Linear and quadratic reformulations of nonlinear optimization problems in binary variables

Août 2018

Thèse présentée
en vue de l'obtention du grade
de Docteur en Sciences Économiques et de Gestion par

Elisabeth RODRÍGUEZ-HECK

Promoteur :	
Yves CRAMA	Université de Liège
Membres du comité :	
Quentin LOUVEAUX	Université de Liège
Michaël SCHYNS	Université de Liège
Membres du jury :	
Endre BOROS	Rutgers University
Christoph BUCHHEIM	Technische Universität Dortmund

Abstract

The problem of optimizing a multilinear polynomial on binary variables without additional constraints arises in a variety of applications. We are interested in resolution methods based on reformulations of this nonlinear problem into a linear or a quadratic one, an approach that attempts to draw benefit from the existing literature on integer linear and quadratic programming.

In the context of linear reformulations we consider the standard linearization, a well-known reformulation method that consists in introducing an auxiliary variable to represent each higher-degree monomial, where the association of auxiliary variables to monomials is achieved using linear constraints. A first contribution of this thesis is a characterization of cases for which the continuous relaxation of the standard linearization provides integer solutions. Additionally, we define a class of valid inequalities called 2-links modeling interactions between pairs of intersecting monomials. For functions with exactly two nonlinear monomials, we prove that the 2-links together with the standard linearization inequalities completely describe the convex hull of their feasible integer points. Moreover, the 2-link inequalities strengthen the standard linearization and greatly improve resolution times for a class of specially structured problems.

A broader definition is considered for quadratic reformulations: a quadratization is a quadratic function depending on the original variables and on a set of auxiliary binary variables, such that, when minimized only over the auxiliary variables, the original multilinear polynomial is recovered. We study several properties of quadratizations such as requiring a small number of auxiliary variables. A notable contribution is the definition of a quadratization for monomials with a positive coefficient using only a logarithmic number of variables, which improves previously published bounds by orders of magnitude. This result is especially interesting because every multilinear polynomial can be quadratized by reformulating its monomials separately. We also consider quadratizations of a different nature defined by splitting each monomial into two subterms to be associated with an auxiliary variable each. Defining such quadratizations using the smallest possible number of variables is an NP-hard problem, for which we define heuristic algorithms that identify sets of variables appearing frequently as a subterm of the original monomials, and substituting each set by the same auxiliary variable.

Finally, this thesis presents a comparison of the resolution times of several linear and quadratic reformulations, using a commercial solver, over different classes of instances. Experimental results show that reformulations exploiting the structure of the original nonlinear problems, for example by better modeling interactions between monomials, have best resolution times for many instances.

Acknowledgements

Five years have past since I first arrived in Liège. This has been a journey of scientific and personal growth, which would have not been possible without the support of many people.

First and foremost, I am most grateful to my thesis advisor, Professor Yves Crama. I would like to thank you for your support and guidance, for your trust and for always pushing me a step further. You are a true inspiration as a scientist and professor, but also as a person. I particularly admire your honesty and integrity, and your capability of getting the best out of people. I could not imagine a better advisor than I had.

I would also like to thank the members of my thesis committee, Professors Quentin Louveaux and Michaël Schyns, for having followed my work during the past few years, and for always having asked excellent questions that pushed my research in interesting directions. I would like to acknowledge the members of my jury and co-authors. Professor Endre Boros, I would like to thank you for your trust, encouragements and availability, I have learned immensely at each of our conversations. Professor Christoph Buchheim, thank you for your openness and availability, for your valuable suggestions and for welcoming me in Dortmund and in Köln.

I also gratefully acknowledge the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office (grant P7/36) and the University of Liège for funding this thesis.

These five years would have never been the same without my amazing colleagues at office 334. Stefano, you have been a great friend since day one, always supportive and ready to help, and your sense of humour can turn every day into the funniest. Célia, thank you for your honesty, support and for the energy you put in every activity, which is truly contagious. Stéphanie, thank you for many uplifting conversations on research, teaching, and on life in general. Anne-Sophie, thank you for being always ready to help, for your spontaneity and transparency. I could not forget Virginie, thank you for always being so welcoming and for showing me that also work can be lots of fun. Alessandro, thank you for all the great moments shared, not only at work but also uncountable dinners, lunches and excursions during the weekends. Martine and Christine, I have enjoyed very much our get togethers and the conferences in Scotland and Canada. Thank you Hande, for having been an essential pillar, especially during my first year. Thank you to all QuantOM'ians: Anaïs, Ashwin, Bart, Cédric, Christian, Guillaume, Julien, Maud, Quentin, Reza, Sabine, Thierry, Thu, Valentin, Véronique and Yasemin for many lunches, coffee breaks, dinners, and conferences shared, and to all collegues in HEC Liège for their friendliness.

Marta, gracias por estar a mi lado incondicionalmente, contar contigo ha sido muy importante estos años aquí. Gracias también a Nicole por haberme acogido desde el primer día, y por tantas cenas y comidas juntas. Merci aussi Charlotte et Itxi, pour avoir toujours été à mes côtés et pour les bons moments partagés.

I am also very grateful to all my friends from Barcelona and Grenoble, for their continuous support and encouragements despite the many kilometers of distance.

Por supuesto quiero dar las gracias a mis padres y hermano, por su apoyo incondicional y por su paciencia. Gracias Mama, por tu fortaleza, por estar siempre a mi lado, por escucharme y comprenderme aunque muchas veces no sea tarea fácil. Gracias Papa, por estar a mi lado y animarme siempre con todos mis proyectos, especialmente, cuando dudaba en embarcarme en esta tesis. Carlos, gracias por escucharme, entenderme y estar a mi lado pase lo que pase. Grisel·la, gracias por tu apoyo y cariño en todo momento.

Mi paso por Lieja también me ha acercado a una parte de mi familia, Teresa, Agustín y Guillermo, el reencuentro con vosotros ha sido uno de esos regalos inesperados que da la vida.

Ich möchte mich auch herzlich an meine Familie in Deutschland bedanken, für die vielen schönen Momente zusammen und für eure Unterstützung. Spezial möchte ich mich an meinen Opa bedanken, für deine Interesse in meiner Doktorarbeit und weil du auch ein Vorbild als Forscher bist.

Gracias a mi nueva familia en Córdoba y en El Vacar. En especial, gracias a Mari Carmen, quien desde el primer día me hizo sentir como una más en la familia. Gracias también a José, Rocío y Alejandro, por vuestro cariño y apoyo, especialmente en los últimos días de escritura de esta tesis.

Y finalmente, gracias a Ángel. Sólo por el hecho de conocerte ha valido la pena vivir en Liège. A tu lado la vida ha ganado en color y alegría, en confianza y comprensión. Cada día a tu lado disfruto, aprendo y crezco como persona. No imagino mejor compañero de viaje que tú.

To my family.

Contents

Chapter 1

Introduction

A pseudo-Boolean function is a mapping $f : \{0, 1\}^n \to \mathbb{R}$ that assigns a real value to each tuple of n binary variables $(x_1, \ldots, x_n)$. Pseudo-Boolean functions have been extensively used and studied during the last century and especially in the last 50 years, due to its theoretical interest and also because they model problems in a wide range of areas such as reliability theory, computer science, statistics, economics, finance, operations research, management science, discrete mathematics, or computer vision (see [24, 40] for a list of applications and references).

In most of these applications f has to be optimized, therefore we are interested in the problem

$$\min_{x \in \{0,1\}^n} f(x), \tag{1.1}$$

which is NP-hard even when f is quadratic.

It is well-know that every pseudo-Boolean function f can be represented uniquely as a multilinear polynomial [66, 67].

$$f(x_1, \ldots, x_n) = \sum_{S \in 2^{[n]}} a_S \prod_{i \in S} x_i, \tag{1.2}$$

where $[n] = \{1, \ldots, n\}$, $2^{[n]}$ is the set of subsets $S \subseteq [n]$ and $a_S \in \mathbb{R}$ is a coefficient assigned uniquely to each subset S. Notice that, given a pseudo-Boolean function f, finding its unique multilinear expression depends on the size of the input f which can be exponential in n. Let $\deg(f)$ denote the degree of the polynomial representation of f.

Unless otherwise specified, we consider that a pseudo-Boolean function is given by its unique multilinear expression. We will refer to a problem of type (1.1), where f is given by its unique multilinear expression (1.2) as a *pseudo-Boolean*, *nonlinear*, *polynomial* or *multilinear* optimization problem.

Pseudo-Boolean functions are closely related to set functions, which are of great interest in discrete mathematics and operations research. Set functions assign a real value to each subset S of a given set of elements $\{1, \ldots, n\}$. Given a set function, we can obtain a pseudo-Boolean function by replacing every set S by its characteristic vector. This link between pseudo-Boolean functions and set functions enables the use of pseudo-Boolean optimization techniques to solve classical problems that are formulated in terms of set functions.

General nonlinear optimization problems currently attract great interest from the mathematical programming and optimization community, and several techniques have been proposed for their resolution, such as enumerative methods, algebraic methods, linear reformulations and quadratic reformulations, which are then solved using a linear or quadratic solver, respectively. It is not clear whether one of the previous techniques is generally better than the others. In fact, the performance of the different approaches seems to depend on the underlying structure of the problem, among other factors. Unconstrained nonlinear optimization problems in binary variables are surveyed in [24, 40], and the constrained case is reviewed in [68]. Nonlinear integer programming has been reviewed in [71, 85], and there exist many recent surveys on mixed-integer nonlinear programming (MINLP) [16, 32, 42, 61, 84] and global optimization [26, 53], which are more general problems.

This thesis focuses on methodological aspects of the resolution of unconstrained nonlinear optimization problems in binary variables. More precisely, we examine resolution techniques based on linear and quadratic reformulations of nonlinear problems by introducing artificial variables. We also refer to linear reformulations as *linearizations* and to quadratic reformulations as *quadratizations*. This approach attempts to draw benefit from the extensive literature on integer linear and quadratic programming and has been recently considered by several authors [4, 5, 30, 31, 46, 47, 49, 51]. The remainder of this chapter consists of Section 1.1, which presents some relevant applications of the pseudo-Boolean optimization framework, and Section 1.2, which describes the structure of this thesis and summarizes its main contributions.

1.1 Selected applications

In order to highlight the applicability of the framework of pseudo-Boolean optimization problems, we describe in detail a set of applications that are especially interesting in the context of this thesis. We start with four classical applications in applied mathematics and theoretical computer science: the *maximum satisfiability* problem, the *maximum cut* problem and the *simple plant location* problem. Then, we briefly describe an energy minimization framework that models several problems in the area of *computer vision*. These applications come from an engineering context, and interestingly much recent progress in methodological questions related to reformulations has been made by the computer vision community. Quadratic reformulations have proven to be extremely useful in the resolution of problems such as image restoration. Moreover, Chapter 4 and Part III present the results of computational experiments, many instances of which are inspired from the image restoration problem in computer vision. Finally we describe an application in *supply chain design* which is especially interesting because the formulation of the problem is *not* directly given as a multilinear polynomial, which represents an additional challenge to the use of our reformulation techniques.

1.1.1 Classical applications

Maximum satisfiability The *maximum satisfiability* problem or MAX-SAT is currently one of the most relevant and well-studied problems in fields as theoretical computer science, artificial intelligence or applied mathematics. We present here a natural pseudo-Boolean formulation for the MAX-SAT problem as given by Boros and Hammer [24].

Let us first formally define the MAX-SAT problem. Given a set of binary variables $(x_1, \ldots, x_n) \in \{0,1\}^n$, the set of *literals* $\mathbf{L} = \{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n,\}$ consists of the set of variables x_i and their complements $\bar{x}_i = 1 - x_i$ for all $i = 1, \ldots, n$. A *clause* $C \subseteq \mathbf{L}$ is a subset of *literals*. A binary assignment $x \in \{0,1\}^n$ *satisfies* a clause C if the Boolean disjunction of the literals in C takes value 1 for this assignment, that is, if

$$\vee_{u \in C} u = 1.$$

Consider a family $\mathcal{C} \subseteq 2^{\mathbf{L}}$ of clauses. The MAX-SAT problem consists in finding a binary assignment x satisfying a maximum number of clauses of $\mathcal{C}$.

The pseudo-Boolean formulation of the MAX-SAT problem is a direct consequence of the observation that a binary assignment $x \in \{0,1\}^n$ satisfies a clause C if, and only if $\prod_{u \in C} \bar{u} = 0$. Then, the maximum satisfiability problem is equivalent with the maximization problem

$$\max_{x \in \{0,1\}^n} \left(\sum_{C \in \mathcal{C}} \left(1 - \prod_{u \in C} \bar{u} \right) \right).$$

A particular case is the MAX-k-SAT problem where each clause $C \in \mathcal{C}$ contains exactly k literals. MAX-SAT is an NP-complete problem and so is MAX-k-SAT even when $k = 2$.

Maximum cut The *maximum cut* problem or MAX-CUT is a well-known problem in graph theory. We describe it here in detail due to its importance in quadratic binary optimization. We use the notations in [24].

Let $G = (V, E)$ be a graph and consider a subset of vertices $S \subseteq V$. A *cut* $\delta(S) \subseteq E$ is defined as the edges with exactly one endpoint in S. The MAX-CUT problem consists in finding a subset S that maximizes the cardinality $|\delta(S)|$.

Let us represent a subset S by its characteristic vector $x \in \{0,1\}^V$ where $x_i = 1$ if $i \in S$ and $x_i = 0$ otherwise. Then, a pseudo-Boolean formulation for the MAX-CUT problem is

$$\max_{S \subseteq V} |\delta(S)| = \max_{x \in \{0,1\}^V} \left(\sum_{(i,j) \in E} (x_i \bar{x}_j + \bar{x}_i x_j) \right).$$

The *cut polytope* is the convex hull of all incidence vectors of edge sets representing cuts of G. De Simone showed in [43] that the cut polytope is equivalent to the *boolean quadric polytope*

$$BQP = \{(x, y) \in \{0,1\}^{|V|+|E|} \mid y_{ij} = x_i x_j\},$$

providing the equivalence between the MAX-CUT problem and quadratic optimization in binary variables.

The boolean quadric polytope was first introduced by Padberg, who initiated a systematic study of its characteristics and facets [94]. The facial structure of the cut polytope was studied before by Barahona, Grötschel and Mahjoub [8, 9], but the identification with the corresponding results on the boolean quadric polytope was only possible after the establishment of their equivalence in [43]. Boros, Crama and Hammer derived bounds on the problem of unconstrained quadratic optimization in binary variables as well as a correspondence between facets of the boolean quadric polytope and the cone of nonnegative quadratic pseudo-Boolean functions [17], and defined new facets for the cut polytope based on this correspondence in [23]. Boros and Hammer considered the problems of *maximum-2-satisfiability* and *weighted signed graph balancing* and established classes of facets for these problems, that were derived from equivalent results in quadratic binary optimization [22]. Finally, a further lower bound as well as a compact formulation for the MAX-CUT problem on a particular class of graphs was given in [18].

The simple plant location problem The simple plant location problem (SPLP) is a classical problem in operations research that was first formulated as a pseudo-Boolean optimization problem by Hammer [64]. We present in detail a formulation by Dearing, Hammer and Simeone [44].

Given a set of customers and a set of potential plant locations, the objective of the SPLP is to optimally locate a set of plants in order to minimize costs for operating the plants and for serving the customers. More precisely, we are given the following parameters: let $P = \{i \mid i = 1, \dots, p\}$ be the set of potential plant locations and $f_i,\ i \in P$ the fixed cost for opening a plant at location i. Let $D = \{j \mid j = 1, \dots, d\}$ be the set of customers. Finally, let $c_{ij},\ i \in P, j \in D$ be the unit transportation cost from plant i to customer j.

The demand of each customer is considered to be one unit and it is assumed that the capacity of each opened plant is sufficient to meet the demand of all customers, meaning that we are in the context of what is also called *uncapacitated plant location*.

A standard integer programming formulation of the SPLP is given by defining variables $y_i = 1$ if a plant is open at location i and 0 otherwise, and $x_{ij} = 1$ if customer j is served by plant i and 0 otherwise.

$$\min \sum_{i=1}^{p} \sum_{j=1}^{d} c_{ij} x_{ij} + \sum_{i=1}^{p} f_i y_i \tag{1.3}$$

$$\text{s.t.} \sum_{i=1}^{p} x_{ij} = 1, \qquad j \in D \tag{1.4}$$

$$x_{ij} \le y_i, \qquad j \in D, i \in P \tag{1.5}$$

$$x_{ij} \in \{0,1\}, y_i \in \{0,1\}, \qquad j \in D, i \in P \tag{1.6}$$

An equivalent formulation of the SPLP in the form (1.1) is presented in [44] by using the well-known property that in some optimal solution, each customer j will receive its entire unit of demand from *one* open plant, namely, the plant with minimum transportation cost to customer j. For each $j \in D$, let $j(\cdot)$ be the index of the permutation of the location indices $i \in P$, such that the transportation costs from $j(i)$ to j satisfy

$$c_{j(1)j} \leq c_{j(2)j} \leq \cdots \leq c_{j(p)j},$$

and consider the complemented y-variables $\bar{y}_j = 1 - y_j$.

Since each customer will receive its demand from a single plant, a cost $c_{j(1)j}$ is incurred if, and only if, $y_{j(1)} = 1$, a cost $c_{j(2)j}$ is incurred if, and only if, $\bar{y}_{j(1)} y_{j(2)} = 1$, and similarly a cost $c_{j(k)j}$ is incurred if, and only if, $\bar{y}_{j(1)}\bar{y}_{j(2)} \ldots \bar{y}_{j(k-1)} y_{j(k)} = 1$. The x-variables are then substituted in the objective function (1.3) by noticing that if for a given j, we have $j(r) = i$, then $x_{ij} = \prod_{k=1}^{r-1} \bar{y}_{j(k)} y_{j(r)}$. Constraints (1.5) become redundant with this substitution. When substituting x_{ij} in (1.4), the following equation is satisfied for each j

$$\sum_{i=1}^{p} \prod_{k=1}^{r-1} \bar{y}_{j(k)} y_{j(r)} + \prod_{i=1}^{p} \bar{y}_{j(i)} = 1, \quad j \in D,$$

where term $\prod_{k=1}^{p} \bar{y}_{j(k)}$ represents an infeasible solution, meaning that none of the plants is assigned to customer j. The infeasibility of such a solution is enforced by adding terms $M_j \prod_{i=1}^{p} \bar{y}_{j(i)}$, with M_j being a large cost to the objective function. The resulting pseudo-Boolean formulation of the SPLP is

$$\begin{aligned} \min \sum_{j=1}^{d} \left(\sum_{i=1}^{p} \left(c_{j(i)j} \prod_{k=1}^{i-1} \bar{y}_{j(k)} y_{j(i)} \right) + M_j \prod_{i=1}^{p} \bar{y}_{j(i)} \right) + \sum_{i=1}^{p} f_i y_i \\ \text{s. t. } y_i \in \{0, 1\} \end{aligned} \tag{1.7}$$

In [44], the pseudo-Boolean formulation of the SPLP is then transformed to a set covering and to a weighted vertex packing problem on a graph, which has later proved to be useful in other contexts, like for the resolution of the p-median problem [60].

1.1.2 Computer vision

A relevant area of application of pseudo-Boolean optimization methods in engineering is *computer vision*. These applications are of particular interest in the context of this thesis because much progress on quadratic reformulations of pseudo-Boolean optimization problems has been made in recent years by the computer vision community. Many computer vision problems such as *image restoration*, *stereo* or *segmentation* can be formulated as energy minimization problems, which are closely related to the pseudo-Boolean optimization framework.

We use here the definitions and notations provided by Ishikawa [74] to formally define the energy minimization model. Let P be a set of *pixels* and L a set of *labels*. Let $\mathcal{C} \subseteq 2^P$ be a set of subsets of pixels. Elements $C \in \mathcal{C}$ are called *cliques*. Let L^P be the set of *labelings*, that is the set of functions $X : P \to L$ assigning a label $X_p \in L$ to a pixel $p \in P$. An *energy function*

$$E : L^P \to \mathbb{R}$$

associates a real value to each labeling $X \in L^P$. It is assumed that $E(X)$ is decomposable into a sum

$$E(X) = \sum_{C \in \mathcal{C}} f_C(X_C), \tag{1.8}$$

where $f_C(X_C)$ is a function that depends only on the labels X_C associated to a given clique C by the labeling X.

A labeling X can also be seen as a random variable taking values in the set of labels L at each pixel. The terminology Markov Random Fields (MRF) is frequently used to designate a system consisting of random variables X together with an energy function $E(X)$.

The *order* of an energy function is the number of pixels in the largest clique minus one, for example, a first-order MRF contains cliques of size one and size two. The terms concerning cliques of size one usually represent the disagreement between the label that is assigned to a pixel p and the observed data. The terms concerning cliques of sizes larger than one are used to model interactions between different pixels.

For example, in the image restoration problem, a blurred image is given as an input, and the objective is to produce a sharp and well-defined image as an output. The variables X represent the color that each pixel p should have in the output image. The terms concerning cliques of size one model the fact that the label of the output pixel should not be too different from the label of the same pixel in the blurred input image. The terms concerning larger cliques model the fact that pixels that are within the same neighbourhood or clique should have similar labels, as it happens in "natural" images. Using more complex terms to model clique interactions, one can also model the fact that inside objects, the pixels of a clique should have similar labels while at the edges of objects, labels should be different.

A key idea behind decomposition (1.8) is that for this type of problems, what matters are local interactions between pixels, and interactions between distant pixels, for example at two opposite corners of an image, do not play any role.

If we assume that the set of labels is binary, i.e., $\mathcal{L} = \{0, 1\}$, an energy function (1.8) can be represented by a multilinear polynomial (1.2), where the terms concerning a clique C will have degree $|C|$. Early models in computer vision only considered functions of degree two, because efficient optimization algorithms were only known for the quadratic case. The development of such efficient algorithms was one of the main motivations to define quadratic reformulation techniques within the computer vision community. Using quadratic reformulations, one can consider higher-degree polynomials which potentially provide more accurate models, and then apply efficient minimization algorithms for the quadratic case.

In the context of this thesis we will consider the following reformulations originating from the computer vision field, which aim at defining a quadratization for each term of a multilinear expression (1.2) separately: Kolmogorov and Zabih [80], and later Freedman and Drineas [56] defined a quadratization for non-linear monomials with a negative coefficient, Ishikawa [74] provided a quadratization for non-linear monomials with a positive coefficient. These quadratizations are reviewed in detail in Chapters 5 and 6. We are also interested in quadratizations of a different nature, which consider several monomials of a multilinear expression at the same time, such as those defined by Fix, Gruber, Boros and Zabih [52], Rosenberg [96] and Anthony, Boros, Crama and Gruber [5], which are reviewed in detail in Chapters 5 and 7.

As for the quadratic binary optimization algorithms, let us first notice that in formulation (1.8), the set of labels is not necessarily binary. However, assuming that the set of labels is binary is not a big loss of generality, because several of the optimization algorithms consider at each iteration a binary decision related to the labels that has to be optimized, such as swapping a label or not, or extending a label to neighboring pixels or not. Such algorithms are often called *move-making* algorithms, and two of the most representative algorithms in this category are *α-expansion moves* and *α-β-swap moves* [27]. Later, α-expansion moves were generalized to *fusion moves* [82, 83]. In each iteration of these algorithms, the optimal decision of whether expanding (or swapping) or not, is reached by solving a *minimum cut* problem in a certain graph, and these procedures are therefore often called *graph cut* algorithms in the computer vision community. When the quadratic function is submodular, that is, when it contains no quadratic terms with a positive coefficient, the quadratic binary optimization problem can be solved efficiently with graph cuts, a result that was first established by Hammer [63]. When the function is not submodular, the performance of the resolution of each iteration in this type of algorithms has been greatly improved using *roof duality* bounds and *persistency* techniques, which allow to determine the integer values that a subset of variables will take at the optimum. These concepts were first introduced by Hammer, Hansen and Simeone [65], and the technique is often referred to as *QPBO (Quadratic Pseudo-Boolean Optimization)* in the computer vision community. An extensive set of computational experiments using persistencies, roof duality and an improved roof duality bound has been presented by Boros, Hammer, Sun and Tavares [25]. The idea of persistencies is examined in detail in Section 8.3 of Chapter 8.

Other quadratic minimization algorithms that are not related to graph-cuts are also used in computer vision, such as *belief propagation* or *tree-reweighted message passing*. General purpose minimization algorithms such as *simulated annealing* or *local search* have also been applied to computer vision problems, but they turn out to be very slow in practice. We refer the reader to two recent and surveys for an extensive computational comparison of the performance of various computer vision techniques on a large set of instances [78, 102].

1.1.3 Joint supply chain network design and inventory management

The problem of *joint supply chain network design and inventory management* considers a supply chain consisting of one or more suppliers, a set of retailers and a set of potential locations for distribution centers (DCs).

There are four objectives to optimize simultaneously: the location of the DCs, the assignment of retailers to DCs, the replenishment policy of the DCs and their levels of safety stocks. The first two objectives relate to the supply chain network design problem and the last two relate to the inventory management problem. Moreover, the last two objectives are modeled using the economic order quantity (EOQ) which leads to square roots in the objective function.

The model that is presented here is an integer nonlinear program in binary variables that was first defined by Shen, Coullard and Daskin [101]. Their resolution technique consists in reformulating the problem as a set covering problem, relying on the assumption that each customer has identical variance-to-mean ratio, and then solve the reformulation using a branch-and-price algorithm. Later on, You and Grossmann [105] presented a heuristic method based on Lagrangean relaxation and Lagrangean decomposition methods that does not require to assume identical variance-to-mean ratios.

The precise formulation considers a set of retailers $i \in I$ with independent demands, following a normal distribution $\mathcal{N}(\mu_i, \sigma_i^2)$. The potential DCs are denoted by $j \in J$. The replenishment lead time L of the DCs is considered to be the same for all suppliers. It is assumed that no stock is held by the retailers and that the DCs use a (Q, r) policy for inventory management. The following sets of variables are defined: $x_j = 1$ if a DC is opened at location j, and 0 otherwise and $y_{ij} = 1$ if retailer i is assigned to DC j, and 0 otherwise.

The objective function consists of four terms. The first term is a classical representation of the facility location cost

$$\sum_{j \in J} f_j x_j,$$

where f_j is the fixed installation cost for DC j.

The second term represents the transportation costs from DCs to retailers and is defined as

$$\sum_{j \in J} \sum_{i \in I} \chi d_{ij} \mu_i y_{ij},$$

where χ are the days per year and d_{ij} is unit transportation cost from j to i.

