPU times of the integer-programming approach they applied two efficiency enhancing procedures to minimize the number of observations that need to be considered by their procedures. This resulted in 69 instances with a number of observations that varies between two and 101, for which CARP was tested; for more details, see Cherchye et al. (2008a). We refer to this set of instances as Data I.

Second, on the basis of the above sample of 148 households, we also construct 120 synthetic data sets (instances) with varying size; these are contained in Data II. Every synthetic data set is obtained by randomly drawing households from the basic sample. Since each household is observed eight times, data set sizes are multiples of eight and range from eight to 96. As such, we consider data sets with substantially more observations than existing consumer panels; this allows us to analyze in further detail the performance of our heuristics. As far as we know, existing panel data with detailed consumption only contain a rather limited number of observations per household. For example, Blow, Browning, and Crawford (2008) and Christensen (2007) use, respectively, Danish and Spanish consumer panels with at most 24 observations per household.

4.5.2 Computational results

In this section we discuss the output of the heuristics applied to Data I and Data II.

Data I

The name of the instance is represented by three numbers. The first is the year, the second represents the number of that instance in that year and the last one is the number of observations considered in that instance. Density is the density of the graph given by the percentage of the number of arcs present in that graph divided by the total number of possible arcs.

Table 4.1: *Properties of the Graph representation of the instances of Data I*

Instance	Ref.	Obser.	Vertices	double-sum	Simple arcs	DS arcs	DS vertices	Total arcs	Density	Cyclic
1918-1-3	-	3	5	0	8	0	0	8	40.00	1

1924-1-2	-	2	2	0	2	0	0	2	100.00	1
1924-2-2	-	2	2	0	2	0	0	2	100.00	1
1924-3-7	I_1	7	22	2	52	1	2	53	11.47	1
1924-4-15	I_2	15	95	5	511	2	3	513	5.74	1
1926-1-2	-	2	2	0	2	0	0	2	100.00	1
1926-2-2	-	2	2	0	2	0	0	2	100.00	1
1926-3-3	-	3	5	0	8	0	0	8	40.00	1
1926-4-11	I_3	11	48	4	167	2	4	169	7.49	1
1927-1-3	-	3	5	0	8	0	0	8	40.00	1
1927-2-4	-	4	8	0	14	0	0	14	25.00	1
1927-3-4	-	4	7	0	13	0	0	13	30.95	1
1927-4-12	I_4	12	68	42	280	17	17	297	6.52	1
1927-5-27	I_5	27	279	590	1951	61	34	2012	2.59	1
1928-1-2	-	2	2	0	2	0	0	2	100.00	1
1928-2-7	-	7	23	0	60	0	0	60	11.86	1
1929-1-3	-	3	5	0	8	0	0	8	40.00	1
1929-2-3	-	3	5	0	8	0	0	8	40.00	1
1929-3-5	-	5	12	0	29	0	0	29	21.97	1
1929-4-32	I_6	32	410	447	3639	21	27	3660	2.18	1
1930-1-2	-	2	2	0	2	0	0	2	100.00	1
1930-2-2	-	2	2	0	2	0	0	2	100.00	1
1930-3-6	-	6	21	0	63	0	0	63	15.00	1
1930-4-16	I_7	16	118	30	682	17	15	699	5.06	1
1930-5-17	I_8	17	139	11	976	9	14	985	5.14	1
1931-1-2	-	2	2	0	2	0	0	2	100.00	1
1931-2-2	-	2	2	0	2	0	0	2	100.00	1
1932-1-2	-	2	2	0	2	0	0	2	100.00	1
1932-2-5	-	5	12	0	23	0	0	23	17.42	1
1932-3-6	-	6	19	0	60	0	0	60	17.54	1
1933-1-4	-	4	9	0	19	0	0	19	26.39	1
1935-1-2	-	2	2	0	2	0	0	2	100.00	1
1935-2-7	-	7	22	0	61	0	0	61	13.20	1
1935-3-101	I_9	101	4384	46916	121269	3052	2672	124321	0.65	1
1936-1-2	-	2	2	0	2	0	0	2	100.00	1
1936-2-2	-	2	2	0	2	0	0	2	100.00	1
1936-3-2	-	2	2	0	2	0	0	2	100.00	1
1936-4-2	-	2	2	0	2	0	0	2	100.00	1
1936-5-5	-	5	11	0	25	0	0	25	22.73	1
1936-6-40	I_{10}	40	755	1121	10049	64	46	10113	1.78	1
1937-1-2	-	2	2	0	2	0	0	2	100.00	1
1937-2-4	-	4	9	0	19	0	0	19	26.39	1
1937-3-5	-	5	13	0	30	0	0	30	19.23	1
1937-4-21	I_{11}	21	226	111	1953	26	19	1979	3.89	1
1938-1-2	-	2	2	0	2	0	0	2	100.00	1
1938-2-4	-	4	8	0	15	0	0	15	26.79	1
1938-3-4	-	4	8	0	14	0	0	14	25.00	1

1938-4-6	-	6	17	0	43	0	0	43	15.81	1
1938-5-9	-	9	39	0	129	0	0	129	8.70	1
1938-6-16	-	16	108	0	511	0	0	511	4.42	1
1939-1-2	-	2	2	0	2	0	0	2	100.00	1
1940-1-2	-	2	2	0	2	0	0	2	100.00	1
1940-2-2	-	2	2	0	2	0	0	2	100.00	1
1940-3-3	-	3	5	0	8	0	0	8	40.00	1
1940-4-18	-	18	141	0	852	0	0	852	4.32	1
1941-1-2	-	2	2	0	2	0	0	2	100.00	1
1941-2-3	-	3	4	0	5	0	0	5	41.67	1
1941-3-26	I_{12}	26	294	257	2353	74	66	2427	2.82	1
1945-1-2	-	2	2	0	2	0	0	2	100.00	1
1945-2-2	-	2	2	0	2	0	0	2	100.00	1
1948-1-2	-	2	2	0	2	0	0	2	100.00	1
1948-2-4	-	4	7	0	10	0	0	10	23.81	1
1948-3-4	-	4	8	0	15	0	0	15	26.79	1
1949-1-2	-	2	2	0	2	0	0	2	100.00	1
1950-1-5	-	5	12	0	25	0	0	25	18.94	1
1954-1-2	-	2	2	0	2	0	0	2	100.00	1
1954-2-2	-	2	2	0	2	0	0	2	100.00	1
1962-1-2	-	2	2	0	2	0	0	2	100.00	1
1962-2-3	-	3	5	0	8	0	0	8	40.00	1

Table 4.1 gives the properties of the graph representation of these instances. Notice that each graph contains a cycle. The analysis of this table shows that 57 instances out of 69 lead to a line graph; that is because they have no double-sum arc. This represents more than 82% of the instances, and it clearly shows that it is worthwhile to detect the absence of double-sum arcs in the data: if these arcs are absent one can immediately conclude (using Lemma 4.13 and Theorem 4.7) that the data satisfy CARP (instead of having to solve an IP-model). The second column of Table 4.1, entitled "Ref." contains the name which is used to refer to each instance in the rest of this chapter.

We then apply the heuristics to the remaining 12 instances. Table 4.2 displays the output of the heuristics. Each column (except for the first two columns and the last column) corresponds to a single instance. The row called "Time" (which corresponds to a specific sequence) reports the CPU time in seconds, which is the average value of the time needed for the four strategies using that particular sequence. The row "CS" identifies the coloring strategies for which we have obtained a partition into acyclic subgraphs. Finally, the last column gives, for each sequence, the total number of strategies for which a feasible coloring was found.

Table 4.2: *Output of heuristics for instances of Data I*

Instance		I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}	I_{12}	Total
Sq1	Time	0.00	0.00	0.00	0.00	0.02	0.12	0.00	0.00	34.04	0.73	0.03	0.08	16
	CS	1,2,3,4	1,4	1,4	4	-	-	1,4	4	-	4	1	1,4	
Sq2	Time	0.00	0.00	0.00	0.00	0.09	0.23	0.00	0.00	78.54	1.10	0.04	0.01	14
	CS	1,3,4	1,4	1,2,4	1,4	-	1	1	4	-	-	1	1	
Sq3	Time	0.00	0.00	0.00	0.00	0.01	0.16	0.00	0.00	31.33	0.70	0.04	0.01	16
	CS	1,2,3,4	1,4	1,3,4	4	-	-	1	4	-	-	1,4	1,4	
Sq4	Time	0.00	0.00	0.00	0.00	0.09	0.30	0.00	0.01	123.93	1.35	0.05	0.01	12
	CS	1,4	1,4	1,4	1	-	1	1	-	-	-	1,4	1	
Sq5	Time	0.00	0.00	0.00	0.00	0.03	0.20	0.00	0.00	12.25	1.38	0.01	0.04	5
	CS	1,4	-	1,2	-	-	-	-	1	-	-	-	-	
Sq6	Time	0.00	0.00	0.00	0.00	0.09	0.21	0.00	0.00	6.00	1.54	0.03	0.07	22
	CS	1,2,3,4	1,4	1,3,4	1,2,3	1,4	1	1	1,4	-	1,4	1	1	
Sq7	Time	0.00	0.00	0.00	0.00	0.02	0.12	0.00	0.00	34.12	0.73	0.02	0.08	16
	CS	1,2,3,4	1,4	1,4	4	-	-	1,4	4	-	4	1	1,4	
Sq8	Time	0.00	0.00	0.00	0.00	0.14	0.50	0.01	0.01	788.55	3.15	0.06	0.20	6
	CS	1,4	1	1,4	-	-	1	-	-	-	-	-	-	
Sq9	Time	0.00	0.00	0.00	0.00	0.05	0.14	0.00	0.00	22.54	1.04	0.00	0.04	15
	CS	1,2,3,4	1	1,2	1,4	1	1	1	1	-	1	-	3	
Sq10	Time	0.00	0.00	0.00	0.00	0.06	0.22	0.00	0.00	27.77	1.50	0.05	0.09	7
	CS	1,4	1	1	1,4	-	1	-	-	-	-	-	-	
Sq11	Time	0.00	0.00	0.00	0.00	0.04	0.25	0.00	0.01	6.09	0.88	0.02	0.05	18
	CS	1,2,3,4	1,4	1,2,3,4	-	-	1,4	1	1,4	-	1	1	1	
Sq12	Time	0.00	0.00	0.00	0.00	0.09	0.30	0.01	0.01	363.63	1.71	0.04	0.10	6
	CS	1	1	1	-	-	1	-	1	-	1	-	-	
Sq13	Time	0.00	0.00	0.00	0.00	0.08	0.24	0.00	0.00	17.65	0.95	0.01	0.05	21
	CS	1,2,3,4	1,4	1,2,3,4	1,4	1,4	1	1,3,4	1	-	1	-	1	

From Table 4.2, we see that for each instance except I_9 (1935-3-101), there is at least one heuristic finding a feasible coloring, meaning that each instance (except I_9) can be partitioned into acyclic subgraphs, and hence, by Theorem 4.7, satisfies CARP. This shows that (at least for this set of real-life instances) using the graph construction described in Section 4.3 does not lead to a loss of the ability to test whether the data sets satisfy CARP.

When looking at the results of the heuristics in more detail, we find that strategy 1 and strategy 4 are more successful than the other strategies. In particular, strategy 1 (CS1) is successful (meaning there is a sequence for which a coloring is found) in 11 out of the 12 instances, and strategy 4 (CS4) is successful for ten instances. This contrasts with strategies 2 and 3 which are only successful for two and five instances, respectively. We conclude that when coloring the vertices sequentially, it is better to keep using the same color, and only resort to another color when forced, rather than to build a "balanced" coloring, having approximately the same number of vertices of each color in any partial coloring.

When analyzing the sequences, it can be concluded that the relevance of a particular sequence is limited. Indeed, when a strategy is successful for some instances, there are often (but not always) many sequences for which this strategy is successful. Sequence 6 (Sq6) and Sequence 13 (Sq13) contain the highest number of strategies for which a feasible coloring was found, making them the most attractive sequences. In particular, the heuristic obtained by combining sequence 6 (Sq6) and strategy 1 (CS1) is very successful indeed: it solves all the instances except the one that is not solved by any heuristic (I_9).

In fact, instance I_9 is a particular instance in the sense that it is the only instance that was not solved by the IP-model of Cherchye et al. (2008a) after one hour of computing time. Our best heuristic (combining strategy 1 and sequence 6) led to a partial feasible coloring of 4224 vertices, i.e., about 95% of the vertices. We also verified that this instance passes our necessary condition in Theorem 4.10. On the other hand, we find that this graph can be colored by the heuristics using three colors. In Chapter 5, we show that this particular graph can, in fact, be colored using two colors, using a dedicated enumerative algorithm. This result enforces the usefulness of Theorem 4.7.

Table 4.2 also shows that the heuristics are quite fast. Computation time for most instances are within 0.1 second, and always (except for I_9) within two seconds. This is in

contrast with the computation time of Cherchye et al. (2008a), who report computation times up to five minutes for their instances. It should be noted, though, that solving the IP-model can lead to a conclusive answer, while the possible failure of a heuristic to produce a coloring gives no information about whether the data satisfy CARP. Nonetheless, we conclude that investing a little computation time to test for CARP quickly is a sensible approach for real-life data (Data I).

Data II

The name of a group of instances is represented by "Rand" followed by a number. Each group contains ten randomly generated instances. Rand is used to express the random characteristics of these instances and the number refers to the number of instances with eight observations aggregated. For instance, Rand-5 has $8 \times 5 = 40$ observations as it is the aggregation of five instances, each with eight observations.

Table 4.3: *Properties of the Graph representation of the instances of Data II*

Instance	Ref.	Obser.	Vertices	double-sum	Simple arcs	DS arcs	DS vertices	Total arcs	Density	Cyclic
Rand-1	R_1	8	25	20	45	1	2	46	6.96	1
Rand-2	R_2	16	108	285	445	12	13	457	3.87	5
Rand-3	R_3	24	254	1098	1651	46	46	1697	2.63	8
Rand-4	R_4	32	449	2652	3920	113	105	4033	2.00	10
Rand-5	R_5	40	718	5608	8079	141	133	8220	1.59	10
Rand-6	R_6	48	1023	9420	13671	212	188	13883	1.32	9
Rand-7	R_7	56	1406	16003	22452	494	369	22946	1.15	10
Rand-8	R_8	64	1808	23199	31981	456	351	32437	0.97	10
Rand-9	R_9	72	2276	32338	44917	710	567	45627	0.88	10
Rand-10	R_{10}	80	2845	45464	63526	835	654	64361	0.79	10
Rand-11	R_{11}	88	3448	59654	84963	1112	838	86075	0.72	10
Rand-12	R_{12}	96	4045	73973	106191	1065	832	107256	0.66	10

Table 4.3 gives the properties of the graph representation of the instances in Data II. In this table, each entry (except the entries in the last column) represents the average value of the ten values obtained for each instance in that group. In the last column (Cyclic), we give the number of instances in that group that contain both a cycle and a double-sum arc. Therefore, instances with only a cycle and no double-sum arc are not counted, since these are solved by Lemma 4.13.

Table 4.4 displays the output of the heuristics when applied to the instances in Data II. The notations are the same as in Table 4.2; an entry in the row "CS" is a 4-

Table 4.4: *Output of heuristics for instances of Data II*

Instance		R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}
Sq1	Time	0.00	0.01	0.08	0.35	1.38	4.53	6.60	18.17	19.97	34.51	38.98	56.26
	CS	1,0,0,1	5,1,1,5	7,3,2,7	7,1,1,9	6,0,0,8	5,1,0,7	4,0,0,6	4,0,1,4	1,0,0,3	0,0,0,2	0,0,0,1	0,0,0,1
Sq2	Time	0.00	0.01	0.08	0.35	1.51	3.77	9.12	18.89	25.58	38.11	52.29	80.11
	CS	1,0,0,1	5,3,1,4	7,2,2,7	7,1,1,7	6,2,1,6	4,0,1,4	4,0,0,1	4,0,0,3	0,0,0,2	0,0,0,0	0,0,0,0	0,0,0,0
Sq3	Time	0.00	0.01	0.08	0.37	1.51	4.69	7.58	19.24	24.21	42.03	58.57	92.67
	CS	1,0,0,1	5,1,1,5	7,3,1,6	7,2,2,6	6,0,0,4	5,1,0,4	4,0,0,3	4,0,0,3	1,0,0,1	0,0,0,1	0,0,0,0	0,0,0,0
Sq4	Time	0.00	0.01	0.09	0.38	1.66	4.21	10.19	21.13	31.06	46.70	88.99	136.29
	CS	1,1,1,1	5,1,2,5	7,2,3,7	7,3,1,5	6,0,2,5	4,0,0,3	4,0,0,0	4,0,0,1	0,0,0,2	0,0,0,0	0,0,0,0	0,0,0,0
Sq5	Time	0.00	0.01	0.10	0.37	1.26	3.26	3.35	11.20	12.98	14.81	23.93	21.77
	CS	1,0,1,0	3,1,2,1	7,3,4,3	6,1,2,4	2,1,0,0	2,0,0,0	1,0,0,0	1,0,0,0	0,0,0,0	0,0,0,0	0,0,0,0	0,0,0,0
Sq6	Time	0.00	0.01	0.08	0.33	1.32	2.99	3.87	16.99	33.88	10.54	21.67	0.80
	CS	1,1,1,1	4,4,4,4	7,5,6,6	8,5,5,5	8,3,3,3	5,2,2,2	3,1,1,1	5,2,1,4	5,2,1,3	0,0,1,2	1,0,0,0	0,0,0,0
Sq7	Time	0.00	0.01	0.08	0.35	1.37	4.52	6.59	18.22	19.97	34.43	38.95	56.47
	CS	1,0,0,1	5,1,1,5	7,3,2,7	7,1,1,9	6,0,0,8	5,1,0,7	4,0,0,6	4,0,1,4	1,0,0,3	0,0,0,2	0,0,0,1	0,0,0,1
Sq8	Time	0.00	0.01	0.14	0.73	3.41	10.23	25.40	56.35	100.94	164.76	347.73	537.32
	CS	0,0,0,0	3,1,1,3	5,2,3,3	2,2,1,3	2,0,0,1	2,0,1,1	1,0,0,0	1,0,0,0	0,0,0,0	0,0,0,0	0,0,0,0	0,0,0,0
Sq9	Time	0.00	0.01	0.12	0.37	1.55	4.07	2.32	15.58	28.18	13.63	6.05	4.55
	CS	1,1,1,1	2,3,2,2	7,7,6,6	6,4,3,6	6,3,5,4	4,2,3,3	1,0,0,1	4,1,0,1	2,1,0,3	1,0,0,0	0,0,0,0	0,0,0,0
Sq10	Time	0.00	0.01	0.09	0.39	1.46	4.77	9.18	18.99	29.85	61.56	105.78	132.23
	CS	0,0,0,0	2,2,2,2	5,3,2,5	4,1,1,4	2,0,1,1	2,0,0,3	1,0,0,0	2,0,0,0	0,0,0,0	0,0,0,0	0,0,0,0	0,0,0,0
Sq11	Time	0.00	0.01	0.07	0.30	1.21	2.64	1.80	10.29	13.41	5.41	1.33	4.53
	CS	1,1,1,1	2,2,3,2	6,5,4,6	6,3,4,6	6,3,3,5	4,2,2,2	1,1,0,1	4,1,1,1	1,0,1,1	0,0,0,1	0,0,0,0	0,0,0,0
Sq12	Time	0.00	0.00	0.10	0.42	1.39	4.35	7.36	16.45	29.42	44.76	51.29	96.40
	CS	0,1,1,0	3,2,1,1	4,2,2,4	2,1,2,3	2,1,0,1	2,0,0,0	1,0,0,0	1,0,0,0	0,0,0,0	0,0,0,0	0,0,0,0	0,0,0,0
Sq13	Time	0.00	0.02	0.11	0.46	2.18	4.88	7.47	22.74	40.05	39.15	82.32	92.34
	CS	1,1,1,1	4,4,4,2	7,6,6,7	7,6,4,7	8,5,5,6	5,3,2,2	2,0,1,4	4,1,0,3	3,3,1,2	1,0,0,1	0,0,0,0	0,0,0,0
Nr. solved		10	10	10	10	10	8	6	7	6	3	2	1

tuple indicating the number of instances solved by CS1, CS2, CS3, and CS4 respectively. Notice however that here an entry in the row "Time" is the average over the ten values obtained for the instances in that group. The last row of Table 4.4 (Nr. solved) reported the number of instances in each group for which the heuristics are able to find an optimal partition.

When analyzing the results of Table 4.4, we see that for the instances with at most 40 observations, the heuristics behave excellent. In fact, for each instance, the heuristics found an acyclic partition. Moreover, the CPU time used by the heuristics is less than two seconds. These observations confirm the results from Data I.

When the number of observations grows, the effectiveness of the heuristics drops. This is clearly seen from the last row of Table 4.4. Still, more than 60% of the instances whose number of observations is between 48 and 72 are solved in a reasonable amount of time (less than a minute). However, when the number of observations further increases, the effectiveness of the heuristic goes further down. We recall that there are three possible explanations for this: (i) either a coloring exists, but the heuristics fail to find one, or (ii) the graph does not admit a coloring in spite of the fact that the data satisfy CARP, or (iii) the data simply does not satisfy CARP. More sophisticated heuristics might shed a light on this question.

Overall, Table 4.4 reports that 83 instances out of 120 are solved using the heuristics; that is around 69% of the instances. The findings obtained after the application of heuristics to the instances in Data I are confirmed here. For instance, sequence 6 (Sq6) and sequence 13 (Sq13) are still the most attractive sequences, while coloring strategies 1 (CS1) and 4 (CS4) are the most successful strategies.

Summarizing, the computational results suggest that

(i) verifying whether the graph derived from the data contains double-sum arcs is rewarding for real-life instances,

(ii) the graph construction from Section 4.3 is useful for testing CARP at least for medium-sized instances (up to 75 observations), and

(iii) investing a little computation time (two seconds) trying to find a heuristic coloring often prevents the usage of a much more time-demanding exact algorithm.

4.6 Summary and conclusions

We consider in this chapter the computational problem of testing whether observed data from household consumption behavior satisfies the Collective Axiom of Revealed Preferences (CARP). We construct a directed graph from the data of observed household consumption which is such that the existence of a vertex-partitioning giving rise to two induced subgraphs that are acyclic implies that the data satisfies CARP. We also propose a necessary condition for CARP. We provide an example showing that there exist data sets which do not satisfy the necessary condition and the corresponding graphs do not admit a partition into two acyclic subgraphs while the data set satisfies CARP. Although checking the necessary condition can be achieved in polynomial time, we prove that the problem of checking whether the vertices of the obtained graphs can be partitioned into two acyclic subgraphs is an NP-complete problem.

We show that when the graph is a line graph, the data used to build that graph satisfies CARP. For graphs that do contain double-sum arcs, we propose and implement heuristics which are quite fast and can be used to check large data sets for CARP. The heuristics proposed are used prior to an exact and time-consuming algorithm. Moreover, if the outcome of the heuristics is not conclusive, it can still be used to reduced the size of the IP model to be solved. Applied to real-life and synthetic data, the heuristics turn out to be very effective for testing CARP. Moreover, the running time of the heuristics are usually within a fraction of second.

An important research direction that might be pursued in the future is an attempt to fill the gap between the necessary and the sufficient conditions proposed in this chapter by providing stronger conditions. Based on the success of the combinatorial structure (graph) used in this chapter for testing CARP, the long term research direction might investigate whether such a structure can be used to improve other (more specific) collective rational tests. Further, we may study how the recovery analysis can be performed from the outcome of a test based on a combinatorial structure.

Chapter 5

Acyclic 2-coloring problem

In this chapter, we consider the general problem of deciding whether a given directed graph can be vertex partitioned into two acyclic subgraphs. Applications of this problem include checking the sufficient condition of the Collective Axiom of Revealed Preference (CARP) defined in Chapter 4. We prove that the problem is NP-complete, even for oriented graphs and argue that the existence of a constant-factor approximation algorithm is unlikely for an optimization version which maximizes the number of vertices that can be colored using two colors while avoiding monochromatic cycles. We present three exact algorithms, namely an integer-programming algorithm based on cycle identification, a backtracking algorithm, and a branch-and-check algorithm. We compare these three algorithms both on real-life instances obtained in Chapter 4 and on randomly generated graphs. We find that for the latter set of graphs, every algorithm solves instances of considerable size within few seconds; however, the CPU time of the integer-programming algorithm increases with the number of vertices in the graph while that of the two other procedures does not. For real-life instances, the integer-programming algorithm solves the largest instance in about a half hour while the branch-and-check algorithm takes about ten minutes and the backtracking algorithm less than five minutes. Finally, for every algorithm, we also study empirically the transition from a high to a low probability of a YES answer as function of the number of arcs divided by the number of vertices.

This chapter is the result of a collaboration with Cor Hurkens, Roel Leus and Frits Spieksma. A preliminary version is available as: F. Talla Nobibon et al. 2010. Exact algorithms for coloring graphs while avoiding monochromatic cycles. *Lecture Notes in Computer Science* 6124, 229–242.

5.1 Introduction

Consider the following problem. Given is a finite, directed graph $G = (V, A)$. The goal is to partition the vertices of G into two subsets such that each subset induces an acyclic subgraph. Since the problem can be equivalently phrased as coloring the vertices of G using two colors such that no monochromatic cycle occurs, we refer to this problem as the *acyclic* 2*-coloring problem*. Notice that the acyclic 2-coloring problem is defined for a directed graph. The counterpart for undirected graphs is named *partition into two forests* and is known to be NP-complete (Wu et al., 1996). The problem defined for directed graphs seems to be neither a special case nor a generalization of the problem for undirected graphs; in other words, an algorithm for solving one problem cannot directly be used to solve the other problem and vice versa. Notice also that the acyclic 2-coloring problem is different from the standard graph coloring problem on an undirected graph because two adjacent vertices can have the same color; a directed acyclic graph, for instance, can be colored using a single color.

In this chapter, we prove that the problem is NP-complete, even for oriented graphs. We also show that it is unlikely to find a constant-factor approximation algorithm for solving an optimization formulation which maximizes the number of vertices that can be colored using two colors while avoiding monochromatic cycles. Further, we identify classes of directed graphs for which the problem is easy. We develop and implement three exact algorithms, namely an integer-programming (IP) algorithm based on cycle identification (in the rest of this chapter, we also refer to this algorithm as cycle-identification algorithm), a backtracking algorithm and a branch-and-check algorithm. We compare these algorithms based on their CPU time, both on real-life instances obtained in Chapter 4 and on randomly generated graphs. We find that every algorithm solves random graphs of considerable size within few seconds. The CPU time of the cycle-identification algorithm increases with the number of vertices in the graph while the running time of both the backtracking algorithm and the branch-and-check algorithm does not increase. Further, for every algorithm we study empirically the phase transition of the problem as function of the number of arcs divided by the number of vertices. When applying the three algorithms to real-life instances obtained in Chapter 4, however, we find that the cycle-identification algorithm usually takes more time than the two other procedures: the largest instance with 4384 vertices takes about a half hour, while the branch-and-check algorithm solves that instance in about ten minutes and the backtracking algorithm in less than five minutes.

This chapter is organized as follows. In Section 5.2, we describe some notation and present a brief literature review. In Section 5.3, we prove the complexity and the non-approximability results and present some properties of the acyclic 2-coloring problem. In Section 5.4, we describe the three exact algorithms, present some refinements and identify classes of directed graphs for which the acyclic 2-coloring problem is easy. Section 5.5 presents some issues related to the implementation of the algorithms. In Section 5.6, we comment computational results and study empirically the phase transition of the problem. We conclude in Section 5.7.

5.2 Notation and literature review

In this section, we describe some notation and definitions that will be used throughout this chapter and subsequently, we present a brief literature review.

5.2.1 Notation and definitions

We denote by $G = (V, A)$ a finite directed graph with $|V| = n$ vertices and $|A| = m$ arcs. In this chapter, we are only interested in directed graphs without loops, which are arcs for which start and end vertex are the same. For a vertex $p \in V$, the *outdegree* of p is the number of arcs leaving p while the *indegree* of p is the number of incoming arcs to p. The *degree* of p is the sum of its outdegree and its indegree. For ease of exposition, we will use pq to represent the arc $p \to q$. If G is such that there are no vertices p and q in V with $pq \in A$ and $qp \in A$ then G is an *oriented* graph. An oriented graph is also obtained by choosing an orientation for each edge of an undirected graph. If the undirected graph is planar (outerplanar) then the obtained oriented graph is also planar (outerplanar). A sequence of vertices $[v_0, v_1, \ldots, v_\ell]$ is called a *chain* of length ℓ if $v_{i-1}v_i \in A$ or $v_iv_{i-1} \in A$ for $i = 1, \ldots, \ell$. G is *connected* if between any two vertices there exists a chain in G joining them. Throughout this chapter, we consider only connected graphs. A sequence of vertices $[v_0, v_1, \ldots, v_\ell]$ is called a *path* from v_0 to v_ℓ if $v_{i-1}v_i \in A$ for $i = 1, \ldots, \ell$. A *vertex-induced subgraph* (subsequently called *induced subgraph*) is a subset of vertices of G together with all arcs whose endpoints are both in that subset. An *arc-induced subgraph* is a subset of arcs of G together with any vertices that are their endpoints. A *strongly connected component* (SCC) of G is a maximal induced subgraph $S = (V(S), A(S))$ where for every pair of vertices $p, q \in V(S)$, there is a path from p to q and a path from q to p. A sequence of vertices $[v_0, v_1, \ldots, v_\ell, v_0]$ is

called a *cycle* of length $\ell+1$ in $G=(V,A)$ if $v_{i-1}v_i \in A$ for $i=1,\ldots,\ell$ and $v_\ell v_0 \in A$. A graph is *acyclic* if it contains no cycle; otherwise it is *cyclic*. A *k-coloring* of the vertices of G is a partition $V_1, V_2, \ldots, V_k$ of V; the sets V_j $(j=1,\ldots,k)$ are called *color classes*. Given a k-coloring of G, a cycle $[v_0, v_1, \ldots, v_\ell, v_0]$ in G is *monochromatic* if there exists $i \in \{1,\ldots,k\}$ such that v_0, v_1, ..., $v_\ell \in V_i$. In this chapter, we use the notions vertex coloring and vertex partition of a graph interchangeably.

Given an integer k, an *acyclic k-coloring* of G is a k-coloring in which the subgraph induced by each color class is acyclic. The *acyclic chromatic number* $a(G)$ of G is the smallest k for which G has an acyclic k-coloring. The *directed line graph* LG of G has $V(LG) \equiv A(G)$ and a vertex (u,v) is adjacent to a vertex (w,z) if $v=w$. An arc $pq \in A$ is called a *single arc* if the arc $qp \notin A$. We define the 2*-undirected graph* $G_2=(V,E)$ associated with G as the undirected graph obtained from G by deleting all single arcs and transforming a pair of arcs forming a cycle of length 2 into an edge (undirected arc); more precisely, $\{v_1,v_2\} \in E$ if and only if $v_1v_2 \in A$ and $v_2v_1 \in A$. We define the *single directed graph* $G_s=(V,A_s)$ of G as the subgraph of G containing only single arcs; more precisely, for a given pair of vertices v_1 and v_2 in V, $v_1v_2 \in A_s$ if and only if $v_1v_2 \in A$ and $v_2v_1 \notin A$.

5.2.2 Literature review

To the best of our knowledge, Deb (2008a,b) is the first to explicitly address the acyclic 2-coloring problem. He proves that the problem is NP-complete and extends the results of Chen (2000) for undirected graphs by computing an upper bound on the acyclic chromatic number $a(G)$. In Chapter 4, we propose heuristics for maximizing the number of vertices that can be colored using two colors while avoiding monochromatic cycles; these heuristics are based on greedily coloring the vertices.

The literature on acyclic k-coloring for undirected graphs, however, is more elaborate. For $k=2$, Wu et al. (1996) study the partition of a graph into two induced forests. Thomassen (2008) studies 2-list-coloring planar graphs without monochromatic triangles. Broersma et al. (2006) investigate the coloring problem on planar graphs while avoiding monochromatic subgraphs. Several authors have studied the acyclic coloring problem for planar graphs (Aifeng and Jinjiang, 1991; Goddard, 1991; Raspaud and Wang, 2008; Roychoudhury and Sur-Kolay, 1995). For a general k, Chen (2000) gives an efficient algorithm for computing an upper bound of $a(G)$. Theoretical results on acyclic k-coloring for undirected graphs are contained in the framework of the general-

ized graph coloring problem (Alekseev et al., 2004). Applications of acyclic k-coloring for undirected graphs include wireless spectrum estimation (Khanna and Kumaran, 1998), game theory (Bartnicki et al., 2008) and logic (Bench-Capon, 2002).

5.3 Complexity and properties of the problem

In this section, we study the complexity of the acyclic 2-coloring problem and derive some properties that we use in the next section to build exact algorithms.

5.3.1 Complexity results

We prove that the acyclic 2-coloring problem is NP-complete, even for oriented graphs and we argue that it is unlikely to find a constant-factor approximation algorithm for an optimization version which maximizes the number of vertices that can be colored using two colors while avoiding monochromatic cycles.

The acyclic 2-coloring problem is explicitly defined as the following decision problem.

INSTANCE: A finite directed graph $G = (V, A)$.
QUESTION: Does G have an acyclic 2-coloring?

Notice that the acyclic 2-coloring problem is defined as a vertex partition problem. A different problem can be similarly defined by considering arc partitioning of G into two subsets such that each arc-induced subgraph is acyclic. This variant of the problem can be decided in polynomial time; in fact every directed graph is a YES instance. This argument comes from the fact that by building the corresponding line graph, the problem becomes equivalent to partitioning the vertices of the line graph into two subsets such that each subset induces an acyclic subgraph. The latter is identified later in this chapter as a YES instance of acyclic 2-coloring problem (see Section 5.4.5).

Notice that the acyclic 2-coloring problem is in the class NP. In fact suppose that we are given a coloring of the vertices of G using two colors. We consider each subgraph induced by a color class separately. We conclude that we have an acyclic coloring of G if and only if both subgraphs are acyclic (this can be checked in linear time using the topological ordering algorithm (Ahuja et al., 1993)). The following theorem shows that the acyclic 2-coloring problem is NP-complete, even for oriented graphs.

Theorem 5.1. *The acyclic 2-coloring problem is NP-complete for oriented graphs.*

Proof. The proof is a refinement of Deb's proof (Deb, 2008a) for arbitrary directed graphs G to oriented graphs. It uses a reduction from the Not-All-Equal-3Sat problem defined as follows.

INSTANCE: Set $X = \{x_1, \ldots, x_{n^*}\}$ of n^* variables, collection $C = \{C_1, \ldots, C_{m^*}\}$ of m^* clauses over X such that each clause $C_\ell \in C$ has $|C_\ell| = 3$, $\ell = 1, \ldots, m^*$.
QUESTION: Is there a truth assignment for X such that each clause in C has at least one true literal and at least one false literal?

Garey and Johnson (1979) proved that the Not-All-Equal-3Sat problem is NP-complete.

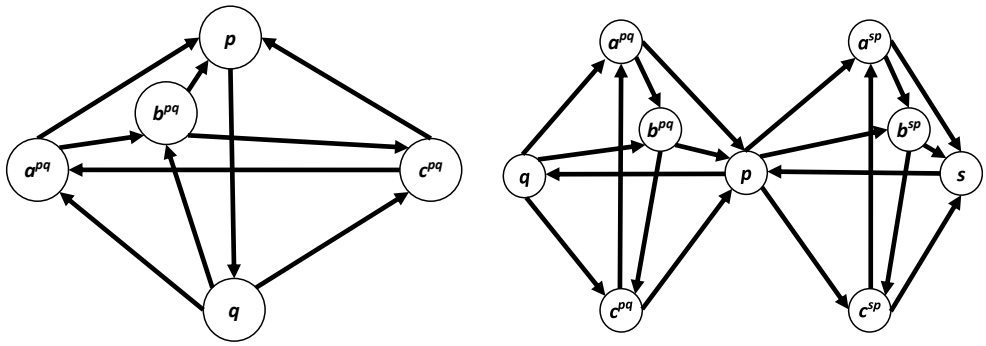
The proof of Theorem 5.1 is structured as follows. First, we build an oriented graph $G = (V, A)$ given the instance of the Not-All-Equal-3Sat problem. Next, we argue the equivalence of a yes-instance of Not-All-Equal-3Sat and the oriented graph G having a partition into two acyclic subgraphs.

In our construction of G, we use a gadget called a *p-q block*, which is a (sub)graph $\mathcal{B}_{pq} = (V_{pq}, A_{pq})$ with five vertices and ten arcs defined by: $V_{pq} = \{p, q, a^{pq}, b^{pq}, c^{pq}\}$ and $A_{pq} = \{pq, qa^{pq}, qb^{pq}, qc^{pq}, a^{pq}p, a^{pq}b^{pq}, b^{pq}p, b^{pq}c^{pq}, c^{pq}p, c^{pq}a^{pq}\}$. The illustration of $\mathcal{B}_{pq}$ is depicted in Figure 5.1(a). In Figure 5.1(b), we draw two blocks sharing one vertex p; these are the p-q block and the s-p block. In the block $\mathcal{B}_{pq}$, the vertices a^{pq}, b^{pq}, c^{pq} are called *block vertices* because they are used to build the block $\mathcal{B}_{pq}$. All the arcs in A_{pq} are called *block arcs*. In our construction of the oriented graph G, there is no arc going from a block vertex a^{pq}, b^{pq} or c^{pq} to vertices other than p and q.

Observe that in any feasible coloring of the block $\mathcal{B}_{pq}$ using two colors, the vertices p and q must always have different colors. Indeed, if p and q are assigned the same color then the three block vertices a^{pq}, b^{pq}, and c^{pq} all must have the same color and therefore will form a monochromatic cycle. To obtain a feasible coloring of $\mathcal{B}_{pq}$, it suffices to assign different colors to p and q, and make sure that the block vertices a^{pq}, b^{pq}, and c^{pq} do not have the same color.

In the first step of the proof, we aim at building an oriented graph $G = (V, A)$ from an arbitrary instance of the Not-All-Equal-3Sat problem. We first determine the set V of vertices followed by the set A of arcs.

Consider an arbitrary instance of the Not-All-Equal-3Sat problem. We build the set V of vertices as follows. For each variable $x_i \in X$, we have five vertices: x_i, $\bar{x}_i$, $a^{x_i\bar{x}_i}$, $b^{x_i\bar{x}_i}$ and $c^{x_i\bar{x}_i}$, where the last three vertices are block vertices; they are used to



(a) p-q block (b) Two blocks sharing vertex p

Figure 5.1: *Illustration of a single p-q block and two blocks sharing one vertex.*

build the block $\mathcal{B}_{x_i\bar{x}_i}$. Therefore, in our oriented graph there will not be an arc going from one of these three vertices to a vertex other than x_i and $\bar{x}_i$. The vertices x_i and $\bar{x}_i$ are called *variable* vertices. Hence, if $|X| = n^*$, we have $5n^*$ vertices corresponding to variables in the Not-All-Equal-3Sat instance. For each clause $C_\ell = \left(x_1^\ell \vee x_2^\ell \vee x_3^\ell\right) \in C$, we define 12 vertices among which nine block vertices. The three vertices x_1^ℓ, x_2^ℓ, and x_3^ℓ are called *literal* vertices. There are block vertices associated with x_1^ℓ, x_2^ℓ, and x_3^ℓ, respectively. For the literal x_1^ℓ there is a variable $x_i \in X$ such that either $x_1^\ell = x_i$ or $x_1^\ell = \bar{x}_i$. On the one hand, if $x_1^\ell = x_i$ then using the block vertices $a^{x_1^\ell \bar{x}_i}$, $b^{x_1^\ell \bar{x}_i}$ and $c^{x_1^\ell \bar{x}_i}$, we build the block $\mathcal{B}_{x_1^\ell \bar{x}_i}$. On the other hand, if $x_1^\ell = \bar{x}_i$ then we use the block vertices $a^{x_1^\ell x_i}$, $b^{x_1^\ell x_i}$ and $c^{x_1^\ell x_i}$ to build the block $\mathcal{B}_{x_1^\ell x_i}$. The block vertices associated with the literal x_2^ℓ and x_3^ℓ are defined similarly. Notice that for each literal $x_r^\ell \in C_\ell$ $(r = 1, 2, 3)$ we have four vertices, namely the literal vertex x_r^ℓ and three block vertices. If there are m^* clauses, we have $12m^*$ vertices coming from clauses. In total, the set V contains $5n^* + 12m^*$ vertices.

To complete the definition of our oriented graph G, we now specify the set A of arcs. We distinguish two types of arcs, depending on whether they are block arcs or not.

1. **Block arcs:** For each variable $x_i \in X$, there is a block $\mathcal{B}_{x_i\bar{x}_i}$, which requires ten block arcs. Hence, if $|X| = n^*$, we have $10n^*$ such block arcs. Further, for each clause $C_\ell = \left(x_1^\ell \vee x_2^\ell \vee x_3^\ell\right) \in C$ there are three blocks, one associated with each literal. Hence, for the m^* clauses there are $30m^*$ block arcs.

2. **Other arcs:** For each clause $C_\ell = \left(x_1^\ell \vee x_2^\ell \vee x_3^\ell\right) \in C$ there are three arcs which are not block arcs. These are $x_1^\ell x_2^\ell$, $x_2^\ell x_3^\ell$, and $x_3^\ell x_1^\ell$, which form a cycle containing the literal vertices x_1^ℓ, x_2^ℓ, and x_3^ℓ. Hence, for the m^* clauses there are $3m^*$ such arcs.

In total, we have $|A| = 33m^* + 10n^*$. This completes the definition of our oriented graph G. Clearly, the above reduction can be done in polynomial time and the obtained graph is an oriented graph.

To illustrate the reduction, we consider the following example of the Not-All-Equal-3Sat problem. The set of variables is $X = \{x_1, x_2, x_3\}$, and there are two clauses $C_1 = (x_1 \vee x_2 \vee x_3)$ and $C_2 = (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$; that is $x_1^1 = x_1$, $x_2^1 = x_2$, $x_3^1 = x_3$, $x_1^2 = \bar{x}_1$, $x_2^2 = x_2$ and $x_3^2 = \bar{x}_3$. Notice that the truth assignment $x_1 = x_2 = 1$ and $x_3 = 0$ is a solution to this Not-All-Equal-3Sat instance. For this example, the set of vertices corresponding to variables is $\{x_1,\ \bar{x}_1,\ a^{x_1\bar{x}_1},\ b^{x_1\bar{x}_1},\ c^{x_1\bar{x}_1},\ x_2,\ \bar{x}_2,\ a^{x_2\bar{x}_2},\ b^{x_2\bar{x}_2},\ c^{x_2\bar{x}_2},\ x_3,\ \bar{x}_3,\ a^{x_3\bar{x}_3},\ b^{x_3\bar{x}_3},\ c^{x_3\bar{x}_3}\}$, and the set of vertices stemming from clauses is $\{x_1^1,\ a^{x_1^1\bar{x}_1},\ b^{x_1^1\bar{x}_1},\ c^{x_1^1\bar{x}_1},\ x_2^1,\ a^{x_2^1\bar{x}_2},\ b^{x_2^1\bar{x}_2},\ c^{x_2^1\bar{x}_2},\ x_3^1,\ a^{x_3^1\bar{x}_3},\ b^{x_3^1\bar{x}_3},\ c^{x_3^1\bar{x}_3},\ x_1^2,\ a^{x_1^2x_1},\ b^{x_1^2x_1},\ c^{x_1^2x_1},\ x_2^2,\ a^{x_2^2\bar{x}_2},\ b^{x_2^2\bar{x}_2},\ c^{x_2^2\bar{x}_2},\ x_3^2,\ a^{x_3^2x_3},\ b^{x_3^2x_3},\ c^{x_3^2x_3}\}$. The set A of arcs obtained by the reduction contains the arcs $x_1^1x_2^1$, $x_2^1x_3^1$, $x_3^1x_1^1$, $x_1^2x_2^2$, $x_2^2x_3^2$, $x_3^2x_1^2$, which are not block arcs, and the block arcs used to build the blocks $\mathcal{B}_{x_1\bar{x}_1}$, $\mathcal{B}_{x_2\bar{x}_2}$, $\mathcal{B}_{x_3\bar{x}_3}$, $\mathcal{B}_{x_1^1\bar{x}_1}$, $\mathcal{B}_{x_2^1\bar{x}_2}$, $\mathcal{B}_{x_3^1\bar{x}_3}$, $\mathcal{B}_{x_1^2x_1}$, $\mathcal{B}_{x_2^2\bar{x}_2}$, and $\mathcal{B}_{x_3^2x_3}$.

In the last step of our proof, we show that the oriented graph G obtained by the above reduction can be partitioned into two acyclic subgraphs if and only if the instance of the Not-All-Equal-3Sat problem is a YES instance. The goal here is to prove that partitioning the oriented graph G built from the instance of the Not-All-Equal-3Sat problem into two acyclic subgraphs is at least as hard as that instance of the Not-All-Equal-3Sat problem.

$\Rightarrow$) If the graph G can be vertex-partitioned into two acyclic subgraphs G_1 and G_2, then for each variable $x_i \in X$, if the associated variable vertex $x_i \in G_1$, then we set the variable $x_i = 1$; otherwise $x_i = 0$. This is a truth assignment for X since each variable in X receives either value 0 or value 1. We now prove that this truth assignment is such that each clause in C has at least one true literal and at least one false literal. We argue by contradiction. Suppose that there exists a clause $C_\ell = (x_1^\ell \vee x_2^\ell \vee x_3^\ell)$ $(\ell \in \{1, \ldots, m^*\})$ in C which is such that either $x_1^\ell = x_2^\ell = x_3^\ell = 1$ or $x_1^\ell = x_2^\ell = x_3^\ell = 0$. Without loss of generality, let us assume that $x_1^\ell = x_2^\ell = x_3^\ell = 1$. We are going to investigate each literal in C_ℓ individually. The first literal x_1^ℓ is either x_i or $\bar{x}_i$ for a given variable $x_i \in X$. We will argue that in both cases, the associated literal vertex,

x_1^ℓ, belongs to G_1.

> On the one hand, if the literal $x_1^\ell = x_i$ then the variable $x_i = 1$. This implies, from the assignment of values to variables, that the associated vertex $x_i \in G_1$. In the construction of G, there is a block $\mathcal{B}_{x_i\bar{x}_i}$ which makes sure that the vertices x_i and $\bar{x}_i$ are not in the same subgraph. Since the vertex $x_i \in G_1$ this implies that the vertex $\bar{x}_i \in G_2$. Next, the presence of the block $\mathcal{B}_{x_1^\ell\bar{x}_i}$ in G (which exists by construction) and the fact that the vertex $\bar{x}_i \in G_2$ imply that the vertex $x_1^\ell \in G_1$.
>
> On the other hand, if $x_1^\ell = \bar{x}_i$ then $\bar{x}_i = 1$ implies that the variable $x_i = 0$ and hence the associated variable vertex $x_i \in G_2$. The block $\mathcal{B}_{x_1^\ell x_i}$ in G and the fact that the vertex $x_i \in G_2$ imply that the vertex $x_1^\ell \in G_1$.

We conclude that whether the literal x_1^ℓ is the variable x_i or its negation $\bar{x}_i$, as long as its value equals 1 the associated vertex $x_1^\ell \in G_1$. Notice that for case $x_1^\ell = 0$ we would conclude that the vertex $x_1^\ell \in G_2$.

By applying a similar reasoning to the literal x_2^ℓ, we obtain that the associated literal vertex $x_2^\ell \in G_1$, while the application of that reasoning to the literal x_3^ℓ leads to $x_3^\ell \in G_1$. We obtain that the vertices $x_1^\ell \in G_1$, $x_2^\ell \in G_1$ and $x_3^\ell \in G_1$; which implies that G_1 contains the cycle $[x_1^\ell, x_2^\ell, x_3^\ell]$. This contradicts the hypothesis that G_1 is acyclic.

$\Leftarrow$) Conversely, suppose that there is a truth assignment for X which is such that each clause in C has at least one true literal and at least one false literal. Consider the subgraphs G_1 and G_2 defined as follows. For each variable $x_i \in X$, if $x_i = 1$ then the variable vertex $x_i \in G_1$ and the variable vertex $\bar{x}_i \in G_2$. Otherwise, if the variable $x_i = 0$ then the vertex $\bar{x}_i \in G_1$ and the vertex $x_i \in G_2$. Further, for the block vertices (used to build the block $\mathcal{B}_{x_i\bar{x}_i}$) we make sure that they are not all three in the same subgraph. For example, we may put the block vertex $a^{x_i\bar{x}_i} \in G_1$, the block vertex $b^{x_i\bar{x}_i} \in G_1$ and the block vertex $c^{x_i\bar{x}_i} \in G_2$. This ensures that each vertex coming from a variable in X is either in G_1 or in G_2. We now deal with vertices stemming from clauses.

Let us consider the vertices coming from a clause $C_\ell = (x_1^\ell \vee x_2^\ell \vee x_3^\ell)$ ($\ell \in \{1, \ldots, m^*\}$) in C. We deal with each literal vertex separately. The first literal vertex, x_1^ℓ, is associated with the first literal x_1^ℓ in C_ℓ, the latter is either x_i or $\bar{x}_i$ ($i \in \{1, \ldots, n\}$). Since the corresponding variables vertices (x_i and $\bar{x}_i$) are either in G_1 or in G_2, we proceed as follows. If the literal $x_1^\ell = x_i$ then we put the literal vertex x_1^ℓ in the same subgraph as the variable vertex x_i. Otherwise (the literal $x_1^\ell = \bar{x}_i$), the literal vertex x_1^ℓ

is in the same subgraph as the variable vertex $\bar{x}_i$. In each case, the block vertices (used in the block $\mathcal{B}_{x_1^\ell\theta}$, where θ is either x_i or $\bar{x}_i$) are distributed in such a way that they are not all three in the same subgraph as sketched before. A similar distribution is done for the literal vertex x_2^ℓ and for the literal vertex x_3^ℓ. This completes the definition of G_1 and G_2. Clearly G_1 and G_2 form a partition of G since each vertex in G is either in G_1 or in G_2.

We now prove that G_1 and G_2 are acyclic. We also argue by contradiction. Suppose, without loss of generality, that G_1 contains a cycle. Notice that this cycle cannot contain both p and q for any p-q block present in G. Further, that cycle cannot be contained in two blocks sharing one vertex. Therefore, if there is a cycle in G_1 then there exists a clause $C_\ell \in C$ such that the cycle uses the literal vertices x_1^ℓ, x_2^ℓ, and x_3^ℓ. Therefore, x_1^ℓ, x_2^ℓ, $x_3^\ell \in G_1$ and hence all the literals of the clause C_ℓ have the same value. This contradicts the fact that the truth assignment for X was such that each clause of C has at least one false and at least one true literal.

This concludes the proof that the instance of the Not-All-Equal-3Sat is a YES instance only if the oriented graph G can be partitioned into two acyclic subgraphs, and hence completes the proof of Theorem 5.1. □

An optimization version of the acyclic 2-coloring problem maximizes the number of vertices of G that can be colored using two colors such that the subgraph induced by each color class is acyclic. We refer to this problem as Max-A2C. We next prove that Max-A2C contains the *maximum bipartite subgraph problem* defined for undirected graphs as a special case. The maximum bipartite subgraph problem is defined a follows: given an undirected graph K, find a bipartite subgraph of K with the maximum number of vertices.

Lemma 5.2. *Max-A2C contains the maximum bipartite subgraph problem as a special case.*

Proof. Consider a given instance of the maximum bipartite subgraph problem for a given undirected graph $K = (V, E)$. We build a directed graph $G = (V, A)$ from K as follows: given two vertices p, $q \in V$, if there is an edge between p and q in E then both the arc from p to q and the arc from q to p are present in A. Observe that a bipartite subgraph in K containing k vertices corresponds to a 2-coloring of the k vertices in the corresponding directed graph G that is acyclic, and vice versa. Therefore, the problem Max-A2C is at least as hard as the maximum bipartite subgraph problem. □

Lund and Yannakakis (1993) prove a non-approximability result for the maximum bipartite subgraph problem. Lemma 5.2, together with their result, implies the following corollary.

Corollary 5.3. *There exists an $\epsilon > 0$ such that Max-A2C cannot be approximated in polynomial time with ratio n^{ϵ} unless $P = NP$.*

5.3.2 Properties of the acyclic 2-coloring problem

We derive two properties of the acyclic 2-coloring problem that are used in the next section to build exact algorithms. Let $G = (V, A)$ be a given directed graph, G_2 its associated 2-undirected graph and G_s its single directed graph.

Proposition 5.4. *If the set V of vertices of G can be partitioned into two subsets, RED and BLUE, such that G_2 is bipartite with all the vertices in RED on one side and those in BLUE on the other side; and the single directed graphs induced by RED, $G_s(RED)$, and by BLUE, $G_s(BLUE)$, respectively, are acyclic then G is a YES instance of the acyclic 2-coloring problem; otherwise G is a NO instance.*

Proof. This follows from the fact that RED and BLUE form an acyclic coloring of G. □

Proposition 5.5. *If G_2 is not bipartite then G is a NO instance of the acyclic 2-coloring problem, while if G_2 is bipartite and G_s is acyclic, then G is a YES instance.*

Proof. Immediate. □

Notice that Proposition 5.5 implies Proposition 5.4 since if G_2 is not bipartite, then there are no two subsets RED and BLUE satisfying the hypothesis of Proposition 5.4. On the other hand, if G_2 is bipartite and G_s is acyclic then there exists two subsets RED and BLUE satisfying the hypothesis of Proposition 5.4. The reverse is not true.

5.4 Exact algorithms

In this section, we describe three exact algorithms for solving the acyclic 2-coloring problem, namely a *cycle-identification* algorithm, a *backtracking* algorithm and a *branch-and-check* (B&C) algorithm. The backtracking algorithm and the B&C algorithm are implicit enumeration algorithms built to solve the acyclic 2-coloring problem

while the cycle-identification algorithm is based on an IP formulation of the problem. We also present two dominance rules which can be used to reduce the size of the considered graph. In the rest of this section, $G = (V, A)$ is a given directed graph, G_2 is its associated 2-undirected graph and G_s its single directed graph.

5.4.1 Cycle-identification algorithm

We consider an IP formulation of the acyclic 2-coloring problem with binary variables x_i $(i = 1, \ldots, n)$, each of which equals one if vertex i is colored red and zero if it is colored blue. We are looking for a coloring x_i $(i = 1, \ldots, n)$ for which there is no monochromatic cycle. We choose to maximize the number of red vertices. Notice that any other objective function can be chosen. To complete the IP formulation, we add for each cycle $\mathcal{C}$ in G, the pair of constraints $1 \leq \sum_{i \in \mathcal{C}} x_i \leq |\mathcal{C}| - 1$, where $|\mathcal{C}|$ is the number of vertices in $\mathcal{C}$. Note that this IP formulation may have an exponential number of constraints.

A formal description of the cycle-identification algorithm is given by CycleId(G). It works as follows. A relaxed IP instance containing only a subset of constraints is solved. If that instance is infeasible, we stop and output NO. Otherwise, we consider the subgraph induced by each color class separately and check whether there is a cycle. If both subgraphs are acyclic then we stop and output YES. On the other hand, if for at least one induced subgraph a cycle is found, we add to the relaxed IP instance the corresponding pair of constraints. The problem is solved again and the above procedure is repeated until either a YES or a NO answer is returned. Notice that the implementation of this algorithm does not need an optimal solution to the IP instances; a feasible solution is enough.

Algorithm 5.1 CycleId(G)

```
1: solve a relaxed IP instance containing only a subset of constraints
2: if there exists a feasible solution
3:    for each subgraph induced by a color class, search for a monochromatic cycle
4:    if monochromatic cycle found
5:       add the corresponding pair of constraints to the relaxed IP instance
6:       solve the relaxed IP instance again and goto 2
7:    else return YES
8: else return NO
```

5.4.2 Backtracking algorithm

An "ordinary" backtracking algorithm for solving the acyclic 2-coloring problem is an adaptation of the well-known backtracking algorithm for graph coloring on undirected graphs. It would work as follows: successively color the vertices of G either red or blue and each time a new vertex is colored, the subgraph induced by the corresponding color class is checked to see whether it is still acyclic; otherwise the color of the last vertex is switched and the subgraph induced by its new color class is then checked. If it is not acyclic, the algorithm backtracks.

In this section, we propose a backtracking algorithm based on Proposition 5.4. This is an enumeration algorithm which explicitly colors every vertex of G. The key difference between our algorithm and an ordinary backtracking algorithm is that the backtracking algorithm described here can anticipate a NO conclusion earlier without having to color many vertices. This is due to the bipartiteness test included in the algorithm. Broadly speaking, this test consistently extends (if possible) the effect of colored vertices to (connected) uncolored vertices.

A formal description of the backtracking algorithm is given by BT(RED, $BLUE$, G) with $RED = \emptyset$ and $BLUE = \emptyset$ at the beginning. In the description, the function bipartite(RED, $BLUE$, G_2) returns YES if G_2 is bipartite given that the vertices in RED are on one side and those in BLUE are on the other side; otherwise it returns NO. We denote by $G_s(A)$ the single directed graph induced by a set A.

Algorithm 5.2 **BT**(RED, $BLUE$, G)

1: if $V = RED \cup BLUE$, then return YES
2: choose a vertex p in $V \setminus \{RED \cup BLUE\}$
3: $RED = RED \cup \{p\}$
4: if bipartite(RED, $BLUE$, G_2) and $G_s(RED)$ acyclic then
5: if BT(RED, $BLUE$, G) then return YES
6: $RED = RED \setminus \{p\}$, $BLUE = BLUE \cup \{p\}$
7: if bipartite(RED, $BLUE$, G_2) and $G_s(BLUE)$ acyclic then
8: if BT(RED, $BLUE$, G) then return YES
9: return NO

Proposition 5.6. *The backtracking algorithm terminates after a finite number of iterations. Further, upon termination, the output decision corresponds to the decision for the original graph G.*

Proof. This follows from the fact that there is a finite number of colorings (at most 2^n) and in the worst case, the backtracking algorithm will enumerate all of them. □

5.4.3 Branch-and-check algorithm

This B&C algorithm is based on Proposition 5.5. Like the backtracking algorithm, it is an enumeration algorithm where at each node we check some conditions and decides whether to proceed or to stop. Unlike the backtracking algorithm, however, the B&C algorithm is an implicit coloring algorithm which branches on an arc, and the directed graph obtained at every child node is different from the graph at the parent node. The expression *branch-and-check* has also been used in the literature to refer to some algorithms that integrate mixed-integer programming and constraint logic programming (Thorsteinsson, 2001).