The third term takes into account the ordering and shipping costs from the supplier to the DCs and the working inventory holding costs and it is obtained by an estimation of a deterministic EOQ model associated to a (Q, r) policy. It reads as follows

$$\beta \sum_{j \in J} a_j \sum_{i \in I} \chi \mu_i y_{ij} + \sum_{j \in J} \sqrt{2\theta h (F_j + \beta g_j) \sum_{i \in I} \chi \mu_i y_{ij}},$$

where β is a weight factor associated with the transportation cost, θ is a weight factor associated with the inventory cost, a_j is the unit transportation cost from the supplier to the DC at candidate site j, g_j is the fixed transportation cost from the supplier to the DC at candidate site j, h is the inventory holding cost per unit of product per year and F_j is the fixed cost of placing an order at distribution center j.

Finally, the fourth term represents the safety stock costs, and is defined as

$$\theta h z_\alpha \sum_{j \in J} \sqrt{\sum_{i \in I} L \sigma_i^2 y_{ij}},$$

where $z_\alpha \sqrt{L}\sigma$ is the optimal safety stock level to guarantee a service level α when demands are normally distributed for one retailer (z_α is a standard normal deviate such that $Pr(z \leq z_\alpha) = \alpha$).

The final model can be summarized as follows,

$$\min \sum_{j\in J}\left(f_j x_j + \sum_{i\in I}\hat{d}_{ij}y_{ij} + K_j\sqrt{\sum_{i\in I}\mu_i y_{ij}} + q\sqrt{\sum_{i\in I}\hat{\sigma}_i^2 y_{ij}}\right) \tag{1.9}$$

$$\text{s.t.} \sum_{j\in J} y_{ij} = 1, \qquad \forall i \in I \tag{1.10}$$

$$y_{ij} \leq x_j, \qquad \forall i \in I, \forall j \in J \tag{1.11}$$

$$x_j \in \{0,1\}, \qquad \forall j \in J \tag{1.12}$$

$$y_{ij} \in \{0,1\}, \qquad \forall i \in I, \forall j \in J \tag{1.13}$$

where we have used the simplified expressions in [105] using

$$\begin{aligned}\hat{d}_{ij} &= \beta\chi\mu_i(d_{ij}+a_j)\\ K_j &= \sqrt{2\theta h\chi(F_j+\beta g_j)}\\ q &= \theta h z_\alpha\\ \hat{\sigma}_i^2 &= L\sigma_i^2.\end{aligned}$$

Notice that problem (1.9) does not exactly fit in our framework because of two reasons: first, the objective function is not given as a multilinear expression and second, the formulation contains additional constraints (1.10)–(1.11). Even if this thesis focuses on unconstrained polynomial optimization, several of the considered methods could potentially be adapted to accommodate constraints. However, the fact of having a non-multilinear objective function in such a precise application is especially challenging, and motivates the exploration of new linearization and quadratization methods for pseudo-Boolean functions that are not given as a multilinear polynomial. As mentioned at the beginning of this chapter, computing the multilinear expression of (1.9) might be very costly, because one might have to enumerate all values of x_j and y_{ij}, which would be equivalent to solving the problem by pure enumeration, and this is clearly impracticable for reasonably large instances.

A possibility that we started to explore is to define an approximate multilinear expression for (1.9) by using a Taylor expansion of the square root terms. Even though this method would not give an exact solution of the problem because it only approximates the objective function, it has the advantage of generating a multilinear polynomial. The Taylor expansion enables to improve the quality of the approximation by increasing the degree, possibly at the expense of higher computing costs, which is a classical quality vs. time trade-off arising in many optimization problems. However, we encountered memory problems by trying to implement a Taylor expansion of reasonable degree even for relatively small instances.

1.1.4 Other applications

We have only highlighted here a few selected applications, but there are many more. A detailed exposition of applications in *graph theory*, *data mining*, *classification*, *learning theory*, *artificial intelligence* or *game theory* among others can be found in [24, 40].

Finally, we also point the reader to the recent surveys [37, 81] on *constraint satisfaction problems* and *valued constraint satisfaction problems*, the latter being rather a generalization than an application of the pseudo-Boolean optimization framework.

1.2 Main contributions and structure of the thesis

This thesis is concerned with a study of methodological aspects of the resolution of unconstrained nonlinear optimization problems in binary variables using linear and quadratic reformulation techniques, which we also call *linearizations* and *quadratizations*, respectively.

This thesis is organized in three parts. Part I is concerned with linear reformulations, Part II is concerned with quadratic reformulations, and Part III presents the results of an extensive set of computational experiments comparing the performance of several linear and quadratic reformulation methods defined in Part I and Part II. The purpose of this introductory section is to highlight the main contributions of this thesis and to describe the structure of each part.

Part I: Linear reformulations

Our contributions regarding linear reformulations are based on a well-known linearization technique, called the *standard linearization*, which defines a linear reformulation of a multilinear problem by introducing a set of auxiliary variables, each one representing a higher-degree monomial. The equivalence between variables and monomials is imposed by introducing constraints, which we call *standard linearization inequalities*. Even though the standard linearization was first proposed in the late fifties [54, 55, 58, 59, 104, 106], it has attracted much recent interest from the optimization community [30, 31, 46, 47, 48, 49, 51].

Chapter 2 presents a detailed literature review on the standard linearization and positions the results of this thesis. We are mainly concerned with polyhedral descriptions of the polytope defined by the standard linearization inequalities, which we call the *standard linearization polytope*.

Chapter 3 presents a characterization of multilinear functions for which the standard linearization polytope has integer vertices, which implies that the corresponding nonlinear binary optimization problems can be solved by continuous linear programming. This characterization is given in terms of the balancedness of the matrix defined by the standard linearization inequalities, and also in terms of the acyclicity of the hypergraph associated with a multilinear polynomial. These results are derived from a more general case, that takes into account the signs of the coefficients of the multilinear polynomial. Moreover, the conditions of the characterization can be checked in polynomial time. The results presented in this chapter can be found in [28].

Chapter 4 defines a class of valid inequalities, called *2-link inequalities* or *2-links*, that strengthen the standard linearization formulation. The main result of this chapter states that for multilinear polynomials with exactly two higher-degree monomials, the polytope obtained by adding the 2-links to the standard linearization polytope has integer vertices. Moreover, it is also shown that the 2-link inequalities are facet-defining for multilinear polynomials with a special structure that we call *nested*, where each higher degree monomial is completely contained in another monomial. This chapter also presents the results of some computational experiments, comparing the resolution times of the standard linearization in a branch & cut framework using different sets of cuts. The use of the 2-link inequalities leads to promising results, especially for a class of instances with a special structure, inspired from the image restoration problem in computer vision. The results presented in this chapter have been published in [41].

Part II: Quadratic reformulations

The second part of this thesis focuses on quadratic reformulations or *quadratizations*. We consider a multilinear polynomial $f(x)$ depending on a set of variables $x \in \{0,1\}^n$. The starting point of this second part of the thesis is a definition by Anthony, Boros, Crama and Gruber [5], stating that a *quadratization* is a quadratic function $g(x,y)$ depending on the original variables x and on a set of new variables $y \in \{0,1\}^m$, such that minimizing g over (x,y) is equivalent to minimizing f over x, without requiring additional constraints. This definition is very general, and not all quadratizations will be equally "good". A desirable property of a quadratization is to use a small number of auxiliary y variables. However, this is not the only criterion one could think of, since having good optimization properties or exploiting the structure of the polynomial (e.g. by modeling interactions between monomials) also are reasonable criteria to define interesting quadratizations. Similar approaches were already defined in the seventies [96], and several recent publications consider quadratic reformulations [4, 5, 30, 31]. There are several factors explaining the recent interest in quadratic reformulations, such as a tendency towards a better understanding and resolution of nonlinear problems, but also the recent improvements in the field of quadratic binary optimization both in theorical aspects and in software developement, which justify the use of quadratic reformulations.

Chapter 5 presents a detailed literature review on quadratizations and positions our contributions. We already point out here the fact that, since every multilinear polynomial consists of monomials with a positive coefficient (*positive monomials*) and monomials with a negative coefficient (*negative monomials*), it is clear that every pseudo-Boolean function can be reformulated by defining a quadratization for negative monomials, and a quadratization for positive monomials. Such procedures are called *termwise* quadratizations. The case of negative monomials is well-solved and there exists a quadratization using a single y variable which additionally has good optimization properties. Interestingly, the case of positive monomials is much less understood and the current best published bound on the number of auxiliary variables to use is linear in n, the dimension of the original variable space.

Chapter 6 presents quadratizations for positive monomials using a logarithmic number of auxiliary variables in n, and a proof that a logarithmic bound is best possible. These results are derived from lower and upper bounds on the number of auxiliary variables required to define a quadratization for more general classes of pseudo-Boolean functions, such as symmetric functions or functions with many zeros. All results in this chapter can be found in [19].

Chapter 7 focuses on quadratizations of a different nature, based on the idea of *pairwise covers* [5], which consists in splitting each monomial in two different parts, and then substituting each part by an auxiliary variable. In this way, each term is "covered" by two auxiliary variables, allowing the definition of a quadratic reformulation associated to the pairwise cover. Determining the smallest number of variables required to define quadratizations based on pairwise covers is an NP-hard problem, even for polynomials of degree three. In this chapter, we define heuristic algorithms to generate quadratizations based on pairwise covers that try to minimize the number of required auxiliary variables. The main idea behind the presented heuristics is that whenever several monomials of the input function contain a certain set of variables as a subterm, then this set can be associated to the same auxiliary variable in all monomials containing it. Moreover, the number of required auxiliary variables can be heuristically minimized by trying to determine sets of variables that occur often as a subterm of the monomials in the original multilinear function. A property that makes this type of quadratizations particularly interesting is that they capture some of the structural properties of the original polynomial problem.

Part III: Computational experiments

The third part of this thesis is concerned with computational experiments aimed at testing the performance of several linearization and quadratization methods defined in Part I and Part II.

An extensive set of instances is considered. Some of these instances are highly unstructured random polynomials while others have very specific structures, arising in application fields like computer vision or statistical mechanics.

Chapter 8 is the main chapter of Part III, containing a description of the instances, of the methods tested and of the technical specifications of the linear and quadratic solver that is used to solve the reformulations. A set of preliminary experiments with the aim of improving the performance of quadratic resolutions using the *persistency* property is also presented. Finally, the results of the final set of experiments comparing the computational performance of the considered linearization and quadratization methods are presented and analyzed. For most instances we compare the standard linearization, the standard linearization with 2-link inequalities, several termwise quadratization methods and several quadratizations based on pairwise covers.

Chapter 9 is an extension of the previous set of experiments, which introduces a heuristic method to choose the subset of "most interesting" 2-link inequalities. The standard linearization with 2-link inequalities presents in general a very good performance but has the drawback of requiring a high memory consumption and model creation time for large instances. The heuristic algorithm presented in this chapter tries to overcome these drawbacks.

Chapter 10 is the final chapter of this thesis, and presents some concluding remarks and several possible future research directions, concerning both theoretical and computational aspects.

Part I

Linear reformulations

Chapter 2

Introduction to linearizations

The first part of this thesis presents several results related to linear reformulation methods of the nonlinear optimization problem

$$\min_{x\in\{0,1\}^n} f(x), \tag{1.1}$$

defined on binary variables $x \in \{0,1\}^n$. An important advantage of linearization approaches is that they allow the resolution of nonlinear binary programs by using integer linear programming techniques, and can thus benefit from the great theoretical and computational advances in integer linear programming of the last decades.

As a reminder, let $[n] = \{1, \ldots, n\}$ and let $2^{[n]}$ be the set of subsets of indices in $[n]$. Let $a_S \in \mathbb{R}$ denote a value associated uniquely with every $S \in 2^{[n]}$. When $|S| = 1$, we write a_i instead of $a_{\{i\}}$ for simplicity. Let us introduce notation $\mathcal{S} \subseteq 2^{[n]}$ to define the set of subsets S such that $a_S \neq 0$ and $|S| \geq 2$. We assume that f is given by its unique multilinear expression

$$f(x_1, \ldots, x_n) = \sum_{S\in\mathcal{S}} a_S \prod_{i\in S} x_i + \sum_{i\in[n]} a_i x_i. \tag{2.1}$$

All contributions presented in this part of the thesis are based on the so-called *standard linearization*, a well-known linearization method that has been defined for functions given in the form (2.1). The standard linearization procedure consists in substituting each nonlinear monomial $\prod_{i\in S} x_i$ by a new variable y_S, and imposing $y_S = \prod_{i\in S} x_i$ as a constraint for all $S \in \mathcal{S}$.

We denote by X_{SL} the set of binary points satisfying these constraints, that is,

$$X_{SL} = \{(x,y) \in \{0,1\}^{n+|\mathcal{S}|} \mid y_S = \prod_{i\in S} x_i, \ \forall S \in \mathcal{S}\}, \tag{2.2}$$

and we denote its convex hull by P^*_{SL}:

$$P^*_{SL} = conv(X_{SL}). \tag{2.3}$$

Then, the problem of optimizing (2.1) is equivalent to the linear programming problem

$$\min_{(x,y)\in P^*_{SL}} L_f(x,y) = \sum_{S\in\mathcal{S}} a_S y_S + \sum_{i\in[n]} a_i x_i. \tag{2.4}$$

In order to obtain a 0–1 linear programming formulation of our problem, the polynomial equation $y_S = \prod_{i \in S} x_i$ can be expressed using the following constraints, to be called *standard linearization inequalities* in the sequel:

$$y_S \leq x_i, \qquad \forall i \in S \tag{2.5}$$
$$y_S \geq \sum_{i \in S} x_i - (|S| - 1), \tag{2.6}$$
$$y_S \geq 0 \tag{2.7}$$

More precisely, when x_i is binary for all $i \in S$, the feasible solutions of the constraints (2.5)–(2.7) are exactly the solutions of the polynomial equation $y_S = \prod_{i \in S} x_i$. (The integrality requirement does not need to be explicitly stated for y_S: when the original variables x_i are binary, then y_S automatically takes a binary value too.) So, if we define the *standard linearization polytope* associated with f as

$$P_{SL} = \{(x, y) \in [0, 1]^{n+|\mathcal{S}|} \mid (2.5), (2.6),\ \forall S \in \mathcal{S}\}, \tag{2.8}$$

then the inequalities defining P_{SL} provide a valid formulation of X_{SL} in the sense that X_{SL} is exactly the set of binary points in P_{SL}.

The standard linearization was proposed by several authors independently [54, 55, 104, 106], in a slightly different form from (2.5)–(2.7) and with integrality constraints on the variables y_S. The initial formulation was later improved by Glover and Woolsey, in a first contribution by adding fewer constraints and variables in the reformulation [58], and in a second contribution by introducing continuous auxiliary variables rather than integer ones [59].

As in [36], we say that a system of linear inequalities is a *perfect formulation* of a set X if the system exactly describes the convex hull of X.

It is a known fact that, when f contains a single nonlinear monomial, the standard linearization inequalities define a perfect formulation for the set of points X_{SL}. Formally,

Remark 1. *When $\mathcal{S}$ contains a single nonlinear monomial S, the inequalities (2.5)–(2.7) and the bound constraints $0 \leq x_i \leq 1$ for $i \in [n]$ provide a perfect formulation of X_{SL}. That is, $P^*_{SL} = P_{SL}$.*

This remark appears to be part of the folklore of the field of nonlinear binary optimization. It can be easily derived by direct arguments, and it also follows from related results, e.g., by McCormick [90] and by Al-Khayyal and Falk [1] for the quadratic case, by Crama [39] and by Ryoo and Sahinidis [99] for the general case of degree higher than two (see also [89]).

However, for the general case when f contains an arbitrary number of nonlinear monomials, finding a concise perfect formulation of P^*_{SL} is probably hopeless (unless P = NP). Moreover, P_{SL} provides a very weak relaxation of P^*_{SL} when f contains an arbitrary number of nonlinear monomials.

2.1 Literature review and contributions

In this first part of the thesis, we are mainly concerned with polyhedral descriptions of the set of integer points satisfying the standard linearization inequalities X_{SL}.

Chapter 3 provides a characterization of multilinear functions f for which $P_{SL} = P^*_{SL}$, that is, cases for which inequalities (2.5)–(2.7) and the bound constraints $0 \leq x_i \leq 1$, for $i \in [n]$ provide a perfect formulation of X_{SL}. This characterization is given in terms of the acyclicity of the hypergraph associated with the monomial set of f, and in terms of the balancedness of the matrix defining constraints (2.5)–(2.7). This result is derived from a more general result that takes into account the signs of the coefficients of the monomials of f. Moreover, the conditions of the characterization can be checked in polynomial time. These results extend those provided for the quadratic case by Hansen and Simeone [69], and for the polynomial case considering coefficient signs by Crama [38, 39]. Some statements of our results have been provided independently and using a proof technique (relying, in particular, on decomposition arguments) by Del Pia and Khajavirad [45, 47]. The details of how our results relate to earlier results are described in Section 3.1.1. The contents of Chapter 3 can also be found in [28].

Chapter 4 goes one step further, by defining a class of valid inequalities that strengthen the standard linearization. These inequalities are called *2-link inequalities* or *2-links* because they model interactions between pairs of monomials with a non-empty intersection. We prove here that the 2-link inequalities, together with the standard linearization inequalities provide a perfect formulation of the set of points X_{SL} associated to a multilinear function f with exactly two higher-degree monomials. It is also shown that the 2-link inequalities are facet-defining for functions where the monomials are *nested*, meaning that every monomial is completely contained in another monomial. The case of nested monomials has also been considered by Fischer et al. [51], in a more general context optimizing a nonlinear objective function with nested monomials over variables belonging to a matroid polytope. Finally, the results of an extensive set of computational experiments show that the 2-link inequalities can be very helpful when solving a class of instances inspired from the image restoration problem in computer vision by linearization. The contents of Chapter 4 have been published in [41].

Several recent papers investigate questions related to the standard linearization technique. We have briefly mentioned recent work on complete descriptions of the standard linearization polytope by Del Pia and Khajavirad [45, 47]. The same authors also defined a class of valid inequalities generalizing the 2-link inequalities presented in Chapter 4 [47]. Del Pia and Khajavirad also recently provided decomposability results for the set of points X_{SL} extending previous results for the quadratic case [48], and a framework for deriving facet-defining inequalities based on lifting operations, among other techniques, for the standard linearization polytope [46]. Buchheim and Rinaldi [30] proved that one can completely describe the convex hull P^*_{SL} of the set of points X_{SL}, by providing a complete polyhedral description of a quadratic reformulation of the original polynomial, and derived as a consequence that the separation problem of the nonlinear problem reduces to the separation problem of the cut polytope. Other recent contributions less related to the results in this thesis can be found in [29, 49, 91].

Chapter 3

Berge-acyclic multilinear 0–1 optimization problems

This chapter provides a characterization of multilinear functions for which the standard linearization inequalities define a perfect formulation of the set of integer points in X_{SL} defined in Chapter 2, implying that the corresponding nonlinear binary optimization problems can be solved by continuous linear programming. [1]

The results of this chapter are given for the maximization of a multilinear polynomial (the minimization case is analogous). As a reminder, the multilinear problem is

$$\max_{x \in \{0,1\}^n} f(x) = \sum_{i \in [n]} c_i x_i + \sum_{S \in \mathcal{S}} a_S \prod_{j \in S} x_j, \tag{3.1}$$

where $[n] = \{1, \ldots, n\}$, $\mathcal{S} \subseteq \{S \in 2^{[n]} \mid |S| \geq 2\}$, $c \in \mathbb{R}^n$, and $a \in \mathbb{R}^{\mathcal{S}}$.

Observe that the monomials in the multilinear expression (3.1) are uniquely associated to a hypergraph $H = ([n], \mathcal{S})$, where $[n]$ is the set of vertices of the hypergraph, and $\mathcal{S}$ is the set of subsets of variables. Similarly, the standard linearization polytope P_{SL} defined in Chapter 2 is uniquely defined for a given hypergraph H, independently of the coefficients of the monomials. Since the results of this chapter rely on this association of the standard linearization inequalities with the hypergraph defining the monomial set, in this chapter we denote the standard linearization polytope associated to a hypergraph by P^H_{SL}. For the sake of simplicity, in this chapter we denote $[n]$ by V. We refer to elements $i \in V$ as vertices, and to elements $S \in \mathcal{S}$ as edges.

We aim at characterizing the instances for which the polytope P^H_{SL} has only integer vertices. One of our main results characterizes integrality of P^H_{SL} in terms of the properties of the hypergraph $H = (V, \mathcal{S})$. More precisely, we show that P^H_{SL} has only integer vertices if and only if H is Berge-acyclic. Furthermore, we show that Berge-acyclicity of H is equivalent to the constraint matrix of P^H_{SL} being balanced; for this, we rely on a fundamental result by Conforti and Cornuéjols [35]. We derive these characterizations from a more general result,

[1]The contents of this chapter have been obtained together with Christoph Buchheim and Yves Crama, and have been submitted for publication [28].

taking into account the signs of the coefficients of the nonlinear terms of f. As a byproduct, we deduce the existence of an efficient algorithm for testing whether a given sign pattern of the nonlinear terms guarantees integrality or not. These results generalize those obtained by Padberg [94] and by Hansen and Simeone [69] for the quadratic case, and by Crama [39] for the general case. In fact, a main objective of this chapter is *to state more explicitly, to unify and to clarify* the relation between these earlier characterizations in terms of associated polyhedra, matrices or hypergraphs, and *to provide simpler, self-standing proofs* for their equivalence. More details will be given in Section 3.1.

3.1 Definitions and statement of the main results

This section formally introduces relevant definitions and states the main results of this chapter. Let $H = (V, \mathcal{S})$ be a finite hypergraph. We assume throughout that $\mathcal{S}$ does not contain singletons. We denote by $\mathcal{P}(H)$ the set of multilinear expressions f of type (3.1), obtained by defining the coefficients $c \in \mathbb{R}^V$ and $a \in \mathbb{R}^{\mathcal{S}}$. For the sake of simplicity, we assume $a_S \neq 0$ for all $S \in \mathcal{S}$.

Let us rewrite the standard linearization inequalities as

$$-y_S + x_i \geq 0 \qquad \forall i \in S, \forall S \in \mathcal{S}, \tag{3.2}$$

$$y_S - \sum_{i \in S} x_i \geq 1 - |S| \qquad \forall S \in \mathcal{S}, \tag{3.3}$$

We denote by M^H_{SL} the matrix of coefficients of the left-hand-sides of (3.2) and (3.3).

Definition 1. *Given a multilinear expression $f \in \mathcal{P}(H)$, its* linearized form *is defined as*

$$L_f(x, y) = \sum_{i \in V} c_i x_i + \sum_{S \in \mathcal{S}} a_S y_S,$$

where the coefficients c_i and a_S are exactly the same as in f.

As already mentioned in Chapter 2, all integer points $(x, y) \in P^H_{SL}$ are such that $y_S = \prod_{j \in S} x_j$ for all $S \in \mathcal{S}$. As a consequence, maximizing the linearized form L_f over the integer points of P^H_{SL} is equivalent to maximizing $f(x)$ over $\{0, 1\}^n$.

Notice that when maximizing a linearized form L_f over P^H_{SL}, constraints (3.2) are not binding when the coefficient a_S is negative, and constraints (3.3) are not binding when a_S is positive. This observation motivates the following definitions.

Definition 2. *A* signed hypergraph *$H(\sigma)$ is a hypergraph $H = (V, \mathcal{S})$ together with a* sign pattern *$\sigma \in \{-1, 1\}^{\mathcal{S}}$. The set of* positive edges *of $H(\sigma)$ is $\mathcal{S}^+ = \{S \in \mathcal{S} : \sigma_S = 1\}$ and the set of* negative edges *is $\mathcal{S}^- = \{S \in \mathcal{S} : \sigma_S = -1\}$.*

Clearly, every element $f \in \mathcal{P}(H)$ (or the associated linearized form L_f) defines a sign pattern by setting $\sigma_S := \text{sgn}(a_S)$ and hence, induces a signed hypergraph $H(\sigma)$. Sign patterns can thus be considered as equivalence classes of $\mathcal{P}(H)$ with respect to the signs of the coefficients.

Definition 3. *The* signed standard linearization polytope $P_{SL}^{H(\sigma)}$ *associated with a signed hypergraph $H(\sigma)$ is the polytope defined by the constraints*

$$-y_S + x_i \geq 0 \qquad \forall i \in S, \forall S \in \mathcal{S}^+, \tag{3.4}$$

$$y_S - \sum_{i \in S} x_i \geq 1 - |S| \qquad \forall S \in \mathcal{S}^-, \tag{3.5}$$

and by the bounds $0 \leq x_i \leq 1$ for all $i \in V$, and $0 \leq y_S \leq 1$ for all $S \in \mathcal{S}$. We denote the matrix of coefficients of the left-hand-sides of (3.4) *and* (3.5) *by $M_{SL}^{H(\sigma)}$.*

The notion of cycles in hypergraphs will frequently be used in this chapter. Several definitions of cycles in hypergraphs have been given in the literature, such as Berge-cycles [11], α-cycles [10], special cycles [3] (also called weak β-cycles [50]), or γ-cycles [50]. In our context, we use the following definitions.

Definition 4. *Given a hypergraph $H = (V, \mathcal{S})$, a* Berge-cycle *C of length p is a sequence $(i_1, S_1, i_2, S_2, \ldots, i_p, S_p, i_{p+1} = i_1)$ where*

1. *$p \geq 2$,*
2. *$i_k, i_{k+1} \in S_k$ for $k = 1, \ldots p-1$ and $i_p, i_1 \in S_p$,*
3. *$i_1, \ldots, i_p$ are pairwise distinct elements of V, and*
4. *$S_1, \ldots, S_p$ are pairwise distinct elements of $\mathcal{S}$.*

If, additionally, $S_k \cap \{i_1, \ldots, i_p\} = 2$ for all $k = 1, \ldots, p$, we call C a special cycle *of H.*

(In the definition of special cycles, it is usually assumed that $p \geq 3$. We only impose here that $p \geq 2$.) Given a Berge-cycle C, we denote by $V_C = \{i_1, \ldots, i_p\}$ its set of vertices and by $\mathcal{S}_C = \{S_1, \ldots, S_p\}$ its set of edges.

Lemma 1. *Any hypergraph containing a Berge-cycle also contains a special cycle.*

Proof. Let $C = (i_1, S_1, i_2, S_2, \ldots, i_p, S_p, i_1)$ be a Berge-cycle of minimal length in $H = (V, \mathcal{S})$. We claim that C is special. Assume on contrary that C is not special and that, without loss of generality, $|S_1 \cap V_C| > 2$. Choose $i_k \in (S_1 \cap V_C) \setminus \{i_1, i_2\}$. Then $(i_1, S_1, i_k, S_{k+1}, \ldots, S_p, x_1)$ is a Berge-cycle strictly shorter than C, contradicting the choice of C. □

We next extend the classical definition of negative cycles in signed graphs [70].

Definition 5. *A* negative (special) cycle *in a signed hypergraph $H(\sigma)$ is a (special) cycle containing an odd number of negative edges.*

Finally, we recall the definition of balanced matrices [11, 35].