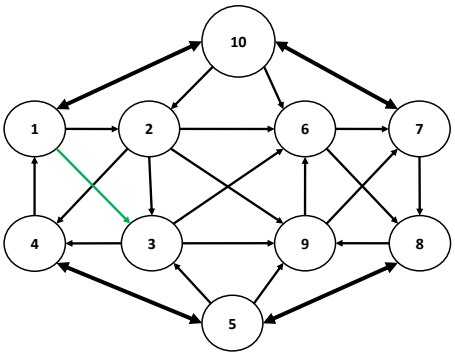
We now explain how to construct two new graphs from a given arbitrary directed graph G. This construction is used in the branching step of the B&C algorithm. Let $p, q \in V$ be two adjacent vertices in G_s such that there is a cycle in G_s containing the arc pq. Consider the directed graphs $H^{pq} = (V'', A'')$ and $F^{pq} = (V', A')$ defined as follows.

The set of vertices of H^{pq} is $V'' = V$ and the set of arcs $A'' = A \cup \{qp\}$. The set of vertices V' of F^{pq} contains V and two additional vertices (pq_1) and (pq_2); that is $V' = V \cup \{(pq_1), (pq_2)\}$. The set of arcs A' is built as follows.

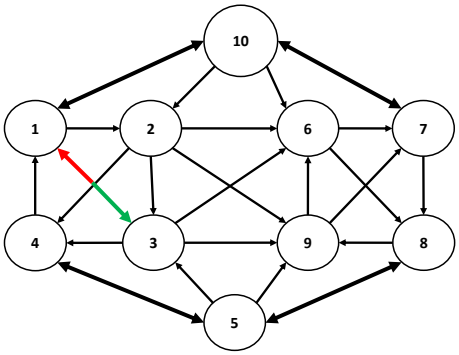
1. Every arc in $A \setminus \{pq\}$ is an arc in A'.
2. For every **single** incoming arc ap into p, add an arc $a(pq_2)$ in A'.
3. For every **single** outgoing arc qa out of q, add an arc $(pq_2)a$ in A'.
4. Finally, add the arcs: $p(pq_1)$, $(pq_1)p$, $q(pq_1)$, $(pq_1)q$, $(pq_1)(pq_2)$, $(pq_2)(pq_1) \in A'$.

Example 5.7. *Figure 5.2 illustrates the construction of H^{13} and F^{13} from the directed graph G by branching on the arc $1 \to 3$.*

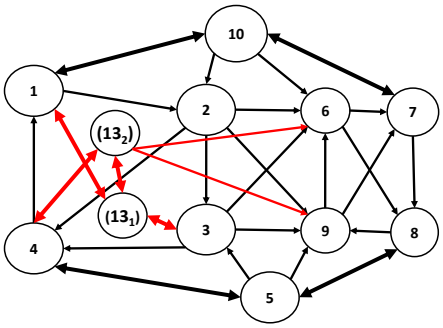
The graph H^{pq} corresponds with a setting where p and q receive different colors, whereas the graph F^{pq} represents the setting where p and q have the same color in any feasible coloring. Informally, the graph H^{pq} arises from G by adding the arc qp; the graph F^{pq} arises from G by replacing the arc pq by a node (pq_2), such that each single arc in G entering p (or leaving q) now enters (pq_2) (or leaves (pq_2)). Further,



(a) The initial graph G



(b) The graph H^{13}



(c) The graph F^{13}

Figure 5.2: *Illustration of the construction of H^{13} and F^{13}. In the graphs, a double-direction arc ($\leftrightarrow$) represents a cycle of length two between the considered vertices.*

we add a node (pq_1) in F^{pq} to enforce that the vertices p, q and (pq_2) have the same color. Remark that each cycle in G containing the arc pq corresponds to a cycle in F^{pq} containing the vertex (pq_2).

Proposition 5.8. *Let p and q be two adjacent vertices contained in a cycle in G_s. F^{pq} or H^{pq} is a YES instance of the acyclic 2-coloring problem if and only if G is a YES instance.*

Proof. $\Leftarrow$) Assume that the graph G can be partitioned into two acyclic subgraphs. There are two options: either the vertices p and q have the same color or they do not.

If p and q have different colors, then the directed graph H^{pq} can be partitioned into two acyclic subgraphs according to the coloring of G; clearly, the 2-cycle $[p, q, p]$ is not

monochromatic.

On the other hand, if p and q have the same color, we prove that the directed graph F^{pq} can be partitioned into two acyclic subgraphs. Consider the following coloring of V'. Each vertex $a \in V$ receives the color obtained by the coloring of G. The vertex (pq_2) is given the color of p and q while (pq_1) receives the color different from that of p and q. We next prove that the subgraphs induced by the color classes are acyclic. Suppose there exists a monochromatic cycle $\mathcal{C}$ in F^{pq}. $\mathcal{C}$ cannot contain (pq_1) because all its neighbors have a different color. $\mathcal{C}$ must contain (pq_2) because otherwise it would lie in G as well. Consider the part of the cycle $x \to (pq_2) \to y$. Now change cycle $\mathcal{C}$ into cycle $\mathcal{C}'$ by replacing $x \to (pq_2) \to y$ by $x \to p \to q \to y$. This would be a monochromatic cycle in G.

$\Rightarrow$) Suppose that F^{pq} or H^{pq} can be partitioned into two acyclic subgraphs. Clearly, a partition of H^{pq} into two acyclic subgraphs immediately yields a partition of G into two acyclic subgraphs. On the other hand, if F^{pq} can be partitioned into two acyclic subgraphs, we consider the coloring of G defined as follows: $p \in V$ receives the same color as in the coloring of F^{pq}. The partition of F^{pq} induces a partition of $G \setminus \{pq\}$ (the graph G minus the arc pq) into two acyclic subgraphs because $G \setminus \{pq\}$ is a subgraph of F^{pq}. Consequently, if there is a monochromatic cycle $\mathcal{C}$ in G, then $\mathcal{C}$ must use the arc pq. However, since a cycle in G that uses the arc pq corresponds to a cycle in F^{pq} using (pq_2), there would be a monochromatic cycle in F^{pq}: a contradiction. □

A formal description of the B&C algorithm for deciding G is given by BnC(G).

Algorithm 5.3 BnC(G)

1: determine G_2, G_s
2: if G_2 is not bipartite, then return NO
3: if G_s is acyclic, then return YES
4: choose an arc pq on a cycle in G_s
5: determine H^{pq}, F^{pq}
6: if BnC(H^{pq}) then return YES
7: else return BnC(F^{pq})

The branching strategy involves the selection of two adjacent vertices p and q in G_s such that there is a cycle in G_s containing the arc pq. The following result proves that using this branching strategy, the B&C algorithm terminates after a finite number of iterations.

Proposition 5.9. *The B&C algorithm terminates after a finite number of iterations.*

Proof. To prove this result we introduce the following parameter of a graph. Given a directed graph G and its single directed graph G_s, we define the *total length of all distinct cycles* in G_s, denoted $L(G)$, as the number of arcs in all distinct cycles in G_s. Notice that an arc is counted as many times as it appears in distinct cycles. We prove that for any two adjacent vertices $p, q \in G_s$ such that there is a cycle in G_s containing the arc pq, $L(H^{pq}) < L(G)$ and $L(F^{pq}) < L(G)$. Clearly, $L(H^{pq}) < L(G)$ because at least one cycle in G_s disappears in H_s^{pq} since the arc pq is not in H_s^{pq}. On the other hand, $L(F^{pq}) < L(G)$ because any cycle in G_s that uses the arc pq has become one arc shorter in the single directed graph F_s^{pq} of F^{pq}. Every cycle in G_s that does not use the arc pq is still present in F_s^{pq}, and so has the same contribution to $L(G)$ and $L(F^{pq})$. □

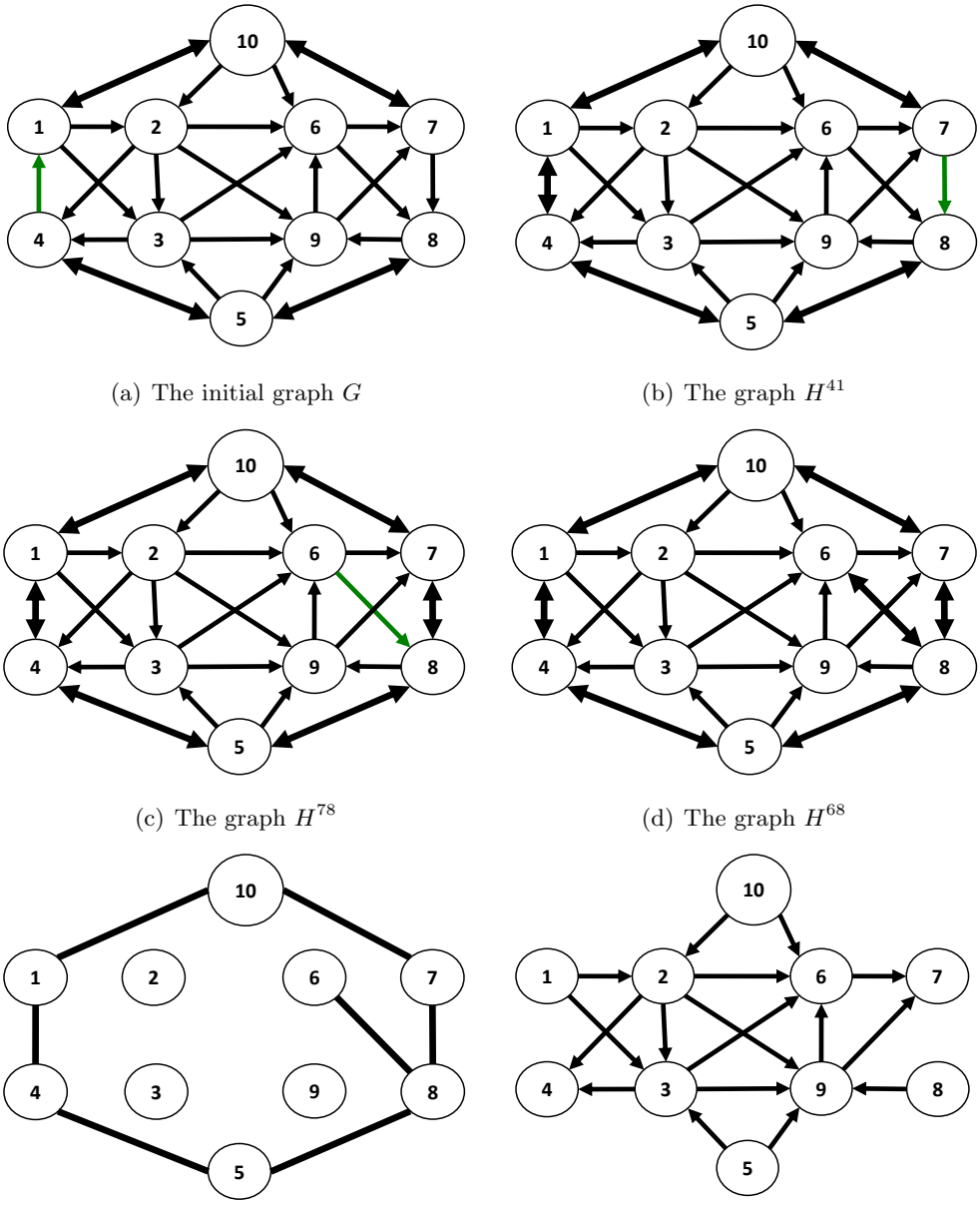
Theorem 5.10. Correctness of the branch-and-check algorithm
Suppose that the B&C algorithm is run on G. Then, its execution terminates after a finite number of iterations and the decision corresponds to the decision for the original graph G.

Proof. This follows from Proposition 5.5, Proposition 5.8 and Proposition 5.9. □

Example 5.11. *Figure 5.3 illustrates the application of the B&C algorithm. The initial graph G, Figure 5.3(a), is the graph in Figure 5.2(a). By branching on the arc $4 \to 1$, we obtain two graphs (H^{41} and F^{41}) and the graph H^{41}, Figure 5.3(b), is selected as the next graph to investigate. In that graph, we choose to branch on the arc $7 \to 8$. The result is two new graphs, H^{78} and F^{78}, and we select H^{78} depicted by Figure 5.3(c) as the next graph. By branching on the arc $6 \to 8$ in H^{78}, we obtain the graphs H^{68} and F^{68}. Considering the graph H^{68} given by Figure 5.3(d), the associated 2-undirected graph depicted by Figure 5.3(e) is bipartite and the single directed H_s^{68} depicted by Figure 5.3(f) is acyclic. Therefore, the initial graph G is a YES instance of the acyclic 2-coloring problem. One acyclic 2-coloring of G has color classes $\{1, 2, 3, 5, 6, 7, 9\}$ and $\{4, 8, 10\}$.*

5.4.4 Refinements

In this section, we present two dominance rules which can be used to reduce the size (the number of arcs and/or the number of vertices) of the directed graph G.



(a) The initial graph G

(b) The graph H^{41}

(c) The graph H^{78}

(d) The graph H^{68}

(e) 2-undirected graph of H^{68}

(f) Single directed graph of H^{68}

Figure 5.3: *Illustration of the B&C algorithm.*

Dominance rule 1: This rule is characterized by the following lemma.

Lemma 5.12. *Given a vertex p in G, if the outdegree or the indegree of p is less than or equal to one then the vertex p can be removed from G without changes in the final outcome.*

Proof. Let $G = (V, A)$ be the directed graph and p be a vertex of G with outdegree or indegree less than or equal to one. Let G_p be the subgraph of G obtained by removing the vertex p and all incident arcs (arcs from p and arcs entering p). Clearly, if G_p cannot be partitioned into two acyclic subgraphs, then G cannot be partitioned into two acyclic subgraphs.

On the other hand, suppose that G_p can be partitioned into two acyclic subgraphs. If the degree of p equals zero, we simply add p to any one of the subgraphs forming the partition of G_p, and the resulting partition is a partition of G into two acyclic subgraphs. If the indegree (outdegree) of p equals one, let q be the vertex of G_p such that the arc qp (pq) exists in G. Then we add the vertex p to the subgraph not containing q. Clearly, the resulting partition is a partition of G into two acyclic subgraphs. □

Dominance rule 2: The aim of this rule is to identify and remove from the graph all single arcs not involved in any cycles in G_s. It proceeds as follows. The vertices of G_s are partitioned into SCCs; notice that such a partition is unique. The arcs between two distinct SCCs are deleted since they are not part of any cycle in G_s.

Notice that if either Dominance rule 1 or Dominance rule 2 removes at least one arc or at least one vertex, then the repeated application of the other rule may further remove new arcs or vertices. For both the cycle-identification algorithm and the backtracking algorithm, these rules can be applied before starting the algorithm. For the branch-and-check algorithm, however, these rules can be applied both before starting the algorithm and at every node of the branching tree since a new directed graph (either H^{pq} or F^{pq}) is built.

5.4.5 Classes of easy graphs

This subsection is devoted to the identification of classes of directed graphs for which the corresponding acyclic 2-coloring problem is always a YES instance. The first class is the class of directed acyclic graphs (DAG). The second class of graphs is the class of line graphs (LG). This class of graphs has been identified as a class of directed graphs

for which the acyclic 2-coloring problem is always a YES instance (see Lemma 4.13 in Chapter 4). The third class of easy graphs is the class of partial directed line (PDL) graphs, see e.g. (Apollonio and Franciosa, 2007). These are graphs obtained from line graphs by removing a set (possibly empty) of arcs. Clearly, the PDL class of graphs contains the class of directed line graphs. Combining the fact that a line graph is a YES instance of the acyclic 2-coloring problem and the fact that any subgraph of an acyclic graph is also acyclic, we conclude that each graph G in the class of PDL graphs is a YES instance of the acyclic 2-coloring problem.

Let us define the following class of directed graphs. The class $\mathcal{G}_i^<$ (with i a positive integer) contains all connected directed graphs with each vertex having degree at most i; and there is at least one vertex with degree less than i. The next corollary follows from repeated application of Lemma 5.12.

Corollary 5.13. *Every graph in $\mathcal{G}_4^<$ is a YES instance of the acyclic 2-coloring problem.*

Further, some results obtained for undirected planar graphs can be extended to oriented planar graphs. These results are included in the following lemma.

Lemma 5.14. *(i) Each oriented planar graph of maximum degree 4 is a YES instance of the acyclic 2-coloring problem.*

(ii) Each oriented outerplanar graph is a YES instance of the acyclic 2-coloring problem.

Proof. This follows from the fact that a similar result is true for undirected planar graphs of maximum degree 4 (Raspaud and Wang, 2008) and for undirected outerplanar graphs (Aifeng and Jinjiang, 1991; Goddard, 1991). □

5.5 Implementation issues

In this section, we present several issues related to the implementation of every algorithm described in Section 5.4.

Bipartiteness, acyclicness and strongly connected components

An adapted breadth-first-search algorithm (Cormen et al., 2001) is implemented to check whether G_2 is bipartite. The same algorithm is also adapted to verify for two given disjoint subsets of vertices, RED and BLUE, whether G_2 is bipartite given that

all the vertices in RED are on one side and those in BLUE are on the other side. A topological ordering algorithm (Ahuja et al., 1993) is used for testing acyclicness of G_s and any induced subgraph $G_s(A)$, where A is a subset of vertices. Tarjan's algorithm (Tarjan, 1972) is used to identify the SCCs of a given graph.

Cycle-identification algorithm

The intuition behind the implementation of this algorithm is that "large" cycles (cycles having many vertices) are likely to share some vertices and arcs with "small" cycles (cycles having few vertices). Therefore, feasibly coloring small cycles may lead to a feasible coloring of large cycles at the same time. In our implementation, we start by including only the smallest cycles and gradually add larger cycles.

Hence, the relaxed IP instance initially contains only constraints coming from cycles of length 2. Therefore, throughout the algorithm we search a monochromatic cycle only in the single directed graphs induced by the color classes. Given a color class, we use the Floyd-Warshall algorithm (Ahuja et al., 1993; Cormen et al., 2001) to find (if there exist) monochromatic cycles which use the smallest number of vertices. If a monochromatic cycle is found, we add the corresponding pair of constraints to the IP, and the IP instance is solved again. The IP instances are solved using the MIP-solver of CPLEX; once a feasible solution is found we stop the solver.

Backtracking algorithm

Branching strategy: The branching strategy of the backtracking algorithm involves the selection of a vertex $p \in V$ which is neither in RED nor in BLUE. We investigate two choices: the first one is simply the first uncolored vertex found while the second choice is an uncolored vertex with the highest degree; ties are broken arbitrarily.

Propagation rule: This rule is applied any time that a new vertex p is added either to RED or to BLUE. It works as follows: suppose a vertex p is added to RED (BLUE). Then for any vertex q which is such that the arcs pq and qp exist (this is equivalent to p and q being adjacent in the undirected graph G_2), if q is not yet in BLUE (RED) then we add q to BLUE (RED). The procedure is repeated for every new vertex added either to RED or to BLUE.

Node selection: Our main objective is to color all the vertices as soon as possible (provided such coloring is possible). Therefore, we use a *depth-first-search* strategy.

Branch-and-check algorithm

Branching strategy: This branching strategy selects a single arc pq which is such that there is a cycle in G_s containing that arc. Before choosing the arc pq, the graph G_s is first reduced by deleting all single arcs linking vertices of the same connected component in G_2 with different colors obtained from the bipartiteness test, and a single arc between vertices of the same connected component in G_2 with the same color is not considered for branching. We investigate two different choices of the arc pq. The first choice is the first arc pq found that meets the above restriction. The second choice is an arc pq with p having the highest degree possible, breaking ties arbitrarily. In both cases, if in addition there is no path in G_s from p to q other than the arc pq, we define a simplified version of $F^{pq} = (V', A')$ by merging p and q. V' contains a vertex (pq) and all vertices in V except p and q such that $|V'| = |V| - 1$ while A' is built as follows. First, every arc $ab \in A$ with $a, b \notin \{p, q\}$ is an arc in A'. Second, for every single incoming arc ax to x with $x \in \{p, q\}$, (respectively every single outgoing arc xa from x), add an arc $a(pq)$ (respectively $(pq)a$) in A' while avoiding the repetition of arcs.

Branch-pruning criterion: This branch-pruning criterion considers each connected component of G_2 and the coloring of its vertices given by the bipartiteness test. If there exists a color class in a connected component which is such that the induced single directed graph is cyclic, then any graph built at a child node of that node is a NO instance of the acyclic 2-coloring problem. Therefore, that node is pruned.

Node selection: For the branch-and-check algorithm, we wish to reach a node with a YES answer as soon as possible (provided it exists). We again use a *depth-first-search* strategy.

5.6 Computational experiments

All algorithms have been coded in C using Visual Studio C++ 2005; all the experiments were run on a Dell Optiplex 760 personal computer with Pentium R processor with 3.16 GHz clock speed and 3.21 GB RAM, equipped with Windows XP. CPLEX 10.2

was used for solving the IP instances. Below, we first provide some details on the real-life instances and the generation of random data sets and subsequently, we discuss the computational results.

5.6.1 Data

The three algorithms were tested both on real-life graphs and on randomly generated graphs. We first present the real-life instances and next we describe how random instances were generated. The instances described in this section can be found at http://www.econ.kuleuven.be/public/NDBAC96/acyclic_coloring.htm

Real-life data

The graphs presented below come from Table 4.1 in Chapter 4. Table 5.1 reports the properties of the real-life instances.

Table 5.1: *Properties of the real-life instances*

Instance	1	2	3	4	5	6	7	8	9	10	11	12
# vertices (n)	22	48	68	95	118	139	226	279	294	410	755	4384
# arcs (m)	53	169	297	513	699	985	1979	2012	2427	3660	10113	124321
# arcs/n	2.40	3.52	4.36	5.40	5.92	7.09	8.76	7.21	8.26	8.93	13.39	28.36

Random data

We have randomly generated graphs with n vertices, where n takes the values 50, 100, 200, 500 and 1 000. These graphs are generated in such a way that they are connected and contain at least one cycle. To diversify as much as possible the instances, we vary the density D of the graph, which equals the number of arcs present in the graph divided by the total number of possible arcs.

The graphs are generated using a two-phase procedure. During the first phase, for each value of n, 400 graphs are randomly generated with 40 different densities, starting from a lower bound of 2.5% for $n = 50$, 1.5% for $n = 100$, 1% for $n = 200$ and 0.5% for $n = 500$ and $n = 1\,000$; and increased with a step of 0.5%. The lower bound is obtained by taking the first multiple of 0.5 greater than or equal to the smallest density for which a connected and cyclic graph can be generated given the number n of vertices.

For every value of D, 10 directed graphs with $m = \lceil D \times (n^2 - n) \rceil$ arcs are generated. Therefore, in total we have $400 \times 5 = 2\,000$ test instances for the first phase.

After preliminary computation on the graphs obtained in the first phase, we identify for each value of n a *critical interval* containing the densities for which we encountered at least one YES instance and at least one NO instance. We observe that densities in this critical interval are exactly those for which potential hard graphs (requiring long running times) can be found. Notice that for each density not in the critical interval, we have obtained for the instances generated in first phase either always a YES or always a NO answer. This, however, does not mean that there is no density outside the critical interval, for which both YES instances and NO instances exist. For a given n, we generate additional graphs with the densities given in Table 5.2.

Table 5.2: *Densities of the graphs generated in the second phase*

n	density (D)			
	from	to	step	total
50	8%	15.75%	0.25%	32
100	3.05%	8.95%	0.05%	119
200	2.01%	3.99%	0.01%	199
500	0.8%	1.498%	0.002%	350
1 000	0.3%	1.2%	0.002%	451

For every value of the density, 100 instances of directed graph are randomly generated following the procedure described above, leading to $1151 \times 100 = 115100$ additional graph instances for the second phase.

5.6.2 Computational results

In this section, we examine different implementations of every algorithm for the set of 50-vertex graphs generated during the first phase (these are 400 graphs in total). The three algorithms are subsequently compared based on their best implementation both on randomly generated graphs and on real-life instances. Finally, we study the phase transition (Hogg, 1985; Monasson et al., 1999) of the acyclic 2-coloring problem as function of the number of arcs divided by n. Throughout this section, the CPU time is expressed in seconds.

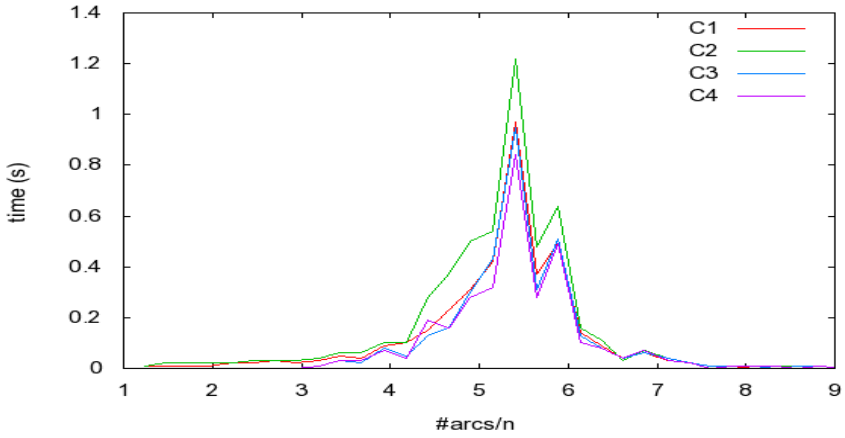


Figure 5.4: *Average CPU time of four different implementations of the cycle-identification algorithm for* 50*-vertex random graphs generated during the first phase.*

Comparison of different implementations of every algorithm

Different implementations of every algorithm have been compared. A time limit of ten minutes is used to stop the algorithm and when this happens, we output undecided.

Cycle-identification algorithm

Figure 5.4 displays the average CPU time as function of the number of arcs divided by n, of four different implementations of the cycle-identification algorithm. The first implementation, identified by **C1**, is CycleId(G) where if there is a monochromatic cycle in one color class, we do not search for a monochromatic cycle in the other color class. Further, at each iteration, we add all the pairs of constraints corresponding with cycles of length 3 in that color class. If there is no such cycle, we search and (if found) add all the pairs of constraints corresponding with cycles of length 4. If there is no cycle of length 4 then we add the pair of constraints corresponding with one monochromatic cycle of any length found. The first color class investigated is the class of red vertices. The second implementation, **C2**, is similar to the first one, except that irrespective of finding monochromatic cycles in the first color class, we search for monochromatic cycles in the second color class. The third implementation, **C3**, considers the first implementation with in addition the use of dominance rules while the fourth implementation, **C4**, adds the dominance rules to the second implementation.

A comparison of the different plots in Figure 5.4 shows that the four implementa-

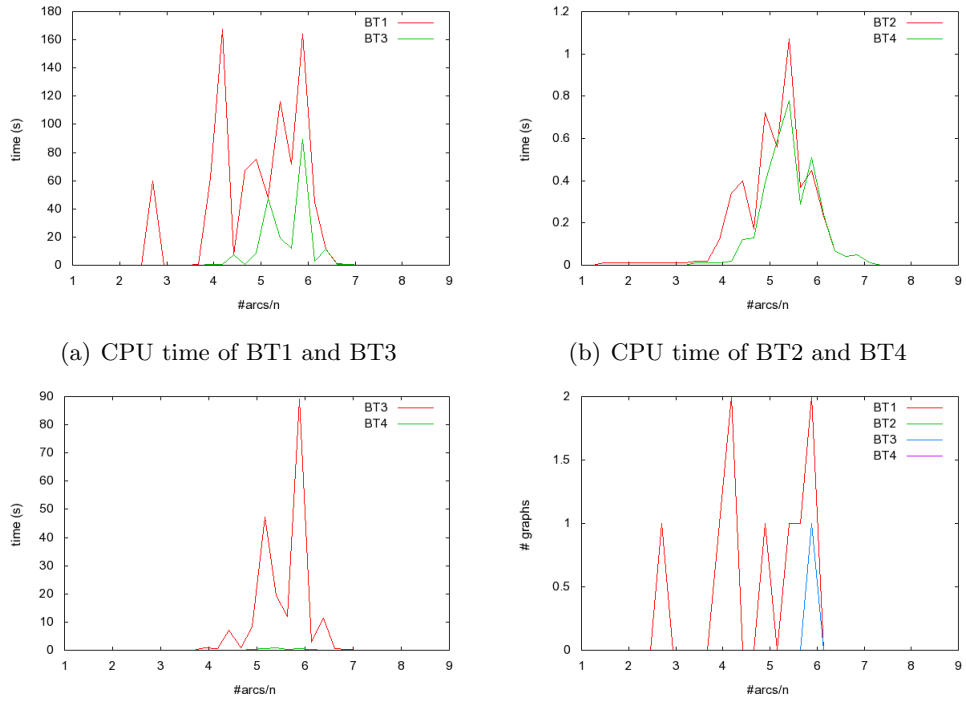
tions use comparable average CPU time. The plot of **C4**, however, is usually below that of the other implementations. In the rest of this chapter, implementation **C4** is adopted for the cycle-identification algorithm, meaning that whenever we refer to this algorithm, we imply that the implementation used is **C4**.

We would like to mention that several other implementations of this algorithms have been tested. Their average CPU time, however, was usually substantially higher than those presented above in Figure 5.4. We have implemented, among others, a variant of CycleId(G) where in the IP formulation a coefficient in the objective function, either 1 and -1, is randomly chosen for each variable x_i, in an attempt to balance the objective function which initially maximizes the number of red vertices. We have also implemented a variant of CycleId(G) where at each iteration, the pair of constraints corresponding with only one monochromatic cycle (of minimal length) is added to the IP instance.

Backtracking algorithm

Figure 5.5 displays the average CPU time, as function of the number of arcs divided by n, of four different implementations of the backtracking algorithm. The first implementation, identified by **BT1**, is the pseudocode BT(RED, $BLUE$, G) with in addition the use of the propagation rule and we branch on the first uncolored vertex found. The second implementation, **BT2**, is similar to the first one, but we choose an uncolored vertex with the highest degree. The third implementation, **BT3**, considers the first implementation with in addition the use of dominance rules while the fourth implementation, **BT4**, adds the dominance rules to the second implementation.

A comparison of plots in Figure 5.5(a) and Figure 5.5(b) shows that the first implementation, **BT1**, has an average CPU time higher than that of the third implementation (**BT3**). The average CPU time of **BT2** is also usually higher than that of **BT4**. These comparisons indicate the positive effect of the use of dominance rules. On the other hand, **BT4** has an average CPU time much more smaller than that of **BT3** (see Figure 5.5(c)). Further using **BT4**, all the instances are solved within a time limit of ten minutes while there is one instance not decided after the time limit when we use **BT3** (see Figure 5.5(d)). To conclude, the implementation **BT4** is used for the rest of experiments when we applied the backtracking algorithm.



(a) CPU time of BT1 and BT3

(b) CPU time of BT2 and BT4

(c) CPU time of BT3 and BT4

(d) Number of undecided instances

Figure 5.5: *Average CPU time of four different implementations of the backtracking algorithm for* 50*-vertex random graphs generated during the first phase.*

B&C algorithm

Figure 5.6 plots the average CPU time of six different implementations of the B&C algorithm. The first implementation, **BnC1**, is the B&C algorithm as described by the pseudocode BnC(G), with in addition the use of the branch-pruning criterion and the arc selected is the first arc found. The second implementation, **BnC2**, is similar to the first one, except that we choose an arc pq with vertex p having the highest degree possible. The third implementation, **BnC3**, considers the first implementation with dominance rules applied at the root node and the fourth implementation, **BnC4**, considers the second implementation with dominance rules also applied at the root node. The fifth implementation, **BnC5**, considers the first implementation with dominance rules at every node of the branching tree while the sixth implementation, **BnC6**, considers the second implementation with dominance rules at every node of the branching

tree.

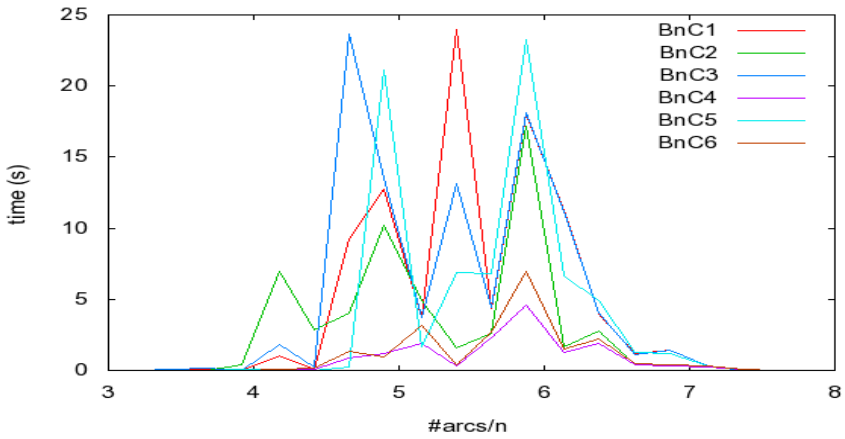


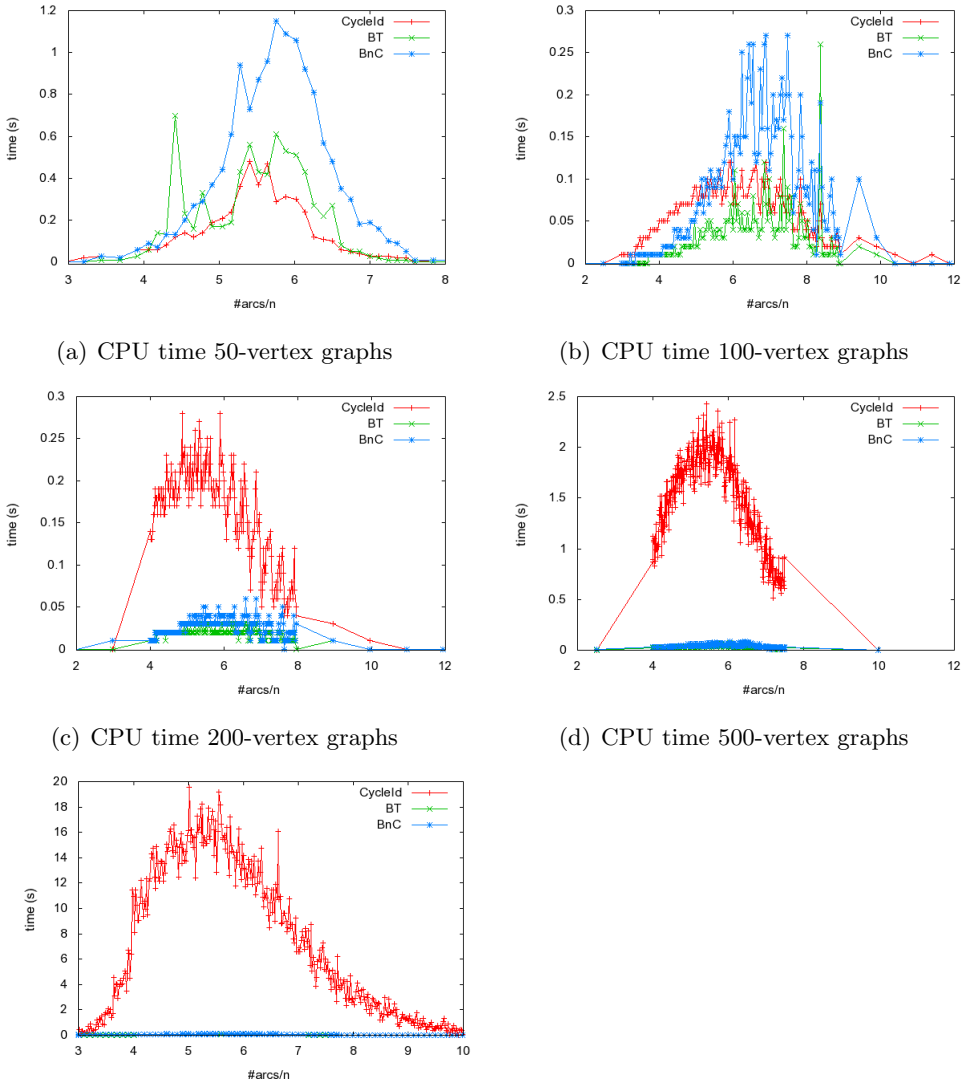
Figure 5.6: *Average CPU time of six different implementations of the B&C algorithm for* 50*-vertex random graphs generated during the first phase.*

A comparison of the six implementations based on the average CPU time is the following. The three implementations using the branching strategy which selects the first arc encountered (**BnC1**, **BnC3** and **BnC5**) have higher average CPU time than the average CPU time of implementations where the arc pq is chosen in such a way that p has the highest degree (**BnC2**, **BnC4** and **BnC6**). Among these last implementations, **BnC4** usually spends the smallest CPU time. We use the implementation **BnC4** of the B&C algorithm for the remaining experiments.

Solving random instances

We compare the three algorithms based on their best implementation on random graphs. In Figure 5.7 we plot, for every value of n, the average CPU time of every algorithm as function of the number of arcs divided by n. Figure 5.7(a) shows the average CPU time for the 50-vertex graphs. The B&C algorithm (BnC) usually reports a higher CPU time than the other algorithms. However, the highest average CPU time is less than 1.2 seconds. The cycle-identification algorithm (CycleId) usually uses, on average, the smallest CPU time. For 100-vertex graphs (Figure 5.7(b)), we see that the average CPU time of CycleId is usually between that of BnC and that of the backtracking algorithm (BT), with BT using, in most cases, the smallest average time. For the large graphs (with more than 100 vertices, see Figures 5.7(c), 5.7(d) and 5.7(e)),

the average CPU time reported for CycleId increases with the value of n, while those of BnC and BT are stable, comparable and usually below one second.



(a) CPU time 50-vertex graphs

(b) CPU time 100-vertex graphs

(c) CPU time 200-vertex graphs

(d) CPU time 500-vertex graphs

(e) CPU time 1 000-vertex graphs

Figure 5.7: *Average CPU time of every algorithm for random graphs.*

Table 5.3: *CPU time of every algorithm for the real-life instances*

Instance	1	2	3	4	5	6	7	8	9	10	11	12
CycleId	0.00	0.00	0.01	0.03	0.06	0.09	0.11	0.20	0.28	0.72	3.97	1812.24
BT	0.00	0.00	0.02	0.03	0.06	0.09	0.36	0.28	0.31	0.28	3.45	283.72
BnC	0.00	0.00	0.01	0.02	0.05	0.09	0.59	0.05	0.28	0.11	3.84	612.41

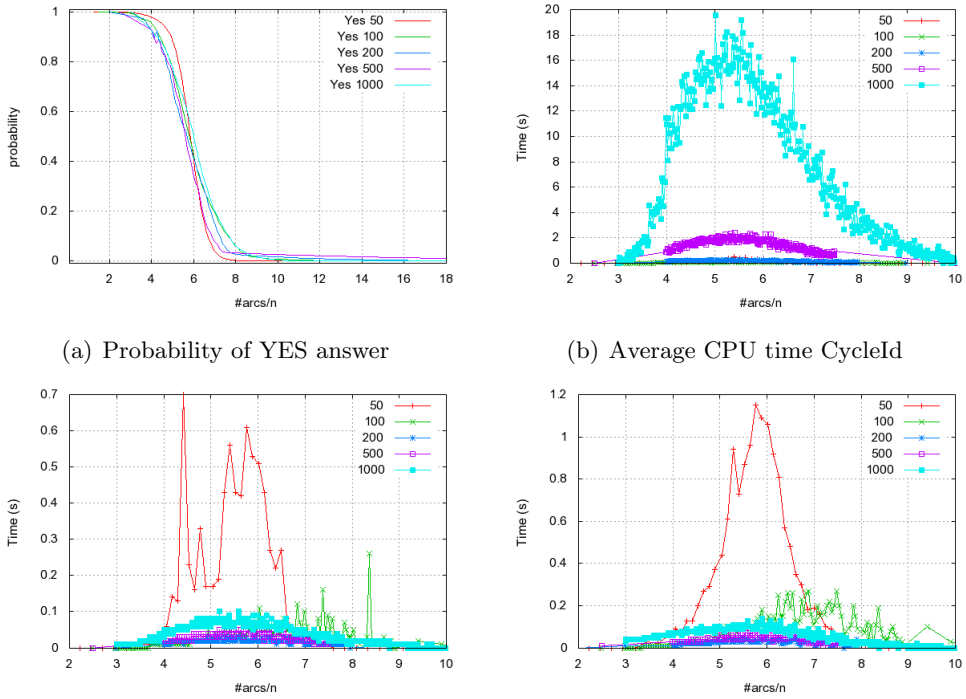
Solving real-life instances

Table 5.3 reports the CPU time of every algorithm when applied to real-life instances. We see that the backtracking algorithm (BT) reports the best CPU time for five instances out of 12 while, the cycle-identification algorithm (CycleId) achieves the best CPU time for six instances and the B&C algorithm (BnC) has the best CPU time for nine instances. For the largest instance with 4384 vertices, however, BT spends less than five minutes, compared to about ten minutes for BnC and about 30 minutes for CycleId.

Phase transition analysis

In this section, we investigate the transition from a high to a low YES probability as function of the number of arcs divided by n (subsequently called *parameter* in this section). Further, we show how the CPU time of every algorithm varies as function of that parameter.

Figure 5.8 presents the probability of a YES answer as well as the average CPU time of every algorithm as function of the parameter. Figure 5.8(a) shows the probability of YES answer as function of the parameter. The plots in Figure 5.8(a) are Bézier approximations (Farin, 2006) of the real plots. This approximation is used mainly to render the plots smoother. For every value of n, the plot has three regions. In the first region, where the value of the parameter is between 0 and 3, almost all the generated instances have a YES answer. The second region, with the value of the parameter between 3 and 8, is called *critical interval* and contains classes of graphs for which both YES instances and NO instances are present. The last region, with the value of the parameter greater than 8, contains graphs for which the probability of YES is almost zero. Overall, we remark that the five plots are similar and the *threshold* value of the parameter, for which the probability of YES answer is equal to $\frac{1}{2}$, is almost the same for every n and is close to 5.75.



(a) Probability of YES answer

(b) Average CPU time CycleId

(c) Average CPU time BT

(d) Average CPU time BnC

Figure 5.8: *Probability of YES answer and average CPU time of every algorithm.*

The plots in Figures 5.8(b), 5.8(c) and 5.8(d) are obtained using the data that are used to generate the plots in Figure 5.7, but here the plots are grouped by algorithm. Figure 5.8(b) plots the average CPU time of CycleId for every value of n. The plots respect the three regions described above. For the first and the third region, the average CPU time is very close to zero while in the critical interval, we have a non-negligible CPU time, showing an easy-hard-easy transition. Further, CycleId has an average CPU time which increases with the value of n, which probably occurs simply because when n increases the IP instance becomes more difficult to solve. Figure 5.8(c) plots the average CPU time of BT for every value of n. The easy-hard-easy transition is also observed here. However, unlike CycleId, BT spends more time in deciding 50-vertex and 100-vertex instances in the critical interval than in deciding instances with more vertices. This decrease in CPU time as the value of n increases stops beyond $n = 200$. The high variability of average CPU time is due to the fact that for very few instances,

the algorithm requires more than one second to decide. In order words, among the instances generated there are very few hard instances. In Figure 5.8(d), the plots of the average CPU time of BnC for every value of n exhibit characteristics similar to those observed for BT. A possible explanation for this decrease in average CPU time is the following: when the value of n increases, the size (number of edges) of the undirected graph G_2 increases, making the bipartiteness test used by both BT and BnC more efficient in detecting NO instances. At the same time, both the propagation rule (used by BT) and the branch-pruning criterion (used by BnC) become stronger, reducing the number of possible nodes to investigate in order to arrive at a YES answer.

In general, for every value of n and irrespective of the algorithm used, the highest average CPU time is usually obtained for values of the parameter around the threshold value. Further, there is a high variability of the average CPU time and there are few hard instances.

5.7 Summary and conclusions

In this chapter, we studied the problem of coloring the vertices of a given directed graph using two colors such that no monochromatic cycle occurs. Applications of this problem include testing of the Collective Axiom of Revealed Preference defined in Chapter 4. We show that the problem is NP-complete, even for oriented graphs and we prove that the existence of a constant-factor-approximation algorithm is unlikely for an optimization version which maximizes the number of vertices that can be colored using two colors while avoiding monochromatic cycles. We present a integer-programming algorithm based on cycle identification, a backtracking algorithm and a branch-and-check algorithm to solve the problem exactly. We compare the three algorithms based on their CPU time, both on real-life instances and on random graphs. For the latter set, graphs with up to 1 000 vertices are solved in few seconds by every algorithm. We also study empirically the phase transition of the problem. We find that the acyclic 2-coloring problem exhibits an easy-hard-easy transition and that hard instances are difficult to generate. For real-life instances coming from the study of rationality of consumption behavior, all the instances are decided using every algorithm and the largest instance with 4384 vertices is solved using the backtracking algorithm in less than five minutes, while the branch-and-check algorithm spends about ten minutes to decide that instance and the cycle-identification algorithm about 30 minutes.

An important research direction that might be pursed in the future is the study of the acyclic 2-coloring problem for some special graphs, including oriented planar graphs. Further, it might be interesting to investigate in more details the optimization variants of the acyclic 2-coloring problem.

Chapter 6

The complexity of testing the Collective Axiom of Revealed Preference

In this chapter, we prove that the problem of testing the Collective Axiom of Revealed Preference (CARP) defined in Chapter 4 is an NP-complete problem. This proof uses a reduction from the Not-All-Equal-3Sat problem.

6.1 Introduction

In this chapter we investigate the complexity of deciding whether an observed data set satisfies the Collective Axiom of Revealed Preference (CARP) defined in Chapter 4. We consider a two-member household that operates in an economy with N goods. At times $t = 1, 2, \ldots, T$, the household purchases a certain quantity of each of the goods $q_t \in \mathbb{R}^N_+$ (also known as a *bundle*), at corresponding prices $p_t \in \mathbb{R}^N_{++}$. We call a pair of N-vectors (p_t, q_t) an *observation*, and the set of observations denoted by $S = \{(p_t, q_t) : \ t \in \mathbb{T} \equiv \{1 \ldots, T\}\}$ is called the data set. For the sake of simplicity, throughout this chapter we will also call $t \in \mathbb{T}$ an observation while referring to (p_t, q_t).

The following definition recalls the rules defining CARP; for more details we refer to Chapter 4.

This chapter is the result of a collaboration with Frits Spieksma and is available as: F. Talla Nobibon and F.C.R. Spieksma. On the complexity of testing the collective axiom of revealed preference. *Mathematical Social Sciences*, 60:123–136, 2010.

Definition 6.1 (CARP)**.** Let $S = \{(p_t, q_t); t \in \mathbb{T}\}$ be a set of observations. S satisfies CARP if there exist hypothetical relations H_0^m, H^m for each member $m \in \{1, 2\}$ that meet for all $s, t, t_1, t_2 \in \mathbb{T}$:

Rule 1**:** if $p_s q_s \geq p_s q_t$ then either $q_s H_0^1 q_t$ or $q_s H_0^2 q_t$;

Rule 2**:** if $p_s q_s \geq p_s q_t$ and $q_t H^m q_s$ then $q_s H_0^\ell q_t$ with $\ell \neq m$;

Rule 3**:** if $p_s q_s \geq p_s(q_{t_1} + q_{t_2})$ and $q_{t_1} H^m q_s$ then $q_s H_0^\ell q_{t_2}$ with $\ell \neq m$;

Rule 4**:** if $p_s q_s > p_s q_t$ then either $\neg(q_t H^1 q_s)$ or $\neg(q_t H^2 q_s)$;

Rule 5**:** if $p_s q_s > p_s(q_{t_1} + q_{t_2})$ then either $\neg(q_{t_1} H^1 q_s)$ or $\neg(q_{t_2} H^2 q_s)$;

where $p_s q_s$ represents the scalar product.

The problem of testing whether a given data set S satisfies CARP can be phrased as the following decision problem.

INSTANCE: A data set $S = \{(p_t, q_t) : \ t \in \mathbb{T}\}$.
QUESTION: Does the data set satisfies CARP? In other words, do there exist H_0^1 and H_0^2 such that *Rules* 1-5 are satisfied?

The main objective of this chapter is to show that testing whether a given data set S satisfies CARP is NP-complete and this is done in Section 6.2; we conclude in Section 6.3.

6.2 Complexity result

In this section we prove that testing CARP is NP-complete. The proof uses a reduction from the Not-All-Equal-3Sat problem, which is defined as follows.

INSTANCE: Set $X = \{x_1, \ldots, x_n\}$ of n variables, collection $C = \{C_1, \ldots, C_m\}$ of m clauses over X such that each clause $C_\ell \in C$ has $|C_\ell| = 3$.
QUESTION: Is there a truth assignment for X such that each clause in C has at least one true literal and at least one false literal?

Garey and Johnson (1979) proved that the Not-All-Equal-3Sat problem is NP-complete.

In the proof, we consider instances of the Not-All-Equal-3Sat problem where no variable occurs more than once in the same clause. This is without loss of generality, since, given an instance of Not-All-Equal-3Sat where a clause contains the same variable twice, we can simplify that clause to get a clause with two distinct variables. By appropriately adding a new variable and a new clause, we can transform that instance into an instance of the Not-All-Equal-3Sat problem where no variable occurs more than once in the same clause. As illustration, the clause $(x_1 \vee x_2 \vee x_2)$ can be replaced by $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x}_3)$.

The idea behind the proof is the following: for each variable and for each clause of the Not-All-Equal-3Sat instance, we build a set of observations. Each of these observations concerns a number of goods; in particular, we have a price-vector, and a quantity-vector for each observation. By choosing appropriate values for the prices and the quantities, we establish for each pair and for each triple of observations the inequality desired. Next, the implications for the hypothetical relations H_0^1 and H_0^2 induced by *Rules* 1-5 are such that their existence is equivalent to the instance of the Not-All-Equal-3Sat problem being satisfiable.

Our result:

Theorem 6.2. *Testing whether a given data set S satisfies CARP is NP-complete.*

The proof of this theorem is structured as follows. First, we build a data set S given the instance of Not-All-Equal-3Sat. Next, we enumerate for each pair of observations (s,t) and for each triple of observations (s,t_1,t_2) whether an inequality of the form $p_s q_s \geq p_s q_t$ (or $p_s q_s > p_s q_t$), or of the form $p_s q_s \geq p_s(q_{t_1} + q_{t_2})$ $\big($or $p_s q_s > p_s(q_{t_1} + q_{t_2})\big)$ is present. This is described in **Claim** 1 and **Claim** 2. Third, we argue the equivalence of a YES instance of Not-All-Equal-3Sat and the data set S satisfying CARP. For the sake of simplicity, throughout this chapter we will also call $t \in \mathbb{T}$ an observation while referring to (p_t, q_t).

Notice that it is not hard to see that the problem of testing CARP is in the class NP: given the relations H_0^1 and H_0^2; (and hence H^1 and H^2) we simply check, for each pair or triple of observations, whether *Rules* 1-5 hold. Clearly, this can be done in polynomial time.

In the first step of the proof, we aim at building the data set S. We shall first determine the set $\mathbb{T}$ of indices of observations. Next, we derive the number of goods in the economy and finally, for each observation, we derive a vector containing the price (respectively quantity) of each good for that observation.

Consider an arbitrary instance of the Not-All-Equal-3Sat problem where no variable occurs more than once in the same clause. We build the set of observations as follows. For each variable $x_i \in X$ $(i = 1, \ldots, n)$, we have two observations specified by x_i and $\bar{x}_i$, where the latter refers to the negation of x_i. We define $\mathbb{T}_1 = \{x_i, \bar{x}_i : i = 1, \ldots, n\}$ with cardinality $|\mathbb{T}_1| = 2n$. The observations in $\mathbb{T}_1$ are called *variable observations*.

For each clause $C_\ell = (\chi_1^\ell \vee \chi_2^\ell \vee \chi_3^\ell) \in C$, where the literal χ_1^ℓ is either the variable x_i or its negation $\bar{x}_i$, χ_2^ℓ is either x_j or $\bar{x}_j$, and χ_3^ℓ is either x_k or $\bar{x}_k$ with $1 \leq i < j < k \leq n$ (this ordering of indices can be achieved by permuting some literals), we define six observations $\mathbb{T}_2^\ell = \{\chi_1^\ell, \chi_2^\ell, \chi_3^\ell, t_1^\ell, t_2^\ell, t_3^\ell\}$. The first three observations in $\mathbb{T}_2^\ell$ are called *literal observations*. The last three observations in $\mathbb{T}_2^\ell$ are *associated observations*; each associated observation is associated with a literal observation. In particular, t_1^ℓ is associated with χ_1^ℓ, t_2^ℓ with χ_2^ℓ and t_3^ℓ with χ_3^ℓ. Let $\mathbb{T}_2 = \cup_{l=1}^m \mathbb{T}_2^\ell$ with $|\mathbb{T}_2| = 6m$. The observations in $\mathbb{T}_2$ are called *clause observations*. That is, a clause observation is either a literal observation or an associated observation. In total, the set of observations $\mathbb{T} = \mathbb{T}_1 \cup \mathbb{T}_2$ contains $T = |\mathbb{T}| = 2n + 6m$ observations.

To illustrate the reduction, we consider the following example of Not-All-Equal-3Sat problem, subsequently referred to as *the example*. The set of variables is $X = \{x_1, x_2, x_3\}$, and there are two clauses $C_1 = (x_1 \vee x_2 \vee x_3)$ and $C_2 = (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$; that is $\chi_1^1 = x_1$, $\chi_2^1 = x_2$, $\chi_3^1 = x_3$, $\chi_1^2 = \bar{x}_1$, $\chi_2^2 = x_2$ and $\chi_3^2 = \bar{x}_3$. Notice that the truth assignment $x_1 = x_2 = 1$ and $x_3 = 0$ is a solution to this Not-All-Equal-3Sat instance. For the example, the variable observations are $\{x_1, \bar{x}_1, x_2, \bar{x}_2, x_3, \bar{x}_3\}$ while the clause observations are $\{\chi_1^1, \chi_2^1, \chi_3^1, t_1^1, t_2^1, t_3^1\}$ for the first clause, and $\{\chi_1^2, \chi_2^2, \chi_3^2, t_1^2, t_2^2, t_3^2\}$ for the second clause. The reduction leads to a set of observations $\mathbb{T} = \{x_1, \bar{x}_1, x_2, \bar{x}_2, x_3, \bar{x}_3, \chi_1^1, \chi_2^1, \chi_3^1, t_1^1, t_2^1, t_3^1, \chi_1^2, \chi_2^2, \chi_3^2, t_1^2, t_2^2, t_3^2\}$ with 18 elements.

To further describe the data set S, we need to fix the number of goods in each bundle, and for each observation in $\mathbb{T}$, we must specify the price and the quantity of each good. We consider an economy with $N = 2T^2$ goods. We now specify the price and the quantity of the N goods for each observation in $\mathbb{T}$. For ease of exposition, a bundle of $N = 2T^2$ goods is represented by two blocks, each block being a $T \times T$ matrix. Each cell in each block represents a good.

We index the rows and columns of the first $T \times T$ matrix (referred to as Block 1 in the rest of this chapter) by the observations in $\mathbb{T}$. We use, both for the rows and for the columns, the following ordering: x_1, $\bar{x}_1$, x_2, $\bar{x}_2$, $\ldots$, x_n, $\bar{x}_n$, χ_1^1, χ_2^1, χ_3^1, t_1^1, t_2^1, t_3^1, χ_1^2, χ_2^2, χ_3^2, t_1^2, t_2^2, t_3^2, $\ldots$, χ_1^m, χ_2^m, χ_3^m, t_1^m, t_2^m, t_3^m. For the example, Block 1 is represented

in Table 6.1.

Table 6.1: *Block* 1 *for the example*

	x_1	$\bar{x}_1$	x_2	$\bar{x}_2$	x_3	$\bar{x}_3$	χ_1^1	χ_2^1	χ_3^1	t_1^1	t_2^1	t_3^1	χ_1^2	χ_2^2	χ_3^2	t_1^2	t_2^2	t_3^2
x_1																		
$\bar{x}_1$																		
x_2																		
$\bar{x}_2$																		
x_3																		
$\bar{x}_3$																		
χ_1^1																		
χ_2^1																		
χ_3^1																		
t_1^1																		
t_2^1																		
t_3^1																		
χ_1^2																		
χ_2^2																		
χ_3^2																		
t_1^2																		
t_2^2																		
t_3^2																		

We use the same indices for naming the rows and columns of the second $T \times T$ matrix (subsequently called Block 2 throughout this chapter). Hence, we can identify a good by specifying a pair (s, t) where s is the row-index (an observation), and where t is the column-index (also an observation), and by specifying the block (either Block 1 or Block 2).

For each variable $x_i \in X$, we define

$$\Gamma_{x_i} = \{\ell \in \{1, \ldots, m\} : \text{ clause } C_\ell \in C \text{ contains literal } x_i\}.$$

Similarly,

$$\Gamma_{\bar{x}_i} = \{\ell \in \{1, \ldots, m\} : \text{ clause } C_\ell \in C \text{ contains literal } \bar{x}_i\}.$$

Further, let

$$\Delta = 1 + \max\big\{8, \max\{2|\Gamma_{x_i}| + 4 : \ i = 1, \ldots, n\}, \ \max\{2|\Gamma_{\bar{x}_i}| + 4 : \ i = 1, \ldots, n\}\big\},$$

(where $|A|$ is the cardinality of A). For the example, we have $\Gamma_{x_1} = \{1\}$ as x_1 appears only in the clause C_1, $\Gamma_{\bar{x}_1} = \{2\}$ because $\bar{x}_1$ is present only in C_2, $\Gamma_{x_2} = \{1, 2\}$, $\Gamma_{\bar{x}_2} = \emptyset$,

$\Gamma_{x_3} = \{1\}$, $\Gamma_{\bar{x}_3} = \{2\}$ and $\Delta = 9$. Since we want to avoid prices equal to 0, we use, in the rest of this chapter, ε to denote a very small, strictly positive, real number.