Definition 6. *A matrix M with all entries in $\{-1, 0, 1\}$ is* balanced *if in every submatrix of M having exactly two non-zeros per row and two non-zeros per column, the sum of the entries is a multiple of four.*

The main result of this chapter is the following characterization. The proof will be given in Section 3.2.

Theorem 1. *Given a hypergraph $H = (V, \mathcal{S})$ and a sign pattern $\sigma \in \{-1, 1\}^{\mathcal{S}}$, the following statements are equivalent:*

(a) For all $f \in \mathcal{P}(H)$ with sign pattern σ, every vertex of P^H_{SL} maximizing L_f is integer.

(b) $M^{H(\sigma)}_{SL}$ is balanced.

(c) $H(\sigma)$ has no negative special cycle.

(d) $P^{H(\sigma)}_{SL}$ is an integer polytope.

Remark 2. *As shown in [34], it can be checked efficiently whether a matrix with entries in $\{-1, 0, 1\}$ is balanced. This implies that all conditions in Theorem 1 can be checked efficiently.*

Remark 3. *Theorem 1 characterizes the sign patterns σ that guarantee integer optimal solutions for* all *$f \in \mathcal{P}(H)$ with sign pattern σ. This does not exclude, however, that some functions with a sign pattern different from σ also lead to integer optimal solutions. This depends on the (relative) values of the coefficients a_S and c_i. As an example, consider the quadratic function*

$$f(x_1, x_2, x_3) = x_1x_2 + x_1x_3 - x_2x_3 - Mx_1$$

with $M \in \mathbb{R}$. The corresponding hypergraph $H(\sigma)$ contains a negative special cycle, but for large enough values of M the optimal vertices of P^H_{SL} with respect to L_f are all integer.

Corollary 1. *Given a hypergraph H, the following statements are equivalent:*

(a) P^H_{SL} is an integer polytope.

(b) M^H_{SL} is balanced.

(c) H is Berge-acyclic.

Proof. We claim that each of the statements (a), (b) and (c) is equivalent to the respective statement (a), (b) and (c) in Theorem 1 holding for all $\sigma \in \{-1, 1\}^{\mathcal{S}}$. The result then follows from Theorem 1. For (a), this equivalence is obvious.

For (b), it is clear that if M^H_{SL} is balanced, then $M^{H(\sigma)}_{SL}$ is balanced for all σ, since every submatrix of $M^{H(\sigma)}_{SL}$ is also a submatrix of M^H_{SL}. Assume now that M^H_{SL} is not balanced. Thus, it contains a submatrix B with exactly two nonzeros per row and per column such that the sum of its entries is congruent with 2 modulo 4. As long as there are two rows in B corresponding to constraints of type (3.2) and (3.3) for the same edge $S \in \mathcal{S}$, the matrix B must contain a submatrix of the form

$$B' = \begin{pmatrix} +1 & -1 \\ -1 & +1 \end{pmatrix},$$

where all other entries of B in the rows and columns of B' are zero by the choice of B. The sum of elements of B' is zero, thus we can recursively delete the corresponding rows and columns from B and finally assume that, for each $S \in \mathcal{S}$, B only contains rows associated with constraints of type (3.2), or only contains a row associated with a constraint of type (3.3). We may then define a sign pattern σ by setting $\sigma_S = 1$ if B contains a row corresponding to (3.2) for S, $\sigma_S = -1$ if B contains the row corresponding to (3.3) for S, and $\sigma_S \in \{-1, 1\}$ arbitrarily otherwise. Then, by construction, the matrix B is a submatrix of $M_{SL}^{H(\sigma)}$, showing that $M_{SL}^{H(\sigma)}$ is not balanced.

For (c), if H is Berge-acyclic, then clearly $H(\sigma)$ has no negative special cycle, for any sign pattern σ. Conversely, assume that H contains a Berge-cycle C. By Lemma 1, we may assume that C is a special cycle. Define a sign pattern σ such that $\sigma_S = -1$ for exactly one edge of C and $\sigma_S = 1$ otherwise. Then C is a negative special cycle in $H(\sigma)$. □

3.1.1 Relation with earlier results

For ordinary graphs, i.e., for hypergraphs $H = (V, \mathcal{S})$ where $|S| = 2$ for all edges $S \in \mathcal{S}$ (corresponding to quadratic functions f), Padberg [94] proved that P_{SL}^{H} has integer vertices if and only if H is an acyclic graph. Corollary 1 generalizes Padberg's result to the case of higher-degree multilinear expressions.

Similarly, Theorem 1 extends results obtained by Hansen and Simeone [69] for the quadratic case (see also Michini [91]). The equivalence of conditions (b) and (d) in Theorem 1 for functions of arbitrary degree was first stated in Crama [38, 39]. However, the proof of this result was omitted from the published version [39]; in the technical report [38], the result was derived from a more general one whose proof was partially erroneous. Essentially, this earlier proof relied on the "observation" that a certain matrix $M(\psi)$ is balanced if and only if a related matrix $M^*(\psi)$ is balanced, a claim which is unfortunately wrong in the general setting of [38]. Although it would be possible to fix the proof in [38] for the particular case of multilinear polynomials – or *canonical expressions*, in the terminology of [38, 39] – it is probably more natural to derive it from strong results subsequently obtained by Conforti and Cornuéjols [33, 35], as we explain in Section 3.2. So, we find it useful to provide here, for the record, a complete correct proof of the equivalence of (b) and (d). Moreover, as compared with [38, 39], Theorem 1 clarifies the link between the integrality properties of P_{SL}^{H}, $P_{SL}^{H(\sigma)}$, and translates the results in terms of the acyclicity of certain hypergraphs.

Theorem 1 and Corollary 1 have been independently derived from Crama's earlier results by Del Pia and Khajavirad [45], and the same authors have used different proof techniques (relying, in particular, on decomposition arguments) to establish Corollary 1 in [47].

3.2 Proof of the main theorem

In this section, we present a proof of Theorem 1 in the form of four separate propositions. We first show that condition (d) implies condition (a) in Theorem 1. This is a direct consequence of the following:

Proposition 1. *Let $f \in \mathcal{P}(\mathcal{H})$ with a sign pattern $\sigma \in \{-1, 1\}^{\mathcal{S}}$. Then the two optimization problems*

$$\max_{(x,y)\in P^H_{SL}} L_f(x, y),$$

and

$$\max_{(x,y)\in P^{H(\sigma)}_{SL}} L_f(x, y) \tag{3.6}$$

have the same sets of optimal solutions.

Proof. Since $P^H_{SL} \subseteq P^{H(\sigma)}_{SL}$, it suffices to show that every optimal solution of problem (3.6) belongs to P^H_{SL}. So let (x^*, y^*) be such an optimal solution and consider any edge $S \in \mathcal{S}$.

Assume first that (x^*, y^*) violates (3.2) for some $j \in S$, so that $y^*_S > x^*_j$. By definition of $P^{H(\sigma)}_{SL}$, this is possible only if $\sigma_S = -1$, we thus have $a_S < 0$. We can now decrease y^*_S to x^*_j without leaving the polyhedron $P^{H(\sigma)}_{SL}$, since the resulting vector satisfies $y^*_S - \sum_{i\in S} x^*_i = -\sum_{i\in S\setminus\{j\}} x^*_i \geq -(|S|-1)$. As $a_S < 0$, this contradicts the assumption that (x^*, y^*) is an optimal solution of (3.6).

Now assume that (x^*, y^*) violates (3.3). Then $y^*_S < \sum_{i\in S} x^*_i - |S| + 1$, $\sigma_S = +1$, and $a_S > 0$. We can now increase y^*_S to $\sum_{i\in S} x^*_i - |S| + 1$ without leaving $P^{H(\sigma)}_{SL}$, as $\sum_{i\in S} x^*_i - |S| + 1 \leq x^*_j$ for all $j \in S$. We again obtain a contradiction to the optimality of (x^*, y^*). □

We next observe that (b) implies (d) in Theorem 1. For this, consider the *generalized set covering polytope* $Q(A)$ corresponding to a matrix A with entries in $\{-1, 0, +1\}$, defined as

$$Q(A) := \{x \colon Ax \geq 1 - n(A), 0 \leq x \leq 1\},$$

where $n(A)$ denotes the column vector whose i^{th} component $n_i(A)$ is the number of negative entries in row i of A. We recall a fundamental result on balanced matrices:

Theorem 2 (Conforti and Cornuéjols [33, 35])**.** *Let M be a matrix with entries in $\{-1, 0, +1\}$. Then M is balanced if and only if for each submatrix A of M, the generalized set covering polytope $Q(A)$ is integral.*

Since $P^{H(\sigma)}_{SL}$ is exactly the generalized set covering polytope $Q(M^{H(\sigma)}_{SL})$, we obtain

Proposition 2. *Given $\sigma \in \{-1, 1\}^{\mathcal{S}}$, if $M^{H(\sigma)}_{SL}$ is balanced, then $P^{H(\sigma)}_{SL}$ is integral.*

We next show that (a) implies (c) in Theorem 1. This is equivalent to showing

Proposition 3. *Let $\sigma \in \{-1, 1\}^{\mathcal{S}}$ and assume that $H(\sigma)$ has a negative special cycle C. Then there exists $f \in \mathcal{P}(H)$ with sign pattern σ such that some optimal vertex of P^H_{SL} with respect to L_f is not integer.*

Proof. Let $C = (i_1, S_1, i_2, S_2, \dots, i_p, S_p, i_{p+1})$ be a negative special cycle in $H(\sigma) = (V, \mathcal{S})$, with $i_{p+1} = i_1$, and consider the sets $\mathcal{S}_C^+ := \mathcal{S}_C \cap \mathcal{S}^+$ and $\mathcal{S}_C^- := \mathcal{S}_C \cap \mathcal{S}^-$. Moreover, consider the partition of the set V_C given by the following subsets:

$$\begin{aligned}
V^{++} &:= && \{i_k \in V_C \colon S_{i_k - 1} \in \mathcal{S}_C^+ \text{ and } S_{i_k} \in \mathcal{S}_C^+\},\\
V^{--} &:= && \{i_k \in V_C \colon S_{i_k - 1} \in \mathcal{S}_C^- \text{ and } S_{i_k} \in \mathcal{S}_C^-\},\\
V^{+-} &:= && \{i_k \in V_C \colon S_{i_k - 1} \in \mathcal{S}_C^+ \text{ and } S_{i_k} \in \mathcal{S}_C^-\},\\
V^{-+} &:= && \{i_k \in V_C \colon S_{i_k - 1} \in \mathcal{S}_C^- \text{ and } S_{i_k} \in \mathcal{S}_C^+\}.
\end{aligned}$$

Note that the following identity holds for all $x \in \mathbb{R}^V$:

$$\sum_{S_k \in \mathcal{S}_C^-} x_{i_k} - \sum_{S_k \in \mathcal{S}_C^+} x_{i_{k+1}} = \sum_{j \in V^{--}} x_j + \sum_{j \in V^{+-}} x_j - \sum_{j \in V^{+-}} x_j - \sum_{j \in V^{++}} x_j = \sum_{j \in V^{--}} x_j - \sum_{j \in V^{++}} x_j. \tag{3.7}$$

We will define an objective function $h = L_f$ corresponding to some $f \in \mathcal{P}(H)$ with sign pattern σ, as well as a fractional point $(\hat{x}, \hat{y}) \in P_{SL}^H$ such that $h(\hat{x}, \hat{y}) > h(x, y)$ for all integer points $(x, y) \in P_{SL}^H$. This will imply the result.

The function h with sign pattern σ is defined by

$$h(x, y) := \mu \left(\sum_{j \in V \setminus V_C} x_j \right) + \sum_{j \in V^{--}} x_j - \sum_{S \in \mathcal{S}_C^-} y_S - \sum_{j \in V^{++}} x_j + \sum_{S \in \mathcal{S}_C^+} y_S + \varepsilon \Big(\sum_{S \in \mathcal{S}^+} y_S - \sum_{S \in \mathcal{S}^-} y_S \Big),$$

where μ is chosen large enough so that all x_j with $j \in V \setminus V_C$ take value one when maximizing h over P_{SL}^H, and where ε is small and positive. Define $(\hat{x}, \hat{y})$ as follows:

- $\hat{x}_i = \frac{1}{2}$, for $x_i \in V_C$,
- $\hat{x}_i = 1$, for $x_i \in V \setminus V_C$,
- $\hat{y}_S = \frac{1}{2}$, for $S \in \mathcal{S}_C^+$,
- $\hat{y}_S = 0$, for $S \in \mathcal{S}_C^-$,
- $\hat{y}_S = \frac{1}{2}$ for $S \in \mathcal{S} \setminus \mathcal{S}_C$ containing at least one vertex in V_C, and
- $\hat{y}_S = 1$ for $S \in \mathcal{S} \setminus \mathcal{S}_C$ containing no vertex in V_C.

It can be verified that $(\hat{x}, \hat{y}) \in P_{SL}^H$ and that

$$h(\hat{x}, \hat{y}) = \mu |V \setminus V_C| + \tfrac{|V^{--}|}{2} - \tfrac{|V^{++}|}{2} + \tfrac{|\mathcal{S}_C^+|}{2} + \varepsilon\, t(\hat{y}) = \mu |V \setminus V_C| + \tfrac{|\mathcal{S}_C^-|}{2} + \varepsilon\, t(\hat{y}),$$

where $t(\hat{y})$ stands for the multiplier of ε, and where the second equality follows from identity (3.7).

It remains to prove that $h(\hat{x}, \hat{y}) > h(x, y)$ for all integer points $(x, y) \in P_{SL}^H$. By the choice of μ, we may assume $x_j = 1$ for all $j \in V \setminus V_C$. Moreover, by integrality of $(x, y) \in P_{SL}^H$, we have $y_S = \prod_{i \in S} x_i$ for all $S \in \mathcal{S}$. Now, since C is a special cycle, and using (3.7) again, we can express $h(x, y)$ as follows:

$$\begin{aligned} h(x,y) &= \mu|V \setminus V_C| + \sum_{j \in V^{--}} x_j - \sum_{e_k \in E_C^-} x_{i_k} x_{i_{k+1}} - \sum_{j \in V^{++}} x_j \\ &+ \sum_{e_k \in E_C^+} x_{i_k} x_{i_{k+1}} + \varepsilon\, t(y) \\ &= \mu|V \setminus V_C| + \sum_{e_k \in E_C^-} [x_{i_k}(1 - x_{i_{k+1}})] - \sum_{e_k \in E_C^+} [(1 - x_{i_k}) x_{i_{k+1}}] \\ &+ \varepsilon\, t(y). \end{aligned}$$

We will prove that

$$\sum_{e_k \in E_C^-} [x_{i_k}(1 - x_{i_{k+1}})] - \sum_{e_k \in E_C^+} [(1 - x_{i_k}) x_{i_{k+1}}] \leq \tfrac{|E_C^-|-1}{2}, \tag{3.8}$$

implying that

$$h(x,y) \leq \mu|V \setminus V_C| + \tfrac{|E_C^-|-1}{2} + \varepsilon\, t(y) < h(\hat{x}, \hat{y})$$

for every integer point $(x, y) \in P_H$ if ε is small enough.

Let us consider two edges $e_k, e_\ell \in E_C^-$ with $k < \ell$ and such that there is no other negative edge between e_k and e_ℓ in the cycle C. If there is no positive edge between e_k and e_ℓ in C, i.e., if $\ell = k + 1$, then we cannot simultaneously have $x_{i_k}(1 - x_{i_{k+1}}) = 1$ and $x_{i_\ell}(1 - x_{i_{\ell+1}}) = 1$. If there is at least one positive edge between e_k and e_ℓ, then it is possible that $x_{i_k}(1 - x_{i_{k+1}}) = x_{i_\ell}(1 - x_{i_{\ell+1}}) = 1$, but then at least one of the positive edges $e_j \in E_C^+$ between e_k and e_ℓ must be such that $(1 - x_{i_j}) x_{i_{j+1}} = 1$. In all cases, the two negative edges and the positive edges between them contribute for at most one unit to the left-hand side of (3.8), and therefore

$$\sum_{e_k \in E_C^-} [x_{i_k}(1 - x_{i_{k+1}})] - \sum_{e_k \in E_C^+} [(1 - x_{i_k}) x_{i_{k+1}}] \leq \tfrac{|E_C^-|}{2}.$$

Moreover, since C has an odd number of negative edges, inequality (3.8) is satisfied, thus completing the proof.

□

The basic idea of the previous proof is to reduce the construction of a fractional vertex of P_{SL}^H to the quadratic case. For this, all variables not corresponding to a node in V_C are set to one, letting them disappear from the monomial expressions corresponding to the edges in $\mathcal{S}_C$. This construction only works for special cycles.

In order to conclude the proof of Theorem 1, it remains to show that (c) implies (b), that is:

Proposition 4. *Given $\sigma \in \{-1, 1\}^{\mathcal{S}}$, if $H(\sigma)$ has no negative special cycle, then the matrix $M_{SL}^{H(\sigma)}$ is balanced.*

Proof. Assume that $M_{SL}^{H(\sigma)}$ is not balanced and let B be a smallest submatrix of $M_{SL}^{H(\sigma)}$ with two non-zero entries per row and two non-zero entries per column, such that the sum of its entries is congruent with 2 modulo 4.

Let $V_B \subseteq V$ be the set of vertices with their associated column in B, let $\mathcal{S}_B \subseteq \mathcal{S}$ be the set of edges associated with at least one row of B, let $\mathcal{S}_B^+$ be the set of positive edges in $\mathcal{S}_B$, and $\mathcal{S}_B^-$ be the set of negative edges in $\mathcal{S}_B$.

Since each column of B has exactly two non-zero entries, each vertex in V_B is contained in exactly two edges in $\mathcal{S}_B$. Moreover, by definition of $M_{SL}^{H(\sigma)}$, every edge $S \in \mathcal{S}_B$ must contain exactly two vertices of V_B (if $S \in \mathcal{S}^-$, then the entries corresponding to both vertices appear in the same row of B; if $S \in \mathcal{S}^+$, then the entries corresponding to the two vertices appear in two rows associated with y_S). Consequently, in view of the minimality of B, the vertices in V_B and the edges in $\mathcal{S}_B$ define a special cycle.

Since rows corresponding to edges $S \in \mathcal{S}_B^+$ are associated with constraints of type (3.2) and contain two non-zero entries by definition, the sum of these entries must be zero. This means that the sum of entries of rows corresponding to edges $S \in \mathcal{S}_B^-$ has a value congruent with 2 modulo 4. Notice that each edge $S \in \mathcal{S}_B^-$ has exactly one row of type (3.3) associated with it, and both entries of this row take value -1. This implies that there must be an odd number of negative edges in the special cycle defined by B. □

Chapter 4

A class of valid inequalities for multilinear 0–1 optimization problems

In this chapter, we introduce a class of valid inequalities for the convex hull of points satisfying the standard linearization constraints, which are called *2-links* or *2-link inequalities*, because they model interactions between pairs of monomials intersecting in at least two variables. [1]

We use the notations and defintions given in Chapter 2 (P_{SL} denotes the polytope called P_{SL}^H in Chapter 3). The main contribution of this chapter is a theorem stating that, when f contains exactly two nonlinear monomials, a complete formulation of P_{SL}^* is obtained by adding the 2-link inequalities to the standard linearization constraints of P_{SL}. We also establish that the 2-links are facet-defining when f consists of *nested* monomials, that is, of a chain of monomials contained in each other. Furthermore, we provide computational experiments showing that for various classes of multilinear polynomials, adding the 2-links to the standard linearization constraints of P_{SL} provides significant improvements in the quality of the bounds of the linear relaxations and on the performance of exact resolution methods.

The rest of the chapter is structured as follows. Section 4.1 formally introduces the 2-links. Section 4.2 establishes their strength for the case of nested monomials, and derives some related properties of the standard linearization inequalities. Section 4.3 presents our main result for the case of two nonlinear monomials. Section 4.4 describes our computational experiments. This set of experiments is extended in Part III of this thesis. Finally, Section 4.5 proposes some conclusions and sketches further research questions.

4.1 Definition and validity of 2-link inequalities

This section formally introduces the 2-links and establishes some of their properties. Let f be the function on variables x_i, $i \in [n]$, represented by the multilinear polynomial (2.1) with

[1]The results presented in this chapter have been published in [41].

$a_S \neq 0$ for all $S \in \mathcal{S}$.

$$f(x_1, \ldots, x_n) = \sum_{S \in \mathcal{S}} a_S \prod_{i \in S} x_i + \sum_{i \in [n]} a_i x_i. \tag{2.1}$$

Let the set X_{SL}, its convex hull P^*_{SL}, and its standard linearization polytope P_{SL} be defined as in Chapter 2. Note that X_{SL}, P_{SL} and P^*_{SL} actually depend on f, or more precisely on the set of monomials $\mathcal{S}$. However we do not indicate this dependence in the notation for simplicity.

As a reminder, the standard linearization inequalities associated with a monomial $S \in \mathcal{S}$ are

$$y_S \leq x_i, \qquad \forall i \in S \tag{2.5}$$

$$y_S \geq \sum_{i \in S} x_i - (|S| - 1), \tag{2.6}$$

$$y_S \geq 0. \tag{2.7}$$

Definition 7. *Consider two monomials indexed by subsets $S, T \in \mathcal{S}$ and consider variables y_S, y_T such that $y_S = \prod_{i \in S} x_i$, $y_T = \prod_{i \in T} x_i$. The* 2-link *associated with (S,T) is the linear inequality*

$$y_S \leq y_T - \sum_{i \in T \setminus S} x_i + |T \setminus S|. \tag{4.1}$$

Proposition 5. Validity of the 2-links. *For any $S, T \in \mathcal{S}$, the 2-link inequality (4.1) is valid for P^*_{SL}.*

Proof. It suffices to show that (4.1) is satisfied by all points (x, y) in X_{SL}. This is trivial when $y_S \leq y_T$. When $(y_S, y_T) = (1, 0)$, the monomial $\prod_{i \in S} x_i$ takes value one and the monomial $\prod_{i \in T} x_i$ takes value zero, which implies that a variable in $T \setminus S$ must be zero. It follows again that (4.1) is satisfied. □

Note that the 2-link inequalities are valid when $|S \cap T| < 2$, but in that case they do not strengthen the relaxation of P_{SL}. Indeed, when $S \cap T = \emptyset$, then (4.1) can be derived by simply adding the standard linearization inequalities $y_S \leq 1$ and $\sum_{i \in T} x_i - (|T| - 1) \leq y_T$. Also, when $|S \cap T| = 1$, say, $S \cap T = \{k\}$, then (4.1) is obtained by adding up $y_S \leq x_k$ and $\sum_{i \in T} x_i - (|T| - 1) \leq y_T$.

4.2 Nested nonlinear monomials

In order to illustrate the strength of 2-link inequalities, we next establish a result (Proposition 6 hereunder) concerning multilinear functions with a particular structure, namely, those for which the nonlinear monomials are nested. We already note that Proposition 6 has been independently found by Fischer, Fischer and McCormick [51] in the more general framework of polynomial functions optimized over a matroid polytope; we will return to this remark at the end of the section.

Observe that the 2-link inequality associated with (S,T) takes the form $y_S \leq y_T$ when $T \subseteq S$.

Proposition 6. Nested monomials. *Consider a function*

$$f(x) = \sum_{k\in[l]} a_{S^{(k)}} \prod_{i\in S^{(k)}} x_i + \sum_{i\in S^{(l)}} a_i x_i$$

defined on l monomials such that $S^{(1)} \subset S^{(2)} \subset \cdots \subset S^{(l)}$, where $|S^{(1)}| \geq 2$ and $S^{(l)} = [n]$ without loss of generality. Let $P^{*,nest}_{SL}$ be the convex hull of the integer points of the standard linearization polytope associated with f. Then, the 2-links

$$y_{S^{(k)}} \leq y_{S^{(k+1)}} - \sum_{i\in S^{(k+1)}\setminus S^{(k)}} x_i + |S^{(k+1)}\setminus S^{(k)}|, \tag{4.2}$$

$$y_{S^{(k+1)}} \leq y_{S^{(k)}}, \tag{4.3}$$

for $k = 1, \dots, l-1$, are facet-defining for $P^{*,nest}_{SL}$.

Proof. Let $x = (x_1, \dots, x_n)$, let u_i be the n-dimensional unit vector with i^{th} component equal to one, let $y = (y_{S^{(1)}}, \dots, y_{S^{(l)}})$, and let v_j be the l-dimensional unit vector with the j^{th} component equal to one.

Observe first that $P^{*,nest}_{SL}$ is full-dimensional; indeed, the n points $(x, y) = (u_i, 0)$, $\forall i \in [n]$, the l points $(x, y) = (\sum_{i\in S^{(k)}} u_i, \sum_{j\leq k} v_j)$, $\forall k \in [l]$, and the point $(0, 0)$ are in $P^{*,nest}_{SL}$ and are affinely independent (see also [46]).

Let F be the face of $P^{*,nest}_{SL}$ represented by (4.2), $F = \{(x, y) \in P^{*,nest}_{SL} \mid y_{S^{(k)}} = y_{S^{(k+1)}} - \sum_{i\in S^{(k+1)}\setminus S^{(k)}} x_i + |S^{(k+1)}\setminus S^{(k)}|\}$, for a fixed $k < l$. To prove that F is a facet, we will show that F is contained in a unique hyperplane and thus $dim(F) = dim(P^{*,nest}_{SL}) - 1$, since $P^{*,nest}_{SL}$ is full-dimensional. Consider $b(x, y) = \sum_{i\in[n]} b_i x_i + \sum_{k\in[l]} b_{S^{(k)}} y_{S^{(k)}}$ and assume that F is contained in the hyperplane $b(x, y) = b_0$. We will see that this is only possible if $b(x, y) = b_0$ is a multiple of

$$y_{S^{(k)}} = y_{S^{(k+1)}} - \sum_{i\in S^{(k+1)}\setminus S^{(k)}} x_i + |S^{(k+1)}\setminus S^{(k)}|. \tag{4.4}$$

1. The point $(x, y) = (\sum_{i\in S^{(k+1)}\setminus S^{(k)}} u_i, 0)$ is in F. Assuming that (x, y) satisfies $b(x, y) = b_0$, we have that $\sum_{i\in S^{(k+1)}\setminus S^{(k)}} b_i = b_0$.

2. Fix an index $j \in S^{(k)}$, and consider $(x, y) = (u_j + \sum_{i\in S^{(k+1)}\setminus S^{(k)}} u_i, 0) \in F$. Assuming that (x, y) satisfies $b(x, y) = b_0$, we have $b_j + \sum_{i\in S^{(k+1)}\setminus S^{(k)}} b_i = b_0$, which implies, together with the previous condition, that $b_j = 0$, $\forall j \in S^{(k)}$.