Next, for each observation in $\mathbb{T}$ we will determine the price, as well as the quantity of each good. We will do this by distinguishing eight types of observations:

- variable observations corresponding to positive (negative) literals. The vector of prices for that observation is denoted by p_{x_i} ($p_{\bar{x}_i}$), and the bundle (purchased quantities) is denoted by q_{x_i} ($q_{\bar{x}_i}$); for $i = 1, \ldots, n$.
- clause observations corresponding to the first (second, third) literal. The vector of prices for that observation is denoted by $p_{\chi_1^\ell}$ ($p_{\chi_2^\ell}$, $p_{\chi_3^\ell}$), and the bundle by $q_{\chi_1^\ell}$ ($q_{\chi_2^\ell}$, $q_{\chi_3^\ell}$); for $\ell = 1, \ldots, m$.
- associated observations corresponding to the first (second, third) literal. The vector of prices for that observation is denoted by $p_{t_1^\ell}$ ($p_{t_2^\ell}$, $p_{t_3^\ell}$), and the bundle by $q_{t_1^\ell}$ ($q_{t_2^\ell}$, $q_{t_3^\ell}$); for $\ell = 1, \ldots, m$.

Choosing the particular values of the prices and the quantities is done with the objective of satisfying some inequalities for pairs or triples of observations. In fact, for each pair of observations (s, t), there are two goods: one in Block 1 and one in Block 2. The good in Block 1 is used to ensure that the desired inequality between $p_s q_s$ and $p_s q_t$ holds. The good in Block 2 is used to enforce the presence or absence of a double-sum inequality involving $p_s q_s$ and $p_s q_t$. All this is achieved by choosing appropriate values for the price and the quantity of each good.

We now continue by describing how the prices of all goods for all observations are set. That is, for each cell in each of the two blocks forming the set of all goods, we fix a strictly positive real value, representing the price. To achieve this, we proceed as follows. For each of the two blocks, we specify the structure of the corresponding matrix by giving a value to each cell representing the price of the good corresponding to that cell. We do this for every observation in $\mathbb{T}$.

Specifying $p_{x_i}, p_{\bar{x}_i}, p_{\chi_1^\ell}, p_{\chi_2^\ell}, p_{\chi_3^\ell}, p_{t_1^\ell}, p_{t_2^\ell}, p_{t_3^\ell}$ for goods corresponding to cells in Block 1.

For each observation $s \in \mathbb{T}$, there is a row in Block 1 indexed by s. We set the price of each good corresponding to a cell in this row equal to 1, except for the price of the good corresponding to cell (s, s): its price equals 2. The goods corresponding to the remaining cells in Block 1 get the price ε.

As an illustration, consider the observation $\bar{x}_2$ of the example. The price of goods corresponding to cells in Block 1 is given by Table 6.2.

Table 6.2: *Price of goods corresponding to cells in Block* 1 *for observation* $\bar{x}_2$.

	x_1	$\bar{x}_1$	x_2	$\bar{x}_2$	x_3	$\bar{x}_3$	χ_1^1	χ_2^1	χ_3^1	t_1^1	t_2^1	t_3^1	χ_1^2	χ_2^2	χ_3^2	t_1^2	t_2^2	t_3^2
x_1	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
$\bar{x}_1$	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
x_2	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
$\bar{x}_2$	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
x_3	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
$\bar{x}_3$	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
χ_1^1	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
χ_2^1	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
χ_3^1	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
t_1^1	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
t_2^1	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
t_3^1	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
χ_1^2	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
χ_2^2	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
χ_3^2	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
t_1^2	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
t_2^2	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
t_3^2	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε

Specifying p_{x_i} for goods corresponding to cells in Block 2.
Given a clause C_ℓ that contains $\bar{x}_i$ (negation of x_i), let r denote the position of $\bar{x}_i$ in the clause C_ℓ. Of course, $r \in \{1, 2, 3\}$ (notice that r depends upon ℓ and i; for reasons of convenience we simply write r instead of $r(i, \ell)$). Thus, for each clause C_ℓ with $\ell \in \Gamma_{\bar{x}_i}$, there is an associated observation t_r^ℓ in $\mathbb{T}$. The price of the good corresponding to cell $(\bar{x}_i, t_r^\ell)$ equals $\frac{1}{2|\Gamma_{\bar{x}_i}|}$. Also, the price of the good corresponding to cell $(t_r^\ell, \bar{x}_i)$ equals $\frac{1}{2|\Gamma_{\bar{x}_i}|}$. This is done for each clause C_ℓ with $\ell \in \Gamma_{\bar{x}_i}$. Notice that in total, we have $2|\Gamma_{\bar{x}_i}|$ cells with value $\frac{1}{2|\Gamma_{\bar{x}_i}|}$ in this block (Block 2). The goods corresponding to the remaining cells in Block 2 get the price ε.

As an illustration, consider the observation x_2 of the example. Since $\Gamma_{\bar{x}_2} = \emptyset$, the price of all goods corresponding to cells in Block 2 is ε.

Specifying $p_{\bar{x}_i}$ for goods corresponding to cells in Block 2.
We use an approach similar to the one used to determine p_{x_i}. Now, let r denote the position of x_i in the clause C_ℓ. For each clause C_ℓ with $\ell \in \Gamma_{x_i}$, there is an associated

observation t_r^ℓ. The price of the good corresponding to cell (x_i, t_r^ℓ) equals $\frac{1}{2|\Gamma_{x_i}|}$. Also, the price of the good corresponding to cell (t_r^ℓ, x_i) equals $\frac{1}{2|\Gamma_{x_i}|}$. This is done for each clause C_ℓ with $\ell \in \Gamma_{x_i}$. The goods corresponding to the remaining cells in Block 2 get the price ε.

As an illustration, consider the observation $\bar{x}_2$ of the example. The prices of goods corresponding to cells in Block 2 are given in Table 6.3. Notice that there are four goods with price $\frac{1}{4}$.

Table 6.3: *Price of goods corresponding to cells in Block* 2 *for observation* $\bar{x}_2$.

	x_1	$\bar{x}_1$	x_2	$\bar{x}_2$	x_3	$\bar{x}_3$	χ_1^1	χ_2^1	χ_3^1	t_1^1	t_2^1	t_3^1	χ_1^2	χ_2^2	χ_3^2	t_1^2	t_2^2	t_3^2
x_1	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
$\bar{x}_1$	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
x_2	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	$\frac{1}{4}$	ε	ε	ε	ε	ε	$\frac{1}{4}$	ε
$\bar{x}_2$	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
x_3	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
$\bar{x}_3$	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
χ_1^1	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
χ_2^1	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
χ_3^1	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
t_1^1	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
t_2^1	ε	ε	$\frac{1}{4}$	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
t_3^1	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
χ_1^2	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
χ_2^2	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
χ_3^2	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
t_1^2	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
t_2^2	ε	ε	$\frac{1}{4}$	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε
t_3^2	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε

Specifying $p_{\chi_1^\ell}$ for goods corresponding to cells in Block 2.
There are two goods corresponding to cells in Block 2 that have a price different from ε. These are the goods corresponding to the two cells (χ_3^ℓ, t_3^ℓ) and (t_3^ℓ, χ_3^ℓ); the price for these goods equals $\frac{1}{2}$. The goods corresponding to the remaining cells in Block 2 get the price ε.

As an illustration, consider the example. For $\ell = 1$ we have $\chi_1^\ell = x_1$ and the two goods with price $\frac{1}{2}$ correspond to cells (χ_3^1, t_3^1) and (t_3^1, χ_3^1). The goods corresponding to the remaining cells in Block 2 get the price ε. For $\ell = 2$, the goods corresponding

to cells (χ_3^2, t_3^2) and (t_3^2, χ_3^2) get the price $\frac{1}{2}$; the goods corresponding to the remaining cells in Block 2 get the price ε.

Specifying $p_{\chi_2^\ell}$ for goods corresponding to cells in Block 2.
Again, there are two goods that have price $\frac{1}{2}$, namely those corresponding to the cells (χ_1^ℓ, t_1^ℓ) and (t_1^ℓ, χ_1^ℓ). The goods corresponding to the remaining cells in Block 2 get the price ε.

As an illustration, consider the example. For $\ell = 1$ we have $\chi_2^\ell = x_2$ and the two goods with price $\frac{1}{2}$ correspond to cells (χ_1^1, t_1^1) and (t_1^1, χ_1^1). The goods corresponding to the remaining cells in Block 2 get the price ε. For $\ell = 2$, the goods corresponding to cells (χ_1^2, t_1^2) and (t_1^2, χ_1^2) get the price $\frac{1}{2}$; the goods corresponding to the remaining cells in Block 2 get the price ε.

Specifying $p_{\chi_3^\ell}$ for goods corresponding to cells in Block 2.
Also here, there are two goods with price $\frac{1}{2}$, namely those corresponding to the cells (χ_2^ℓ, t_2^ℓ) and (t_2^ℓ, χ_2^ℓ). The goods corresponding to the remaining cells in Block 2 get the price ε.

As an illustration, consider the example. For $\ell = 1$ we have $\chi_3^\ell = x_3$ and the two goods with price $\frac{1}{2}$ correspond to cells (χ_2^1, t_2^1) and (t_2^1, χ_2^1). The goods corresponding to the remaining cells in Block 2 get the price ε. For $\ell = 2$, the goods corresponding to cells (χ_2^2, t_2^2) and (t_2^2, χ_2^2) get the price $\frac{1}{2}$; the goods corresponding to the remaining cells in Block 2 get the price ε.

Specifying $p_{t_1^\ell}$ for goods corresponding to cells in Block 2.
Recall that the observation t_1^ℓ is associated with the literal observation χ_1^ℓ. Further, the literal χ_1^ℓ is either x_i or $\bar{x}_i$ for a given $i \in \{1, \ldots, n\}$. In both cases, there are only two goods corresponding to cells in Block 2 that have price different from ε.

> If $\chi_1^\ell = x_i$, then the two goods of Block 2 with price $\frac{1}{2}$ are those corresponding to cells $(\chi_2^\ell, \bar{x}_i)$ and $(\bar{x}_i, \chi_2^\ell)$. The goods corresponding to the remaining cells in Block 2 get the price ε.
>
> On the other hand, if $\chi_1^\ell = \bar{x}_i$, then the two goods of Block 2 corresponding to cells (χ_2^ℓ, x_i) and (x_i, χ_2^ℓ) have price $\frac{1}{2}$. The goods corresponding to the remaining cells in Block 2 get the price ε.

As an illustration, consider the example. For $\ell = 1$ the observation t_1^1 is such that the goods corresponding to cells $(\chi_2^1, \bar{x}_1)$ and $(\bar{x}_1, \chi_2^1)$ in Block 2 have price $\frac{1}{2}$ since $\chi_1^1 = x_1$. The goods corresponding to the remaining cells in Block 2 get the price ε.

For $\ell = 2$, the goods corresponding to cells (χ_2^2, x_1) and (x_1, χ_2^2) in Block 2 have price $\frac{1}{2}$ for observation t_1^2 because $\chi_1^2 = \bar{x}_1$. The goods corresponding to the remaining cells in Block 2 get the price ε.

Specifying $p_{t_2^\ell}$ for goods corresponding to cells in Block 2.

The observation t_2^ℓ is associated with χ_2^ℓ which is either x_j or $\bar{x}_j$ for a given $j \in \{1, \ldots, n\}$.

> If $\chi_2^\ell = x_j$, then the two goods of Block 2 with price $\frac{1}{2}$ are those corresponding to cells $(\chi_3^\ell, \bar{x}_j)$ and $(\bar{x}_j, \chi_3^\ell)$. The goods corresponding to the remaining cells in Block 2 get the price ε.
>
> Otherwise, if $\chi_2^\ell = \bar{x}_j$ then the two goods of Block 2 corresponding to cells (χ_3^ℓ, x_j) and (x_j, χ_3^ℓ) have price $\frac{1}{2}$. The goods corresponding to the remaining cells in Block 2 get the price ε.

As an illustration, consider the example. For $\ell = 1$ the observation t_2^1 is such that the goods corresponding to cells $(\chi_3^1, \bar{x}_2)$ and $(\bar{x}_2, \chi_3^1)$ in Block 2 have price $\frac{1}{2}$ since $\chi_2^1 = x_2$. The goods corresponding to the remaining cells in Block 2 get the price ε. For $\ell = 2$, the goods corresponding to cells $(\chi_3^2, \bar{x}_2)$ and $(\bar{x}_2, \chi_3^2)$ in Block 2 have price $\frac{1}{2}$ for observation t_2^2 because $\chi_2^2 = x_2$. The goods corresponding to the remaining cells in Block 2 get the price ε.

Specifying $p_{t_3^\ell}$ for goods corresponding to cells in Block 2.

The observation t_3^ℓ is associated with χ_3^ℓ which is either x_k or $\bar{x}_k$ for a given $k \in \{1, \ldots, n\}$.

> If $\chi_3^\ell = x_k$, the two goods of Block 2 with price $\frac{1}{2}$ are those corresponding to cells $(\chi_1^\ell, \bar{x}_k)$ and $(\bar{x}_k, \chi_1^\ell)$. The goods corresponding to the remaining cells in Block 2 get the price ε.
>
> If, on the other hand, $\chi_3^\ell = \bar{x}_k$ then the two goods of Block 2 corresponding to cells (χ_1^ℓ, x_k) and (x_k, χ_1^ℓ) have price $\frac{1}{2}$. The goods corresponding to the remaining cells in Block 2 get the price ε.

As an illustration, consider the example. For $\ell = 1$ the observation t_3^1 is such that the goods corresponding to cells $(\chi_1^1, \bar{x}_3)$ and $(\bar{x}_3, \chi_1^1)$ in Block 2 have price $\frac{1}{2}$ since $\chi_3^1 = x_3$. The goods corresponding to the remaining cells in Block 2 get the price ε. For $\ell = 2$, the goods corresponding to cells (χ_1^2, x_3) and (x_3, χ_1^2) in Block 2 have price

$\frac{1}{2}$ for observation t_3^2 because $\chi_3^2 = \bar{x}_3$. The goods corresponding to the remaining cells in Block 2 get the price ε.

This achieves the description of prices: for each observation in $\mathbb{T}$ we have specified the price of every good in the bundle (that is every cell in the two blocks). It remains to fix for each observation in $\mathbb{T}$ the quantity used for each good in the bundle.

Specifying q_{x_i} for goods corresponding to cells in Block 1.
There is a row and a column in Block 1 indexed by x_i. All the goods corresponding to cells of Block 1 other than those corresponding to cells in row x_i and in column x_i get the value 0 as their quantity. As for cells in row x_i and column x_i, the good corresponding to cell (x_i, x_i) gets the value 1. The quantity of the good corresponding to cell $(x_i, \bar{x}_i)$ equals 1, and the quantity of the good corresponding to cell $(\bar{x}_i, x_i)$ equals the value $|\Gamma_{x_i}| + 1$. Moreover, for every clause C_ℓ containing $\bar{x}_i$ ($\ell \in \Gamma_{\bar{x}_i}$), the good corresponding to cell (x_i, t_r^ℓ) gets the value 1, where r denotes the position of $\bar{x}_i$ in the clause C_ℓ. The quantity of the good corresponding to cell (t_r^ℓ, x_i) equals 2. The remaining goods corresponding to cells in row x_i are not used and get the quantity 0, while those remaining in column x_i get the value Δ. Observe that a good corresponding to a cell (x_i, t) in row x_i has a non-zero value if and only if the corresponding to the cell (t, x_i) in column x_i has a value different from (more precisely less than) Δ.

Table 6.4: *Quantity of goods corresponding to cells in Block* 1 *for observation* x_1.

	x_1	$\bar{x}_1$	x_2	$\bar{x}_2$	x_3	$\bar{x}_3$	x_1^1	x_2^1	x_3^1	t_1^1	t_2^1	t_3^1	$\bar{x}_1^2$	x_2^2	$\bar{x}_3^2$	t_1^2	t_2^2	t_3^2
x_1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
$\bar{x}_1$	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_2	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{x}_2$	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_3	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{x}_3$	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_1^1	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_2^1	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_3^1	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_1^1	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_2^1	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_3^1	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{x}_1^2$	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_2^2	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{x}_3^2$	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_1^2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_2^2	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

t_3^2	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

As an illustration, consider the observation x_1 of the example. The quantity of goods corresponding to cells in Block 1 are given in Table 6.4.

Specifying $q_{\bar{x}_i}$ for goods corresponding to cells in Block 1.

All goods corresponding to cells in Block 1 other than those in row $\bar{x}_i$ and in column $\bar{x}_i$ get the value 0 as their quantity. The good corresponding to cell $(\bar{x}_i, \bar{x}_i)$ has quantity 1. Also, the good corresponding to cell $(\bar{x}_i, x_i)$ has quantity 1, and the good corresponding to cell $(x_i, \bar{x}_i)$ has quantity $|\Gamma_{\bar{x}_i}| + 1$. For every clause C_ℓ containing x_i ($\ell \in \Gamma_{x_i}$), the good corresponding to cell $(\bar{x}_i, t_r^\ell)$ has quantity 1 (where r refers to the position of x_i in clause C_ℓ). The good corresponding to cell $(t_r^\ell, \bar{x}_i)$ has quantity 2. For the goods corresponding to the remaining cells in row $\bar{x}_i$, their quantity is 0, while the goods corresponding to the remaining cells in column $\bar{x}_i$ have quantity Δ.

As an illustration, consider the observation $\bar{x}_2$ of the example. The quantity of goods corresponding to cells in Block 1 are given in Table 6.5.

Table 6.5: *Quantity of goods corresponding to cells in Block 1 for observation $\bar{x}_2$.*

	x_1	$\bar{x}_1$	x_2	$\bar{x}_2$	x_3	$\bar{x}_3$	x_1^1	x_2^1	x_3^1	t_1^1	t_2^1	t_3^1	$\bar{x}_1^2$	x_2^2	$\bar{x}_3^2$	t_1^2	t_2^2	t_3^2
x_1	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{x}_1$	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{x}_2$	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0
x_3	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{x}_3$	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_1^1	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_2^1	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_3^1	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_1^1	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_2^1	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_3^1	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{x}_1^2$	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_2^2	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{x}_3^2$	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_1^2	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_2^2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t_3^2	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Specifying $q_{\chi_1^\ell}$ for goods corresponding to cells in Block 1.
All goods corresponding to cells in Block 1 that are neither in row χ_1^ℓ nor in column χ_1^ℓ have quantity 0. The goods corresponding to cells $(\chi_1^\ell, \chi_1^\ell)$, $(\chi_1^\ell, \chi_2^\ell)$, $(\chi_1^\ell, \chi_3^\ell)$ and (χ_1^ℓ, t_3^ℓ) in row χ_1^ℓ have quantity 1. The good corresponding to cell $(\chi_2^\ell, \chi_1^\ell)$ gets a quantity of 4, that corresponding to cell $(\chi_3^\ell, \chi_1^\ell)$ receives a quantity of 6 while the good corresponding to cell (t_3^ℓ, χ_1^ℓ) has quantity 2. For the goods corresponding to the remaining cells in row χ_1^ℓ, their quantity is 0, while those corresponding to the remaining cells in column χ_1^ℓ have quantity Δ.
Specifying $q_{\chi_2^\ell}$ for goods corresponding to cells in Block 1.
All goods corresponding to cells in Block 1 neither in row χ_2^ℓ nor in column χ_2^ℓ have quantity 0. The goods corresponding to cells $(\chi_2^\ell, \chi_2^\ell)$, $(\chi_2^\ell, \chi_1^\ell)$, $(\chi_2^\ell, \chi_3^\ell)$ and (χ_2^ℓ, t_1^ℓ) have quantity 1. The good corresponding to cell $(\chi_1^\ell, \chi_2^\ell)$ has a quantity of 6 and that corresponding to cell $(\chi_3^\ell, \chi_2^\ell)$ gets a quantity 4, while the good corresponding to cell (t_1^ℓ, χ_2^ℓ) has quantity 2. For the goods corresponding to the remaining cells in row χ_2^ℓ, their quantity is 0, while those corresponding to the remaining cells in column χ_2^ℓ have quantity Δ.
Specifying $q_{\chi_3^\ell}$ for goods corresponding to cells in Block 1.
All goods corresponding to cells in Block 1 neither in row χ_3^ℓ nor in column χ_3^ℓ have quantity 0. The goods corresponding to cells $(\chi_3^\ell, \chi_3^\ell)$, $(\chi_3^\ell, \chi_1^\ell)$, $(\chi_3^\ell, \chi_2^\ell)$ and (χ_3^ℓ, t_2^ℓ) have quantity 1. In column χ_3^ℓ, the good corresponding to cell $(\chi_1^\ell, \chi_3^\ell)$ has a quantity of 4, that corresponding to cell $(\chi_2^\ell, \chi_3^\ell)$ has a quantity of 6 while the good corresponding to cell (t_2^ℓ, χ_3^ℓ) has quantity 2. For the goods corresponding to the remaining cells in row χ_3^ℓ, their quantity is 0, while those corresponding to the remaining cells in column χ_3^ℓ have quantity Δ.
Specifying $q_{t_1^\ell}$ for goods corresponding to cells in Block 1.
All goods corresponding to cells in Block 1 that are neither in row t_1^ℓ nor in column t_1^ℓ have quantity 0. Since t_1^ℓ is associated to χ_1^ℓ which is either x_i or $\bar{x}_i$ for a given $i \in \{1, \ldots, n\}$, we distinguish two cases.

> If $\chi_1^\ell = x_i$, then the goods corresponding to cells (t_1^ℓ, t_1^ℓ), (t_1^ℓ, χ_2^ℓ) and $(t_1^\ell, \bar{x}_i)$ have quantity 1. In column t_1^ℓ, the good corresponding to cell (χ_2^ℓ, t_1^ℓ) has quantity 3, while the good corresponding to cell $(\bar{x}_i, t_1^\ell)$ has quantity $|\Gamma_{x_i}| + 1$. The goods corresponding to the remaining cells in row t_1^ℓ have quantity 0, while those corresponding to the remaining cells in column t_1^ℓ have quantity Δ.

If, on the other hand, $\chi_1^\ell = \bar{x}_i$ then the goods corresponding to cells (t_1^ℓ, t_1^ℓ), (t_1^ℓ, χ_2^ℓ) and (t_1^ℓ, x_i) have quantity 1. In column t_1^ℓ, the good corresponding to cell (χ_2^ℓ, t_1^ℓ) has quantity 3, while the good corresponding to cell (x_i, t_1^ℓ) has quantity $|\Gamma_{\bar{x}_i}| + 1$. The goods corresponding to the remaining cells in row t_1^ℓ have quantity 0, while those corresponding to the remaining cells in column t_1^ℓ have quantity Δ.

Specifying $q_{t_2^\ell}$ for goods corresponding to cells in Block 1.
All goods corresponding to cells in Block 1 that are neither in row t_2^ℓ nor in column t_2^ℓ have quantity 0. Since t_2^ℓ is associated to χ_2^ℓ which is either x_j or $\bar{x}_j$ for a given $j \in \{1, \ldots, n\}$, we distinguish two cases.

If $\chi_2^\ell = x_j$, then the goods corresponding to cells (t_2^ℓ, t_2^ℓ), (t_2^ℓ, χ_3^ℓ) and $(t_2^\ell, \bar{x}_j)$ have quantity 1. The good corresponding to cell (χ_3^ℓ, t_2^ℓ) has quantity 3, while the good corresponding to cell $(\bar{x}_j, t_2^\ell)$ has quantity $|\Gamma_{x_j}| + 1$. The goods corresponding to the remaining cells in row t_2^ℓ have quantity 0, while those corresponding to the remaining cells in column t_2^ℓ have quantity Δ.

Otherwise, if $\chi_2^\ell = \bar{x}_j$ then the goods corresponding to cells (t_2^ℓ, t_2^ℓ), (t_2^ℓ, χ_3^ℓ) and (t_2^ℓ, x_j) have quantity 1. In column t_2^ℓ, the good corresponding to cell (χ_3^ℓ, t_2^ℓ) has quantity 3, while the good corresponding to cell (x_j, t_2^ℓ) has quantity $|\Gamma_{\bar{x}_j}| + 1$. The goods corresponding to the remaining cells in row t_2^ℓ have quantity 0, while those corresponding to the remaining cells in column t_2^ℓ have quantity Δ.

Specifying $q_{t_3^\ell}$ for goods corresponding to cells in Block 1.
All goods corresponding to cells in Block 1 that are neither in row t_3^ℓ nor in column t_3^ℓ have quantity 0. Since t_3^ℓ is associated to χ_3^ℓ which is either x_k or $\bar{x}_k$ for a given $k \in \{1, \ldots, n\}$, we distinguish two cases.

If $\chi_3^\ell = x_k$ then the goods corresponding to cells (t_3^ℓ, t_3^ℓ), (t_3^ℓ, χ_1^ℓ) and $(t_3^\ell, \bar{x}_k)$ have quantity 1. In column t_3^ℓ, the good corresponding to cell (χ_1^ℓ, t_3^ℓ) has quantity 3, while the good corresponding to cell $(\bar{x}_k, t_3^\ell)$ has quantity $|\Gamma_{x_k}| + 1$. The goods corresponding to the remaining cells in row t_3^ℓ have quantity 0, while those corresponding to the remaining cells in column t_3^ℓ have quantity Δ.

Otherwise, if $\chi_3^\ell = \bar{x}_k$, then the goods corresponding to cells (t_3^ℓ, t_3^ℓ), (t_3^ℓ, χ_1^ℓ) and (t_3^ℓ, x_k) have quantity 1. In column t_3^ℓ, the good corresponding to cell (χ_1^ℓ, t_3^ℓ) has quantity 3, while the good corresponding to cell (x_k, t_3^ℓ) has quantity $|\Gamma_{\bar{x}_k}| + 1$.

The goods corresponding to the remaining cells in row t_3^ℓ have quantity 0, while those corresponding to the remaining cells in column t_3^ℓ have quantity Δ.

We now proceed with the quantities of the goods corresponding to cells in Block 2.

Specifying q_{x_i} for goods corresponding to cells in Block 2.

The goods corresponding to cells in Block 2 that have a non-ε price get the quantity $|\Gamma_{\bar{x}_i}| + 1$ while those with ε price get the value 0 as quantity.

As an illustration, consider the observation x_2 of the example. For that observation, all the goods corresponding to cells in Block 2 get the price ε. Therefore, all the goods corresponding to cells in Block 2 have quantity 0.

Specifying $q_{\bar{x}_i}$ for goods corresponding to cells in Block 2.

The goods corresponding to cells in Block 2 have quantity $|\Gamma_{x_i}| + 1$, if their price in that observation is different from ε; otherwise their quantity equals 0.

Specifying $q_{\chi_1^\ell}, q_{\chi_2^\ell}, q_{\chi_3^\ell}$ for goods corresponding to cells in Block 2.

For goods corresponding to cells in Block 2, the following holds: if the price of such a good in some observation is ε, then the quantity of that good for that observation is 0, otherwise the quantity is 3.

Specifying $q_{t_1^\ell}, q_{t_2^\ell}, q_{t_3^\ell}$ for goods corresponding to cells in Block 2.

For goods corresponding to cells in Block 2, the following holds: if the price of such a good in some observation is ε, then the quantity of that good for that observation is 0, otherwise the quantity is 2.

This completes the description of the quantity of each good in the bundle for every observation in $\mathbb{T}$. Thus, we have built the data set S. Notice that this construction of S is done in polynomial time.

The second step of our proof identifies some characteristics of the data set S constructed above; these are the inequalities and double-sum inequalities satisfied by the vectors of quantity and price of observations. Our goal is to compare for each pair s, t (respectively triple s, t_1, t_2) of observations the values $p_s q_s$ and $p_s q_t$ (respectively $p_s q_s$ and $p_s(q_{t_1} + q_{t_2})$).

Claim 1. *Given the data set S defined above, we have the following inequalities.*

For each $i = 1, \ldots, n$,

$$p_{x_i} q_{x_i} > p_{x_i} q_{\bar{x}_i}, \tag{6.1}$$

$$p_{\bar{x}_i} q_{\bar{x}_i} > p_{\bar{x}_i} q_{x_i}. \tag{6.2}$$

For each $\ell = 1, \ldots, m$,

$$p_{\chi_1^\ell} q_{\chi_1^\ell} > p_{\chi_1^\ell} q_{\chi_2^\ell}, \tag{6.3}$$

$$p_{\chi_1^\ell} q_{\chi_1^\ell} > p_{\chi_1^\ell} q_{\chi_3^\ell}, \tag{6.4}$$

$$p_{\chi_1^\ell} q_{\chi_1^\ell} > p_{\chi_1^\ell} q_{t_3^\ell}, \tag{6.5}$$

$$p_{\chi_2^\ell} q_{\chi_2^\ell} > p_{\chi_2^\ell} q_{\chi_1^\ell}, \tag{6.6}$$

$$p_{\chi_2^\ell} q_{\chi_2^\ell} > p_{\chi_2^\ell} q_{\chi_3^\ell}, \tag{6.7}$$

$$p_{\chi_2^\ell} q_{\chi_2^\ell} > p_{\chi_2^\ell} q_{t_1^\ell}, \tag{6.8}$$

$$p_{\chi_3^\ell} q_{\chi_3^\ell} > p_{\chi_3^\ell} q_{\chi_1^\ell}, \tag{6.9}$$

$$p_{\chi_3^\ell} q_{\chi_3^\ell} > p_{\chi_3^\ell} q_{\chi_2^\ell}, \tag{6.10}$$

$$p_{\chi_3^\ell} q_{\chi_3^\ell} > p_{\chi_3^\ell} q_{t_2^\ell}. \tag{6.11}$$

For each $\ell = 1, \ldots, m$, *for each* $i = 1, \ldots, n$ *with* $\chi_1^\ell = x_i$ *or* $\chi_1^\ell = \bar{x}_i$

$$p_{t_1^\ell} q_{t_1^\ell} > p_{t_1^\ell} q_{\chi_2^\ell}, \tag{6.12}$$

$$p_{t_1^\ell} q_{t_1^\ell} > \begin{cases} p_{t_1^\ell} q_{\bar{x}_i} & \text{if } \chi_1^\ell = x_i, \\ p_{t_1^\ell} q_{x_i} & \text{if } \chi_1^\ell = \bar{x}_i, \end{cases} \tag{6.13}$$

$$\left.\begin{array}{rcll} p_{\bar{x}_i} q_{\bar{x}_i} & > & p_{\bar{x}_i} q_{t_1^\ell} & \text{if } \chi_1^\ell = x_i, \\ p_{x_i} q_{x_i} & > & p_{x_i} q_{t_1^\ell} & \text{if } \chi_1^\ell = \bar{x}_i. \end{array}\right\} \tag{6.14}$$

For each $\ell = 1, \ldots, m$, *for each* $j = 1, \ldots, n$ *with* $\chi_2^\ell = x_j$ *or* $\chi_2^\ell = \bar{x}_j$

$$p_{t_2^\ell} q_{t_2^\ell} > p_{t_2^\ell} q_{\chi_3^\ell}, \tag{6.15}$$

$$p_{t_2^\ell} q_{t_2^\ell} > \begin{cases} p_{t_2^\ell} q_{\bar{x}_j} & \text{if } \chi_2^\ell = x_j, \\ p_{t_2^\ell} q_{x_j} & \text{if } \chi_2^\ell = \bar{x}_j, \end{cases} \tag{6.16}$$

$$\left.\begin{array}{rcll} p_{\bar{x}_j} q_{\bar{x}_j} & > & p_{\bar{x}_j} q_{t_2^\ell} & \text{if } \chi_2^\ell = x_j, \\ p_{x_j} q_{x_j} & > & p_{x_j} q_{t_2^\ell} & \text{if } \chi_2^\ell = \bar{x}_j. \end{array}\right\} \tag{6.17}$$

For each $\ell = 1, \ldots, m$, *for each* $k = 1, \ldots, n$ *with* $\chi_3^\ell = x_k$ *or* $\chi_3^\ell = \bar{x}_k$

$$p_{t_3^\ell} q_{t_3^\ell} > p_{t_3^\ell} q_{\chi_1^\ell}, \tag{6.18}$$

$$p_{t_3^\ell} q_{t_3^\ell} > \begin{cases} p_{t_3^\ell} q_{\bar{x}_k} & \text{if } \chi_3^\ell = x_k, \\ p_{t_3^\ell} q_{x_k} & \text{if } \chi_3^\ell = \bar{x}_k, \end{cases} \tag{6.19}$$

$$\left.\begin{array}{rcll} p_{\bar{x}_k} q_{\bar{x}_k} & > & p_{\bar{x}_k} q_{t_3^\ell} & \text{if } \chi_3^\ell = x_k, \\ p_{x_k} q_{x_k} & > & p_{x_k} q_{t_3^\ell} & \text{if } \chi_3^\ell = \bar{x}_k. \end{array}\right\} \tag{6.20}$$

For all other pair s, t of distinct observations in $\mathbb{T}$

$$p_s q_s < p_s q_t. \tag{6.21}$$

Claim 2. *Given the data set S defined above, the following double-sum inequalities hold.*

For each $\ell = 1, \ldots, m$,

$$p_{\chi_1^\ell} q_{\chi_1^\ell} > p_{\chi_1^\ell}(q_{\chi_3^\ell} + q_{t_3^\ell}), \tag{6.22}$$
$$p_{\chi_2^\ell} q_{\chi_2^\ell} > p_{\chi_2^\ell}(q_{\chi_1^\ell} + q_{t_1^\ell}), \tag{6.23}$$
$$p_{\chi_3^\ell} q_{\chi_3^\ell} > p_{\chi_3^\ell}(q_{\chi_2^\ell} + q_{t_2^\ell}). \tag{6.24}$$

For each $\ell = 1, \ldots, m$, for each $i = 1, \ldots, n$ with $\chi_1^\ell = x_i$ or $\chi_1^\ell = \bar{x}_i$

$$p_{t_1^\ell} q_{t_1^\ell} > \begin{cases} p_{t_1^\ell}(q_{\chi_2^\ell} + q_{\bar{x}_i}) & \text{if } \chi_1^\ell = x_i, \\ p_{t_1^\ell}(q_{\chi_2^\ell} + q_{x_i}) & \text{if } \chi_1^\ell = \bar{x}_i, \end{cases} \tag{6.25}$$

$$\left.\begin{array}{rcll} p_{\bar{x}_i} q_{\bar{x}_i} & > & p_{\bar{x}_i}(q_{x_i} + q_{t_1^\ell}) & \text{if } \chi_1^\ell = x_i, \\ p_{x_i} q_{x_i} & > & p_{x_i}(q_{\bar{x}_i} + q_{t_1^\ell}) & \text{if } \chi_1^\ell = \bar{x}_i. \end{array}\right\} \tag{6.26}$$

For each $\ell = 1, \ldots, m$, for each $j = 1, \ldots, n$ with $\chi_2^\ell = x_j$ or $\chi_2^\ell = \bar{x}_j$

$$p_{t_2^\ell} q_{t_2^\ell} > \begin{cases} p_{t_2^\ell}(q_{\chi_3^\ell} + q_{\bar{x}_j}) & \text{if } \chi_2^\ell = x_j, \\ p_{t_2^\ell}(q_{\chi_3^\ell} + q_{x_j}) & \text{if } \chi_2^\ell = \bar{x}_j, \end{cases} \tag{6.27}$$

$$\left.\begin{array}{rcll} p_{\bar{x}_j} q_{\bar{x}_j} & > & p_{\bar{x}_j}(q_{x_j} + q_{t_2^\ell}) & \text{if } \chi_2^\ell = x_j, \\ p_{x_j} q_{x_j} & > & p_{x_j}(q_{\bar{x}_j} + q_{t_2^\ell}) & \text{if } \chi_2^\ell = \bar{x}_j. \end{array}\right\} \tag{6.28}$$

For each $\ell = 1, \ldots, m$, for each $k = 1, \ldots, n$ with $\chi_3^\ell = x_k$ or $\chi_3^\ell = \bar{x}_k$

$$p_{t_3^\ell} q_{t_3^\ell} > \begin{cases} p_{t_3^\ell}(q_{\chi_1^\ell} + q_{\bar{x}_k}) & \text{if } \chi_3^\ell = x_k, \\ p_{t_3^\ell}(q_{\chi_1^\ell} + q_{x_k}) & \text{if } \chi_3^\ell = \bar{x}_k, \end{cases} \tag{6.29}$$

$$\left.\begin{array}{rcll} p_{\bar{x}_k} q_{\bar{x}_k} & > & p_{\bar{x}_k}(q_{x_k} + q_{t_3^\ell}) & \text{if } \chi_3^\ell = x_k, \\ p_{x_k} q_{x_k} & > & p_{x_k}(q_{\bar{x}_k} + q_{t_3^\ell}) & \text{if } \chi_3^\ell = \bar{x}_k. \end{array}\right\} \tag{6.30}$$

For each $i = 1, \ldots, n$ and for each $\ell, \ell' \in \Gamma_{\bar{x}_i}$ with r and r' being the position of $\bar{x}_i$ in the clause C_ℓ and $C_{\ell'}$ respectively,

$$p_{x_i} q_{x_i} > p_{x_i}(q_{t_r^\ell} + q_{t_{r'}^{\ell'}}). \tag{6.31}$$

For each $i = 1, \ldots, n$ and for each $\ell, \ell' \in \Gamma_{x_i}$ with r and r' being the position of x_i in the clause C_ℓ and $C_{\ell'}$ respectively,

$$p_{\bar{x}_i} q_{\bar{x}_i} \quad > \quad p_{\bar{x}_i}(q_{t^{\ell}_{r}} + q_{t^{\ell'}_{r'}}). \tag{6.32}$$

There are no double-sum inequalities other than those mentioned above.

The proofs of **Claim** 1 and **Claim** 2 are given in the Appendix.

In the last step of our proof, we prove that the data set S obtained by the above reduction satisfies CARP if and only if the instance of the Not-All-Equal-3Sat problem is a YES instance. The goal here is to prove that the instance of CARP built from the arbitrary instance of the Not-All-Equal-3Sat problem is at least as hard as that instance of the Not-All-Equal-3Sat problem. This proof strongly relies on **Claim** 1-2.

On one hand, suppose that S satisfies CARP. Thus there exist sets (hypothetical relations) H_0^1 and H_0^2 that satisfy *Rules* 1-5. The following is true for H_0^1 and H_0^2.

Lemma 6.3. *If the data set S satisfies CARP, then there are no two distinct observations s and t satisfying $p_s q_s \geq p_s q_t$ such that $(q_s, q_t) \in H_0^1$ and $(q_s, q_t) \in H_0^2$.*

Proof. From the inequalities listed in **Claim** 1, we observe that for a pair of distinct observations $s, t \in \mathbb{T}$, if $p_s q_s \geq p_s q_t$ then $p_s q_s > p_s q_t$ and $p_t q_t > p_t q_s$. Next, we argue by contradiction: if $(q_s, q_t) \in H_0^1$ and $(q_s, q_t) \in H_0^2$ then $(q_s, q_t) \in H^1$ and $(q_s, q_t) \in H^2$. This, however, contradicts *Rule* 4. Therefore, Lemma 6.3 holds. □

We now build a truth assignment for the instance of Not-All-Equal-3Sat and show that it is a YES instance. For each variable $x_i \in X$ we set $x_i = 1$ if $(q_{x_i}, q_{\bar{x}_i}) \in H_0^1$; otherwise $x_i = 0$. Thus, the value of each x_i is well-defined. In fact, using (6.1) and *Rule* 1 we conclude that for each i, $(q_{x_i}, q_{\bar{x}_i}) \in H_0^1$ or $(q_{x_i}, q_{\bar{x}_i}) \in H_0^2$. Since by construction, $x_i = 1$ corresponds to the case $(q_{x_i}, q_{\bar{x}_i}) \in H_0^1$, it follows that $x_i = 0$ corresponds to the case $(q_{x_i}, q_{\bar{x}_i}) \in H_0^2$.

We now prove that each clause in C has at least one true literal and at least one false literal. We argue by contradiction. Suppose that there exists a clause $C_\ell = (\chi_1^\ell \vee \chi_2^\ell \vee \chi_3^\ell)$ ($\ell \in \{1, \ldots, m\}$) in C which is such that either $\chi_1^\ell = \chi_2^\ell = \chi_3^\ell = 1$ or $\chi_1^\ell = \chi_2^\ell = \chi_3^\ell = 0$. Without loss of generality, let us assume that $\chi_1^\ell = \chi_2^\ell = \chi_3^\ell = 1$. We are going to investigate each literal in C_ℓ individually. The first literal χ_1^ℓ is either x_i or $\bar{x}_i$. We will argue that in both cases, we have $(q_{t^{\ell}_{1}}, q_{\chi_2^\ell}) \in H_0^1$.

Indeed, if $\chi_1^\ell = x_i$ then $x_i = 1$ implies that $(q_{x_i}, q_{\bar{x}_i}) \in H_0^1$ from the assignment of values to variables. The double-sum inequality (6.26) for the clause C_ℓ is $p_{\bar{x}_i} q_{\bar{x}_i} > p_{\bar{x}_i}(q_{x_i} + q_{t_1^\ell})$. Since $(q_{x_i}, q_{\bar{x}_i}) \in H_0^1$, *Rule* 3 implies that $(q_{\bar{x}_i}, q_{t_1^\ell}) \in H_0^2$. Using the double-sum inequality $p_{t_1^\ell} q_{t_1^\ell} > p_{t_1^\ell}(q_{\chi_2^\ell} + q_{\bar{x}_i})$ given by (6.25) and the fact that $(q_{\bar{x}_i}, q_{t_1^\ell}) \in H_0^2$, *Rule* 3 leads to $(q_{t_1^\ell}, q_{\chi_2^\ell}) \in H_0^1$.

On the other hand, if $\chi_1^\ell = \bar{x}_i$ then $\bar{x}_i = 1$ implies that $x_i = 0$ and $(q_{x_i}, q_{\bar{x}_i}) \in H_0^2$. Using *Rule* 2 and (6.2) we obtain $(q_{\bar{x}_i}, q_{x_i}) \in H_0^1$. The double-sum inequality (6.26) is $p_{x_i} q_{x_i} > p_{x_i}(q_{\bar{x}_i} + q_{t_1^\ell})$ and $(q_{\bar{x}_i}, q_{x_i}) \in H_0^1$. Thus *Rule* 3 implies that $(q_{x_i}, q_{t_1^\ell}) \in H_0^2$. The inequality (6.25) is $p_{t_1^\ell} q_{t_1^\ell} > p_{t_1^\ell}(q_{\chi_2^\ell} + q_{x_i})$ and $(q_{x_i}, q_{t_1^\ell}) \in H_0^2$; therefore $(q_{t_1^\ell}, q_{\chi_2^\ell}) \in H_0^1$ from *Rule* 3.

Since $(q_{t_1^\ell}, q_{\chi_2^\ell}) \in H_0^1$ we use the double-sum inequality $p_{\chi_2^\ell} q_{\chi_2^\ell} > p_{\chi_2^\ell}(q_{\chi_1^\ell} + q_{t_1^\ell})$, (given by (6.23)) and *Rule* 3 to obtain that $(q_{\chi_2^\ell}, q_{\chi_1^\ell}) \in H_0^2$. Finally, using *Rule* 2 and (6.8) we have $(q_{\chi_1^\ell}, q_{\chi_2^\ell}) \in H_0^1$.

We conclude that whether the literal χ_1^ℓ is x_i or $\bar{x}_i$, as long as its value equals 1 we have $(q_{\chi_1^\ell}, q_{\chi_2^\ell}) \in H_0^1$. Notice that, in case $\chi_1^\ell = 0$ we can conclude that $(q_{\chi_1^\ell}, q_{\chi_2^\ell}) \in H_0^2$.

By applying a similar reasoning to the second literal χ_2^ℓ, we obtain $(q_{\chi_2^\ell}, q_{\chi_3^\ell}) \in H_0^1$, while the application of that reasoning to the third literal χ_3^ℓ leads to $(q_{\chi_3^\ell}, q_{\chi_1^\ell}) \in H_0^1$. We obtain $(q_{\chi_1^\ell}, q_{\chi_2^\ell}) \in H_0^1$, $(q_{\chi_2^\ell}, q_{\chi_3^\ell}) \in H_0^1$ and $(q_{\chi_3^\ell}, q_{\chi_1^\ell}) \in H_0^1$. Thus $(q_{\chi_1^\ell}, q_{\chi_2^\ell})$, $(q_{\chi_2^\ell}, q_{\chi_3^\ell})$, $(q_{\chi_3^\ell}, q_{\chi_1^\ell}) \in H^1$, which imply from *Rule* 2, (6.4), (6.6), and (6.10) that $(q_{\chi_1^\ell}, q_{\chi_2^\ell})$, $(q_{\chi_2^\ell}, q_{\chi_3^\ell})$, $(q_{\chi_3^\ell}, q_{\chi_1^\ell}) \in H_0^2$. Thus $(q_{\chi_1^\ell}, q_{\chi_2^\ell}) \in H_0^1 \cap H_0^2$ and $\chi_2^\ell \neq \chi_1^\ell$. This contradicts Lemma 6.3. This concludes the proof that if the data set S satisfies CARP then the instance of Not-All-Equal-3Sat is a YES instance.

On the other hand, suppose that there is a truth assignment for X which is such that each clause in C has at least one true literal and at least one false literal. Consider H_0^1 and H_0^2 defined as follows. For each variable $x_i \in X$, if $x_i = 1$ then $(q_{x_i}, q_{\bar{x}_i}) \in H_0^1$ and $(q_{\bar{x}_i}, q_{x_i}) \in H_0^2$. Otherwise, if $x_i = 0$ then $(q_{\bar{x}_i}, q_{x_i}) \in H_0^1$ and $(q_{x_i}, q_{\bar{x}_i}) \in H_0^2$. This ensures that for each pair of observations (s, t) occurring in inequalities (6.1) or (6.2) the corresponding bundle pair (q_s, q_t) is either in H_0^1 or in H_0^2. We now deal with pairs of observations occurring in inequalities (6.3)-(6.20). We will specify for each ordered pair of observations occurring in each of these inequalities whether the corresponding bundle pair is in H_0^1 or in H_0^2. For every clause $C_\ell = (\chi_1^\ell \vee \chi_2^\ell \vee \chi_3^\ell)$ in C, we consider each literal in C_ℓ in turn. The construction of H_0^1 and H_0^2 for a given clause C_ℓ is given in Table 6.6.

Table 6.6: *Construction of H_0^1 and H_0^2 for a given clause C_ℓ.*

	$\chi_1^\ell = x_i$				$\chi_1^\ell = \bar{x}_i$			
	$x_i = 1$	ineq.	$x_i = 0$	ineq.	$x_i = 1$	ineq.	$x_i = 0$	ineq.
H_0^1	$(q_{x_i}, q_{\bar{x}_i})$	(6.1)	$(q_{\bar{x}_i}, q_{x_i})$	(6.2)	$(q_{x_i}, q_{\bar{x}_i})$	(6.1)	$(q_{\bar{x}_i}, q_{x_i})$	(6.2)
	$(q_{t_1^\ell}, q_{\bar{x}_i})$	(6.13)	$(q_{\bar{x}_i}, q_{t_1^\ell})$	(6.14)	$(q_{x_i}, q_{t_1^\ell})$	(6.14)	$(q_{t_1^\ell}, q_{x_i})$	(6.13)
	$(q_{t_1^\ell}, q_{\chi_2^\ell})$	(6.12)	$(q_{\chi_2^\ell}, q_{t_1^\ell})$	(6.8)	$(q_{\chi_2^\ell}, q_{t_1^\ell})$	(6.8)	$(q_{t_1^\ell}, q_{\chi_2^\ell})$	(6.12)
	$(q_{\chi_1^\ell}, q_{\chi_2^\ell})$	(6.3)	$(q_{\chi_2^\ell}, q_{\chi_1^\ell})$	(6.6)	$(q_{\chi_2^\ell}, q_{\chi_1^\ell})$	(6.6)	$(q_{\chi_1^\ell}, q_{\chi_2^\ell})$	(6.3)
H_0^2	$(q_{\bar{x}_i}, q_{x_i})$	(6.2)	$(q_{x_i}, q_{\bar{x}_i})$	(6.1)	$(q_{\bar{x}_i}, q_{x_i})$	(6.2)	$(q_{x_i}, q_{\bar{x}_i})$	(6.1)
	$(q_{\bar{x}_i}, q_{t_1^\ell})$	(6.14)	$(q_{t_1^\ell}, q_{\bar{x}_i})$	(6.13)	$(q_{t_1^\ell}, q_{x_i})$	(6.13)	$(q_{x_i}, q_{t_1^\ell})$	(6.14)
	$(q_{\chi_2^\ell}, q_{t_1^\ell})$	(6.8)	$(q_{t_1^\ell}, q_{\chi_2^\ell})$	(6.12)	$(q_{t_1^\ell}, q_{\chi_2^\ell})$	(6.12)	$(q_{\chi_2^\ell}, q_{t_1^\ell})$	(6.8)
	$(q_{\chi_2^\ell}, q_{\chi_1^\ell})$	(6.6)	$(q_{\chi_1^\ell}, q_{\chi_2^\ell})$	(6.3)	$(q_{\chi_1^\ell}, q_{\chi_2^\ell})$	(6.3)	$(q_{\chi_2^\ell}, q_{\chi_1^\ell})$	(6.6)
	$\chi_2^\ell = x_j$				$\chi_2^\ell = \bar{x}_j$			
	$x_j = 1$	ineq.	$x_j = 0$	ineq.	$x_j = 1$	ineq.	$x_j = 0$	ineq.
H_0^1	$(q_{x_j}, q_{\bar{x}_j})$	(6.1)	$(q_{\bar{x}_j}, q_{x_j})$	(6.2)	$(q_{x_j}, q_{\bar{x}_j})$	(6.1)	$(q_{\bar{x}_j}, q_{x_j})$	(6.2)
	$(q_{t_2^\ell}, q_{\bar{x}_j})$	(6.16)	$(q_{\bar{x}_j}, q_{t_2^\ell})$	(6.17)	$(q_{x_j}, q_{t_2^\ell})$	(6.17)	$(q_{t_2^\ell}, q_{x_j})$	(6.16)
	$(q_{t_2^\ell}, q_{\chi_3^\ell})$	(6.15)	$(q_{\chi_3^\ell}, q_{t_2^\ell})$	(6.11)	$(q_{\chi_3^\ell}, q_{t_2^\ell})$	(6.11)	$(q_{t_2^\ell}, q_{\chi_3^\ell})$	(6.15)
	$(q_{\chi_2^\ell}, q_{\chi_3^\ell})$	(6.7)	$(q_{\chi_3^\ell}, q_{\chi_2^\ell})$	(6.10)	$(q_{\chi_3^\ell}, q_{\chi_2^\ell})$	(6.10)	$(q_{\chi_2^\ell}, q_{\chi_3^\ell})$	(6.7)
H_0^2	$(q_{\bar{x}_j}, q_{x_j})$	(6.2)	$(q_{x_j}, q_{\bar{x}_j})$	(6.1)	$(q_{\bar{x}_j}, q_{x_j})$	(6.2)	$(q_{x_j}, q_{\bar{x}_j})$	(6.1)
	$(q_{\bar{x}_j}, q_{t_2^\ell})$	(6.17)	$(q_{t_2^\ell}, q_{\bar{x}_j})$	(6.16)	$(q_{t_2^\ell}, q_{x_j})$	(6.16)	$(q_{x_j}, q_{t_2^\ell})$	(6.17)
	$(q_{\chi_3^\ell}, q_{t_2^\ell})$	(6.11)	$(q_{t_2^\ell}, q_{\chi_3^\ell})$	(6.15)	$(q_{t_2^\ell}, q_{\chi_3^\ell})$	(6.15)	$(q_{\chi_3^\ell}, q_{t_2^\ell})$	(6.11)
	$(q_{\chi_3^\ell}, q_{\chi_2^\ell})$	(6.10)	$(q_{\chi_2^\ell}, q_{\chi_3^\ell})$	(6.7)	$(q_{\chi_2^\ell}, q_{\chi_3^\ell})$	(6.7)	$(q_{\chi_3^\ell}, q_{\chi_2^\ell})$	(6.10)
	$\chi_3^\ell = x_k$				$\chi_3^\ell = \bar{x}_k$			
	$x_k = 1$	ineq.	$x_k = 0$	ineq.	$x_k = 1$	ineq.	$x_k = 0$	ineq.
H_0^1	$(q_{x_k}, q_{\bar{x}_k})$	(6.1)	$(q_{\bar{x}_k}, q_{x_k})$	(6.2)	$(q_{x_k}, q_{\bar{x}_k})$	(6.1)	$(q_{\bar{x}_k}, q_{x_k})$	(6.2)
	$(q_{t_3^\ell}, q_{\bar{x}_k})$	(6.19)	$(q_{\bar{x}_k}, q_{t_3^\ell})$	(6.20)	$(q_{x_k}, q_{t_3^\ell})$	(6.20)	$(q_{t_3^\ell}, q_{x_k})$	(6.19)
	$(q_{t_3^\ell}, q_{\chi_1^\ell})$	(6.18)	$(q_{\chi_1^\ell}, q_{t_3^\ell})$	(6.5)	$(q_{\chi_1^\ell}, q_{t_3^\ell})$	(6.5)	$(q_{t_3^\ell}, q_{\chi_1^\ell})$	(6.18)
	$(q_{\chi_3^\ell}, q_{\chi_1^\ell})$	(6.9)	$(q_{\chi_1^\ell}, q_{\chi_3^\ell})$	(6.4)	$(q_{\chi_1^\ell}, q_{\chi_3^\ell})$	(6.4)	$(q_{\chi_3^\ell}, q_{\chi_1^\ell})$	(6.9)
H_0^2	$(q_{\bar{x}_k}, q_{x_k})$	(6.2)	$(q_{x_k}, q_{\bar{x}_k})$	(6.1)	$(q_{\bar{x}_k}, q_{x_k})$	(6.2)	$(q_{x_k}, q_{\bar{x}_k})$	(6.1)
	$(q_{\bar{x}_k}, q_{t_3^\ell})$	(6.20)	$(q_{t_3^\ell}, q_{\bar{x}_k})$	(6.19)	$(q_{t_3^\ell}, q_{x_k})$	(6.19)	$(q_{x_k}, q_{t_3^\ell})$	(6.20)
	$(q_{\chi_1^\ell}, q_{t_3^\ell})$	(6.5)	$(q_{t_3^\ell}, q_{\chi_1^\ell})$	(6.18)	$(q_{t_3^\ell}, q_{\chi_1^\ell})$	(6.18)	$(q_{\chi_1^\ell}, q_{t_3^\ell})$	(6.5)
	$(q_{\chi_1^\ell}, q_{\chi_3^\ell})$	(6.4)	$(q_{\chi_3^\ell}, q_{\chi_1^\ell})$	(6.9)	$(q_{\chi_3^\ell}, q_{\chi_1^\ell})$	(6.9)	$(q_{\chi_1^\ell}, q_{\chi_3^\ell})$	(6.4)

Table 6.6 displays two forms of symmetry. First, at the level of a literal χ_i^ℓ, $i = 1, 2, 3$ we observe that the set H_0^1 when $\chi_i^\ell = 1$ equals the set H_0^2 when $\chi_i^\ell = 0$. Second, when substituting $\bar{x}_i$ ($\bar{x}_j$, $\bar{x}_k$) for x_i (x_j, x_k), and x_i (x_j, x_k) for $\bar{x}_i$ ($\bar{x}_j$, $\bar{x}_k$), the set H_0^1 (respectively H_0^2) when $\chi_1^\ell = x_i$ ($\chi_2^\ell = x_j$, $\chi_3^\ell = x_k$) becomes the set H_0^2 (respectively H_0^1) when $\chi_1^\ell = \bar{x}_i$ ($\chi_2^\ell = \bar{x}_j$, $\chi_3^\ell = \bar{x}_k$).

To complete the definition of H_0^1 and H_0^2, we set $(q_s, q_s) \in H_0^1 \cap H_0^2$ for every $s \in \mathbb{T}$.
Remark 1**:** Notice that there is no pair of distinct observations (s, t) such that $p_s q_s \geq p_s q_t$ and $(q_s, q_t) \in H_0^1 \cap H_0^2$.

We next prove two properties of the sets H_0^1, H_0^2, H^1 and H^2 described above.
Property 1**:** For any pair of observations (s, t), if $(q_s, q_t) \in H^i$ and $p_s q_s \geq p_s q_t$ then $(q_s, q_t) \in H_0^i$, for $i = 1, 2$.

Proof. Without loss of generality, suppose that $(q_s, q_t) \in H^1$ with $p_s q_s \geq p_s q_t$. We argue by contradiction; suppose that $(q_s, q_t) \notin H_0^1$. Then by construction, $(q_s, q_t) \in H_0^2$. Since $(q_s, q_t) \in H_0^2$, we have, by construction of H_0^1 and H_0^2, that $(q_t, q_s) \in H_0^1$. Further, since $(q_s, q_t) \in H^1$ there exists a sequence (non empty, because of **Remark** 1) of observations u, v, ..., w such that (q_s, q_u), (q_u, q_v), ..., (q_w, q_t) are in H_0^1. By construction of H_0^1 and H_0^2, however, this implies that (q_t, q_w), ..., (q_v, q_u), $(q_u, q_s) \in H_0^2$. Together with the fact that $(q_s, q_t) \in H_0^2$ and $(q_t, q_s) \in H_0^1$, we get (q_s, q_u), (q_u, q_v), ..., (q_w, q_t), $(q_t, q_s) \in H_0^1$ and (q_s, q_t), (q_t, q_w), ..., (q_v, q_u), $(q_u, q_s) \in H_0^2$. In other words, for every observation $a \in \{s, t, u, \ldots, w\}$ there exist two observations b and c in $\{s, t, u, \ldots, w\}$ such that $(q_a, q_b) \in H_0^1$ and $(q_a, q_c) \in H_0^2$. It follows from the construction (see Table 6.6) that this can happen only if $a \in \{\chi_1^\ell, \chi_2^\ell, \chi_3^\ell\}$ for a given $\ell = 1, \ldots, m$. Therefore, s, t, u, ..., $w \in \{\chi_1^\ell, \chi_2^\ell, \chi_3^\ell\}$ for a given $\ell = 1, \ldots, m$. The latter result implies that the length of the sequence u, v, ..., w is one (suppose that the sequence contains only u) and we have (q_s, q_u) and (q_u, q_t) in H_0^1. If, in addition $(q_t, q_s) \in H_0^1$, then we have $(q_{\chi_1^\ell}, q_{\chi_2^\ell})$, $(q_{\chi_2^\ell}, q_{\chi_3^\ell})$ and $(q_{\chi_3^\ell}, q_{\chi_1^\ell})$ in H_0^1, which is only possible if the variables χ_1^ℓ, χ_2^ℓ and χ_3^ℓ are assigned the same value. Since these three variables are in the same clause C_ℓ, this contradicts the fact that we have a truth assignment that is a solution to the Not-All-Equal-3Sat instance. □

Property 2**:** For any triple of observations (s, t_1, t_2) satisfying $p_s q_s \geq p_s(q_{t_1} + q_{t_2})$, the pair of bundles (q_s, q_{t_1}) and (q_s, q_{t_2}) are in the same set H_0^i for $i = 1, 2$.