3. If $k+1 < l$, fix an index $j \in S^{(l)}\setminus S^{(k+1)}$, and consider $(x, y) = (\sum_{i\in S^{(k+1)}\setminus S^{(k)}} u_i + u_j, 0) \in F$. Assuming that (x, y) satisfies $b(x, y) = b_0$, we have that $b_j + \sum_{i\in S^{(k+1)}\setminus S^{(k)}} b_i = b_0$, which implies together with the first condition that $b_j = 0$, $\forall j \in S^{(l)}\setminus S^{(k+1)}$.

4. We next show that $b_{S^{(j)}} = 0$ for $j < k$. Assume first that $j = 1 < k$ and let $(x, y) = (\sum_{i\in S^{(k+1)}\setminus S^{(k)}} u_i + \sum_{i\in S^{(1)}} u_i, v_1) \in F$. Assuming that (x, y) satisfies $b(x, y) = b_0$, we have that $\sum_{i\in S^{(1)}} b_i + \sum_{i\in S^{(k+1)}\setminus S^{(k)}} b_i + b_{S^{(1)}} = b_0$, which implies, together with the previous conditions, that $b_{S^{(1)}} = 0$. Repeating this procedure for $j = 2, \dots, k-1$ (in this order), we obtain that $b_{S^{(j)}} = 0$ for all $j < k$.

5. Fix a $j \in S^{(k+1)} \backslash S^{(k)}$, and take $(x, y) = (\sum_{i \in S^{(k+1)} \backslash \{j\}} u_i, \sum_{i \leq k} v_i) \in F$. Assuming that (x, y) satisfies $b(x, y) = b_0$, we have $\sum_{i \in S^{(k+1)}, i \neq j} b_i + \sum_{i \leq k} b_{S^{(i)}} = b_0$, which implies, together with the previous conditions and repeating for $j \in S^{(k+1)} \backslash S^{(k)}$, that $b_j = b_{S^{(k)}}$, for all $j \in S^{(k+1)} \backslash S^{(k)}$.

6. Consider $(x, y) = (\sum_{i \in S^{(k+1)}} u_i, \sum_{i \leq k+1} v_i) \in F$. Assuming that (x, y) satisfies $b(x, y) = b_0$, we obtain that $\sum_{i \in S^{(k+1)}} b_i + \sum_{i \leq k+1} b_{S^{(i)}} = b_0$, which implies, together with the previous conditions, that $b_{S^{(k)}} + b_{S^{(k+1)}} = 0$.

7. Consider subset $S^{(k+2)}$, and take $(x, y) = (\sum_{i \in S^{(k+2)}} u_i, \sum_{j \leq k+2} v_j) \in F$. Assuming that (x, y) satisfies $b(x, y) = b_0$, and using the previous conditions we have that $b_{S^{(k+2)}} = 0$. Repeating this reasoning for $j = k + 3, \dots l$ (in this order), we obtain $b_{S^{(j)}} = 0$, for all $j > k + 1$.

Putting together the previous conditions, we have that $b(x, y) = b_0$ takes the form $b_{S^{(k)}} \sum_{i \in S^{(k+1)} \backslash S^{(k)}} x_i + b_{S^{(k)}} y_{S^{(k)}} - b_{S^{(k)}} y_{S^{(k+1)}} = |S^{(k+1)} \backslash S^{(k)}| \, b_{S^{(k)}}$, which is a multiple of equation (4.4) as required.

In a similar way, it can be proved that the face represented by (4.3) is a facet, by using the following set of points (in the given order): $(0, 0)$; $(u_i, 0)$, for $i = 1, \dots, n$; $(\sum_{i \in S^{(j)}} u_i, \sum_{r \leq j} v_r)$, for $j \in \{1, \dots, n\} \backslash \{k\}$. □

Remark 4. *The 2-link inequalities are only facet-defining for consecutive monomials in the nested sequence. In fact, the 2-links corresponding to non-consecutive monomials are implied by the 2-links associated with consecutive monomials.*

The following remarks can be proved using similar arguments as those presented in the proof of Proposition 6. They imply, in particular, that the standard linearization inequalities (2.5)–(2.7) are not always facet-defining for $P^{*,nest}_{SL}$.

Remark 5. *The lower bounding inequality $0 \leq y_{S^{(l)}}$ is facet-defining for $P^{*,nest}_{SL}$. However, the inequalities $0 \leq y_{S^{(k)}}$, $k = 1, \dots, l-1$ are redundant, since they are implied by $0 \leq y_{S^{(k+1)}}$ and by $y_{S^{(k+1)}} \leq y_{S^{(k)}}$.*

Remark 6. *The standard linearization inequality $y_{S^{(1)}} \geq \sum_{i \in S^{(1)}} x_i - (|S^{(1)}| - 1)$ is facet-defining for $P^{*,nest}_{SL}$. However $y_{S^{(k)}} \geq \sum_{i \in S^{(k)}} x_i - (|S^{(k)}| - 1)$, $k = 2, \dots, l$ are redundant, since they are implied by $y_{S^{(k-1)}} \geq \sum_{i \in S^{(k-1)}} x_i - (|S^{(k-1)}| - 1)$ and $y_{S^{(k-1)}} \leq y_{S^{(k)}} - \sum_{i \in S^{(k)} \backslash S^{(k-1)}} x_i + |S^{(k)} \backslash S^{(k-1)}|$.*

Remark 7. *The standard linearization inequalities $y_{S^{(k)}} \leq x_i$, $i \in S^{(k)} \backslash S^{(k-1)}$ are facet-defining for $P^{*,nest}_{SL}$ for all $k = 1, \dots, l$, where $S^{(0)} = \emptyset$. However $y_{S^{(k)}} \leq x_i$, $i \in S^{(k-1)}$ are redundant, since they are implied by $y_{S^{(k-1)}} \leq x_i$, $i \in S^{(k-1)}$ and $y_{S^{(k)}} \leq y_{S^{(k-1)}}$.*

The results by Fischer et al. [51] actually imply that inequalities (4.2), (4.3), together with the facet-defining inequalities of Remarks 5, 6 and 7, define the convex hull $P^{*,nest}_{SL}$ for the nested case.

4.3 The case of two nonlinear monomials

In this section, we present some results for the special case of a multilinear function $f(x) = a_S \prod_{i \in S} x_i + a_T \prod_{i \in T} x_i + \sum_{i \in [n]} a_i x_i$ containing exactly two nonlinear monomials indexed by S and T. Our first result states that the 2-links associated with S and T are facet-defining for P^*_{SL}, whenever $|S \cap T| \geq 2$. As observed in Section 4.1, the 2-links are valid but redundant for $|S \cap T| < 2$. Our second and main result is a theorem stating that the 2-links, together with the standard linearization inequalities, provide a complete description of P^*_{SL}. Since the case of nested monomials has been covered above, we assume throughout this section that $S \not\subseteq T$ and $T \not\subseteq S$. Moreover, we also assume for simplicity that $S \cup T = [n]$: indeed, when $S \cup T \subset [n]$, the value of the variables in $R = [n] \backslash (S \cup T)$ is unrestricted, and hence $conv(X_{SL}) = conv(X'_{SL}) \times [0,1]^{|R|}$, where X'_{SL} is the projection of X_{SL} on the subspace defined by the variables x_i ($i \in S \cup T$), y_S, and y_T. So, it is sufficient to understand the case where $S \cup T = [n]$.

Proposition 7. *The standard linearization inequalities (2.5), (2.6) and (2.7) are facet-defining for the case of a function f containing exactly two nonlinear monomials defined by subsets S and T such that $S \not\subseteq T$ and $T \not\subseteq S$.*

Proof. Let $x = (x_1, \ldots, x_n)$, let u_i denote the n-dimensional unit vector with i^{th} component equal to one, let $y = (y_S, y_T)$, and let $v_S = (1,0)$, $v_T = (0,1)$, respectively.

To prove that inequality (2.5) (for monomial S and variable $i \in S$) is facet-defining, one can use the same technique as in the proof of Proposition 6, using the following set of points (in the given order): $(0,0)$; $(u_j, 0)$, for $j = 1, \ldots, n$, $j \neq i$; $(\sum_{j \in S} u_j, v_S)$; $(\sum_{j \in S \cup T} u_j, v_S + v_T)$.

Similarly, for inequality (2.6) (for monomial S), one can use the following points: $(\sum_{i \in S} u_i, v_S)$; for each $k \in S$, $(\sum_{i \in S, i \neq k} u_i, 0)$; for each $k \in T \backslash S$, $(u_k + \sum_{i \in S, i \neq j} u_i, 0)$ (where $j \in S$ if $S \cap T = \emptyset$, and $j \in S \cap T$ otherwise); $(\sum_{i \in S \cup T} u_i, v_S + v_T)$.

Finally, for inequality (2.7) (for monomial S) one can use the following points: $(0,0)$; $(u_i, 0)$, for $i = 1, \ldots, n$; $(\sum_{i \in T} u_i, v_T)$.

For monomial T, the proofs are symmetric. □

Note that Proposition 7 is valid for any value of $|S \cap T|$.

Proposition 8. *The 2-links*

$$y_S \leq y_T - \sum_{i \in T \backslash S} x_i + |T \backslash S| \tag{4.5}$$

$$y_T \leq y_S - \sum_{i \in S \backslash T} x_i + |S \backslash T|, \tag{4.6}$$

are facet-defining for P^*_{SL}, the convex hull of the integer points of the standard linearization polytope associated with a function f containing exactly two nonlinear monomials defined by subsets S and T such that $|S \cap T| \geq 2$.

Proof. As in the proof of Proposition 7, let $x = (x_1, \dots, x_n)$, let u_i denote the n-dimensional unit vector with i^{th} component equal to one, let $y = (y_S, y_T)$, and let $v_S = (1, 0)$, $v_T = (0, 1)$, respectively. Since Proposition 6 covers the case of nested monomials, we assume that $S \nsubseteq T$ and $T \nsubseteq S$. We will prove that (4.5) is facet-defining (the proof for (4.6) is symmetric).

Observe that P^*_{SL} is full-dimensional (i.e., of dimension $n + 2$), given that the n points $(u_i, 0)$, $\forall i \in [n]$, the two points $(\sum_{i\in S} u_i, v_S)$ and $(\sum_{i\in T} u_i, v_T)$, and the point $(0, 0)$ are contained in P^*_{SL} and are affinely independent (see also [46]).

Now, let F be the face of P^*_{SL} represented by (4.5), $F = \{(x, y) \in P^*_{SL} \mid y_S = y_T - \sum_{i\in T\setminus S} x_i + |T\setminus S|\}$. Let $b(x, y) = \sum_{i\in[n]} b_i x_i + b_S y_S + b_T y_T$ and assume that F is contained in the hyperplane $b(x, y) = b_0$. We will use the same technique as for Proposition 6 to see that F is a facet.

1. Consider $(x, y) = (\sum_{i\in T\setminus S} u_i, 0) \in F$. Assuming that (x, y) satisfies $b(x, y) = b_0$, we obtain that $\sum_{i\in T\setminus S} b_i = b_0$.

2. Fix an index $j \in S$ and consider $(x, y) = (\sum_{i\in T\setminus S} u_i + u_j, 0) \in F$. Assuming that (x, y) satisfies $b(x, y) = b_0$ and using the previous condition we deduce that $b_j = 0$ for all $j \in S$.

3. Fix an index $j \in T\setminus S$. Consider $(x, y) = (\sum_{i\in(S\cup T)\setminus\{j\}} u_i, v_S) \in F$. Assuming that (x, y) satisfies $b(x, y) = b_0$ we obtain $\sum_{i\in(S\cup T)\setminus\{j\}} b_i + b_S = b_0$, which, together with the previous conditions, implies $b_j = b_S$ for all $j \in T\setminus S$.

4. Consider $(x, y) = (\sum_{i\in S\cup T} u_i, v_S + v_T) \in F$. Assuming that (x, y) satisfies $b(x, y) = b_0$, and together with the previous conditions, we obtain $b_S = -b_T$.

Putting together the previous conditions, we have that $b(x, y) = b_0$ takes the form $b_S y_S - b_S y_T + b_S \sum_{i\in T\setminus S} x_i = b_S|T\setminus S|$. □

Proposition 8 establishes that the 2-links are strong valid inequalities for P^*_{SL}. We will see that, in addition, when we add the 2-links to P_{SL}, we obtain a complete description of P^*_{SL}. For this, let

$$P^{2links}_{SL} = P_{SL} \cap \{(x, y_S, y_T) \in \mathbb{R}^{n+2} \mid (4.5),\ (4.6) \text{ are satisfied}\}.$$

It is easy to see that the bound constraints $y_S \le 1$, $y_T \le 1$ and $x_i \ge 0$, $\forall i \in [n]$ are implied by the standard linearization inequalities (2.5) and by the remaining bound constraints. We keep them in the description of P^{2links}_{SL} for simplicity of exposition. Note that for $|S \cap T| < 2$, we have that $P^{2links}_{SL} = P_{SL}$, since the 2-links are redundant.

Theorem 3. *$P^*_{SL} = P^{2links}_{SL}$ when the function f contains two nonlinear monomials.*

For disjoint monomials, this can be derived directly from Remark 1. For the general case, the proof relies on a classical result by Balas ([6, 7], see also [36] for the bounded case) aimed at modeling the convex hull of the union of q polytopes $P^1, \dots, P^q \subseteq \mathbb{R}^m$ such that, for $k \in [q]$, P^k is described by the inequalities

$$\begin{aligned} A^k x &\le b^k, \\ 0 \le x &\le d^k. \end{aligned} \tag{4.7}$$

The union $\cup_{k\in[q]}P^k$ can be modeled by introducing q binary variables z^k, indicating whether a point x is in the k^{th} polytope, and q vectors of variables $x^k \in \mathbb{R}^m$. Then, a point $x \in \mathbb{R}^m$ belongs to $\cup_{k\in[q]}P_k$ if and only if there exist $x^1, \ldots, x^q$ and $z^1, \ldots, z^q$ such that

$$\sum_{k\in[q]} x^k = x \tag{4.8}$$

$$A^k x^k \le b^k z^k, \quad k \in [q] \tag{4.9}$$

$$0 \le x^k \le d^k z^k, \quad k \in [q] \tag{4.10}$$

$$\sum_{k\in[q]} z^k = 1 \tag{4.11}$$

$$z^k \in \{0,1\}, \quad k \in [q]. \tag{4.12}$$

Let Q be the set of points $(x, x^1, \ldots, x^q, z^1, \ldots, z^q)$ satisfying (4.8)–(4.12). Balas' result states that this disjunctive model is perfect:

Proposition 9. *[6, 7, 36] The convex hull of solutions to (4.8)–(4.12), that is, conv(Q), is described by inequalities (4.8)–(4.11) and* $z^k \in [0,1]$ *for* $k \in [q]$.

For any set $W \subseteq \mathbb{R}^{n+l}$ (defined on variables $(x,w) \in \mathbb{R}^{n+l}$), let $Proj_x(W)$ be the projection of W on the space of the x variables. With these notations we can write the union of the polytopes as $\cup_{k\in[q]}P^k = Proj_x(Q)$ and, by commutativity of the operators *conv* and $Proj_x$, we have that

$$conv(\cup_{k\in[q]}P^k) = Proj_x(conv(Q)). \tag{4.13}$$

So, Proposition 9 provides a perfect extended formulation of $conv(\cup_{k\in[q]}P^k)$.

We are now ready for a proof of Theorem 3.

Proof. We will show that all vertices of P^{2links}_{SL} are integer and therefore the inequalities defining P^{2links}_{SL} provide a perfect formulation of P^*_{SL} (i.e., $P^{2links}_{SL} = P^*_{SL}$). Consider the following set of inequalities, where (4.14)–(4.15) result from the standard linearization of $y_{S\cap T} = \prod_{i\in S\cap T} x_i$, (4.16)–(4.18) result from the standard linearization of $y_S = y_{S\cap T}\prod_{i\in S\setminus T} x_i$, and (4.19)–(4.21) result from the standard linearization of $y_T = y_{S\cap T}\prod_{i\in T\setminus S} x_i$:

$$y_{S\cap T} \le x_i, \quad \forall i \in S\cap T, \tag{4.14}$$

$$y_{S\cap T} \ge \sum_{i\in S\cap T} x_i - (|S\cap T| - 1), \tag{4.15}$$

$$y_S \le y_{S\cap T}, \tag{4.16}$$

$$y_S \le x_i, \quad \forall i \in S\setminus T, \tag{4.17}$$

$$y_S \ge \sum_{i\in S\setminus T} x_i + y_{S\cap T} - |S\setminus T|, \tag{4.18}$$

$$y_T \le y_{S\cap T}, \tag{4.19}$$

$$y_T \le x_i, \quad \forall i \in T\setminus S, \tag{4.20}$$

$$y_T \geq \sum_{i \in T \setminus S} x_i + y_{S \cap T} - |T \setminus S|, \tag{4.21}$$

$$0 \leq y_S \leq 1, \tag{4.22}$$

$$0 \leq y_T \leq 1, \tag{4.23}$$

$$0 \leq y_{S \cap T} \leq 1, \tag{4.24}$$

$$0 \leq x_i \leq 1, \qquad \forall i \in S \cup T. \tag{4.25}$$

Let P denote the polytope

$$P = \{(x, y_S, y_T, y_{S \cap T}) \in \mathbb{R}^{n+3} \mid (4.14) - (4.25) \text{ are satisfied}\},$$

and let P^0 (respectively, P^1) denote the faces of P defined by fixing $y_{S \cap T} = 0$ (respectively, $y_{S \cap T} = 1$) in (4.14)–(4.25). So, P^0 is described by the constraints

$$\begin{aligned}
&\sum_{i \in S \cap T} x_i - (|S \cap T| - 1) \leq 0 \\
&y_{S \cap T} = y_S = y_T = 0 \\
&0 \leq x_i \leq 1, && \forall i \in S \cup T
\end{aligned}$$

and P^1 is described by

$$\begin{aligned}
&\sum_{i \in S \setminus T} x_i - (|S \setminus T| - 1) \leq y_S \\
&\sum_{i \in T \setminus S} x_i - (|T \setminus S| - 1) \leq y_T \\
&y_S \leq x_i, && \forall i \in S \setminus T \\
&y_T \leq x_i, && \forall i \in T \setminus S \\
&0 \leq y_S \leq 1 \\
&0 \leq y_T \leq 1 \\
&y_{S \cap T} = x_i = 1, && \forall i \in S \cap T \\
&0 \leq x_i \leq 1, && \forall i \in (S \setminus T) \cup (T \setminus S).
\end{aligned}$$

Observe that P^0 is an integer polytope because it is defined by a (totally unimodular) cardinality constraint. Polytope P^1 is also integer, because it is defined by the standard linearization constraints corresponding to two monomials on disjoint sets of variables, namely, $\prod_{i \in S \setminus T} x_i$ and $\prod_{i \in T \setminus S} x_i$; hence we can use the fact that the inequalities defining P_{SL} are a perfect formulation for a single nonlinear monomial.

As a consequence, $conv(P^0 \cup P^1)$ also is an integral polytope. Our objective is now to describe this polytope and, namely, to show that $conv(P^0 \cup P^1) = P$. In view of Proposition 9, a point $(x, y_S, y_T, y_{S \cap T})$ belongs to $conv(P^0 \cup P^1)$ if and only if there exist $x_i^0, x_i^1 \in \mathbb{R}^n$ $(i \in S \cup T)$ and $y_S^0, y_S^1, y_T^0, y_T^1, y_{S \cap T}^0, y_{S \cap T}^1, z^0, z^1 \in \mathbb{R}$ such that

$$x_i^0 + x_i^1 = x_i, \qquad \forall i \in S \cup T, \tag{4.26}$$
$$y_S^0 + y_S^1 = y_S, \tag{4.27}$$
$$y_T^0 + y_T^1 = y_T, \tag{4.28}$$
$$y_{S\cap T}^0 + y_{S\cap T}^1 = y_{S\cap T}, \tag{4.29}$$
$$\sum_{i\in S\cap T} x_i^0 \leq (|S \cap T| - 1)\, z^0, \tag{4.30}$$
$$y_{S\cap T}^0 = 0, \tag{4.31}$$
$$y_S^0 = 0, \tag{4.32}$$
$$y_T^0 = 0, \tag{4.33}$$
$$x_i^0 \leq z^0, \qquad \forall i \in S \cup T, \tag{4.34}$$
$$0 \leq x_i^0, \qquad \forall i \in S \cup T, \tag{4.35}$$
$$\sum_{i\in S\setminus T} x_i^1 - y_S^1 \leq (|S\setminus T| - 1)\, z^1, \tag{4.36}$$
$$\sum_{i\in T\setminus S} x_i^1 - y_T^1 \leq (|T\setminus S| - 1)\, z^1, \tag{4.37}$$
$$y_S^1 \leq x_i^1, \qquad \forall i \in S\setminus T, \tag{4.38}$$
$$y_T^1 \leq x_i^1, \qquad \forall i \in T\setminus S, \tag{4.39}$$
$$y_{S\cap T}^1 = z^1, \tag{4.40}$$
$$y_S^1 \leq z^1, \tag{4.41}$$
$$0 \leq y_S^1, \tag{4.42}$$
$$y_T^1 \leq z^1, \tag{4.43}$$
$$0 \leq y_T^1, \tag{4.44}$$
$$x_i^1 = z^1, \qquad \forall i \in S \cap T, \tag{4.45}$$
$$x_i^1 \leq z^1, \qquad \forall i \in (S\setminus T) \cup (T\setminus S) \tag{4.46}$$
$$0 \leq x_i^1, \qquad \forall i \in (S\setminus T) \cup (T\setminus S) \tag{4.47}$$
$$z^0 + z^1 = 1, \tag{4.48}$$
$$z^0 \leq 1, \tag{4.49}$$
$$0 \leq z^0, \tag{4.50}$$
$$z^1 \leq 1, \tag{4.51}$$
$$0 \leq z^1. \tag{4.52}$$

Let W denote the polytope defined by constraints (4.26)-(4.52). We will explicitly calculate the projection $Proj_{(x,y_S,y_T,y_{S\cap T})}(W) = conv(P^0 \cup P^1)$.

First, we simplify constraints (4.26)-(4.52) using the following observations:

- Substituting (4.32) in (4.27) we obtain $y^1_S = y_S$.
- Substituting (4.33) in (4.28) we obtain $y^1_T = y_T$.
- Substituting (4.31) in (4.29) we obtain $y^1_{S\cap T} = y_{S\cap T}$, which in turn gives $z^1 = y_{S\cap T}$ using (4.40).
- Using $z^1 = y_{S\cap T}$ in (4.48) we have that $z^0 = 1 - y_{S\cap T}$.
- Substituting (4.45) in (4.26) for $i \in S \cap T$ and using $z^1 = y_{S\cap T}$, we have that $x^0_i = x_i - y_{S\cap T}, \forall i \in S \cap T$.
- Finally, (4.26) also gives that $x^1_i = x_i - x^0_i, \forall i \in (S\setminus T) \cup (T\setminus S)$.

Applying these substitutions to (4.26)–(4.52), we obtain

$$\sum_{i\in S\cap T} x_i - (|S \cap T| - 1) \le y_{S\cap T}, \tag{4.53}$$
$$y_{S\cap T} \le x_i, \qquad \forall i \in S \cap T, \tag{4.54}$$
$$y_S \le y_{S\cap T}, \tag{4.55}$$
$$y_T \le y_{S\cap T}, \tag{4.56}$$
$$x_i \le 1, \qquad \forall i \in S \cap T, \tag{4.57}$$
$$0 \le y_S, \tag{4.58}$$
$$0 \le y_T, \tag{4.59}$$
$$0 \le y_{S\cap T} \le 1, \tag{4.60}$$
$$\sum_{i\in S\setminus T} x_i - \sum_{i\in S\setminus T} x^0_i \le y_S + y_{S\cap T}(|S\setminus T| - 1), \tag{4.61}$$
$$\sum_{i\in T\setminus S} x_i - \sum_{i\in T\setminus S} x^0_i \le y_T + y_{S\cap T}(|T\setminus S| - 1), \tag{4.62}$$
$$y_S \le x_i - x^0_i, \qquad \forall i \in S\setminus T, \tag{4.63}$$
$$y_T \le x_i - x^0_i, \qquad \forall i \in T\setminus S, \tag{4.64}$$
$$y_{S\cap T} \le 1 - x^0_i, \qquad \forall i \in (S\setminus T) \cup (T\setminus S), \tag{4.65}$$
$$x_i - x^0_i \le y_{S\cap T}, \qquad \forall i \in (S\setminus T) \cup (T\setminus S), \tag{4.66}$$
$$x^0_i \le x_i, \qquad \forall i \in (S\setminus T) \cup (T\setminus S), \tag{4.67}$$
$$0 \le x^0_i, \qquad \forall i \in (S\setminus T) \cup (T\setminus S). \tag{4.68}$$

We will now use the Fourier-Motzkin elimination method to project out all variables x^0_i from (4.53)–(4.68), for $i \in (S\setminus T) \cup (T\setminus S)$, so as to obtain a description of $conv(P^0 \cup P^1)$ in the space of variables $(x_i, y_S, y_T, y_{S\cap T})$.

Notice that constraints (4.53)–(4.60) will not play any role in the projection, since they do not involve the variables x^0_i.

Proceeding by induction on the number of eliminated variables, let $I \subseteq S \setminus T$ and $J \subseteq T \setminus S$ be the sets of indices such that variables x_i^0 have been projected out for all $i \in I \cup J$, and let $|I| = p$, $|J| = q$. As induction hypothesis, suppose that after eliminating the variables in $I \cup J$, the formulation is defined by constraints (4.53)–(4.60) together with the following inequalities:

$$\begin{aligned}
&0 \le x_i \le 1, && \forall i \in I \cup J, && (4.69)\\
&y_S \le x_i, && \forall i \in I, && (4.70)\\
&y_T \le x_i, && \forall i \in J, && (4.71)\\
&\sum_{i \in S \setminus T} x_i - \sum_{i \in (S \setminus T) \setminus I} x_i^0 \le y_S + y_{S \cap T} (|S \setminus T| - (p+1)) + p, && && (4.72)\\
&\sum_{i \in T \setminus S} x_i - \sum_{i \in (T \setminus S) \setminus J} x_i^0 \le y_T + y_{S \cap T} (|T \setminus S| - (q+1)) + q, && && (4.73)
\end{aligned}$$

$$\begin{aligned}
&y_S \le x_i - x_i^0, && \forall i \in (S \setminus T) \setminus I, && (4.74)\\
&y_T \le x_i - x_i^0, && \forall i \in (T \setminus S) \setminus J, && (4.75)\\
&y_{S \cap T} \le 1 - x_i^0, && \forall i \in ((S \setminus T) \setminus I) \cup ((T \setminus S) \setminus J), && (4.76)\\
&x_i - x_i^0 \le y_{S \cap T}, && \forall i \in ((S \setminus T) \setminus I) \cup ((T \setminus S) \setminus J), && (4.77)\\
&x_i^0 \le x_i, && \forall i \in ((S \setminus T) \setminus I) \cup ((T \setminus S) \setminus J), && (4.78)\\
&0 \le x_i^0, && \forall i \in ((S \setminus T) \setminus I) \cup ((T \setminus S) \setminus J). && (4.79)
\end{aligned}$$

Note that the induction hypothesis holds when $I = J = \emptyset$ and $p = q = 0$, since (4.69)–(4.79) boils down to (4.61)–(4.68) in this case. Given I, J, p and q, let us now eliminate variable x_j^0, where $j \in (S \setminus T) \setminus I$, by the Fourier-Motzkin method (the analysis would be symmetric for $j \in (T \setminus S) \setminus J$). This leads to inequality $x_j \le 1$ by combining constraints (4.76) and (4.77) for j, to inequality $y_S \le x_j$ by combining (4.74) and (4.79) for j, and to inequality $0 \le x_j$ by combining (4.78) and (4.79) for j. Combining (4.72) and (4.76) yields

$$\sum_{i \in S \setminus T} x_i - \sum_{i \in (S \setminus T) \setminus (I \cup \{j\})} x_i^0 \le y_S + y_{S \cap T} (|S \setminus T| - (p+2)) + p + 1.$$

All other combinations of inequalities containing x_j^0 in (4.69)–(4.79) lead to redundant constraints. So, clearly, the formulation obtained after projecting out x_j^0 is the same as (4.69)–(4.79), with I replaced by $I \cup \{j\}$. This shows that the induction hypothesis holds for all I, J, p, q.

Assume now that we have eliminated all variables x_i^0, $i \in (S \setminus T) \cup (T \setminus S)$. In this case, it follows from the inductive reasoning that constraints (4.74)–(4.79) become vacuous. Moreover, the remaining constraints (4.53)–(4.60) and (4.69)–(4.73) are exactly (4.14)–(4.25), the defining constraints of polytope P (except for the bounds $y_S \le 1$, $y_T \le 1$ and $x_i \ge 0$, $\forall i \in S \cap T$, which, as stated previously, are among the redundant constraints and can be easily derived from the remaining inequalities). Therefore, we have proved that $P = conv(P^0 \cup P^1)$, which implies that P has integer vertices.

To conclude the proof, we are going to show next that P_{SL}^{2links} is exactly the projection of P on the space of (x, y_S, y_T) variables. Indeed, if we use the Fourier-Motzkin elimination method to project out variable $y_{S \cap T}$ from (4.14)-(4.25), then we obtain the standard linearization inequality (2.6) for S by combining constraints (4.15) and (4.18), and for T by combining (4.15) and (4.21). Constraints (2.5) for y_S, y_T, and $i \in S \cap T$ are obtained by combining (4.14) and (4.16), and (4.14) and (4.19), respectively. Finally, the 2-links (4.5) and (4.6) are obtained from inequalities (4.16), (4.21) and (4.18), (4.19), respectively.

So, we have established that $P_{SL}^{2links} = Proj_{(x,y_S,y_T)}(P)$. Since P is bounded, every vertex of P_{SL}^{2links} is the projection of a vertex of P. This implies that all vertices of P_{SL}^{2links} are integer, since P is integral. Thus, the inequalities defining P_{SL}^{2links} provide a perfect formulation for X_{SL}, that is, $P_{SL}^{2links} = P_{SL}^*$.

□

4.4 Computational experiments

We have seen in Section 4.3 that adding all possible 2-links to P_{SL} provides a complete description of P_{SL}^* when the associated function contains two nonlinear monomials. This is not true anymore for functions with three nonlinear monomials. A counterexample is given by the function $f_{3mon}(x) = 5x_1x_2x_4 - 3x_1x_3x_4 - 3x_1x_2x_3 + 2x_3$. If we define P_{SL}^{2links} for f_{3mon} and optimize the corresponding linearized function L_f over P_{SL}^{2links}, we obtain the fractional solution $x_i = 0.5$ for $i = 1, 2, 3, 4$, $y_{134} = 0.5$, $y_{124} = 0$ and $y_{123} = 0.5$.

However, the 2-links might still be helpful for the general case of functions containing more than two nonlinear monomials. In this section, we provide computational evidence showing that the 2-links improve the LP-relaxation of the standard linearization, as well as the computational performance of exact resolution methods. This may not be totally expected, since the 2-links are in relatively small number (quadratic in the number of terms). It appears, however, that capturing relations between pairs of terms improves the standard linearization formulation to a certain extent. Buchheim and Klein [29] provide results of a related nature for constrained binary quadratic problems, in the sense that they derive valid inequalities for simplified problems involving a single quadratic term, and observe that these inequalities result in significant improvements when applied to the general case.

In our experiments, we consider two classes of instances of the integer linear program

$$\min \quad L_f(x, y) = \textstyle\sum_{S \in \mathcal{S}} a_S y_S + \sum_{i \in [n]} a_i x_i \tag{4.80}$$
$$\text{subject to} \quad (x, y) \in P_{SL} \tag{4.81}$$
$$x \in \{0, 1\}^n. \tag{4.82}$$

The first class contains RANDOM instances that are randomly generated in the same way as in [30]. The second class contains so-called VISION instances; they are inspired by an image restoration problem which is widely studied in the field of computer vision. A description of all instances is provided in the next subsections.

We have used CPLEX 12.6 [72] to run our experiments. We report two types of results. First, we compare the bound obtained when solving the relaxed problem (4.80)-(4.81) with the bound obtained when optimizing (4.80) over P_{SL}^{2links}. Next, we focus on the computational performance of the CPLEX IP-solver when solving the instances to optimality. We compare four different versions of branch & cut to solve (4.80)-(4.82), namely:

1. NO CUTS: the automatic cut generation mechanism of CPLEX is disabled to solve the plain standard linearization model (4.80)-(4.82).
2. USER CUTS: we solve the standard linearization model enhanced with the addition of 2-links (i.e., over the polytope P_{SL}^{2links}) but without additional automatic cut generation by CPLEX.
3. CPLEX CUTS: the automatic cut generation mechanism of CPLEX is enabled (with the default setting of cut generation parameters) to solve the standard linearization model (4.80)-(4.82).
4. CPLEX & USER CUTS (C & U): CPLEX is allowed to use two types of cuts, namely, the 2-links and any additional cuts that it can automatically generate to solve the standard linearization model.

Note that when the 2-link inequalities are used in the branch & cut process, they are treated as a pool of so-called “user cuts”. During the process, CPLEX first tries to cut off the current solution by relying on these user cuts only, and next generates its own cuts as needed.

Except for the cut generation parameters, all other IP resolution parameters are set to default. Several preliminary tests have been performed in order to determine the best settings of CPLEX pre-processing parameters. As a result, we chose to set the `Linear Reduction Switch` parameter to the non-default value “perform only linear reductions” since this is the recommended setting by CPLEX whenever there are user cuts. A time limit of 1 hour was set for each instance. All experiments were run on a PC with processor Intel(R) Core(TM) i7-4510U CPU @ 2GHz-2.60GHz, RAM memory of 8 GB, and a Windows 7 64-bit Operating System.

4.4.1 RANDOM instances

Instance definition Random instances are generated as in [30]. All functions in this class are to be maximized. They are of two different types.

- RANDOM SAME DEGREE. The number of variables n, the number of monomials m and the degree d are given as input. For each triplet (d, n, m), five functions are generated by randomly, uniformly and independently choosing the variables to include in each of the m monomials. All monomials have the same degree d. Their coefficients are drawn uniformly in the interval $[-10, 10]$. All instances in this class have small degree, namely, $d \in \{3, 4\}$.

- Random high degree. n and m are given as an input. Each of the m monomials is generated as follows: first, the degree d of the monomial is chosen from the set $\{2, \dots, n\}$ with probability 2^{1-d}. In this way, we capture the fact that a random polynomial is likely to have more monomials of lower degree than monomials of higher degree. Then, the variables and coefficient of the monomial are chosen as for the Random same degree instances. Again, we generate five instances for each pair (n, m). These instances are of much higer degree than the Random same degree instances. Their average degree will be reported hereunder.

Results Table 4.1 presents the results of our experiments on instances Random same degree. Each line displays averages over 5 instances. The first three columns specify parameters d, n, m. The fourth and fifth columns display the relative gaps between the optimal value of the integer programming problem on one hand, and the optimal value of the LP-relaxations of the plain standard linearization (P_{SL}), or of the standard linearization with 2-links (P_{SL}^{2links}) on the other hand. Columns 6 to 9 present the execution times of each of the four tested methods (>3600 is reported whenever no instance was solved to optimality), and columns 10 to 13 give the number of nodes of the branch & cut tree ("–" indicates that no instance was solved to optimality). If the time limit was reached for one or more instances, the unsolved instances are not taken into account in the averages. In addition, we write in parentheses () how many instances were solved to optimality in this case.

Table 4.1 shows that as a general trend, the addition of 2-links to the standard linearization is useful. Concerning the LP-relaxation bounds, we see that adding the 2-links always improves the bound associated with P_{SL}, by a gap percentage of 0.25% up to 8%. For execution times, it is clear that cuts of any type are helpful, since method no cuts is, in most cases, significantly worse than the other methods. In almost all cases, the fastest method is either user cuts or cplex & user cuts (plain cplex cuts is fastest only three times). For large instance sizes, cplex & user cuts is able to solve more instances than the competing methods. Looking at the number of nodes, it is interesting to notice that even when user cuts is the fastest method, it usually generates more nodes than either cplex or cplex & user cuts. This suggests that its performance is due to the smaller amount of time spent in generating the cuts and in solving the corresponding LPs. In contrast, it seems that the performance of cplex & user cuts is due to the fact that it produces smaller branch & cut trees. It may also be interesting to observe that the difficulty of the problems clearly increases with the density of the instances, that is, with the ratio $\frac{m}{n}$. This observation was also made by Buchheim and Rinaldi [30] (for slightly smaller values of m). Dense instances feature more interactions among monomials. This may increase the intrinsic difficulty of the instances and reduce the effect of adding the 2-links (or other cuts).

Table 4.1: Random same degree: computing times.

Instance			LP bounds: gap %		IP execution times (secs)				IP number of nodes			
d	n	m	P_{SL}	P_{SL}^{2links}	NO CUTS	USER	CPLEX	C & U	NO CUTS	USER	CPLEX	C & U
3	200	500	16.37	12.19	28.00	3.75	9.67	8.24	11557	1208	771	598
3	200	600	27.32	22.80	198.51 (3)	292.46 (4)	416.71	445.25	74409 (3)	93554 (4)	58004	60808
3	200	700	34.96 (2)	28.46 (2)	> 3600 (0)	433.78 (1)	1541.8 (2)	1426.72 (2)	– (0)	104504 (1)	128827 (2)	138958 (2)
3	400	800	4.51	3.49	3.65	2.57	7.46	6.68	423	251	210	135
3	400	900	9.31	7.93	502.41	243.58	104.52	87.75	65848	27489	6481	5405
3	400	1000	14.77 (3)	13.13 (3)	841.36 (1)	434.76 (1)	1334.96 (2)	1884.21 (3)	91939 (1)	37172 (1)	61899 (2)	84172 (3)
3	600	1100	2.78	2.32	14.09	9.88	16.07	14.52	1551	1121	891	626
3	600	1200	6.06	5.37	645.16	333.94	197.13	270.07	46502	25967	8616	12159
3	600	1300	10.17 (3)	9.15 (3)	> 3600 (0)	> 3600 (0)	2157.84 (2)	2234.61 (3)	– (0)	– (0)	84366 (2)	84655 (3)
4	200	350	16.50	11.23	6.50	3.20	9.98	5.89	2218	885	1468	722
4	200	400	22.25	15.84	663.89	207.28	341.68	108.36	262758	64383	70215	26307
4	200	450	28.72	20.81	999.44	324.28	664.39	382.55	285857	81764	98206	49588
4	200	500	35.09 (4)	24.84 (4)	2461.88 (1)	2268.63 (3)	1281.11 (1)	1340.34 (3)	586370 (1)	364125 (3)	143895 (1)	177784 (3)
4	400	550	4.37	3.26	36.97	17.10	14.76	11.6	6753	2743	1806	1318
4	400	600	8.15	5.91	58.79	13.86	63.1	20.19	7416	1458	5563	1184
4	400	650	10.22	7.72	177.74 (4)	681.06	348.79	514.13	22268 (4)	76797	25517	44714
4	400	700	12.25 (3)	8.92 (3)	1343.18 (2)	1179.95 (3)	602.68 (3)	329.05 (3)	130349 (2)	110322 (3)	36622 (3)	21418 (3)
4	600	750	1.54	1.28	3.42	3.05	6.15	5.89	278	234	222	142
4	600	800	2.59	2.14	16.54	12.08	18.37	15.5	1423	940	987	744
4	600	850	5.20	4.02	475.43 (4)	359.65	664.29	316.73	34555 (4)	28255	38502	21381
4	600	900	9.38 (4)	7.59 (4)	103.49 (1)	42.29 (1)	1526.84 (2)	1475.3 (4)	5865 (1)	2183 (1)	63850 (2)	61697 (4)

Table 4.2 presents the results of our experiments on instances Random high degree. The structure of the table is the same as for Table 4.1 except that d represents now the average degree of the five instances considered in each line.

The interpretation of the results is very similar to the interpretation of Table 4.1. The 2-link inequalities, by themselves, already improve the LP bound, the execution time and the size of the branch & cut tree, as compared to using no cuts. Method cplex cuts is usually more effective than user cuts here. However, cplex & user cuts still provides an improvement over cplex cuts, both in terms of execution time and size of the enumeration tree, and especially for dense instances. Observe that for this class of instances, we can handle much higher densities $\frac{m}{n}$ than for the Random same degree instances. This is again similar to the observations in Buchheim and Rinaldi [30], and might be due to the fact that many short monomials (of size 2) tend to appear in this type of instances and may reduce their complexity.

Table 4.2: RANDOM HIGH DEGREE: computing times.

Instance			LP bounds: gap %		IP execution times (secs)				IP number of nodes			
d (avg)	n	m	P_{SL}	P_{SL}^{2links}	NO CUTS	USER	CPLEX	C & U	NO CUTS	USER	CPLEX	C & U
12.6	200	600	12.21	10.15	10.42	8.08	7.15	5.81	5838	3595	398	368
11.2	200	700	12.73	10.73	78.72	30.12	34.74	28.17	35521	12821	3979	2997
11	200	800	18.99	16.10	748.15	254.81	118.55	111.64	257212	76479	10584	9936
13.6	200	900	27.29	23.72	889.37 (2)	690.72 (2)	1029.25	863.39	242729 (2)	135884 (2)	93124	75445
11.2	400	900	3.03	2.43	3.09	1.72	4.15	3.88	859	330	82	61
11	400	1000	3.50	2.82	19.56	6.77	8.87	8.44	4404	1396	286	259
11.4	400	1100	7.27	6.64	55.64 (4)	347.27	59.86	53.66	11289 (4)	61545	2970	2459
11.8	400	1200	7.04 (4)	6.45 (4)	256.80 (3)	117.35 (3)	254.46 (4)	147.80 (4)	42754 (3)	18483 (3)	13987 (4)	9123 (4)
13.8	600	1300	1.38	1.21	2.97	2.53	5.42	5.42	252	207	58	51
11.4	600	1400	3.86	3.57	294.03	238.87	124.30	135.38	36485	27234	5650	5516
12.2	600	1500	4.63	4.10	593.70	228.02	100.28	86.36	67493	24272	3942	3444
12.6	600	1600	5.00	4.53	1374.74 (4)	561.85 (4)	345.37	280.95	110267	47097	11063	8844

4.4.2 Vision instances

This class of instances is inspired from the image restoration problem, which is widely investigated in computer vision. The problem consists in taking a *blurred image* as an input and in reconstructing an original *sharp base image* based on this input. The interest of the vision instances, beside the practical importance of the underlying problem, is that they have a special structure for which linearization and related pseudo-Boolean optimization methods have proved to perform well (see, e.g., [79], [74], [52], [78]). It is out of the scope of the present chapter to work with real-life images: we will rely on a simplified version of the problem and on relatively small scale instances in order to generate structured instances and to evaluate the impact of the 2-link inequalities in this setting. Accordingly, we do not focus on the quality of image restoration (as engineers would typically do), but we devote more attention to the generation of relatively hard instances.

Input image definition An *image* is a rectangle consisting of $l \times h$ pixels. We model it as a matrix of dimension $l \times h$, where each element represents a pixel which takes value 0 or 1. An input *blurred image* is constructed by considering a *base image* and by applying a perturbation to it, that is, by changing the value of each pixel with a given probability. A base image is denoted as I^{base} and its pixels by p_{ij}^{base}. A blurred image is denoted by I^{blur} and its pixels by p_{ij}^{blur}.

We consider three base images, namely, TOP LEFT RECTANGLE, CENTRE RECTANGLE and CROSS (see Figure 8.1), with three different sizes 10×10, 10×15 and 15×15.

$$\begin{matrix}
1&1&1&1&1&0&0&0&0&0\\
1&1&1&1&1&0&0&0&0&0\\
1&1&1&1&1&0&0&0&0&0\\
1&1&1&1&1&0&0&0&0&0\\
1&1&1&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0
\end{matrix}$$

(a) TOP LEFT RECTANGLE

$$\begin{matrix}
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&1&1&1&1&1&0&0&0\\
0&0&1&1&1&1&1&0&0&0\\
0&0&1&1&1&1&1&0&0&0\\
0&0&1&1&1&1&1&0&0&0\\
0&0&1&1&1&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0
\end{matrix}$$

(b) CENTRE RECTANGLE

$$\begin{matrix}
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&1&1&0&0&0&0\\
0&0&1&1&1&1&1&1&0&0\\
0&0&1&1&1&1&1&1&0&0\\
0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0
\end{matrix}$$

(c) CROSS

Figure 4.1: Vision: base images of size 10×10.

We define three different types of perturbations that can be applied to a base image I^{base} in order to generate I^{blur}, namely:

- None: $p_{ij}^{blur} = p_{ij}^{base}$ with probability 1, $\forall (i, j) \in [l] \times [h]$.
- Low: $p_{ij}^{blur} = p_{ij}^{base}$ with probability 0.95, $\forall (i, j) \in [l] \times [h]$.
- High: $p_{ij}^{blur} = p_{ij}^{base}$ with probability 0.5, $\forall (i, j) \in [l] \times [h]$ with $p_{ij}^{base} = 0$.

Regarding the class High, note that changing the value of every pixel with probability 0.5 would lead to blurred images that are totally unrelated to the original base image; that is why we only apply the perturbation to the "white" pixels (originally taking value $p_{ij}^{base} = 0$) in this case.

Image restoration model The *image restoration model* associated with a blurred image I^{blur} is defined as an objective function $f(x) = L(x) + P(x)$ that must be minimized. The variables x_{ij}, for all $(i, j) \in [l] \times [h]$, represent the value assigned to each pixel in the output image. $L(x)$ is the linear part and models *similarity* between the input blurred image I_{blur} and the output. $P(x)$ is the nonlinear polynomial part and emphasizes *smoothness*: it aims at taking into account the fact that images typically consist of distinct objects, with pixels inside each object having similar colors, while pixels outside the objects have a different color. Much has been studied on the complex statistics of natural images, but we use here a simplified model.

- *Similarity*: $L(x) = a_L \sum_{i\in[l],j\in[h]}(p_{ij}^{blur} - x_{ij})^2$ minimizes the difference between the value of a pixel in the input image and the value that is assigned to the pixel in the output. Since $x_{ij} \in \{0, 1\}$, $L(x)$ is indeed linear. The coefficient of $L(x)$ is chosen as $a_L = 25$.

- *Smoothness*: $P(x)$ is a polynomial defined by considering 2×2 pixel windows $W_{ij} = \{x_{ij}, x_{i,j+1}, x_{i+1,j}, x_{i+1,j+1}\}$, for $i = 1, \dots, l-1$, $j = 1, \dots, h-1$. Smoothness is imposed by penalizing the objective function with a nonlinear monomial for each window W_{ij}. The more the assignment of variables in the window W_{ij} looks like a checkerboard, the higher the coefficient of the monomial, thus giving preference to smoother assignments. Table 4.3 provides the penalties used for each of the 16 assignments of values to a 2×2 window. So for example, the assignment of values $x_{ij} = x_{i,j+1} = 1$, $x_{i+1,j} = x_{i+1,j+1} = 0$ (third row in Table 4.3) gives rise to the monomial $30x_{ij}x_{i,j+1}(1 - x_{i+1,j})(1 - x_{i+1,j+1})$ in the objective function. We made the implementation choice of developing expressions of the type $30x_{ij}x_{i,j+1}(1 - x_{i+1,j})(1 - x_{i+1,j+1})$ into a multilinear function. Notice that one could also make the choice of not developing them and consider the function defined on the set of variables x_i and their complements $\bar{x}_i = 1 - x_i$, which possibly represents the structure of the underlying problem in a more accurate way and might have an impact on the results. We describe this point in more detail in Chapter 10.

The choice of coefficients in Table 4.3 and of the linear coefficient a_L was made by running a series of preliminary calibration tests aimed at finding a good balance between the importance given to smoothness and to similarity, so that the resulting instances are not too easy to solve.

Instance definition For each image size in $\{10 \times 10, 10 \times 15, 15 \times 15\}$ and for each base image, we have generated five instances, namely: one sharp image (the base image with perturbation type none), two blurred images with perturbation type low, and two blurred images with perturbation type high.

Table 4.3: VISION: variable assignments of 2×2 pixel windows and associated penalty coefficients.

Variable assignments								Coefficient
0 0	1 1							10
0 0	1 1							
0 0	0 0	0 1	1 0	1 1	1 1	1 0	0 1	20
0 1	1 0	0 0	0 0	1 0	0 1	1 1	1 1	
1 1	0 0	1 0	0 1					30
0 0	1 1	1 0	0 1					
1 0	0 1							40
0 1	1 0							

Notice that the difference between the five instances associated with a given size and a given base image is due to the input blurred image, which results from a random perturbation. This only affects the similarity term $L(x)$, while the smoothness model $P(x)$ remains the same for all instances of a given size.

Results Tables 4.4, 4.5 and 4.6 report the results obtained for images of size 10×10, 10×15 and 15×15, respectively. The structure of the tables is the same as for random instances, except for the first two columns, which respectively specify the base image and the perturbation applied. For the perturbation type NONE, we report the result obtained for a single instance. For the perturbation type LOW or HIGH, we report the averages for two instances.

We can see that, in all cases, the bounds derived from P_{SL} are very bad (ranging from 400% to 2000% above the optimal value). The bounds are significantly improved (by about 50%) when we add 2-links to the formulation (see column P_{SL}^{2links}), but they still remain very weak. Concerning execution times, methods NO CUTS and USER CUTS perform poorly and reach the time limit for almost every instance. A drastic improvement in computing times is achieved by CPLEX CUTS, which solves the easiest instances in just a few seconds and the most difficult ones in 110 seconds at most. Interestingly, however, a further significant improvement is obtained by CPLEX & USER CUTS, which solves all instances in less than 13 seconds. CPLEX & USER CUTS is in some cases up to ten times faster than CPLEX CUTS and always solves the problem at the root node, which suggests that its excellent performance is indeed due to the addition of the 2-links.

It is interesting to notice the major effect played by the structure of the instances. Indeed, vision instances have much worse LP gaps than random instances, and are much more dense (reaching $n = 225$ variables and $m = 1598$ terms for the 15×15 images). For the vision instances, we observe dramatic differences among the four solution methods that we have tested. Nevertheless, these instances turn out to be much easier to solve to optimality than random instances: it appears that the cuts generated by CPLEX and the 2-link inequalities are very complementary and provide remarkable benefits for the class of vision instances. Of course, the larger the size of the image, the more difficult the problem becomes. Perturbation types also have a big influence on complexity, since HIGH perturbation type instances are always harder to solve, as one might expect. Finally, the choice of base images does not seem to have any impact on the difficulty of the instances.

Table 4.4: VISION 10×10 ($n = 100, m = 668$): computing times.

Instance (10×10)		LP bounds: gap %		IP execution times (secs)				IP number of nodes			
Base image	Perturbation	P_{SL}	P_{SL}^{2links}	NO CUTS	USER	CPLEX	C & U	NO CUTS	USER	CPLEX	C & U
TOP LEFT RECT	NONE	584.07	296.70	> 3600	61.31	2.75	4.76	–	122037	0	0
TOP LEFT RECT	LOW	679.57	352.33	> 3600	105.91	4.70	0.74	–	220003	4	0
TOP LEFT RECT	HIGH	482.95	253.18	> 3600	> 3600	16.22	2.52	–	–	77.5	0
CENTRE RECT	NONE	1074.53	581.13	> 3600	304.89	6.05	0.81	–	625644	0	0
CENTRE RECT	LOW	1038.39	562.50	> 3600	494.95	7.41	0.94	–	1027936	0	0
CENTRE RECT	HIGH	525.25	277.48	> 3600	> 3600	11.44	1.48	–	–	0	0
CROSS	NONE	1989.29	1100	> 3600	206.25	3.25	0.95	–	418973	0	0
CROSS	LOW	1679.44	931.69	> 3600	669.79	8.49	1.63	–	1407712	0	0
CROSS	HIGH	379.48	192	> 3600	3062.91 (1)	10.15	1.45	–	5727483 (1)	3.5	0

Table 4.5: VISION 10×15 ($n = 150, m = 1033$): computing times.

Instance (10×15)		LP bound: gap %		IP execution times (secs)				IP number of nodes			
Base image	Perturbation	P_{SL}	P_{SL}^{2links}	NO CUTS	USER	CPLEX	C & U	NO CUTS	USER	CPLEX	C & U
TOP LEFT RECT	NONE	621.80	318.05	> 3600	> 3600	6.22	1.98	–	–	0	0
TOP LEFT RECT	LOW	749.58	396.66	> 3600	> 3600	15.50	2.04	–	–	3.5	0
TOP LEFT RECT	HIGH	480.87	251.87	> 3600	> 3600	38.49	3.35	–	–	42.5	0
CENTRE RECT	NONE	859.13	458.65	> 3600	> 3600	7.94	2.04	–	–	0	0
CENTRE RECT	LOW	1015.13	552.04	> 3600	> 3600	15.74	2.59	–	–	3.5	0
CENTRE RECT	HIGH	464.31	242.59	> 3600	> 3600	49.42	3.11	–	–	64.5	0
CROSS	NONE	1608.33	883.33	> 3600	> 3600	32.37	2.26	–	–	0	0
CROSS	LOW	1790.63	999.23	> 3600	> 3600	20.78	2.54	–	–	7.5	0
CROSS	HIGH	468.24	245.07	> 3600	> 3600	38.22	3.46	–	–	38.5	0

Table 4.6: VISION 15×15 ($n = 225, m = 1598$): computing times.