Proof. It is not difficult to check this result from Table 6.6. □

We now prove that the hypothetical relations H_0^1 and H_0^2 defined above satisfy *Rules* 1-5.

Rule 1**:** This rule is satisfied because on the one hand, a pair of distinct observations s, t in $\mathbb{T}$ occurring in $p_s q_s \geq p_s q_t$ is identified by one of the inequalities (6.1)-(6.20) and therefore is by construction either in H_0^1 or in H_0^2. On the other hand, $(q_s, q_s) \in H_0^1 \cap H_0^2$ by construction.

Rule 2**:** Suppose that $p_s q_s \geq p_s q_t$ and $(q_t, q_s) \in H^1$. We know by construction that $p_s q_s \geq p_s q_t$ implies $p_t q_t \geq p_t q_s$ and from Property 1 we have $(q_t, q_s) \in H_0^1$; which implies $(q_s, q_t) \in H_0^2$.

Rule 3**:** Suppose that $p_s q_s \geq p_s(q_{t_1} + q_{t_2})$ and $(q_{t_1}, q_s) \in H^1$. Property 1 implies that $(q_{t_1}, q_s) \in H_0^1$ and Property 2 implies that (q_s, q_{t_1}) and (q_s, q_{t_2}) are in the same set. Since $(q_{t_1}, q_s) \in H_0^1$, we conclude that $(q_s, q_{t_2}) \in H_0^2$.

Rule 4**:** This rule follows from the fact that $H_0^1 \cap H_0^2$ contains only (q_s, q_s) where s is a given observation in $\mathbb{T}$. Thus, for two distinct observations s and t with $p_s q_s > p_s q_t$, $(q_s, q_t) \notin H_0^1 \cap H_0^2$.

Rule 5**:** Suppose that there exist $s, t_1, t_2 \in \mathbb{T}$ such that $p_s q_s > p_s(q_{t_1} + q_{t_2})$ and $(q_{t_1}, q_s) \in H^1$. Then $(q_{t_1}, q_s) \in H_0^1$ from Property 1 and $(q_s, q_{t_2}) \in H_0^2$ from Property 2. Therefore $(q_{t_2}, q_s) \notin H_0^2$ because of Lemma 6.3.

This concludes the proof that if the instance of the Not-All-Equal-3Sat is a YES instance then the data set S satisfies CARP.

6.3 Conclusion

In this chapter, we prove that the problem of testing the Collective Axiom of Revealed Preference (CARP) is NP-complete even for two-member household. This result justifies the enumerative approaches that are used in Cherchye et al. (2008a) and the heuristic approaches presented in Chapter 4 to test a given data set for CARP.

Appendix

In Appendix A, we derive the value of the scalar product $p_s q_t$ for each pair of observations s and t in $\mathbb{T}$. Next, in Appendix B we prove **Claim** 1, and in Appendix C we

prove **Claim** 2.

A. Scalar product of observations in $\mathbb{T}$

In what follows, we first specify the quantity $p_s q_t$ for every pair of observations s and t in $\mathbb{T}$. The symbol $\cong$ is used to mean that the value reported of $p_s q_t$ is the limit when ϵ tends to 0 of the exact value. We consider five cases.

Case 1. Both observations are variable observations; that is $s, t \in \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$.

Below, we distinguish eight types of combinations as follows:

> **How to compute $p_{x_i} q_{x_i}$ when $s = t = x_i$ with $i = 1, \ldots, n$.**
>
> Notice that the scalar product $p_{x_i} q_{x_i}$ is not affected by goods corresponding to cells with quantity zero. Further, as we take the limit when ε tends to zero, only goods corresponding to cells in Block 1 and in Block 2 with price different from ε are considered. In Block 1, this restriction allows to consider only goods corresponding to cells in row x_i. The quantity $p_{x_i} q_{x_i}$ contains a part coming from the good corresponding to cell (x_i, x_i) in Block 1. This accounts for $2 \times 1 = 1$ in $p_{x_i} q_{x_i}$ since the price is 2 and the quantity is 1. Looking at the vector of quantity, the good corresponding to cell $(x_i, \bar{x}_i)$ has a quantity of one and contributes for $1 \times 1 = 1$ in $p_{x_i} q_{x_i}$. Moreover, we know that for every clause C_ℓ with $\ell \in \Gamma_{\bar{x}_i}$, the good corresponding to cell (x_i, t^ℓ_r) in row x_i gets the value one (here, $r \in \{1, 2, 3\}$ is the position of $\bar{x}_i$ in the clause C_ℓ). Thus each such good adds the value of one to $p_{x_i} q_{x_i}$ and there are $|\Gamma_{\bar{x}_i}|$ such goods. There are no additional value coming from the goods corresponding to the remaining cells in Block 1. In total, goods corresponding ti cells in Block 1 contribute for $2 + 1 + |\Gamma_{\bar{x}_i}|$ in $p_{x_i} q_{x_i}$. For goods corresponding to cells in Block 2, we know by construction that there are $2 \times |\Gamma_{\bar{x}_i}|$ goods with price $\frac{1}{2 \times |\Gamma_{\bar{x}_i}|}$ and the quantity of $|\Gamma_{\bar{x}_i}| + 1$. Therefore, if $\Gamma_{\bar{x}_i} \neq \emptyset$ then the goods corresponding to cells in Block 2 contribute for $2 \times |\Gamma_{\bar{x}_i}| \left(\frac{1}{2 \times |\Gamma_{\bar{x}_i}|} \times (|\Gamma_{\bar{x}_i}| + 1) \right) = |\Gamma_{\bar{x}_i}| + 1$. Notice that if $\Gamma_{\bar{x}_i} = \emptyset$ then that contribution is zero. Putting together the contribution of goods corresponding to cells in Block 1 and in Block 2, we obtain $p_{x_i} q_{x_i} \cong 2 + 1 + |\Gamma_{\bar{x}_i}| + |\Gamma_{\bar{x}_i}| + 1 = 2|\Gamma_{\bar{x}_i}| + 4$ if $\Gamma_{\bar{x}_i} \neq \emptyset$ and $p_{x_i} q_{x_i} \cong 2 + 1$ if $\Gamma_{\bar{x}_i} = \emptyset$. Therefore,

$$p_{x_i} q_{x_i} \quad \cong \quad \begin{cases} 3 & \text{if } \Gamma_{\bar{x}_i} = \emptyset \\ 2|\Gamma_{\bar{x}_i}| + 4 & \text{if } \Gamma_{\bar{x}_i} \neq \emptyset. \end{cases} \tag{6.33}$$

How to compute $p_{\bar{x}_i} q_{\bar{x}_i}$ when $s = t = \bar{x}_i$ with $i = 1, \ldots, n$.
Following the procedure above, the scalar product

$$p_{\bar{x}_i} q_{\bar{x}_i} \cong \begin{cases} 3 & \text{if } \Gamma_{x_i} = \emptyset \\ 2|\Gamma_{x_i}| + 4 & \text{if } \Gamma_{x_i} \neq \emptyset. \end{cases} \tag{6.34}$$

How to compute $p_{x_i} q_{\bar{x}_i}$ when $s = x_i$ and $t = \bar{x}_i$ with $i = 1, \ldots, n$.
To compute the scalar product $p_{x_i} q_{\bar{x}_i}$, observe that goods corresponding to cells in Block 2 do not affect that scalar product. In fact, for the observation x_i, the only goods in Block 2 with price different from ε corresponding to cells either in row $\bar{x}_i$ or in column $\bar{x}_i$ while the goods of observation $\bar{x}_i$ in Block 2 with non-zero quantity corresponding to cells either in row x_i or in column x_i. Therefore, the scalar product $p_{x_i} q_{\bar{x}_i}$ is based on goods corresponding to cells in Block 1. In that block, only goods corresponding to cells in row x_i are interesting as they have a non-ε price. However, for observation $\bar{x}_i$ the only good in row x_i of Block 1 with non-zero quantity corresponding to cell $(x_i, \bar{x}_i)$ with quantity $|\Gamma_{\bar{x}_i}| + 1$. This implies that

$$p_{x_i} q_{\bar{x}_i} \cong |\Gamma_{\bar{x}_i}| + 1. \tag{6.35}$$

How to compute $p_{\bar{x}_i} q_{x_i}$ when $s = \bar{x}_i$ and $t = x_i$ with $i = 1, \ldots, n$.
An analysis following the reasoning used above leads to

$$p_{\bar{x}_i} q_{x_i} \cong |\Gamma_{x_i}| + 1. \tag{6.36}$$

How to compute $p_{x_i} q_{x_j}$ when $s = x_i$ and $t = x_j$ with $i, j \in \{1, \ldots, n\}$, $i \neq j$.
We are not going to compute $p_{x_i} q_{x_j}$ but we will rather provide a lower bound to $p_{x_i} q_{x_j}$. Notice that the scalar product $p_{x_i} q_{x_j}$ is at least as large as the contribution of good corresponding to cell (x_i, x_j) in Block 1. The latter good contributes $1 \times \Delta = \Delta$ to $p_{x_i} q_{x_j}$ because the cell (x_i, x_j) is in the column x_j and in that column, the observation x_j is such that only goods corresponding to cells (x_j, x_j), $(\bar{x}_j, x_j)$ and (t_r^ℓ, x_j) where the clause C_ℓ contains $\bar{x}_j$ (r being the position of $\bar{x}_j$ in C_ℓ), have quantity different from Δ. Therefore

$$p_{x_i} q_{x_j} \geq \Delta. \tag{6.37}$$

How to compute $p_{\bar{x}_i} q_{x_j}$ when $s = \bar{x}_i$ and $t = x_j$ with $i, j \in \{1, \ldots, n\}$, $i \neq j$.
Similarly, $p_{\bar{x}_i} q_{x_j}$ is greater than or equal to Δ using the same reasoning as above. Thus,

$$p_{\bar{x}_i} q_{x_j} \geq \Delta. \tag{6.38}$$

How to compute $p_{x_i} q_{\bar{x}_j}$ when $s = x_i$ and $t = \bar{x}_j$ with $i, j \in \{1, \ldots, n\}$, $i \neq j$.
The quantity $p_{x_i} q_{\bar{x}_j}$ is at least as large as the contribution of the good corresponding to cell $(x_i, \bar{x}_j)$ in Block 1. That contribution is $1 \times \Delta = \Delta$. Therefore

$$p_{x_i} q_{\bar{x}_j} \geq \Delta. \tag{6.39}$$

How to compute $p_{\bar{x}_i} q_{\bar{x}_j}$ when $s = \bar{x}_i$ and $t = \bar{x}_j$ with $i, j \in \{1, \ldots, n\}$, $i \neq j$.
The quantity $p_{\bar{x}_i} q_{\bar{x}_j}$ is at least as large as the contribution of the good corresponding to cell $(\bar{x}_i, \bar{x}_j)$ in Block 1. However, that good contributes for $1 \times \Delta = \Delta$. Therefore

$$p_{\bar{x}_i} q_{\bar{x}_j} \geq \Delta. \tag{6.40}$$

Case 2. Both observations are clause observations, corresponding to some clause C_ℓ with $\ell = 1, \ldots, m$.
This means that $s, t \in \{\chi_1^\ell, \chi_2^\ell, \chi_3^\ell, t_1^\ell, t_2^\ell, t_3^\ell\}$. There are 36 possibilities listed below.

How to compute $p_{\chi_1^\ell} q_{\chi_1^\ell}$, $p_{\chi_2^\ell} q_{\chi_2^\ell}$ and $p_{\chi_3^\ell} q_{\chi_3^\ell}$.
Consider the scalar product $p_{\chi_1^\ell} q_{\chi_1^\ell}$. It is neither affected by goods corresponding to cells with quantity zero nor by goods corresponding to cells in Block 1 and Block 2 with price ε. In Block 1, only goods corresponding to cells in row χ_1^ℓ are considered. The quantity $p_{\chi_1^\ell} q_{\chi_1^\ell}$ contains a part coming from the goods corresponding to cells $(\chi_1^\ell, \chi_1^\ell)$, $(\chi_1^\ell, \chi_2^\ell)$, $(\chi_1^\ell, \chi_3^\ell)$ and (χ_1^ℓ, t_3^ℓ) in Block 1. Each of these goods has a price of one and a quantity of one except the good corresponding to cell $(\chi_1^\ell, \chi_1^\ell)$ which has a price of two and a quantity of one. Therefore, they contribute $2 + 1 + 1 + 1 = 5$ in $p_{\chi_1^\ell} q_{\chi_1^\ell}$. As for goods corresponding to cells in Block 2, we know that there are two goods corresponding to cells (χ_3^ℓ, t_3^ℓ) and (t_3^ℓ, χ_3^ℓ) with price $\frac{1}{2}$ and quantity three. These two goods contribute $2(\frac{1}{2} \times 3) = 3$. In total,

$$p_{\chi_1^\ell} q_{\chi_1^\ell} \cong 5 + 3 = 8. \tag{6.41}$$

A similar analysis leads to

$$p_{\chi_2^\ell} q_{\chi_2^\ell} \cong 8 \tag{6.42}$$

and

$$p_{\chi_3^\ell} q_{\chi_3^\ell} \cong 8. \tag{6.43}$$

How to compute $p_{\chi_1^\ell} q_{\chi_2^\ell}$, $p_{\chi_2^\ell} q_{\chi_3^\ell}$ and $p_{\chi_3^\ell} q_{\chi_1^\ell}$.
The scalar product $p_{\chi_1^\ell} q_{\chi_2^\ell}$ is affected only by the good of Block 1 corresponding to cell $(\chi_1^\ell, \chi_2^\ell)$. In fact, this good is in row χ_1^ℓ, and therefore gets the price of one in observation χ_1^ℓ. Moreover, the observation χ_2^ℓ uses six units of that good. It is not difficult to see that the goods corresponding to the remaining cells in row χ_1^ℓ of Block 1 get the quantity zero for observation χ_2^ℓ and the goods (χ_3^ℓ, t_3^ℓ) and (t_3^ℓ, χ_3^ℓ) which are the only goods of Block 2 with non-ε price for observation χ_1^ℓ have a quantity of zero for observation χ_2^ℓ; therefore do not contribute in $p_{\chi_1^\ell} q_{\chi_2^\ell}$. Thus

$$p_{\chi_1^\ell} q_{\chi_2^\ell} \cong 6. \tag{6.44}$$

Similarly, we get

$$p_{\chi_2^\ell} q_{\chi_3^\ell} \cong 6 \tag{6.45}$$

and

$$p_{\chi_3^\ell} q_{\chi_1^\ell} \cong 6. \tag{6.46}$$

How to compute $p_{\chi_1^\ell} q_{\chi_3^\ell}$, $p_{\chi_2^\ell} q_{\chi_1^\ell}$ and $p_{\chi_3^\ell} q_{\chi_2^\ell}$.
The scalar product $p_{\chi_1^\ell} q_{\chi_3^\ell}$ is affected only by the good of Block 1 corresponding to cell $(\chi_1^\ell, \chi_3^\ell)$. That good is in row χ_1^ℓ therefore gets the price of one for observation χ_1^ℓ. Moreover, the observation χ_3^ℓ uses four units of that good. Thus

$$p_{\chi_1^\ell} q_{\chi_3^\ell} \cong 4. \tag{6.47}$$

Similarly, we get

$$p_{\chi_2^\ell} q_{\chi_1^\ell} \cong 4 \tag{6.48}$$

and

$$p_{\chi_3^\ell} q_{\chi_2^\ell} \cong 4. \tag{6.49}$$

How to compute $p_{t_1^\ell} q_{t_1^\ell}$, $p_{t_2^\ell} q_{t_2^\ell}$ and $p_{t_3^\ell} q_{t_3^\ell}$.
The observation t_1^ℓ is associated to χ_1^ℓ. There are two options for the literal χ_1^ℓ, either $\chi_1^\ell = x_i$ or $\chi_1^\ell = \bar{x}_i$.

If $\chi_1^\ell = x_i$ then the quantity $p_{t_1^\ell} q_{t_1^\ell}$ contains a part coming from the goods corresponding to cells (t_1^ℓ, t_1^ℓ), (t_1^ℓ, χ_2^ℓ) and $(t_1^\ell, \bar{x}_i)$ in Block 1. The first good has a price of two and a quantity of one, while the two others have a price of one and a quantity of one. Therefore, they account for $2+1+1 = 4$ in $p_{t_1^\ell} q_{t_1^\ell}$. In Block 2, the two goods corresponding to cells $(\chi_2^\ell, \bar{x}_i)$ and $(\bar{x}_i, \chi_2^\ell)$ with price $\frac{1}{2}$ and quantity two are the only goods contributing to $p_{t_1^\ell} q_{t_1^\ell}$. They contribute for $2(\frac{1}{2} \times 2) = 2$. In sum, $p_{t_1^\ell} q_{t_1^\ell} \cong 4 + 2 = 6$.

On the other hand, if $\chi_1^\ell = \bar{x}_i$ then the quantity $p_{t_1^\ell} q_{t_1^\ell}$ contains a part coming from the goods corresponding to cells (t_1^ℓ, t_1^ℓ), (t_1^ℓ, χ_2^ℓ) and (t_1^ℓ, x_i) in Block 1. As above, these goods account for $2 + 1 + 1 = 4$ in $p_{t_1^\ell} q_{t_1^\ell}$. As for goods in Block 2, only two goods corresponding to cells (χ_2^ℓ, x_i) and (x_i, χ_2^ℓ) with price $\frac{1}{2}$ and quantity two contribute to $p_{t_1^\ell} q_{t_1^\ell}$. They contribute for $2(\frac{1}{2} \times 2) = 2$. In total, we obtain $p_{t_1^\ell} q_{t_1^\ell} \cong 4 + 2 = 6$.

To summarize, whether $\chi_1^\ell = x_i$ or $\chi_1^\ell = \bar{x}_i$, we have

$$p_{t_1^\ell} q_{t_1^\ell} \cong 6. \tag{6.50}$$

We also obtain, using similar reasoning that

$$p_{t_2^\ell} q_{t_2^\ell} \cong 6 \tag{6.51}$$

and

$$p_{t_3^\ell} q_{t_3^\ell} \cong 6. \tag{6.52}$$

How to compute $p_s q_t$ when $s, t \in \{t_1^\ell, t_2^\ell, t_3^\ell\}$ with $s \neq t$.
The scalar product $p_s q_t$ is greater than or equal to the contribution of the good corresponding to cell (s, t) in Block 1. However, the cell (s, t) being in row s of Block 1, it has the price of one for observation s. But that cell is in column t in Block 1 and gets the value Δ as quantity. Therefore

$$p_s q_t \geq \Delta. \tag{6.53}$$

The set of inequalities (6.53) represents six values of $p_s q_t$.

How to compute $p_{\chi_1^\ell} q_{t_3^\ell}$, $p_{\chi_2^\ell} q_{t_1^\ell}$ and $p_{\chi_3^\ell} q_{t_2^\ell}$.
The scalar product $p_{\chi_1^\ell} q_{t_3^\ell}$ is determined only by the good corresponding to cell (χ_1^ℓ, t_3^ℓ) in Block 1. This good has a price of one for observation χ_1^ℓ and a quantity of three for observation t_3^ℓ. Therefore, it accounts for $1 \times 3 = 3$ in $p_{\chi_1^\ell} q_{t_3^\ell}$. Notice that the good corresponding to cell (χ_1^ℓ, t_3^ℓ) is the only good of observation t_3^ℓ in row χ_1^ℓ of Block 1 with non-zero quantity. As for goods corresponding to cells in Block 2, we know that the two goods of observation t_3^ℓ in Block 2 with non-zero quantity have the price of ε for observation χ_1^ℓ. Therefore,

$$p_{\chi_1^\ell} q_{t_3^\ell} \cong 3. \tag{6.54}$$

The following similar results hold.

$$p_{\chi_2^\ell} q_{t_1^\ell} \cong 3, \tag{6.55}$$

$$p_{\chi_3^\ell} q_{t_2^\ell} \cong 3. \tag{6.56}$$

How to compute $p_{t_3^\ell} q_{\chi_1^\ell}$, $p_{t_1^\ell} q_{\chi_2^\ell}$ and $p_{t_2^\ell} q_{\chi_3^\ell}$.
The scalar product $p_{t_3^\ell} q_{\chi_1^\ell}$ is mainly determined by the good corresponding to cell (t_3^ℓ, χ_1^ℓ) in Block 1. This good has a price of one for observation t_3^ℓ and a quantity of two for observation χ_1^ℓ. Therefore, it accounts for $1 \times 2 = 2$ in $p_{t_3^\ell} q_{\chi_1^\ell}$ and

$$p_{t_3^\ell} q_{\chi_1^\ell} \cong 2. \tag{6.57}$$

The following similar results hold.

$$p_{t_1^\ell} q_{\chi_2^\ell} \cong 2, \tag{6.58}$$

$$p_{t_2^\ell} q_{\chi_3^\ell} \cong 2. \tag{6.59}$$

How to compute $p_s q_t$ and $p_t q_s$ when $s \in \{t_1^\ell, t_2^\ell, t_3^\ell\}$, $t \in \{\chi_1^\ell, \chi_2^\ell, \chi_3^\ell\}$ and $(s,t) \notin \{(t_1^\ell, \chi_2^\ell), (t_2^\ell, \chi_3^\ell), (t_3^\ell, \chi_1^\ell)\}$, $(t,s) \notin \{(\chi_2^\ell, t_1^\ell), (\chi_3^\ell, t_2^\ell), (\chi_1^\ell, t_3^\ell)\}$.
The scalar product $p_s q_t$ is at least as large as the contribution of the good corresponding to cell (s,t) in Block 1. However, the cell (s,t) being in row s of Block 1, it has the price of one for observation s. But that cell is in column t of Block 1 and gets the value Δ as quantity for observation t. Therefore

$$p_s q_t \geq \Delta. \tag{6.60}$$

Similarly, we prove that

$$p_t q_s \geq \Delta. \tag{6.61}$$

These are 12 additional scalar products; completing the description of the 36 scalar products announced.

Case 3. One observation is a variable observation, the other is a clause observation such that the corresponding clause does not contain that variable. This means one observation is in $\{\chi_1^\ell, \chi_2^\ell, \chi_3^\ell, t_1^\ell, t_2^\ell, t_3^\ell\}$ from clause $C_\ell = (\chi_1^\ell \vee \chi_2^\ell \vee \chi_3^\ell)$ while the other is a variable observation x_i or $\bar{x}_i$ $(i = 1, 2, \ldots, n)$ which is such that x_i or $\bar{x}_i$ is not in $\{\chi_1^\ell, \chi_2^\ell, \chi_3^\ell\}$. Let $s \in \{\chi_1^\ell, \chi_2^\ell, \chi_3^\ell, t_1^\ell, t_2^\ell, t_3^\ell\}$ and t be a variable observation satisfying the above condition.

How to compute $p_t q_s$ and $p_s q_t$.
The value of the scalar product $p_t q_s$ is least as large as the contribution of good corresponding to cell (t, s) in Block 1. Since the price of that good equals 1, and its quantity equals Δ, we get

$$p_t q_s \geq \Delta. \tag{6.62}$$

Using similar arguments, we can prove that

$$p_s q_t \geq \Delta. \tag{6.63}$$

Case 4. One observation is a variable observation, the other is a clause observation such that the corresponding clause contains that variable.
This means one observation is in $\{\chi_1^\ell, \chi_2^\ell, \chi_3^\ell, t_1^\ell, t_2^\ell, t_3^\ell\}$ from clause $C_\ell = (\chi_1^\ell \vee \chi_2^\ell \vee \chi_3^\ell)$ while the other is a variable observation x_i or $\bar{x}_i$ which is such that x_i or $\bar{x}_i$ is in $\{\chi_1^\ell, \chi_2^\ell, \chi_3^\ell\}$. Let $s \in \{\chi_1^\ell, \chi_2^\ell, \chi_3^\ell, t_1^\ell, t_2^\ell, t_3^\ell\}$ and t be a variable observation satisfying the above condition.

How to compute $p_{x_i} q_{t_1^\ell}$ and $p_{t_1^\ell} q_{x_i}$, when $\chi_1^\ell = x_i$.
The value of the scalar product $p_{x_i} q_{t_1^\ell}$ is at least as large as the contribution of good corresponding to cell (x_i, t_1^ℓ) in Block 1. Since the price of that good equals 1, and its quantity equals Δ, we get

$$p_{x_i} q_{t_1^\ell} \geq \Delta. \tag{6.64}$$

Using similar arguments, we get

$$p_{t_1^\ell} q_{x_i} \geq \Delta. \tag{6.65}$$

Similar inequalities hold when $\chi_2^\ell = x_j$ and $\chi_3^\ell = x_k$. These are

$$p_{x_j} q_{t_2^\ell} \geq \Delta, \tag{6.66}$$

$$p_{t_2^\ell} q_{x_j} \geq \Delta, \tag{6.67}$$

$$p_{x_k} q_{t_3^\ell} \geq \Delta, \tag{6.68}$$

$$p_{t_3^\ell} q_{x_k} \geq \Delta. \tag{6.69}$$

How to compute $p_{\bar{x}_i} q_{t_1^\ell}$ and $p_{t_1^\ell} q_{\bar{x}_i}$ when $\chi_1^\ell = x_i$.
The scalar product $p_{\bar{x}_i} q_{t_1^\ell}$ is not affected by goods corresponding to cells in Block 2. The scalar product $p_{\bar{x}_i} q_{t_1^\ell}$ is determined by the good corresponding to cell $(\bar{x}_i, t_1^\ell)$ in Block 1. That good has a price of one and a quantity of $|\Gamma_{x_i}| + 1$. Therefore, it accounts for $1 \times (|\Gamma_{x_i}| + 1) = |\Gamma_{x_i}| + 1$ in $p_{\bar{x}_i} q_{t_1^\ell}$. Hence,

$$p_{\bar{x}_i} q_{t_1^\ell} \cong |\Gamma_{x_i}| + 1. \tag{6.70}$$

The scalar product $p_{t_1^\ell} q_{\bar{x}_i}$ is determined by the contribution of good corresponding to cell $(t_1^\ell, \bar{x}_i)$ in Block 1. That good has a price of one and a quantity of two. Thus,

$$p_{t_1^\ell} q_{\bar{x}_i} \cong 2. \tag{6.71}$$

The inequalities similar to those above hold for the pair of observations t_2^ℓ and $\bar{x}_j$; and t_3^ℓ and $\bar{x}_k$ when $\chi_2^\ell = x_j$ and $\chi_3^\ell = x_k$. There are given by

$$p_{\bar{x}_j} q_{t_2^\ell} \cong |\Gamma_{x_j}| + 1, \tag{6.72}$$

$$p_{t_2^\ell} q_{\bar{x}_j} \cong 2, \tag{6.73}$$

$$p_{\bar{x}_k} q_{t_3^\ell} \cong |\Gamma_{x_k}| + 1, \tag{6.74}$$

$$p_{t_3^\ell} q_{\bar{x}_k} \cong 2. \tag{6.75}$$

How to compute $p_{\bar{x}_i} q_{t_1^\ell}$ and $p_{t_1^\ell} q_{\bar{x}_i}$ when $\chi_1^\ell = \bar{x}_i$.
Inequalities similar to those obtained when $\chi_1^\ell = x_i$ hold. These are

$$p_{\bar{x}_i} q_{t_1^\ell} \geq \Delta, \tag{6.76}$$

$$p_{t_1^\ell} q_{\bar{x}_i} \geq \Delta, \tag{6.77}$$

If $\chi_2^\ell = \bar{x}_j$ and $\chi_3^\ell = \bar{x}_k$ then

$$p_{\bar{x}_j} q_{t_2^\ell} \geq \Delta, \tag{6.78}$$

$$p_{t_2^\ell} q_{\bar{x}_j} \geq \Delta, \tag{6.79}$$

$$p_{\bar{x}_k} q_{t_3^\ell} \geq \Delta, \tag{6.80}$$

$$p_{t_3^\ell} q_{\bar{x}_k} \geq \Delta. \tag{6.81}$$

How to compute $p_{x_i} q_{t_1^\ell}$ and $p_{t_1^\ell} q_{x_i}$ when $\chi_1^\ell = \bar{x}_i$.

$$p_{x_i} q_{t_1^\ell} \cong |\Gamma_{\bar{x}_i}| + 1, \tag{6.82}$$

$$p_{t_1^\ell} q_{x_i} \cong 2, \tag{6.83}$$

$$p_{x_j} q_{t_2^\ell} \cong |\Gamma_{\bar{x}_j}| + 1, \tag{6.84}$$

$$p_{t_2^\ell} q_{x_j} \cong 2, \tag{6.85}$$

$$p_{x_k} q_{t_3^\ell} \cong |\Gamma_{\bar{x}_k}| + 1, \tag{6.86}$$

$$p_{t_3^\ell} q_{x_k} \cong 2. \tag{6.87}$$

How to compute $p_{x_i}q_{t_r^\ell}$ and $p_{\bar{x}_i}q_{t_r^\ell}$ when $r \in \{1,2,3\}$ and $\chi_r^\ell \notin \{x_i, \bar{x}_i\}$.
The value of the scalar product $p_{x_i}q_{t_r^\ell}$ is least as large as the contribution of good corresponding to cell (x_i, t_r^ℓ) in Block 1. Since the price of that good equals 1, and its quantity equals Δ, we get

$$p_{x_i}q_{t_r^\ell} \geq \Delta. \tag{6.88}$$

Similarly,

$$p_{t_r^\ell}q_{x_i} \geq \Delta, \tag{6.89}$$

$$p_{\bar{x}_i}q_{t_r^\ell} \geq \Delta, \tag{6.90}$$

and

$$p_{t_r^\ell}q_{\bar{x}_i} \geq \Delta. \tag{6.91}$$

How to compute p_sq_t and p_tq_s when $s \in \{\chi_1^\ell, \chi_2^\ell, \chi_3^\ell\}$ and t is a variable observation.
It is not difficult to prove that all these scalar products are at least as large as Δ. That is

$$p_sq_t \geq \Delta \tag{6.92}$$

and

$$p_tq_s \geq \Delta. \tag{6.93}$$

Case 5. Both observations are clause observations; one of them from clause C_{ℓ_1}, the other from clause C_{ℓ_2} with $\ell_1 \neq \ell_2$.
Let $s \in \{\chi_1^{\ell_1}, \chi_2^{\ell_1}, \chi_3^{\ell_1}, t_1^{\ell_1}, t_2^{\ell_1}, t_3^{\ell_1}\}$ and $t \in \{\chi_1^{\ell_2}, \chi_2^{\ell_2}, \chi_3^{\ell_2}, t_1^{\ell_2}, t_2^{\ell_2}, t_3^{\ell_2}\}$. These are 36 pairs of observations (s,t).