Instance (15×15)		LP bounds (gap %)		IP execution times (secs)				IP number of nodes			
Base image	Perturbation	P_{SL}	P_{SL}^{2links}	NO CUTS	USER	CPLEX	C & U	NO CUTS	USER	CPLEX	C & U
TOP LEFT RECT	NONE	660.90	340.26	> 3600	> 3600	19.5	3.49	–	–	0	0
TOP LEFT RECT	LOW	714.29	374.27	> 3600	> 3600	28.06	6.41	–	–	0	0
TOP LEFT RECT	HIGH	565.72	302.48	> 3600	> 3600	111.3	12.86	–	–	126.5	0
CENTRE RECT	NONE	698.13	366.75	> 3600	> 3600	30.12	4.71	–	–	0	0
CENTRE RECT	LOW	851.09	457.40	> 3600	> 3600	38.33	8.44	–	–	6.5	0
CENTRE RECT	HIGH	483.33	253.69	> 3600	> 3600	97.17	10.34	–	–	222	0
CROSS	NONE	1284.52	698.57	> 3600	> 3600	16.54	5.63	–	–	0	0
CROSS	LOW	1457.22	801.10	> 3600	> 3600	22.30	7.26	–	–	0	0
CROSS	HIGH	530.46	282.23	> 3600	> 3600	103.75	11.02	–	–	80	0

4.5 Conclusions

In this chapter, we have provided new results on the standard linearization technique, a well-known approach to the optimization of multilinear polynomials in binary variables. We have introduced the 2-link inequalities, a set of valid inequalities that express a relation between pairs of monomials, and that strengthen the LP-relaxation of the standard linearization. Our main result is that, for a function containing at most two nonlinear terms, the 2-links, together with the classical standard linearization inequalities, provide a perfect formulation of the standard linearization polytope P^*_{SL}.

For the general case of objective functions with more than two nonlinear terms, the 2-links are not enough to obtain a complete description of the standard linearization polytope. However, our computational experiments show that the 2-links are still helpful for various classes of instances. On one hand, the 2-links always improve the LP-relaxation bounds derived from the standard linearization. The improvement is much larger for the computer vision instances, which have very bad standard linearization bounds to begin with, than for unstructured random instances. On the other hand, our results show that 2-links can be very effective within a branch & cut framework. This is especially true when solving Vision instances, where the addition of 2-links to the pool of available cuts allows CPLEX to obtain the optimal solution without any branching and, as a consequence, significantly reduces the solution time. The magnitude of this effect is even more surprising given that the 2-links are rather simple inequalities and that they are in relatively small number (quadratic in the number of terms of the objective function).

There are many interesting open questions arising from our research. Of course, it is unlikely to obtain a complete description of the standard linearization polytope in the general case (unless P = NP). It remains however interesting to investigate whether there are other special cases of functions for which the 2-links provide a complete description of P^*_{SL}. A related question is to identify specially structured instances for which the impact of the 2-links is computationally significant, as is the case for our Vision instances. Finally, another natural question is whether it is possible to generate similar inequalities by establishing a link between three or more monomials, and whether these inequalities would further tighten the lower bounds and improve computational performance. A partial answer to this question has been provided by Del Pia and Khajavirad [47], where the authors define the *flower* inequalities, generalizing the 2-links to the case of several monomials such that a particular monomial has an intersection of at least two variables with the rest of monomials. The authors prove that flower inequalities, together with the standard linearization define a complete description for a certain acyclic hypergraphs, a result of a similar nature than the results presented in Chapter 3.

Part II

Quadratic reformulations

Chapter 5

Introduction to quadratizations

The second part of this thesis is concerned with quadratic reformulations of problem (1.1), also called quadratizations. A systematic study of quadratizations and their properties has been initiated by Anthony, Boros, Crama and Gruber [5], where a quadratization is formally defined as follows,

Definition 8. *Given a pseudo-Boolean function $f(x)$ on $\{0,1\}^n$, we say that $g(x,y)$ is a* quadratization *of f if $g(x,y)$ is a quadratic polynomial depending on x and on m* auxiliary variables *$y_1,\ldots,y_m$, such that*

$$f(x) = \min_{y\in\{0,1\}^m} g(x,y), \quad \forall x \in \{0,1\}^n. \tag{5.1}$$

It is clear that given a pseudo-Boolean function f and a quadratization g, minimizing f over $x \in \{0,1\}^n$ is equivalent to minimizing g over $(x,y) \in \{0,1\}^{n+m}$. Therefore, when a quadratization of f is available, the nonlinear minimization problem (1.1) can be reformulated as a quadratic one, without introducing additional constraints. Of course, the quadratic problem is still difficult to solve, but such reformulations attempt to draw benefit from the extensive literature and software for the quadratic case. Moreover, it is a well-known fact that every pseudo-Boolean function f admits a quadratization [96].

The starting point of the results presented in this part of the thesis is Definition 8. This is a very general definition encompassing many different quadratization methods, as opposed to the standard linearization considered in Part I, which is a precise linear reformulation procedure. Furthermore, not all quadratizations have the same properties or perform equally well when solving the resulting quadratic problems, and even the definition of which properties lead to "good" quadratizations is an open question. In the next paragraphs we highlight some interesting properties for quadratizations.

Small number of auxiliary variables A desirable property of a quadratization is to introduce a small number of auxiliary variables m, so that the size of the reformulation does not increase too much with respect to the size of the original problem. Anthony et al. [5] established tight upper and lower bounds on the number of variables that a quadratization requires for any pseudo-Boolean function. Concerning the lower bound, the authors prove that there exist pseudo-Boolean functions of n variables for which every quadratization must involve at least $\Omega(2^{\frac{n}{2}})$ auxiliary variables, independently of the procedure used to define the quadratization. As for the upper bound, the authors define a polynomial-time quadratization procedure which uses at most $O(2^{\frac{n}{2}})$ variables for any pseudo-Boolean function (in particular, for any function involving 2^n terms). Furthermore, when considering functions of fixed degree d, similar results are established, with a lower bound of $\Omega(n^{\frac{d}{2}})$ variables and an upper bound of $O(n^{\frac{d}{2}})$ variables.

Chapter 6 is concerned with the question of defining quadratizations using a small number of auxiliary variables for special classes of pseudo-Boolean functions. We establish upper and lower bounds on the number of auxiliary variables representing improvements of orders of magnitude, with respect to the best known bounds. The most remarkable result concerns the positive monomial (see also Section 5.1.2), for which the best upper bound published so far was linear in n, whereas our new upper bound is logarithmic. Additionally, a lower bound exactly matching the upper bound is provided. Logarithmic bounds are also established for parity, exact k-out-of-n and at least k-out-of-n functions, improving the best known bounds for these functions provided by Anthony, Boros, Crama and Gruber [4].

Submodularity Another interesting property for quadratic reformulations is the fact of having good optimization properties such as *submodularity*. Submodular functions are sometimes seen as the discrete analogous of convex functions [88]. They play a key role in optimization because problem (1.1), which is in general NP-hard, can be solved in polynomial time if f is submodular (see [62, 75, 100]).

A quadratic pseudo-Boolean function is submodular if, and only if, all quadratic terms have non-positive coefficients. This simple characterization gives a vague measure of distance from submodularity of a quadratization, which is the number of positive quadratic terms. Moreover, this property was used by Hammer [63] to give a simple polynomial time optimization algorithm for quadratic pseudo-Boolean functions based on an equivalence with the minimum cut problem in a graph, which has proven very useful in the computer vision literature (see Section 1.1.2 in Chapter 1). Large positive quadratic coefficients also seem to have a negative impact on computational performance [21, 52, 74].

It is clear that if a function f admits a submodular quadratization, then f itself must be submodular. However, it is difficult to identify which submodular functions can be quadratized keeping this property. In fact, Živný, Cohen and Jeavons [103] proved that there exist submodular pseudo-Boolean functions of degree four that do not admit a submodular quadratization.

The results in this thesis have not considered the fact of generating submodular quadratizations as an explicit objective. However, in Chapter 8 we briefly analyze the influence of the number of positive quadratic terms on the computational performance of the resolution of the corresponding quadratizations. It would be an interesting question to analyze this aspect in a more thorough way.

Exploiting structural properties of multilinear polynomials Another interesting property of quadratic reformulations is their ability of better exploiting structural properties of the original multilinear polynomials and the underlying applications.

An example of what we call structural property of a multilinear polynomial is the interaction between pairs of monomials having a non-empty intersection. Concerning linear reformulations, experimental results in Chapter 4 showed that adding information on interactions between monomials to the model, for example using the 2-link inequalities, can be very useful to solve instances inspired from the image restoration problem. These inequalities related the auxiliary variables corresponding to intersecting monomials using constraints. Concerning quadratizations, we will see in the remainder of this chapter that there exist quadratization procedures that associate separate sets of auxiliary variables to each monomial, while other procedures use common sets of auxiliary variables for different terms, which again might add information on intersecting monomials to the model. The computational experiments presented in Part III of this thesis clearly indicate that quadratizations using common sets of auxiliary variables for different monomials present a better computational performance for instances inspired from the image restoration problem.

From now on, the term *structure* of the original nonlinear functions will mostly refer to interactions between monomials. Nevertheless, there exist other structural properties that should be considered and that might have been already implicitly exploited by some reformulations in the experiments of Part III.

For example, many functions arising in problems from the fields of computer vision, signal processing or machine learning can be decomposed or separated as a sum of terms, each of which only depends on a subset of variables $X_C \subseteq \{x_1, \dots, x_n\}$

$$E(X) = \sum_{C \in \mathcal{C}} f_C(X_C), \tag{1.8}$$

for $\mathcal{C} \subseteq 2^{[n]}$ [57, 74, 92]. In fact, every pseudo-Boolean optimization problem can be written in the form (1.8), but this decomposition becomes interesting for example when the intersections between pairs of sets $C \in \mathcal{C}$ are small, when the subsets C have themselves small cardinality, when the functions f_C are the same for all $C \in \mathcal{C}$, or when the functions f_C have interesting properties such as submodularity.

Some structural aspects of multilinear functions might be more easily viewed in terms of the configuration of the edges of the hypergraph associated with the multilinear polynomial. One can for example consider multilinear functions associated with acyclic hypergraphs or with hypergraphs containing only cycles of short length, such as the functions considered in Chapters 3 and 4, or by Del Pia and Khajavirad [47].

Similar properties have been studied by Del Pia and Khajavirad [48], who considered cases in which the configuration of the monomials of the hypergraph allows the computation of the convex hull P^*_{SL} by decomposing the set of monomials into simpler subsets.

The submodularity property mentioned in the previous subsection is also an aspect of the structure of pseudo-Boolean functions, but as proved by Živný, Cohen and Jeavons [103] it cannot always be carried over to quadratic reformulations, even for functions of degree four.

To this point it is not clear how exactly the previous properties are taken into account by our reformulation methods, but experiments in Part III show that reformulations better modeling interactions between monomials result in very good computing times for non-random polynomials. Therefore, an interesting open question is to understand which of the previous structural properties are better exploited by our reformulations.

5.1 Literature review and contributions

The remainder of this chapter reviews the literature concerning the most relevant quadratization methods and positions the contributions of this thesis. Interestingly, much progress in the understanding of quadratizations from both a methodological and a computational point of view has been made in the field of computer vision, where these type of techniques perform especially well for problems such as image restoration [52, 56, 74, 80, 97].

All the procedures reviewed in this section heavily rely on the assumption made in previous chapters that a pseudo-Boolean function f is represented by its unique multilinear polynomial expression,

$$f(x_1,\dots,x_n) = \sum_{S\in\mathcal{S}} a_S \prod_{i\in S} x_i + \sum_{i\in[n]} a_i x_i, \tag{2.1}$$

where $[n] = \{1,\dots,n\}$, $\mathcal{S}$ is the set of subsets $S \in 2^{[n]}$ such that $a_S \neq 0$ and $|S| \geq 2$, and $\deg(f)$ denotes the degree of the polynomial representation of f.

5.1.1 Rosenberg's quadratization

Rosenberg [96] defined a general quadratization method applicable to every pseudo-Boolean function. Assuming that a pseudo-Boolean function f is given by its multilinear expression, the quadratic reformulation is achieved via an iterative procedure, where at each iteration a product x_ix_j is chosen from a highest-degree monomial f, and substituted by a new variable y_{ij}. A penalty term $M(x_ix_j - 2x_iy_{ij} - 2x_jy_{ij} + 3y_{ij})$ (where M is a large positive number) is added to the objective function, enforcing that $y_{ij} = x_ix_j$ at all optimal solutions. If there are other highest-degree terms not involving product x_ix_j, a different product can be substituted by a different variable in the same iteration, thus decreasing the degree of f by one unit at each iteration. This procedure is then repeated until obtaining a quadratic function. There are many ways of choosing the order of substituting products x_ix_j, and this choice can make a difference in the number of auxiliary variables and the properties of the resulting quadratic function. Rosenberg's quadratization can be applied to any f, proving that every pseudo-Boolean function admits a quadratization. Moreover, this transformation can be computed in an efficient way. An important drawback of this approach is that the penalty terms introduce many large positive coefficients, resulting in a highly non-submodular quadratization, even if the input function f was submodular, meaning that the resulting quadratic problem becomes difficult to minimize.

5.1.2 Termwise quadratizations

Termwise quadratizations are a family of quadratization techniques that have attracted much interest in the literature, relying on the intuitive idea that one can define a quadratization for f by providing a quadratization for each term in its multilinear representation (2.1) independently, using separate sets of auxiliary variables. More precisely, if $g_S(x, y_S)$ is a quadratization of the monomial $a_S \prod_{i\in S} x_i$, where the vectors of auxiliary variables y_S are distinct for all monomials, then $g(x, y) = \sum_{S\in 2^{[n]}} g_S(x, y_S)$ is a quadratization of f. Notice that, when relying on termwise quadratizations, we will always require at least t variables, where t is the number of terms. When t is large (say $\Omega(2^n)$), then the quadratization defined by Anthony et al. [5] provides a much better bound of $O(2^{\frac{n}{2}})$ auxiliary variables. In order to construct termwise quadratizations, it is necessary to understand quadratizations of positive monomials ($a_S > 0$) and of negative monomials ($a_S < 0$).

Negative monomials

The case of negative monomials, or monomials with a negative coefficient, is well understood. A simple expression to quadratize cubic negative monomials has been introduced by Kolmogorov and Zabih [80]. This expression was later extended to higher degrees by Freedman and Drineas [56], where a quadratization for a degree n negative monomial $N_n(x) = -\prod_{i=1}^n x_i$ is given by

$$N_n(x) = \min_{y\in\{0,1\}} (n-1)y - \sum_{i=1}^{n} x_i y. \tag{5.2}$$

This quadratization uses a single auxiliary variable, which is the best that one can expect for $n \geq 3$. Moreover, this quadratization is submodular because all quadratic terms have negative coefficients.

Anthony et. al proved in [5] that when attempting to quadratize a negative monomial using a single auxiliary variable and in such a way that there exists no other quadratization which is strictly smaller point by point, one only has two options, either using Freedman and Drineas' quadratization (5.2) or the following expression

$$s_n^+(x, y) = (n-2)x_n y - \sum_{i=1}^{n-1} x_i (y - \bar{x}_n). \tag{5.3}$$

Other more complex quadratizations for negative monomials are the *type-I* and *type-I transformations* introduded by Rother, Kohli, Feng and Jia [97]. These quadratizations are defined on the set of variables x_i and their complements $\bar{x}_i = 1 - x_i$. The type-I transformation transformation requires two auxiliary variables and the type-II transformation only one. However, these quadratizations are not submodular.

Positive monomials

Surprisingly, the case of monomials with a positive coefficient is much less understood than the case of negative monomials.

Defining a quadratization for the degree n positive monomial $P_n(x) = \prod_{i=1}^{n} x_i$ using Rosenberg's procedure requires the introduction of $n-2$ auxiliary variables. Another quadratization requiring $n-2$ auxiliary variables was given by Boros and Gruber [21], by noticing that a positive monomial of degree n can be substituted by a positive quadratic term plus a sum of negative monomials on the x_i variables and their complements $\bar{x}_i = 1 - x_i$. Using (5.2) for negative monomials, the authors obtain the following quadratization

$$x_1 \dots x_n - x_{n-1}x_n = -\sum_{i=1}^{n-2} \bar{x}_i \prod_{j=i+1}^{n} x_j = \min_{y \in \{0,1\}^{n-2}} \sum_{i=1}^{n-2} y_i \left(n - i - \bar{x}_i - \sum_{j=i+1}^{n} x_j \right). \tag{5.4}$$

Moreover, it is also observed in [21] that transformations type-I and type-II [97] for negative monomials can also be used to quadratize positive monomials with at least $n-2$ auxiliary variables, by using the first equality in (5.4).

More recently, Ishikawa [73, 74] defined the following quadratization for $P_n(x)$:

$$P_n(x) = \min_{y \in \{0,1\}^m} \sum_{i=1}^{m} y_i(c_{i,n}(-|x| + 2i) - 1) + \frac{|x|\,(|x| - 1)}{2}, \tag{5.5}$$

where $|x| = \sum_{i=1}^{n} x_i$, $m = \lfloor \frac{n-1}{2} \rfloor$ and

$$c_{i,n} = \begin{cases} 1, \text{if } n \text{ is odd and } i = m, \\ 2, \text{otherwise.} \end{cases}$$

Quadratization (5.5) uses $\lfloor \frac{n-1}{2} \rfloor$ auxiliary variables, and this is currently the best published upper bound on the number of variables required to define a quadratization for the positive monomial. Anthony et al. [4] gave an independent proof of the upper bound $\lfloor \frac{n-1}{2} \rfloor$, based on a representation result for arbitrary discrete functions. Interestingly, the quadratization of the positive monomial defined in [4] is identical to Ishikawa's for even values n but it is different for odd values of n.

In Chapter 6 we provide a quadratization for the positive monomial using $\lceil \log(n) \rceil - 1$ auxiliary variables. This upper bound improves Ishikawa's linear bound by orders of magnitude. We also prove that $\lceil \log(n) \rceil - 1$ is a lower bound on the number of variables required to quadratize the positive monomial, exactly matching the upper bound. The bounds for the positive monomial are derived from logarithmic bounds for more general classes of functions, such as symmetric, exact k-out-of-n, at least k-out-of-n or parity functions.

Several other quadratizations are defined in [74], by considering different versions of quadratization (5.5) where a subset of the x_i variables are substituted by their complements $\bar{x}_i$. Interestingly, type-I and type-II transformations can be obtained as a special case of Ishikawa's quadratization by considering the complements of a certain subset of the x_i variables.

As a last remark, Ishikawa's quadratization (5.5) introduces $\binom{n}{2}$ positive quadratic terms (the ones corresponding to S_2). Despite being a highly non-submodular quadratization, it provides very good computational results for computer vision instances [73].

5.1.3 Substituting common sets of variables

The previous section focused on quadratization methods based on the idea of defining a quadratic reformulation for each monomial separately. Several approaches of a different nature have also been considered in the literature. Let $\mathcal{S}$ be the set of monomials of the multilinear representation of a pseudo-Boolean function f. We have regrouped in this section several contributions that are fundamentally different but that at some level share the idea of identifying sets of variables that occur as *subterms* (subsets of variables) of several monomials in $\mathcal{S}$, and associating these subterms to the same auxiliary variable in all monomials containing them.

Rosenberg's quadratization could be perhaps considered the first method proposed within this class, depending on its implementation. More specifically, on could implement Rosenberg's procedure by substituting a product x_ix_j in a monomial by an auxiliary variable y^1_{ij}, and then substitute the same product x_ix_j in another monomial by an auxiliary variable y^2_{ij}, and so on (which would lead to a termwise quadratization). However one could also use the same auxiliary variable y_{ij} to substitute product x_ix_j in each of its occurrences in the original monomial set. In this case, Rosenberg's procedure would fall into the category of substituting common sets of variables. However, we have presented this procedure separately in Section 5.1.1 for historical reasons, because it was the first quadratization method proposed and because no explicit implementation is proposed in the original article [96].

Generating all possible covers of a monomial by two subsets

Buchheim and Rinaldi present in [30, 31] a quadratic reformulation of a multilinear optimization problem in binary variables that is based on the idea of, for every monomial S in the original monomial set $\mathcal{S}$, introducing an artificial binary variable $y_{\{I,J\}}$ for every possible decomposition of S into two subsets, that is, for every pair of subsets $I, J \in \mathcal{S}$ such that $S = I \cup J$. The introduction of these variables results in a quadratic reformulation on variables $y_{\{I,J\}}$. The method proposed in [30] is not aimed at solving this reformulation directly, but at using efficient separation techniques. The authors prove that the standard linearization polytope P^*_{SL} defined in Chapter 2 is isomorphic to a face of the polytope of the quadratic problem defined on variables $y_{\{I,J\}}$. This implies that a complete polyhedral description of P^*_{SL} can be derived from a complete polyhedral description of the polytope associated to the quadratic problem on $y_{\{I,J\}}$, which is isomorphic to the boolean quadric polytope for the unconstrained case. As a result, all the information and separation techniques of the boolean quadric polytope can be used in the context of polynomial binary optimization.

These results in [30] are applicable when the monomial set $\mathcal{S}$ is *reducible*, meaning that every monomial $S \in \mathcal{S}$ is the union of two other monomials in $\mathcal{S}$. The authors show that every monomial set can be made reducible and provide a heuristic algorithm to achieve this property. This heuristic algorithm relates to the idea of finding common subsets of variables in the original monomial set $\mathcal{S}$, and works as follows: as long as $\mathcal{S}$ is not reducible, determine two distinct variables $i, j \in \{1, \dots, n\}$ such that the cardinality of the set

$$\mathcal{P}(i, j) = \{S \in \mathcal{S} \mid i, j \in S \text{ and } S \neq S_1 \cup S_2 \text{ for any pair } S_1, S_2 \in \mathcal{S}\backslash\{S\}\}$$

is maximized. Add the sets $\{i, j\}$ and $S \setminus \{i, j\}$ to $\mathcal{S}$ for all $S \in \mathcal{P}(i, j)$, then iterate.

This algorithm attempts to minimize the number of sets $\{i, j\}$ to be added to the original monomial set. Finding the smallest set of monomials to add is an NP-hard problem even for degree three (see for example Observation 3 in [24]). In approaches based on substituting common sets of variables by the same auxiliary variable, heuristics relying on similar ideas might be useful. In fact, Chapter 7 presents two heuristic algorithms based on the same idea as Buchheim and Rinaldi's heuristic to make an instance reducible, with the objective of generating pairwise covers of small size.

Pairwise covers

A similar idea to the one of making instances reducible has been explored by Anthony et al. [5], where the authors introduce the notion of *pairwise covers*, which consists in, given a monomial set $\mathcal{S}$, defining a hypergraph $\mathcal{H}$ such that $S \in \mathcal{S}$ with $|S| \geq 3$, there are two sets $A(S), B(S) \in \mathcal{H}$ such that $|A(S)| < |S|$, $|B(S)| < |S|$, and $A(S) \cup B(S) = S$.

A key difference with Buchheim and Rinaldi's approach [30] is that, once a pairwise cover is defined, Anthony et al. define a quadratization of a pseudo-Boolean function f, by introducing auxiliary variables associated to elements in the pairwise cover $\mathcal{H}$, which is subsequently minimized. Therefore, it is interesting to define a pairwise cover of small cardinality, in order to introduce only a small number of many auxiliary variables. As mentioned previously, the problem of defining a pairwise cover of smallest size is NP-hard, even for polynomials of degree three [24], but a small pairwise cover can be defined heuristically.

Chapter 7 is concerned with defining quadratizations that heuristically minimize the number of auxiliary variables introduced quadratizations based on pairwise covers. Corresponding computational experiments are presented in Part III, where several termwise and pairwise covers based quadratizations and linearizations are compared. Among quadratization methods, it seems that quadratizations based on pairwise covers perform much better computationally, at least for non-random instances. Two possible explanations for this behavior are that quadratizations based on pairwise covers seem to be better at modeling interactions between monomials of the original nonlinear functions, and that they introduce less positive quadratic terms.

A hypergraph-based reduction

Fix, Boros, Gruber and Zabih [52] define a quadratization method based on the idea of substituting by an auxiliary variable a subset of variables that is common to several monomials in the monomial set of the original multilinear polynomial. In this case, a different quadratic expression is given depending on whether the monomials with a common subset have a positive or a negative coefficient. Theorems 4 and 5 state the results for positive and negative monomials, respectively.

Theorem 4 (Theorem 3.1 in [52]). *Let*

$$f(x) = \sum_{S \in \mathcal{S}} \alpha_S \prod_{j \in S} x_j$$

be a multilinear polynomial with monomial set $\mathcal{S} \subseteq 2^{[n]}$, such that $\alpha_S > 0$ for all $S \in \mathcal{S}$, and that there exists a common set $C \subseteq S$, for all $S \in \mathcal{S}$. Then,

$$f(x) = \min_{y \in \{0,1\}} \left(\sum_{S \in \mathcal{S}} \alpha_S \right) y \prod_{j \in C} x_j + \sum_{S \in \mathcal{S}} \left(\alpha_S \prod_{j \in S \setminus C} x_j - \alpha_S y \prod_{j \in S \setminus C} x_j \right) \tag{5.6}$$

Theorem 5 (Theorem 3.3 in [52]). *Let*

$$f(x) = \sum_{S \in \mathcal{S}} \alpha_S \prod_{j \in S} x_j$$

be a multilinear polynomial with monomial set $\mathcal{S} \subseteq 2^{[n]}$, such that $\alpha_S < 0$ for all $S \in \mathcal{S}$, and that there exists a common set $C \subseteq S$, for all $S \in \mathcal{S}$. Then,

$$f(x) = \min_{y \in \{0,1\}} \sum_{S \in \mathcal{S}} -\alpha_S \left(1 - \prod_{j \in C} x_j - \prod_{j \in S \setminus C} x_j \right) y \tag{5.7}$$

For positive monomials, the authors consider the case where C consists of a single variable. In this case, expression (5.6) substitutes each positive monomial S of degree d_S by a positive quadratic term, a positive term of degree $d_S - 1$ and a negative term of degree d_S, meaning that the degree of the positive terms is reduced by one. Expression (5.6) can then be applied repeatedly until obtaining positive terms of degree not larger than two. Notice however that for the case of negative monomials it is necessary that $|C| \geq 2$, because otherwise negative terms S of degree d_S would be replaced by other negative terms of the same degree, and the procedure might not terminate.

The choice of the common subset C might have a great impact on the performance of the method, in a similar way as one would like to add a smallest possible number of subsets to an instance for it to be reducible in Buchheim and Rinaldi's approach [30], or to introduce a small number of variables in Rosenberg's procedure [96] and in quadratizations based on pairwise covers [5].

In the implementation of this hypergraph-based reduction, the authors first reduce all positive terms to degree two by using Theorem 4, and negative terms are reformulated afterwards using Freedman and Drineas' quadratization (5.2). Fix et al. present the results of several computational experiments comparing their hypergraph-based reduction to the termwise quadratization using Ishikawa's expression for positive monomials (5.5) and to a different type of procedure, called *generalized roof duality*, which is not based on quadratic reformulations but on finding a "most submodular" approximation of a pseudo-Boolean function [76, 77]. The hypergraph-based reduction in [52] performs very well for several computer vision problems, in particular for those problems where the monomial set is *locally complete*, meaning that if a subset S is part of the initial monomial set, then all subsets $S' \subset S$ are also part of the initial monomial set.