How to compute p_sq_t and p_tq_s.
It is not difficult to obtain the following lower bounds.

$$p_sq_t \geq \Delta \tag{6.94}$$

and

$$p_tq_s \geq \Delta. \tag{6.95}$$

Notice that for two distinct observations s and t in $\mathbb{T}$, we have $p_s q_t \geq \Delta$ if and only if $p_t q_s \geq \Delta$.

In Table 6.7, we summarize the scalar products computed above by presenting only those which have values less than Δ.

B. Proof of Claim 1

We are now in a position to finish the proof of **Claim** 1. Here we show how the inequalities identified by **Claim** 1 follow from the scalar product computed in Appendix A.

The first set of inequalities (6.1) comes from (A) and (C) in Table 6.7.

The inequalities (6.2) stem from (B) and (D) in Table 6.7.

The inequalities (6.3), (6.7) and (6.9) stem from (E) and (G) in Table 6.7.

The inequalities (6.4), (6.6) and (6.10) stem from (E) and (H) in Table 6.7.

The inequalities (6.5), (6.8) and (6.11) stem from (E) and (I) in Table 6.7.

The inequalities (6.13), (6.16) and (6.19) stem from either (F) and (L) or (F) and (N), in Table 6.7.

The inequalities (6.12), (6.15) and (6.18) stem from (F) and (J) in Table 6.7.

The inequalities (6.14), (6.17) and (6.20) stem from either (B) and (K) or (A) and (M), in Table 6.7.

The set of inequalities (6.21) come from the fact that for any other pair of observations $s, t \in \mathbb{T}$, the scalar product $p_s q_t$ is greater than or equal to Δ.

C. Proof of Claim 2

Here, we show how the double-sum inequalities described by **Claim** 2 originate from the scalar products computed in Appendix A. For every clause $C_\ell = (\chi_1^\ell \vee \chi_2^\ell \vee \chi_3^\ell) \in C$, $\ell \in \{1, \ldots, m\}$ with the given clause observations χ_1^ℓ, χ_2^ℓ, χ_3^ℓ, t_1^ℓ, t_2^ℓ and t_3^ℓ, we have:

The double-sum inequalities (6.22), (6.23) and (6.24) come from (E), (H) and (I) in Table 6.7.

Table 6.7: *Summary of scalar products with values less than* Δ

id.	product		value			proof
(A)	$p_{x_i} q_{x_i}$	$=$	$\begin{cases} 3 & \text{if } \Gamma_{\bar{x}_i} = \emptyset \\ 2\lvert\Gamma_{\bar{x}_i}\rvert + 4 & \text{if } \Gamma_{\bar{x}_i} \neq \emptyset \end{cases}$	$i = 1, \ldots, n$		(6.33)
(B)	$p_{\bar{x}_i} q_{\bar{x}_i}$	$=$	$\begin{cases} 3 & \text{if } \Gamma_{x_i} = \emptyset \\ 2\lvert\Gamma_{x_i}\rvert + 4 & \text{if } \Gamma_{x_i} \neq \emptyset \end{cases}$	$i = 1, \ldots, n$		(6.34)
(C)	$p_{x_i} q_{\bar{x}_i}$	$=$	$\lvert\Gamma_{\bar{x}_i}\rvert + 1$	$i = 1, \ldots, n$		(6.35)
(D)	$p_{\bar{x}_i} q_{x_i}$	$=$	$\lvert\Gamma_{x_i}\rvert + 1$	$i = 1, \ldots, n$		(6.36)
(E)	$p_{\chi_r^\ell} q_{\chi_r^\ell}$	$=$	8	$l = 1, \ldots, m,\ r = 1, 2, 3$		(6.41), (6.42), (6.43)
(F)	$p_{t_r^\ell} q_{t_r^\ell}$	$=$	6	$l = 1, \ldots, m,\ r = 1, 2, 3$		(6.50), (6.51), (6.52)
(G)	$p_{\chi_i^\ell} q_{\chi_j^\ell}$	$=$	6	$l = 1, \ldots, m,\ (i, j) \in \{(1,2), (2,3), (3,1)\}$		(6.44), (6.45), (6.46)
(H)	$p_{\chi_i^\ell} q_{\chi_j^\ell}$	$=$	4	$l = 1, \ldots, m,\ (i, j) \in \{(1,3), (2,1), (3,2)\}$		(6.47), (6.48), (6.49)
(I)	$p_{\chi_i^\ell} q_{t_j^\ell}$	$=$	3	$l = 1, \ldots, m,\ (i, j) \in \{(1,3), (2,1), (3,2)\}$		(6.54), (6.55), (6.56)
(J)	$p_{t_i^\ell} q_{\chi_j^\ell}$	$=$	2	$l = 1, \ldots, m,\ (i, j) \in \{(3,1), (1,2), (2,3)\}$		(6.57), (6.58), (6.59)
(K)	$p_{\bar{x}_i} q_{t_r^\ell}$	$=$	$\lvert\Gamma_{x_i}\rvert + 1$ if $\chi_r^\ell = x_i$	$l = 1, \ldots, m,\ r = 1, 2, 3,\ i = 1, \ldots, n$		(6.70), (6.72), (6.74)
(L)	$p_{t_r^\ell} q_{\bar{x}_i}$	$=$	2 if $\chi_r^\ell = x_i$	$l = 1, \ldots, m,\ r = 1, 2, 3,\ i = 1, \ldots, n$		(6.71), (6.73), (6.75)
(M)	$p_{x_i} q_{t_r^\ell}$	$=$	$\lvert\Gamma_{\bar{x}_i}\rvert + 1$ if $\chi_r^\ell = \bar{x}_i$	$l = 1, \ldots, m,\ r = 1, 2, 3,\ i = 1, \ldots, n$		(6.82), (6.84), (6.86)
(N)	$p_{t_r^\ell} q_{x_i}$	$=$	2 if $\chi_r^\ell = \bar{x}_i$	$l = 1, \ldots, m,\ r = 1, 2, 3,\ i = 1, \ldots, n$		(6.83), (6.85), (6.87)

The inequalities (6.25), (6.27) and (6.29) stem from (F), (J) and either (L) or (N) in Table 6.7.

The inequalities (6.26), (6.28) and (6.30) stem from either (B), (D) and (K) or (A), (C) and (M) in Table 6.7.

The inequalities (6.31) stem from (A) and (M) and the inequalities (6.32) from (B) and (K) in Table 6.7.

The non-existence of the other possible inequalities is justified by the fact that for those inequalities, at least one scalar product appearing in the right hand side has a value greater than or equal to Δ.

List of Figures

List of Tables

Bibliography

Aarts, E., Lenstra, J. K. (Eds.), 1997. Local Search in Combinatorial Optimization. Wiley.

Afriat, S., 1967. The construction of utility functions from expenditure data. International Economic Review 8, 67–77.

Ahuja, R. K., Magnanti, T. L., Orlin, J. B., 1993. Network Flows: Theory, Algorithms, and Applications. Prentice-Hall.

Aifeng, Y., Jinjiang, Y., 1991. On the vertex arboricity of planar graphs of diameter two. Discrete Mathematics 307, 2438–2447.

Air Transport Action Group, December 2007. The economic and social benefits of air transport. Tech. rep., Air Transport Action Group.

Alekseev, V. E., Farrugia, A., Lozin, V. V., 2004. New results on generalized graph coloring. Discrete Mathematics and Theoretical Computer Science 6, 215–222.

Alidaee, B., Kochenberger, G., Amini, M., 2001. Greedy solutions of selection and ordering problems. European Journal of Operational Research 134, 203–215.

Alsuwaiyel, M. H., 1999. Algorithms Design and Techniques and Analysis. World Scientific Publishing.

Aluru, S., 2006. Handbook of Computational Molecular Biology. Chapman & Hall/CRC.

Apollonio, N., Franciosa, P. G., 2007. A characterization of partial directed line graphs. Discrete Mathematics 307, 2598–2614.

Baker, M. J., 2003. The Marketing Book, 5th Edition. Butterworth-Heinemann.

Barnhart, C., Johnson, E., Savelsbergh, M., 1998. Branch-and-price: column generation for solving huge integer programs. Operations Research 46, 316–329.

Bartnicki, T., Grytczuk, J. A., Kierstead, H. A., 2008. The game of arboricity. Discrete Mathematics 308, 1388–1393.

Bench-Capon, T. J. M., 2002. Value-based argumentation frameworks. In Proceedings of Non Monotonic Reasoning 2002, 444–453.

Bertsimas, D., Tsitsiklis, J. N., 1997. Introduction to Linear Optimization. Athena Scientific.

Bhaskar, T., Sundararajan, R., Krishnan, P. G., 2009. A fuzzy mathematical programming approach for cross-sell optimization in retail banking. Journal of the Operational Research Society 60, 717–727.

Bigras, L., Gamache, M., Savard, G., 2008. Time-indexed formulations and the total weighted tardiness problem. INFORMS Journal on Computing 20, 133–142.

Bilginturk, Z., Oguz, C., Salman, S., 28-31 August 2007. Order acceptance and scheduling decisions in make-to-order systems. In: Baptiste, P., Kendall, G., Munier-Kordon, A., Sourd, F. (Eds.), Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA). Paris, France, pp. 80–87.

Blow, L., Browning, M., Crawford, I., 2008. Revealed preference analysis of characteristic models. Review of Economic Studies 75, 371–389.

Brandeau, M. L., Sainfort, F., Pierskalla, W. P., 2004. Operations Research and Health Care: A Handbook of Methods and Applications. Kluwer Academic Publishers.

Brassington, F., Pettitt, S., 2003. Principles of Marketing, 3rd Edition. Financial Times/Prentice Hall.

Broersma, H. J., Fomin, F. V., Kratochvil, J., Woeginger, G. J., 2006. Planar graph coloring avoiding monochromatic subgraphs: Trees and paths make it difficult. Algorithmica 4, 343–361.

Brown, D., Matzkin, R., 1996. Testable restrictions on the equilibrium manifold. Econometrica 64, 1249–1262.

Brown, D., Shannon, C., 2000. Uniqueness, stability and comparative statics in rationalizable walrasian markets. Econometrica 68, 1529–1540.

Browning, M., Chiappori, P., 1998. Efficient intra-household allocations: a general characterization and empirical tests. Econometrica 68, 1241–1278.

Bruner, R. C., Eades, K. M., Harris, R. S., Higgins, R. C., 1998. Best practices in estimating the cost of capital: survey and synthesis. Financial Practice and Education 8, 13–28.

Caprara, A., Kellerer, H., Pferschy, U., Pisinger, D., 2000. Approximation algorithms for knapsack problems with cardinality constraints. European Journal of Operational Research 123, 333–345.

Carvajal, A., Ray, I., Snyder, S., 2004. Equilibrium behavior in markets and games: testable restrictions and identification. Journal of Mathematical Economics 40, 1–40.

Chan, A. H. S., Ao, S.-I., 2008. Advances in Industrial Engineering and Operations Research. Springer.

Chang, G. J., Chen, C., Chen, Y., 2004. Vertex and tree arboricities of graphs. Journal of Combinatorial Optimization 8, 295–306.

Chen, Z., 2000. Efficient algorithm for acyclic colorings of graphs. Theoretical Computer Science 230, 75–95.

Cherchye, L., Crawford, I., De Rock, B., Vermeulen, F., 2009a. The revealed preference approach to demand. Chapter 9 in *Quantifying Consumer Preferences: Estimating Demand Systems*, Daniel Slottje (ed.), Contributions to Economic Analysis, Emerald Press.

Cherchye, L., De Rock, B., Sabbe, J., Vermeulen, F., 2008a. Nonparametric tests of collectively rational consumption behavior: an integer programming procedure. Journal of Econometrics 147, 258–265.

Cherchye, L., De Rock, B., Vermeulen, F., 2007. The collective model of household consumption: a nonparametric characterization. Econometrica 75, 553–574.

Cherchye, L., De Rock, B., Vermeulen, F., 2008b. Analyzing cost efficient production behavior under economies of scope: A nonparametric methodology. Operations Research 56, 204–221.

Cherchye, L., De Rock, B., Vermeulen, F., 2009b. An Afriat theorem for the collective model of household consumption. Journal of Economic Theory forthcoming.

Chiappori, P., 1988. Rational household labor supply. Econometrica 56, 63–89.

Chiappori, P., 1992. Collective labor supply and welfare. Journal of Political Economy 100, 437–467.

Chiappori, P., Ekeland, I., 2006. The micro economics of group behavior: general characterization. Journal of Economic Theory 130, 1–26.

Chiappori, P., Ekeland, I., 2009. The micro economics of efficient group behavior: identification. Econometrica 77, 763–800.

Christensen, M., 2007. Integrability of demand accounting for unobservable heterogeneity: a test on panel data. IFS Working paper W14/07, University of Manchester.

Chung-Piaw, T., Vohra, R., 2003. Afriat's theorem and negative cycles. mimeo, Northwestern University.

Chuzhoy, J., Ostrovsky, R., Rabani, Y., 2006. Approximation algorithms for the job interval selection problem and related scheduling problems. Mathematics of Operations Research 31 (4), 730–738.

Chvátal, V., 1983. Linear Programming. W.H. Freeman.

Cohen, M., 2004. Exploiting response models - optimizing cross-sell and up-sell opportunities in banking. Information Systems 29, 327–341.

Collins, W., 1987. Collins Cobuild English Language Dictionary. William Collins Sons & Co Ltd.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., 2001. Introduction to Algorithms, Second Edition, 2nd Edition. MIT Press and McGraw-Hill.

Cornuejols, G., Tutuncu, R., 2007. Optimization Methods in Finance. Cambridge University Press.

De, P., Ghosh, J. B., Wells, C. E., 1991. On the minimization of the weighted number of tardy jobs with random processing times and deadline. Computers & Operations Research 18 (5), 457–463.

De Reyck, B., Degraeve, Z., 2003. Broadcast scheduling for mobile advertising. Operations Research 51, 509–517.

Deb, R., 2008a. Acyclic partitioning problem is NP-complete for $k = 2$. Private communication, Yale University, United States.

Deb, R., 2008b. An efficient nonparametric test of the collective household model. Working paper, Yale University, United States.

Deo, N., 1974. Graph Theory with Application to Engineering and Computer Science. Prentice-Hall.

Diewert, E., 1973. Afriat and revealed preference theory. Review of Economic Studies 40, 419–425.

Dobell, A. R., 1965. A comment on A. Y. C. Koo's an empirical test of revealed preference theory. Econometrica 33, 451–455.

Donni, O., 2008. Household behavior and family economics. The Encyclopedia of Life Support Systems Contribution 6.154.9.

Downes, J., Goodman, J. E., 1995. Dictionary of Finance and Investment Terms, 4th Edition. Barron's.

Dwyer, F. R., 1997. Customer lifetime valuation to support marketing decision making. Journal of Direct Marketing 11, 6–13.

Emmons, H., 1969. One machine sequencing to minimize certain functions of job tardiness. Operations Research 17, 701–715.

Engels, D. W., Karger, D. R., Kolliopoulos, S. G., Sengupta, S., Uma, R. N., Wein, J., 2003. Techniques for scheduling with rejection. Journal of Algorithms 49, 175–191.

Epstein, L., Nogab, J., Woeginger, G. J., 2002. On-line scheduling of unit time jobs with rejection: minimizing the total completion time. Operations Research Letters 30, 415–420.

Farin, G., 2006. Class A Bézier curves. Computer Aided Geometric Design 23, 573–581.

Garey, M. R., Johnson, D. S., 1979. Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco.

Ghosh, J. B., 1997. Job selection in a heavily loaded shop. Computers & Operations Research 24 (2), 141–145.

Goddard, W., 1991. Acyclic colorings of planar graphs. Discrete Mathematics 91, 91–94.

Goldberg, A. V., Kennedy, R., 1995. An efficient cost scaling algorithm for the assignment problem. Mathematical Programming 71, 153–177.

Golumbic, M. C., 2004. Algorithmic Graph Theory and Perfect Graphs, 2nd Edition. ELSEVIER B.V.

Gross, L. J., Yellen, J., 2004. Handbook of Graph Theory. CRC Press.

Guerrero, H. H., Kern, G. M., 1998. How to more effectively accept and refuse orders. Production and Inventory Management Journal 4, 59–62.

Gupta, S. K., Kyparisis, J., Ip, C., 1992. Project selection and sequencing to maximize net present value of the total return. Management Science 38, 751–752.

Hellinckx, E., 2004. Customer relationship management: De optimalisatie van de planning van campagnes. Master's thesis, KULeuven (in Dutch).

Herbots, J., Herroelen, W., Leus, R., 2007. Dynamic order acceptance and capacity planning on a single bottleneck resource. Naval Research Logistics 54 (8), 874–889.

Hogg, T., 1985. Refining the phase transition in combinatorial search. Artificial Intelligence 81, 127–154.

Ivănescu, V. C., Fransoo, J. C., Bertrand, J. W., 2006. A hybrid policy for order acceptance in batch process industries. OR Spectrum 28, 199–222.

Kellerer, H., Pferschy, U., Pisinger, D., 2004. Knapsack Problems. Springer.

Keskinocak, P., Tayur, S., 2004. Due date management policies. In: Simchi-Levi, D., Wu, S. D., Shen, Z. J. (Eds.), Handbook of Quantitative Supply Chain Analysis: Modeling in the E-Business Era. Kluwer, Ch. 12, pp. 485–547.

Khanna, S., Kumaran, K., 1998. On wireless spectrum estimation and generalized graph coloring. Proc. of INFOCOM'98, 1273–1283.

Kleywegt, A. J., Papastavrou, J. D., 2001. The dynamic and stochastic knapsack problem with random sized items. Operations Research 49 (1), 26–41.

Knott, A., Hayes, A., Neslin, S. A., 2002. Next-product-to-buy models for cross-selling applications. Journal of Interactive Marketing 16, 59–75.

Kotler, P., Armstrong, G., 2006. Principles of Marketing, 12th Edition. Pearson Prentice Hall.

Kumar, V., Ramani, G., Bohling, T., 2004. Customer lifetime value approaches and best practice applications. Journal of Interactive Marketing 18, 60–72.

Laguna, M., Kelly, J. P., Gonzlez-Verlarde, J. L., Glover, F., 1991. Tabu search for the multilevel generalized assignment problem. European Journal of Operational Research 82, 176–189.

Lawler, E. L., 1977. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathemathics 1, 331–342.

Lenstra, J. K., Rinnooy Kan, A. H., Brucker, P., 1977. Complexity of machine scheduling problems. Annals of Discrete Mathemathics 1, 343–362.

Lewis, H. F., Slotnick, S. A., 2002. Multi-period job selection: planning work loads to maximize profit. Computers & Operations Research 29, 1081–1098.

Li, S., Sun, B., Wilcox, R. T., 2005. Cross-selling sequentially ordered products: An application to consumer banking services. Journal of Marketing Research XLII, 233–239.

Lim, A., Wang, F., Xu, Z., 2004. On the selection and assignment with minimum quantity commitments. Lecture Notes in Computer Science 3106, 102–111.

Lim, A., Wang, F., Xu, Z., 2006. A transportation problem with minimum quantity commitment. Transportation Science 40, 117–129.

Lim, A., Xu, Z., 2006. The bottleneck problem with minimum quantity commitment. Naval Research Logistics 53, 91–100.

Lu, L., Zhang, L., Yuan, J., 2008. The unbounded parallel batch machine scheduling with release dates and rejection to minimize makespan. Theoretical Computer Science 396, 283–289.

Lund, C., Yannakakis, M., 1993. The approximation of maximum subgraph problems. Lecture Notes in Computer Science 700, 40–51.

Lundberg, S., Pollak, R., 2007. Family decision-making. The New Palgrave, Dictionary of Economics, 2nd Edition forthcoming.

Luo, Z., Tang, L., Zhang, W., 2007. Using branch-and-price algorithm to solve raw materials logistics planning problem in iron and steel industry. 2007 International Conference on Management Science and Engineering, 529–536.

Martello, S., Toth, P., 1990. Knapsack Problems: Algorithms and Computer Implementation. Jown Wiley and Sons.

Mazer, A., 2007. Electric Power Planning for Regulated and Deregulated Markets. John Wiley & Sons.

Meng-Gérard, J., Chrétienne, P., Baptiste, P., Sourd, F., 2009. On maximizing the profit of a satellite launcher: selecting and scheduling tasks with time windows and setups. Discrete Applied Mathematics 157, 3656–3664.

Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L., 1999. Determining computational complexity from characteristic 'phase transitions'. Nature 400, 133–137.

Nemhauser, G. L., Wolsey, L. A., 1988. Integer and Combinatorial Optimization. Wiley.

Pan, Y., Shi, L., 2007. On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. Mathematical Programming, Serie A 110, 543–559.

Pochet, Y., Wolsey, L. A., 2006. Production Planning by Mixed Integer Programming. Springer.

Potts, C. N., Van Wassenhove, L. N., 1985. A branch and bound algorithm for the total weighted tardiness problem. Operations Research 33 (2), 363–377.

Prinzie, A., Van Den Poel, D., 2006. Investigating purchasing-sequence patterns for financial services using Markov, MTD and MTDg models. European Journal of Operational Research 170, 710–734.

Prinzie, A., Van Den Poel, D., 2007. Predicting home-appliance acquisition sequences: Markov/Markov for discrimination and survival analysis for modeling sequential information in NPTB models. Decision Support Systems 44, 28–45.

Raspaud, A., Wang, W., 2008. On the vertex-arboricity of planar graphs. European Journal of Combinatorics 29, 1064–1075.

Reinartz, W., Thomas, J. S., Kumar, V., 2005. Balancing acquisition and retention resources to maximize customer profitability. Journal of Marketing 69, 63–79.

Rinnooy Kan, A. H. G., Lageweg, B. J., Lenstra, J. K., 1975. Minimizing total cost in one-machine scheduling. Operations Research 23, 908–927.

Rom, W. O., Slotnick, S. A., 2009. Order acceptance using genetic algorithms. Computers & Operations Research 36 (6), 1758–1767.

Roos, C., Terlaky, T., Vial, J. P., 2006. Interior Point Methods for Linear Optimization. Springer.

Roundy, R., Chen, D., Chen, P., Cakanyildirim, M., Freimer, M. B., Melkonian, V., 2005. Capacity-driven acceptance of customer orders for a multi-stage batch manufacturing system: models and algorithms. IIE Transactions 37, 1093–1105.

Roychoudhury, A., Sur-Kolay, S., 1995. Efficient algorithms for vertex arboricity of planar graphs. Lecture Notes in Computer Science 1026, 37–51.

Ryals, L., 2005. Making customer relationship management work: The measurement and the profitable management of customer relationships. Journal of Marketing 69, 252–261.

Savelsbergh, M., 1997. A branch-and-price algorithm for the generalized assignment problem. Operations Research 45, 831–841.

Sengupta, S., 2003. Algorithms and approximation schemes for minimum lateness/tardiness scheduling with rejection. Lecture Notes in Computer Science 2748, 79–90.

Slotnick, S. A., Morton, T. E., 1996. Selecting jobs for a heavily loaded shop with lateness penalties. Computers & Operations Research 23, 131–140.

Slotnick, S. A., Morton, T. E., 2007. Order acceptance with weighted tardiness. Computers & Operations Research 34 (10), 3029–3042.

Spieksma, F. C. R., 1999. On the approximability of an interval scheduling problem. Journal of Scheduling 2, 215–227.

Starr, R., 1969. Quasi-equilibria in markets with non-convex preferences. Econometrica 37, 25–38.

Starret, D., 1972. Fundamental nonconvexities in the theory of externalities. Journal of Economic Theory 4, 180–199.

Talla Nobibon, F., Herbots, J., Leus, R., 2009. Order acceptance and scheduling in a single-machine environment: exact and heuristic algorithms. Working paper KBI-0903, Department of Quantitative Methods and Information Management, Faculty of Business and Economics, KULeuven (Leuven, Belgium).

Talla Nobibon, F., Leus, R., Spieksma, F. C. R., 2008. Models for the optimization of promotion campaigns: exact and heuristic algorithms. Research report KBI_0814, Department of Quantitative Methods and Information Management, Faculty of Business and Economics, KULeuven (Leuven, Belgium).

Tanaka, S., Fujikuma, S., Araki, M., 2009. An exact algorithm for single-machine scheduling without machine idle time. Journal of Scheduling 12, 575–593.

Tarjan, R. E., 1972. Depth-first search and linear graph algorithms. SIAM Journal on Computing 2, 146–160.

Thomassen, C., 2008. 2-list-coloring planar graphs without monochromatic triangles. Journal of Combinatorial theory 98, 1337–1348.

Thorsteinsson, E. S., 2001. Branch-and-check: a hybrid framework integrating mixed integer programming and constraint logic programming. Lecture Notes in Computer Science 2239, 16–30.

Van Den Akker, J. M., Hoogeveen, J. A., Van Kempen, J. W., 2006. Parallel machine scheduling through column generation: Minimax objectives (extended abstract). ESA 2006, Lecture Notes in Computer Science 4168, 648–659.

Van Den Akker, J. M., Hurkens, C. A. J., Savelsbergh, M. W. P., 2000. Time-indexed formulations for single-machine scheduling problems: column generation. INFORMS Journal on Computing 12, 111–124.

Van Praag, N., 2010. Optimization of promotion campaigns using tabu search. Master's thesis, KULeuven.

Vance, P. H., Atamturk, A., Barnhart, C., Gelman, E., Johnson, E. L., Krishna, A., Nemhauser, G. L., Rebello, R., 1997. A heuristic branch-and-price approach for the airline crew pairing problem. Tech. Rep. LEC-97-06, Georgia Institute of Technology.

Varian, H., 1982. The nonparametric approach to demand analysis. Econometrica 50, 945–974.

Varian, H., 2006. Revealed preference. in M. Szenberg, L. Ramrattan and A.A. Gottesman (eds.), Samuelsonian economics and the 21st century, Oxford University Press.

Wolsey, L. A., 1998. Integer Programming. John Wiley & Sons.

Wright, S. J., 1997. Primal-Dual Interior Point Methods. SIAM.

Wu, Y., Yuan, J., Zhao, Y., 1996. Partition a graph into two induced forests. Journal of Mathematical Study 1, 1–6.

Yang, B., Geunes, J., 2007. A single resource scheduling problem with job-selection flexibility, tardiness costs and controllable processing times. Computers & Industrial Engineering 53, 420–432.

Yugma, C., 2005. Dynamic management of a portfolio of orders. 4OR: A Quarterly Journal of Operations Research 3, 167–170.

Zhu, J., 2009. Optimization of Power System Operation. John Wiley & Sons.

Zijm, W. H. M., 2000. Towards intelligent manufacturing planning and control systems. OR Spektrum 22, 313–345.

Doctoral dissertations from the Faculty of Business and Economics

A list of doctoral dissertations from the Faculty of Business and Economics can be found at the following website:
`http://www.econ.kuleuven.be/phd/doclijst.htm`.