Chapter 6

Compact quadratizations for pseudo-Boolean functions

This chapter presents upper and lower bounds on the number of auxiliary variables required to define a quadratization for several classes of specially structured functions, such as functions with many zeros, Symmetric functions, Exact k-out-of-n, At least k-out-of-n and Parity functions and Positive monomials. [1]

For Positive monomials, we provide a quadratization using only $m = \lceil \log(n) \rceil - 1$ auxiliary variables, which is a significant improvement with respect to Ishikawa's linear bound [73, 74], reducing the upper bound on the number of auxiliary variables by orders of magnitude. Moreover, we prove that one cannot quadratize the positive monomial using less than $m = \lceil \log(n) \rceil - 1$ variables, thus providing a lower bound that exactly matches the upper bound. This result is especially interesting in the context of termwise quadratizations.

Our quadratization of the Positive monomial is presented as a direct consequence of two more general results that define quadratizations for Exact k-out-of-n and At least k-out-of-n functions. Moreover, lower bounds on the number of variables required to quadratize Exact k-out-of-n and At least k-out-of-n functions and hence the Positive monomial are derived from a lower bound for an even more general class of functions, that we call Zero until k functions, and are characterized by taking value zero for all $x \in \{0, 1\}^n$ with $|x| = \sum_{i=1}^n x_i < k$.

In this chapter, we also define a quadratization for Symmetric functions using $O(\sqrt{n}) = 2\lceil \sqrt{n+1} \rceil$ variables, which matches the lower bound of $\Omega(\sqrt{n})$ variables that was given by Anthony et al. [4]. We also establish lower and upper bounds for the Parity function.

This chapter is structured as follows. Section 6.1 formally defines the considered functions, illustrates their relations and provides a summary of the bounds. The precise statements for lower and upper bounds are presented in Section 6.2 and in Section 6.3, respectively. Finally, Section 6.4 establishes some complementary lower bounds which are derived from a generalization of some of functions considered in the first sections. These last bounds are weaker than the bounds presented in Sections 6.2 and 6.3, but might nevertheless be useful

[1]The results presented in this chapter have been obtained together with Endre Boros and Yves Crama, and have been submitted for publication [19].

in other situations.

6.1 Definitions, notations and summary of contributions

Let us first define some notations. We assume throughout the chapter that $n \geq 1$. Let $x = (x_1, \dots, x_n) \in \{0,1\}^n$ and let $[n] = \{1, \dots, n\}$. The *Hamming weight* of x is $|x| = \sum_{i=1}^n x_i$, that is, the number of ones in x. We denote the *complement* of x by $\bar{x} = (\bar{x}_1, \dots, \bar{x}_n) = (1 - x_1, \dots, 1 - x_n)$. Notice that $|\bar{x}| = \sum_{i=1}^n \bar{x}_i = n - |x|$.

The original variables of the considered functions will be denoted by x, while auxiliary variables of quadratizations will be denoted by y and in some cases z.

Definition 9. Zero until *k* functions. *Let $0 \leq k \leq n$ be an integer. A* Zero until *k function $f : \{0,1\}^n \to \mathbb{R}$ is a pseudo-Boolean function such that $f(x) = 0$ if $|x| < k$, and such that there exists a point $x^* \in \{0,1\}^n$ with $|x^*| = k$ and $f(x^*) > 0$.*

It is easy to check that f is a Zero until k function if and only if, in its unique multilinear representation (1.2), all terms of degree smaller than k have coefficient zero, and there is one term of degree k with a positive coefficient. In fact, when this is the case, the coefficients of the multilinear representation are such that $a_S = f(x^S)$ for all $|S| \leq k$, where $x^S \in \{0,1\}^n$ is the characteristic vector of S, with components $x_i^S = 1$ for $i \in S$ and $x_i^S = 0$ for $i \notin S$.

Definition 10. Symmetric *functions.* *A pseudo-Boolean function $f : \{0,1\}^n \to \mathbb{R}$ is* symmetric *if its value only depends on $|x|$, that is, if there exists a function $r : \{0, \dots, n\} \to \mathbb{R}$ such that $f(x) = r(|x|)$.*

Definition 11. *The* Exact *k*-out-of-*n* *function.* *Let $0 \leq k \leq n$ be an integer. The* exact *k-*out-of*-n function is defined as*

$$f_{=k}(x) = \begin{cases} 1, & \textit{if } |x| = k \\ 0, & \textit{otherwise.} \end{cases} \tag{6.1}$$

Definition 12. *The* **At least *k*-out-of-*n* *function.*** *Let $0 \leq k \leq n$ be an integer. The* At least *k-*out-of*-n function is defined as*

$$f_{\geq k}(x) = \begin{cases} 1, & \textit{if } |x| \geq k \\ 0, & \textit{otherwise.} \end{cases} \tag{6.2}$$

Definition 13. *The* **Positive monomial.** *When $k = n$, the* exact *n-*out-of*-n function is equal to the* At least *n-*out-of*-n function. We call this function* Positive monomial *and denote it $P_n(x)$. Its polynomial expression is*

$$P_n(x) = \prod_{i=1}^n x_i \tag{6.3}$$

Figure 6.1: Compact quadratizations: relation between the considered classes of functions.

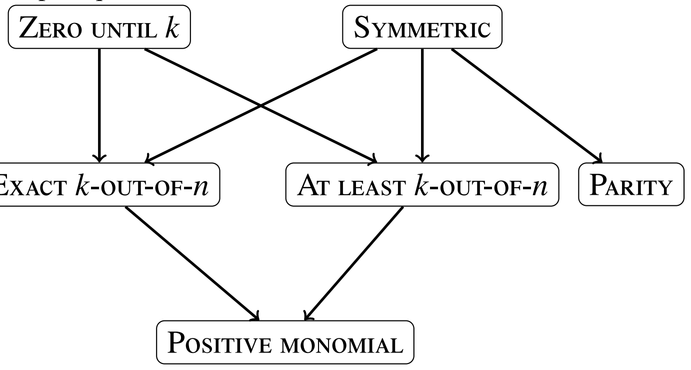


Table 6.1: Compact quadratizations: summary of lower and upper bounds.

Function	**Lower Bound**	**Upper Bound**
Zero until k	$\Omega(2^{\frac{n}{2}})$ for some function (see [5]) $\lceil \log(k) \rceil - 1$ for all functions	$O(2^{\frac{n}{2}})$ (see [5])
Symmetric	$\Omega(\sqrt{n})$ for some function (see [4])	$O(\sqrt{n}) = 2\lceil \sqrt{n+1} \rceil$
Exact k-out-of-n	$\max(\lceil \log(k) \rceil, \lceil \log(n-k) \rceil) - 1$	$\max(\lceil \log(k) \rceil, \lceil \log(n-k) \rceil)$
At least k-out-of-n	$\lceil \log(k) \rceil - 1$	$\max(\lceil \log(k) \rceil, \lceil \log(n-k) \rceil)$
Positive monomial	$\lceil \log(n) \rceil - 1$	$\lceil \log(n) \rceil - 1$
Parity	$\lceil \log(n) \rceil - 1$	$\lceil \log(n) \rceil - 1$

Definition 14. *The* **Parity** ***function.*** *The* parity *function $\pi_n(x)$ is defined as follows:*

$$\pi_n(x) = \begin{cases} 1, & \textit{if } |x| \textit{ is even,} \\ 0, & \textit{otherwise.} \end{cases} \tag{6.4}$$

Observe that Zero until k and Symmetric functions refer to *classes* of functions satisfying certain properties, while Exact k-out-of-n, At least k-out-of-n, Positive monomial and Parity refer to *uniquely defined* functions, for a given n and a given k.

Figure 6.1 schematizes the relations between the previously defined classes of functions. Classes on top of the figure are more general, and an arrow indicates whether a function is a particular case of another one.

Table 6.1 presents a summary of the values of the lower and upper bounds described in Section 6.2 and Section 6.3. (Here, and everywhere in the chapter we use the convention that $\log(0) = -\infty$.)

It should be noted that the meaning of the lower and upper bounds presented in Table 6.1 for the class of Symmetric functions, and the $\Omega(2^{\frac{n}{2}})$ bound for Zero until k functions are different from the rest. For example, for the Exact k-out-of-n function, the lower bound means that *we cannot quadratize the* Exact k-out-of-n *function with fewer than* $\max\{\lceil log(k) \rceil, \lceil log(n-k) \rceil\} - 1$ *variables* and the upper bound means that *we have defined a precise quadratization* *for the* EXACT k-OUT-OF-n *function using* $\max\{\lceil log(k)\rceil, \lceil log(n-k)\rceil\}$ *variables.* For SYMMETRIC functions, the lower bound means that *there exists at least one symmetric function requiring* $\Omega(\sqrt{n})$ *variables*, while the upper bound means that *all symmetric functions can be quadratized using* $O(\sqrt{n})$ *variables.* See Section 6.2.1 and Section 6.3.1 for precise statements.

Anthony et al. established in [4] an upper bound of $\lceil \frac{n}{2} \rceil$ variables for the AT LEAST k-OUT-OF-n function, and an upper bound of $\lfloor \frac{n}{2} \rfloor$ variables for the EXACT k-OUT-OF-n function. These bounds are significantly improved in this chapter, where we define tight upper and lower bounds that are logarithmic in n.

The lower bound for SYMMETRIC functions presented in Table 6.1 is given in [4]. In the same paper, a lower bound of $\Omega(\sqrt{n})$ variables is also proved for quadratizations of the PARITY function that are linear in the y variables (so called y-linear quadratizations), and an upper bound of $\lfloor \frac{n}{2} \rfloor$ variables is provided for the PARITY function. These earlier results for PARITY are also improved here, firstly because we do not restrict ourselves to y-linear quadratizations, and secondly because the number of necessary auxiliary variables is reduced from linear to logarithmic in the upper bound; the resulting lower and upper bounds are exactly equal.

For ZERO UNTIL k functions, we present two different lower bounds. The lower bound $\lceil \log(k) \rceil - 1$ is valid for *all* ZERO UNTIL k functions, while the lower bound $\Omega(2^{\frac{n}{2}})$ is valid for *almost all* ZERO UNTIL k functions. The corresponding result states that there exist ZERO UNTIL k functions requiring $\Omega(2^{\frac{n}{2}})$ auxiliary variables in any quadratization. The upper bound for ZERO UNTIL k functions is the same as for general pseudo-Boolean functions (see [5]). Indeed, the lower bound $\Omega(2^{\frac{n}{2}})$ implies that the order of magnitude of the upper bound cannot be less than this value; this is actually a rather natural observation, since for small values of k, most pseudo-Boolean functions are ZERO UNTIL k functions.

6.2 Lower bounds

This section formally states and proves the lower bounds on the number of auxiliary variables summarized in Table 6.1.

6.2.1 SYMMETRIC functions

A lower bound for the number of variables required to quadratize symmetric functions was established by Anthony et al. [4]. We state their theorem for completeness.

Theorem 6 (Theorem 5.3 in [4]). *There exist* SYMMETRIC *functions of n variables for which any quadratization must involve at least* $\Omega(\sqrt{n})$ *auxiliary variables.*

6.2.2 ZERO UNTIL k, EXACT k-OUT-OF-n, AT LEAST k-OUT-OF-n functions, and the POSITIVE MONOMIAL

We start this section with a lower bound for ZERO UNTIL k functions which is a direct extension of a result due to Anthony et al. [5].

Theorem 7. *For every fixed integer k, there exist* Zero until *k functions of n variables for which every quadratization must involve at least* $\Omega(2^{\frac{n}{2}})$ *auxiliary variables.*

Proof. The proof is analogous to the proof of Theorem 1 in [5], and we only briefly sketch it here. For any m, let V_m be the set of pseudo-Boolean functions of n variables which can be quadratized using at most m auxiliary variables. It was observed in [5] that V_m, viewed as a subset of the vector space of all pseudo-Boolean functions, is contained in a finite union of subspaces, each of dimension $\ell(n, m) = O(nm + n^2 + m^2)$. On the other hand, for any fixed k, the set of Zero until k pseudo-Boolean functions of n variables contains a subspace of dimension $2^n - O(n^k) = \Omega(2^n)$, namely, the subspace spanned by the monomials $\prod_{i\in S} x_i$ with $|S| > k$. It follows that, if m auxiliary variables are sufficient to quadratize every Zero until k function, then $\ell(n, m) = \Omega(2^n)$, and $m = \Omega(2^{\frac{n}{2}})$. $\square$

Observe that, as was the case for general pseudo-Boolean functions in [5], the bound in Theorem 7 actually holds for *almost all* Zero until k functions, in the sense that the set of Zero until k functions that require less than $\Omega(2^{\frac{n}{2}})$ auxiliary variables has Lebesgue measure zero, when compared to the whole space of Zero until k functions.

Furthermore, Theorem 7 implies that it is not possible to find an upper bound on the number of auxiliary variables to define a quadratization for Zero until k that is smaller than exponential in n – see Table 6.1. This also makes sense intuitively, since for a fixed k, the proportion of Zero until k functions among all pseudo-Boolean functions tends to 1 as n goes to infinity, and therefore, the same lower and upper bounds apply to Zero until k functions as to general pseudo-Boolean functions.

We next present another lower bound for Zero until k functions.

Theorem 8. *Assume that f is a* Zero until *k function with* $k \geq 1$ *and that* $g(x, y)$ *is a quadratization of f with m auxiliary variables. Then,*

$$m \geq \lceil \log(k) \rceil - 1.$$

Proof. Let us define

$$r(x) = \prod_{y\in\{0,1\}^m} g(x, y). \tag{6.5}$$

For every point $x \in \{0, 1\}^n$ with $|x| < k$, $f(x) = 0$ by definition of Zero until k functions. Also, since $g(x, y)$ is a quadratization of $f(x)$, there exists $y \in \{0, 1\}^m$ such that $g(x, y) = 0$, which implies that $r(x) = 0$ for all points x with $|x| < k$.

Moreover, since f is a Zero until k function we know that there exists a point $x^* \in \{0, 1\}^n$ such that $|x^*| = k$ and $f(x^*) > 0$, which implies that $g(x^*, y) > 0$ for all $y \in \{0, 1\}^m$, and hence $r(x^*) > 0$. Let $S^* = \{i \in [n] \mid x_i^* = 1\}$ with $|S^*| = k$.

In view of the observations following Definition 9, the unique multilinear expression of r can be written as

$$r(x) = \sum_{\substack{S\subseteq[n]\\ |S|\geq k}} a_S \prod_{i\in S} x_i \tag{6.6}$$

where $a_{S^*} = r(x^*) > 0$. Thus,

$$\deg(r) \geq k. \tag{6.7}$$

Now, the right-hand-side of (6.5) is a product of 2^m functions of degree two, meaning that

$$\deg(r) \leq 2^{m+1}, \tag{6.8}$$

which together with (6.7) implies that $m + 1 \geq \lceil \log(k) \rceil$. □

Notice the difference, of orders of magnitude, between the bounds given in Theorem 7, which is valid for almost all functions, and in Theorem 8, which is valid for all functions. The lower bound $\lceil \log(k) \rceil - 1$ given in Theorem 8 is rather weak for low values of k. However, for the particular case of the Positive monomial it leads to a lower bound that exactly matches the upper bound that we provide in Section 6.3.

Corollary 2. *The* Positive monomial $P_n(x)$ *is a* Zero until n *function, and therefore it cannot be quadratized using less than* $\lceil \log(n) \rceil - 1$ *auxiliary variables.*

For At least k-out-of-n and Exact k-out-of-n functions, the lower bound of Theorem 8 remains weak for small values of k.

Corollary 3. *For every fixed* $k \geq 1$, *the* At least k-out-of-n *function is a* Zero until k *function, and therefore it cannot be quadratized using less than* $\lceil \log(k) \rceil - 1$ *auxiliary variables.*

Corollary 4. *For every fixed* $k \geq 1$, *the* Exact k-out-of-n *function is a* Zero until k *function, and therefore it cannot be quadratized using less than* $\lceil \log(k) \rceil - 1$ *auxiliary variables.*

However, for Exact k-out-of-n functions we can derive a tighter lower bound on the number of auxiliary variables, with a difference of only one unit with respect to the upper bound that will be defined in Section 6.3.2, by relying on the following property:

Observation 1. *The* Exact k-out-of-n *function is such that*

$$f_{=k}(x_1, \ldots, x_n) = f_{=n-k}(\bar{x}_1, \ldots, \bar{x}_n).$$

Theorem 9. *Let* $k \geq 1$ *and assume that* $g(x, y)$ *is a quadratization of the* Exact k-out-of-n *function* $f_{=k}$ *with* m *auxiliary variables. Then,*

$$m \geq \max(\lceil \log(k) \rceil, \lceil \log(n-k) \rceil) - 1.$$

Proof. By Corollary 4, $m \geq \lceil \log(k) \rceil - 1$. By Observation 1, $f_{=k}(x_1, \ldots, x_n) = f_{=n-k}(\bar{x}_1, \ldots, \bar{x}_n)$, thus, by changing the names of the x variables we see that $h(x, y) = g(\bar{x}, y)$, viewed as a function of (x, y), is a quadratization of $f_{=n-k}(x_1, \ldots, x_n)$ using m auxiliary variables. Corollary 4 implies that any quadratization of $f_{=n-k}(x_1, \ldots, x_n)$ uses at least $\lceil \log(n-k) \rceil - 1$ auxiliary variables, thus $m \geq \lceil \log(n-k) \rceil - 1$, which completes the proof. □

Remark 8. *Observe that* $\max(\lceil \log(k) \rceil, \lceil \log(n-k) \rceil) - 1$ *is not a valid lower bound for all* Zero until k *functions. Indeed, consider for example a* Positive monomial *of degree* k, *seen as a function* f *of* n *variables:* $f(x_1, \ldots, x_n) = \prod_{i=1}^{k} x_j$, *where* n *is such that* $\lceil \log(n-k) \rceil > \lceil \log(k) \rceil$. *This is a* Zero until k *function. In Theorem 14 hereunder, we define a quadratization for this* Positive monomial *that uses* $\lceil \log(k) \rceil - 1$ *auxiliary variables. This shows that* $\lceil \log(n-k) \rceil - 1$ *cannot be a lower bound on the number of auxiliary variables for all* Zero until k *functions.*

6.2.3 The Parity function

In this section we prove that $\lceil \log(n) \rceil - 1$ is a lower bound on the number of variables required to define a quadratization for the Parity function.

Theorem 10. *Assume that $g(x, y)$ is a quadratization of the* Parity *function $\pi_n(x)$ with m auxiliary variables. Then,*

$$m \geq \lceil \log(n) \rceil - 1.$$

Proof. Let us define

$$r(x) = \prod_{y \in \{0,1\}^m} g(x, y). \tag{6.9}$$

Note that $\deg(r) \leq 2^{m+1}$, since $g(x, y)$ is quadratic.

Note also that $r(x) \geq 1$ whenever $|x|$ is even and $r(x) = 0$ whenever $|x|$ is odd. Therefore we can write

$$r(x) = \sum_{\substack{z \in \{0,1\}^n \\ |z| \textbf{ even}}} r(z) \prod_{i:z_i=1} x_i \prod_{i:z_i=0} (1 - x_i).$$

The sign of the coefficient of $\prod_{i=1}^n x_i$ in each of the above terms is $(-1)^n$. Thus, in the unique multilinear representation of $r(x)$, $\prod_{i=1}^n x_i$ has coefficient

$$(-1)^n \sum_{\substack{z \in \{0,1\}^n \\ |z| \text{ is even}}} r(z) \neq 0,$$

and consequently, $\deg(r) = n$. The inequality

$$n = \deg(r) \leq 2^{m+1}$$

follows, implying the claim. □

6.3 Upper bounds

This section defines the quadratizations for Symmetric pseudo-Boolean functions, for the Exact k-out-of-n and At least k-out-of-n functions, and for the Positive monomial and the Parity function that lead to the upper bounds displayed in Table 6.1.

6.3.1 Symmetric functions

Let $\mathbb{N}$ be the set of nonnegative integers and $Z = \{0, 1, \ldots, n\}$. Theorem 11 defines a quadratization of Symmetric functions using $2\lceil \sqrt{n+1} \rceil$ auxiliary variables.

Theorem 11. *Let $f(x_1,\dots,x_n)$ be a* Symmetric *pseudo-Boolean function such that $f(x) = r(|x|)$, with $r : \mathbb{N} \to \mathbb{R}$ and $r(k) = 0$ for $k > n$, by convention. Let $l = \lceil \sqrt{n+1} \rceil$, and choose $M \in \mathbb{R}$ such that $M > |r(k)|$ for all $k \in Z$. Then,*

$$\begin{aligned} g(x,y,z) = &\sum_{i=0}^{l-1}\sum_{j=0}^{l-1} r(il+j) y_i z_j \\ &+ 2M\left(1-\sum_{i=0}^{l-1} y_i\right)^2 + 2M\left(1-\sum_{j=0}^{l-1} z_j\right)^2 \\ &+ 2M\left(|x| - \left(l\sum_{i=0}^{l-1} i y_i + \sum_{j=0}^{l-1} j z_j\right)\right)^2 \end{aligned} \tag{6.10}$$

is a quadratization of f using $2\lceil \sqrt{n+1} \rceil = O(\sqrt{n})$ auxiliary variables y_i, z_i, $i = 0,\dots,l-1$.

Proof. Observe first that every integer $k \in Z$ has a unique representation $k = il + j$ with $0 \le i, j \le l-1$. So, for every $x \in \{0,1\}^n$, let us define integers $i(x)$ and $j(x)$ such that $|x| = i(x)\,l + j(x)$, $0 \le i(x) \le l-1$ and $0 \le j(x) \le l-1$ hold.

Let us then define auxiliary vectors $y^*, z^* \in \{0,1\}^l$ (with components indexed from 0 to $l-1$), such that

$$y_i^* = \begin{cases} 1 \text{ if } i = i(x), \\ 0 \text{ otherwise,} \end{cases}$$

$$z_j^* = \begin{cases} 1 \text{ if } j = j(x), \\ 0 \text{ otherwise.} \end{cases}$$

Let us observe next that due to the three terms involving M in (6.10), $g(x,y,z) < M$ if and only if $y = y^*$ and $z = z^*$. Due to the definition of the first term of g, in this case $g(x,y^*,z^*) = r(|x|) = f(x)$. □

The upper bound in Theorem 11 matches the order of magnitude of the lower bound given in Theorem 6. Interestingly, in combination with Lemma 5.1 of [4], it also implies that *every* pseudo-Boolean function can be quadratized using $O(2^{n/2})$ auxiliary variables, a result proved by another approach in [5].

Finally, observe that Theorem 11 can be generalized to a more general class of pseudo-Boolean functions, for which the value of a given $x \in \{0,1\}^n$ is determined by a weighted sum of the values of the components x_i instead of the the Hamming weight $|x| = \sum_{i=1}^n x_i$ of x. More precisely, given a linear function $L : \{0,1\}^n \to \{0,1,\dots,R\}$ and a function $r : \mathbb{N} \to \mathbb{R}$ with $r(k) = 0$ for $k > R$, consider the pseudo-Boolean function $f(x) = r(L(x))$. Theorem 11 holds for f by considering $l = \lceil \sqrt{R+1} \rceil$ and substituting $|x|$ by $L(x)$ in equation (6.10).

6.3.2 Exact k-out-of-n and At least k-out-of-n functions

Theorem 12 and Theorem 13 are the main results of this section. They define, respectively, a quadratization for the Exact k-out-of-n function and a quadratization for the At least k-out-of-n function using at most $\lceil \log(n) \rceil$ variables. The following observations will be useful in the proofs of Theorems 12 and 13.

Let us define the following sets

$$I^l_{even} = \{0, 2, \ldots, 2^l - 2\}, \tag{6.11}$$

and

$$I^l_{odd} = \{1, 3, \ldots, 2^l - 1\}. \tag{6.12}$$

Observation 2. *Observe that for all $l \geq 2$,*

$$I^l_{even} = \left\{ \sum_{i=1}^{l-1} 2^i y_i \mid (y_1, \ldots, y_{l-1}) \in \{0,1\}^{l-1} \right\}, \tag{6.13}$$

and

$$I^l_{odd} = \left\{ 1 + \sum_{i=1}^{l-1} 2^i y_i \mid (y_1, \ldots, y_{l-1}) \in \{0,1\}^{l-1} \right\}. \tag{6.14}$$

Observation 3. *Given integers $0 \leq |x| \leq n$, $0 \leq k \leq n$ and $l = \max(\lceil \log(k) \rceil, \lceil \log(n-k) \rceil)$, observe that*

$$0 \leq |x| - k + 2^l \leq 2^l - 1, \textit{ for } |x| < k, \tag{6.15}$$

and

$$0 \leq |x| - k - 1 \leq 2^l - 1, \textit{ for } |x| > k. \tag{6.16}$$

Proof. Note first that, by definition of l, $2^l \geq k$ and $2^l \geq n - k$ are satisfied for all k. The first inequality of (6.15) holds because $|x| \geq 0$ and $2^l \geq k$, and the second one holds because $|x| < k$. The first inequality of (6.16) holds because $k < |x|$, and the second one holds because $|x| - k \leq n - k \leq 2^l$. □

Let us define

$$A_k(x, y, z) = |x| - (k - 2^l)z - (k+1)(1-z) - \sum_{i=1}^{l-1} 2^i y_i,$$

where $l = \max(\lceil \log(k) \rceil, \lceil \log(n-k) \rceil)$, $z \in \{0,1\}$ and $y = (y_1, ..., y_{l-1}) \in \{0,1\}^{l-1}$.

Theorem 12. *For each integer $0 \leq k \leq n$, the function*

$$G_k(x, y, z) = \frac{1}{2} A_k(x, y, z)(A_k(x, y, z) - 1) \tag{6.17}$$

is a quadratization of the Exact k-out-of-n *function $f_{=k}$ using*

$$l = \max(\lceil \log(k) \rceil, \lceil \log(n-k) \rceil) \leq \lceil \log(n) \rceil$$

auxiliary variables $y \in \{0,1\}^{l-1}$ and $z \in \{0,1\}$.

Proof. Note first that $G_k(x, y, z) \geq 0$ for all (x, y, z) and for all k, since it is the half-product of two consecutive integers. Therefore, when $|x| \neq k$ we only have to show that there exists (y, z) such that $G_k(x, y, z) = f_{=k}(x) = 0$.

We consider three cases:

1. If $0 \leq |x| < k$, set $z = 1$ so that $A_k(x, y, 1) = |x| - k + 2^l - \sum_{i=1}^{l-1} 2^i y_i$. By Observation 3, $0 \leq |x| - k + 2^l \leq 2^l - 1$. Hence, using Observation 2 if $|x| - k + 2^l \in I^l_{odd}$, one can choose y such that $A_k(x, y, 1) - 1 = 0$, and if $|x| - k + 2^l \in I^l_{even}$, one can choose y such that $A_k(x, y, 1) = 0$.

2. If $k < |x| \leq n$, set $z = 0$ so that $A_k(x, y, 0) = |x| - k - 1 - \sum_{i=1}^{l-1} 2^i y_i$. By Observation 3, $0 \leq |x| - k - 1 \leq 2^l - 1$. Hence, using Observation 2 if $|x| - k - 1 \in I^l_{odd}$, one can choose y such that $A_k(x, y, 0) - 1 = 0$, and if $|x| - k - 1 \in I^l_{even}$, one can choose y such that $A_k(x, y, 0) = 0$.

3. Consider finally the case where $|x| = k$. When $z = 1$, we obtain $A_k(x, y, 1) = 2^l - \sum_{i=1}^{l-1} 2^i y_i \geq 2$, and hence $G_k(x, y, 1) \geq 1$. When $z = 0$, $A_k(x, y, 0) = -1 - \sum_{i=1}^{l-1} 2^i y_i \leq -1$, and hence again $G_k(x, y, 0) \geq 1$. The minimum value $G_k(x, y, z) = 1$ is obtained by setting either $z = y_i = 1$, or $z = y_i = 0$, for $i = 1, \ldots, l - 1$.

□

As announced earlier, the upper bound established in Theorem 12 almost perfectly matches the lower bound given in Theorem 9. Moreover, Theorem 12 provides as a corollary an upper bound on the number of variables required to obtain a quadratization of Symmetric functions.

Corollary 5. *If $f : \{0, 1\}^n \to \mathbb{R}$ is a* Symmetric *function, the value of which is strictly above its minimum value for at most d different Hamming weights $|x|$, then f can be quadratized with at most $d\,(\lceil \log(n) \rceil)$ variables.*

Proof. Let $r : \{0, 1, \ldots, n\} \to \mathbb{R}$ be such that $f(x) = r(|x|)$. Let α be the minimum value of f (and of r), and let $k_1, \ldots, k_d$ be the values of $|x|$ such that $f(x)$ (and $r(|x|)$) is larger than α. The result follows from Theorem 12 by observing that f can be expressed as

$$f(x) = \alpha + \sum_{i=1}^{d} (r(k_i) - \alpha)\, f_{=k_i}(x).$$

□

Let us now turn to the case of At least k-out-of-n functions.

Theorem 13. *For each integer $0 \leq k \leq n$, the function*

$$G_k(x, y, z) = \frac{1}{2} (A_k(x, y, z))\, (A_k(x, y, z) - 1) + (1 - z) \tag{6.18}$$

is a quadratization of the At least k-out-of-n *function* $f_{\geq k}$ *using*

$$l = \max(\lceil \log(k) \rceil, \lceil \log(n-k) \rceil) \leq \lceil \log(n) \rceil$$

auxiliary variables $y \in \{0,1\}^{l-1}$ *and* $z \in \{0,1\}$.

Proof. Note again that $G_k(x,y,z) \geq 0$ for all (x,y,z), because its first term is the half-product of two consecutive integers and $1 - z \geq 0$. Therefore, when $|x| < k$ we only have to show that there exists (y,z) such that $G_k(x,y,z) = f_{\geq k}(x) = 0$.

1. If $0 \leq |x| < k$, set $z = 1$. We obtain exactly the same expression as in the proof of Theorem 12 and the same argument holds.

2. If $|x| = k$ and $z = 1$, we obtain $A_k(x,y,1) = 2^l - \sum_{i=1}^{l-1} 2^i y_i \geq 2$, and hence $G_k(x,y,1) \geq 1$. If $z = 0$, we have that $G_k(x,y,0) \geq 2$. As in the proof of Theorem 12, we attain the minimum value $G_k(x,y,z) = 1$ by setting $z = y_i = 1$, for $i = 1, \ldots, l-1$.

3. Finally, let $|x| > k$. For $z = 1$,

$$G_k(x,y,1) = \frac{1}{2}\left(|x| - k + 2^l - \sum_{i=1}^{l-1} 2^i y_i\right)\left(|x| - k + 2^l - \sum_{i=1}^{l-1} 2^i y_i - 1\right).$$

Now, $\left(|x| - k + 2^l - \sum_{i=1}^{l-1} 2^i y_i\right) \geq 2$ because $2^l - \sum_{i=1}^{l-1} 2^i y_i \geq 2$ and $|x| - k$ is strictly positive. Hence, $G_k(x,y,1) \geq 1$.

For $z = 0$,

$$G_k(x,y,0) = \frac{1}{2}\left(|x| - k - 1 - \sum_{i=1}^{l-1} 2^i y_i\right)\left(|x| - k - 1 - \sum_{i=1}^{l-1} 2^i y_i - 1\right) + 1.$$

By Observation 3, $0 \leq |x| - k - 1 \leq 2^l - 1$. Hence, using Observation 2 if $|x| - k - 1 \in I^l_{odd}$, one can choose y such that $A_k(x,y,0) - 1 = 0$, and if $|x| - k - 1 \in I^l_{even}$, one can choose y such that $A_k(x,y,0) = 0$, thus achieving $G_k(x,y,0) = f_{\geq k}(x) = 1$ in both cases.

□

The upper bound in Theorem 13 is larger than the lower bound given in Corollary 3 when $k < \frac{n}{2}$, but the bounds are equal (up to one unit) for larger values of k.

6.3.3 The Positive monomial

In this section we define a quadratization using $\lceil \log(n) \rceil - 1$ auxiliary variables for the Positive monomial. Since the Positive monomial is a particular case of the Exact k-out-of-n function and of the At least k-out-of-n function (with $k = n$), Theorem 12 and Theorem 13 imply a slighlty weaker upper bound. Nevertheless, the stronger upper bound in Theorem 14 can be easily derived from the proofs of these theorems when $k = n$.

Theorem 14. *Let $l = \lceil \log(n) \rceil$. Then,*

$$g(x,y) = \frac{1}{2}(|x| + 2^l - n - \sum_{i=1}^{l-1} 2^i y_i)\,(|x| + 2^l - n - \sum_{i=1}^{l-1} 2^i y_i - 1) \tag{6.19}$$

is a quadratization of the positive monomial $P_n(x) = \prod_{i=1}^n x_i$ using $\lceil \log n \rceil - 1$ auxiliary variables.

Proof. This is a direct consequence of the proofs of Theorem 12 and Theorem 13: indeed, when setting $k = n$ it is easy to verify that we can always fix $z = 1$ in the quadratizations (6.17) and (6.18). □

As mentioned in the introduction of this chapter, Theorem 14 provides a significant improvement over the best previously known quadratizations for the Positive monomial, and the upper bound on the number of auxiliary variables precisely matches the lower bound presented in Section 6.2.

Remark 9. *Although the quadratization* (6.19)*, and the related expressions* (6.17) *or* (6.18)*, may seem somewhat mysterious, it is instructive to realize that they derive from rather simple modifications of a more natural result: indeed, the readers may easily convince themselves that, when $n = 2^l$, then*

$$g'(x,y) = (|x| - \sum_{i=0}^{l-1} 2^i y_i)^2$$

is a quadratization of P_n using $\log(n)$ *auxiliary variables. This result clearly highlights the underlying intuition, which is that $|x|$ can always be expressed as $|x| = \sum_{i=0}^{l-1} 2^i y_i$, except when $|x| = n$.*

When $n < 2^l$, the quadratization g' can be adjusted by fixing $2^l - n$ variables x_i to 1 in P_{2^l} and in g'. Moreover, the number of auxiliary variables can be marginally reduced by distinguishing between even and odd values of $|x|$. Altogether, this leads to Theorem 14.

Let us finally present the following family of quadratizations of the positive monomial, the best of which uses $\lceil \frac{n}{4} \rceil$ auxiliary variables (see [20] for a proof).

Theorem 15. *For all integers $n \geq 2$, if $\frac{n}{4} \leq m \leq \frac{n}{2}$, and $N = n - 2m$, then*

$$g(x,y) = \frac{1}{2}\,(|x| - 2|y| - (N-2)y_1)\,(|x| - 2|y| - (N-2)y_1 - 1) \tag{6.20}$$

is a quadratization of the positive monomial $P_n = \prod_{i=1}^n x_i$ using m auxiliary variables.

These quadratizations require a linear number of auxiliary variables but still improve the $\lfloor \frac{n-1}{2} \rfloor$ bound of (5.5). Notice that Ishikawa's quadratization (5.5) uses coefficients of absolute values varying approximately between 1 and n, while the absolute values of the coefficients in (6.19) and (6.20) vary roughly between 1 and n^2, which might result in functions containing coefficients of very different orders of magnitude, potentially inducing numerical problems when n is large. Moreover, note that $\lceil \log n \rceil - 1$ is equal to $\lceil \frac{n}{4} \rceil$ when $3 \leq n \leq 12$, so that the difference in the number of auxiliary variables only becomes relevant for very high degrees. Finally, the number of positive quadratic terms is also different in quadratizations (5.5), (6.20) and (6.19), especially for large degrees, which might also impact computational performance. All in all, the behavior of these different quadratizations in an optimization setting is unclear and should be computationally tested. Some preliminary results can be found in Chapter 8.

6.3.4 The Parity function

Theorem 16. *Let $l = \lceil \log(n) \rceil$. When n is even, the function*

$$g_e(x, y) = \left(|x| - n + 2^l - \sum_{i=1}^{l-1} 2^i y_i - 1 \right)^2 \tag{6.21}$$

is a quadratization of the Parity *function $\pi_n(x)$.*

When n is odd, the function

$$g_o(x, y) = \left(|x| - n + 2^l - \sum_{i=1}^{l-1} 2^i y_i \right)^2 \tag{6.22}$$

is a quadratization of the Parity *function $\pi_n(x)$.*

Both $g_e(x, y)$ and $g_o(x, y)$ use $\lceil \log(n) \rceil - 1$ auxiliary variables.

Proof. Assume that n is even. Then, $2^l - n$ is even and the parity of $|x|$ and $|x| - n + 2^l$ is the same.

If $|x|$ is odd, we only have to show that for each x, there exists a y such that $g_e(x, y) = 0$, because $g_e(x, y) \geq 0$ holds for all (x, y). Since $|x| - n + 2^l$ is odd, we have that $0 \leq |x| - n + 2^l \leq 2^l - 1$. Now, Observation 2 implies that for the right choice of y, $g_e(x, y) = 0$ is satisfied.

If $|x|$ is even, $g_e(x, y) \geq 1$ holds for all (x, y), because $|x| - n + 2^l - \sum_{i=1}^{l-1} 2^i y_i - 1$ is odd. Moreover, since $0 \leq |x| - n + 2^l \leq 2^l$ we obtain $\min_{y \in \{0,1\}^{l-1}} g_e(x, y) = 1$ with an appropriate choice of y.

When n is odd, the proof is analogous by considering I^l_{even} instead of I^l_{odd} and by noticing that $|x|$ and $|x| - n + 2^l$ have different parities. $\square$

6.4 Further lower bounds

This last section presents lower bounds on the number of variables required to define a quadratization for a class of pseudo-Boolean functions generalizing At least k-out-of-n, Exact k-out-of-n, Parity functions and a particular type of Symmetric functions. These functions are called d-sublinear, and are characterized by the fact that they take value zero everywhere, except on d hyperplanes. The main result of this section is Theorem 17, which gives a logarithmic lower bound on the number of auxiliary variables required to define a quadratization for d-SUBLINEAR functions. We choose to present these results in a separate section because the bounds derived from Theorem 17 are in general weaker than those presented in Section 6.2. Nevertheless, Theorem 17 may prove useful in other situations, and establishes an interesting link with results obtained by Linial and Radhakrishnan [87] in a different context (see also Alon and Füredi [2]).

Definition 15. *A pseudo-Boolean function $f : \{0,1\}^n \to \mathbb{R}$ is d-SUBLINEAR, if there exist linear functions $q_1, \dots, q_d$ such that $\prod_{j=1}^d q_j(x) = 0$ whenever $f(x) \neq 0$. We say that the linear functions $q_1, \dots, q_d$* dominate *f.*

We say that a pseudo-Boolean function is SUBLINEAR *if it is* 1-SUBLINEAR.

In other words, f is d-SUBLINEAR if every point x^* such that $f(x^*) \neq 0$ belongs to at least one of the hyperplanes $q_1(x) = 0, \dots, q_d(x) = 0$.

For a linear function $q = a_0 + \sum_{i=1}^n a_i x_i$, let $\beta(q)$ denote the number of variables with a non-zero coefficient in q, that is,

$$\beta(q) = |\{i \in \{1, \dots, n\} \text{ such that } a_i \neq 0\}|.$$

We are going to use the following lemma.

Lemma 2 (Lemma 2 in [87])**.** *Assume that $r : \{0,1\}^n \to \mathbb{R}$ is a* SUBLINEAR *function dominated by a linear function q, and assume that there exists a point x^* such that $r(x^*) \neq 0$. Then,*

$$\deg(r) \geq \frac{\beta(q)}{2}.$$

Theorem 17. *Assume that f is a d-SUBLINEAR function dominated by linear functions $q_1, \dots, q_d$, and that there exists $x^* \in \{0,1\}^n$ such that $f(x^*) > 0$, $q_1(x^*) = 0$ and $\prod_{j=2}^d q_j(x^*) \neq 0$. Then, the number m of auxiliary variables in any quadratization of f is such that*

$$2^{m+1} \geq \frac{\beta(q_1)}{2} - d + 1.$$

Proof. Let us define

$$r(x) = \prod_{j=2}^{d} q_j(x) \prod_{y \in \{0,1\}^m} g(x,y),$$

and note that

$$\deg(r) \leq d - 1 + 2^{m+1},$$

because r is a product of $d-1$ linear functions and 2^m quadratic functions.

Since g is a quadratization of f and $f(x^*) > 0$, we have $g(x^*, y) > 0$ for all $y \in \{0,1\}^m$. By assumption, $\prod_{j=2}^d q_j(x^*) \neq 0$, and hence $r(x^*) \neq 0$.

Moreover, r is SUBLINEAR dominated by the function q_1. Indeed, for points x with $r(x) \neq 0$, the definition of r implies that $q_j(x) \neq 0$ for all $j = 2, \dots, n$ and $g(x,y) \neq 0$ for all $y \in \{0,1\}^m$, thus $f(x) \neq 0$. But f is d-SUBLINEAR, thus $q_1(x) = 0$, which implies that $r(x)$ is SUBLINEAR.

The conditions of Lemma 2 are satisfied, therefore

$$\deg(r) \geq \frac{\beta(q_1)}{2},$$

which together with $\deg(r) \leq d - 1 + 2^{m+1}$ proves the claim. □

The bound of Theorem 17 is of interest when d is small. In particular, for the special case of Exact k-out-of-n functions (including the Positive monomial), we can take $d = 1$ and $q_1(x) = \sum_{i=1}^{n} x_i - k$. This yields a lower bound $\log(n) - 2$ that is only slightly weaker than the bound established in Theorem 9. More generally, for arbitrary symmetric functions, we have the following corollary.

Corollary 6. *If f is a* Symmetric *function, the value of which is strictly above its minimum value for at most d different Hamming weights $|x|$, where $d \leq \mu n + 1$ and $0 \leq \mu < \frac{1}{2}$, then the number m of auxiliary variables in any quadratization of f is such that*

$$m \geq \log(\frac{1}{2} - \mu) + \log(n) - 1.$$

Proof. As in the proof of Corollary 5, let α be the minimum value of f. Then, $h = f - \alpha$ is strictly positive for d values of $|x|$, and Theorem 17 applies directly to h with d dominating linear functions of the form $q_i(x) = \sum_{i=1}^{n} x_i - k_i$, for $i = 1, \ldots, d$. □

Again, this bound is relatively weak when compared, for instance, to the upper bound in Corollary 5, but it could prove useful in some cases.

6.5 Conclusions

In this chapter we have established new upper and lower bounds on the number of auxiliary variables required to define a quadratization for several classes of specially structured pseudo-Boolean functions defined on n binary variables. These bounds greatly improve the best bounds previously proposed in the literature. Most remarkably, the best upper bound published so far for the Positive monomial was linear in n, whereas our new upper bound is logarithmic. Moreover, for the Positive monomial and for the Parity function, we have also established lower bounds that exactly match the upper bounds.

Furthermore, we have provided logarithmic upper and lower bounds for Exact k-out-of-n and At least k-out-of-n functions. For Symmetric functions we have proved an upper bound of the order of $O(\sqrt{n})$, matching the order of magnitude of the best lower bound proposed in the literature.

For the more general class of Zero until k functions, we have established two different types of lower bounds, namely, a logarithmic bound in k which is valid for all functions in this class, and an exponential bound in n, which is valid for almost all Zero until k functions and which implies that the upper bounds available for general pseudo-Boolean functions are also applicable for this class.

Many additional questions arise from these results. From a theoretical point of view, it would be interesting to explore further generalizations and classes of pseudo-Boolean functions. For Exact k-out-of-n and At least k-out-of-n functions, the lower and upper bounds are of the same order of magnitude, but it would be nice to close the remaining unit gap. From an experimental perspective, it would be worth to examine the computational behavior of the different proposed quadratizations when applied to generic pseudo-Boolean optimization problems.

Chapter 7

Heuristic algorithms for small pairwise covers

The previous chapter provided, among other results, a quadratization using the smallest possible number of variables for positive monomials, allowing the definition of compact quadratizations for any multilinear polynomial using termwise procedures. A potential disadvantage of termwise procedures is that they might be worse at modeling structural properties of the original polynomial problem, such as the interaction between monomials, because each monomial is reformulated independently, using a different set of auxiliary variables. As reviewed in Chapter 5, quadratic reformulations relying on the idea of substituting subsets of variables that appear in more than one monomial by the same auxiliary variable have also been considered in the literature. This chapter focuses on an approach of this type, using the concept of *pairwise covers* introduced by Anthony et al. [5].

Definition 16 ([5])**.** *Given two hypergraphs $\mathcal{S}, \mathcal{H} \subseteq 2^{[n]}$, we say that $\mathcal{H}$ is a pairwise cover of $\mathcal{S}$ if, for every set $S \in \mathcal{S}$ with $|S| \geq 3$, there are two sets $A(S), B(S) \in \mathcal{H}$ such that $|A(S)| < |S|$, $|B(S)| < |S|$, and $A(S) \cup B(S) = S$.*

Pairwise covers split every higher-degree monomial into two subterms, and each of this subterms can be associated to an auxiliary variable to define a quadratization. Anthony et al. [5] defined a quadratization in this way. We state here a slightly weaker version of the corresponding theorem.

Theorem 18 (Theorem 4 in [5])**.** *If $\mathcal{S}, \mathcal{H} \subseteq 2^{[n]}$ are two hypergraphs such that $\mathcal{H} \subseteq \mathcal{S}$ and $\mathcal{H}$ is a pairwise cover of $\mathcal{S}$, then every pseudo-Boolean function of the form $f(x) = \sum_{S \in \mathcal{S}} a_S \prod_{j \in S} x_j$ is such that*

$$f(x) = \min_{y \in \{0,1\}^m} \sum_{S \in \mathcal{S}} a_S \prod_{j \in S} x_j + \sum_{H \in \mathcal{H}} b_H \left(y_H \left(|H| - \frac{1}{2} - \sum_{j \in H} x_j \right) + \frac{1}{2} \prod_{j \in H} x_j \right), \tag{7.1}$$

where $b_H = 0$ for $H \in \mathcal{S} \setminus \mathcal{H}$ and

$$\frac{1}{2} b_H = \sum_{\substack{S \in \mathcal{S}: \\ H \in \{A(S), B(S)\}}} \left(|a_S| + \frac{1}{2} b_S \right),$$

for $H \in \mathcal{H}$. Moreover, if for each monomial $S \in \mathcal{S}$ we set $y_{A(S)} = \prod_{j \in A(S)} x_j$ and $y_{B(S)} = \prod_{j \in B(S)} x_j$ and replace $\prod_{j \in S} x_j$ by $y_{A(S)} y_{B(S)}$, then (7.1) *defines a quadratization of f, using at most $|\mathcal{H}|$ auxiliary variables.*

Intuitively, when substituting subproducts $\prod_{j \in A(S)} x_j$ and $\prod_{j \in B(S)} x_j$ by auxiliary variables, the first summation of equation (7.1) is transformed into a quadratic function, and the second summation is a penalty imposing that auxiliary variables are equal to subproducts. Note that the hypothesis $\mathcal{H} \subseteq \mathcal{S}$ is necessary because otherwise we could have higher-degree terms in the second summation. The proof of Theorem 18 is based on the idea that $y^*_H = \prod_{j \in H} x_j$ minimizes the right-hand side of (7.1) for all $x \in \{0, 1\}^n$, and for this value, the second summation in the right-hand side vanishes.

Theorem 18 implies that every pseudo-Boolean function f admits a quadratization using at most as many auxiliary variables as the size of a pairwise cover of its monomial set $\mathcal{S}$, raising the question of finding the smallest size of a pairwise cover $\mathcal{H}$. This problem is NP-hard even for functions of degree three (see Observation 3 in [24]). Moreover, Theorem 18 provides a constructive procedure to define a quadratization for f, when a pairwise cover satisfying $\mathcal{H} \subseteq \mathcal{S}$ is given. However, no precise method to construct such a pairwise cover $\mathcal{H}$ has been defined in [5]. The remainder of this chapter describes three heuristic algorithms to generate pairwise covers of small size.

Notice that, given a monomial set $\mathcal{S}$, defining a pairwise cover $\mathcal{H}$ such that $\mathcal{H} \subseteq \mathcal{S}$ is very similar to the idea of making an instance *reducible* in [30], or to choosing an order of substituting products of variables $x_i x_j$ by a new variable y_{ij} in Rosenberg's procedure [96]. The main difference with [30] is that if the monomial set $\mathcal{S}$ is already reducible, our heuristic methods still aim at finding a pairwise cover of a small size and reformulate the problem using Theorem 18, while the approach in [30] applies a separation algorithm. Heuristics 2 and 3 are based on very similar ideas to the heuristic used in [30] to make an instance reducible. More precisely, Remark 1 in [5] states that a monomial set $\mathcal{S} \subseteq 2^{[n]}$ is *reducible* if it is a pairwise cover of itself, which implies hypothesis $\mathcal{H} \subseteq \mathcal{S}$. The hypergraph-based reduction by Fix et al. [52] is different in that sets of positive monomials and sets of negative monomials are reformulated with different expressions, while the sign of the coefficient of the monomials does not play any role here. Moreover, after reformulating sets of positive monomials, Fix et al. use termwise quadratizations for negative monomials.

7.1 Building bricks of the heuristic algorithms

The purpose of this chapter is to describe three heuristic algorithms to generate pairwise covers of small size. These algorithms have been implemented and tested in an experimental setting comparing them with termwise quadratizations and linearizations. The results of these experiments are presented in Chapter 8.

Given a pseudo-Boolean function f represented by its unique multilinear expression, let $\mathcal{S}$ denote its monomial set. The three heuristic methods described in this chapter rely on a set of ideas that we call *building bricks* and describe hereunder. A detailed pseudocode is described later in this chapter.

Building brick 1. *The idea behind the three heuristics is to identify sets of variables that appear as subsets of one or more monomials in the input monomial set $\mathcal{S}$. From now on, we call such sets of variables* subterms. *The heuristics differ in the* order *in which subterms are considered to generate the pairwise cover. Therefore, the first step of each heuristic is to construct priority list of subterms, that will be called* `PrioritySubterms`*, and it is the criterion used to construct this list that differentiates the three heuristics.*

Assume for a moment that we are given a priority list `PrioritySubterms` and that R is an element of this list. Consider a monomial $S \in \mathcal{S}$ such that $R \subset S$. Then R and $S \setminus R$ are subterms of S, and will correspond to $A(S)$ and $B(S)$ in the pairwise cover. We also say that S is covered by R and $S \setminus R$.

Building brick 2. *A second list of monomials called* `TermsToSplit` *is maintained during the execution of the algorithm. This list initially contains all terms of $\mathcal{S}$ having degree at least three. As the algorithm progresses, whenever an element $A(S)$ or $B(S)$ consists of more than three variables and is not part of the original monomial set $\mathcal{S}$, it is added to the end of the list* `TermsToSplit`*.*

Remember that Theorem 18 requires $\mathcal{H} \subseteq \mathcal{S}$. If $A(S)$ or $B(S)$ are not initially in $\mathcal{S}$, we can add them to the objective function with a zero coefficient, but this also means that we have to cover $A(S)$ or $B(S)$ if they contain more than three variables.

Building brick 3. *The algorithm is executed while* `PrioritySubterms` *and* `TermsToSplit` *are not empty. If the algorithm finished because* `PrioritySubterms` *is empty, this means that there might remain some elements in* `TermsToSplit` *with degree larger than three. A post-processing phase is then applied.*

Building brick 4. *Consider a general iteration of the algorithm where R denotes the current element of* `PrioritySubterms`*. In this general iteration, we examine all monomials in $S \in \mathcal{S}$, and whenever $R \subset S$, monomial S will be covered by $A(S) = R$ and $B(S) = S \setminus R$.*

1. *If $|A(S)| \geq 3$, then $A(S)$ is added to* `TermsToSplit` *(idem for $B(S)$).*

2. *If $|B(S)| \geq 2$, then $B(S)$ is added at the end of* `PrioritySubterms`*.*

The second point in building brick 4 means that we try to give some priority to substitute $B(S)$ in other monomials S' such that $B(S) \subset S'$. Indeed, we are trying to minimize the size of the pairwise cover, and since $B(S)$ is already in the pairwise cover, we would like to use it to cover other monomials.

Building brick 5. *During the execution of the algorithm, whenever we cover a monomial S by $A(S)$ and $B(S)$, we check whether one of the subsets $A(S)$ and $B(S)$ has been already considered in the pairwise cover. Assume for example that $A(S)$ already appeared in the pairwise cover of another monomial S' as $A(S')$, meaning that $A(S) = A(S')$. When associating auxiliary variables to the sets in the pairwise cover, we will impose $y_{A(S)} = y_{A(S')}$, because both auxiliary variables actually represent the same subterm.*

The choice of setting $y_{A(S)} = y_{A(S')}$ might seem obvious, but we specify it here because one could also introduce two different variables $y_{A(S)} \neq y_{A(S')}$ to represent the same subterm, which would induce a termwise procedure if applied to all monomials.

Building brick 6. *Finally, once a pairwise cover is constructed, the corresponding quadratization is implemented using equation* (7.1).

The remainder of this section presents the particular details of each heuristic. First, a general description of the algorithm and of the procedure used to create the priority list `PrioritySubterms` is described. Then a high-level pseudocode is provided for each heuristic, together with an illustrative example for Heuristics 2 and 3. Some parts of the pseudocode might be repetitive and can be skipped; they are described here for the sake of reproducibility, in order to offer a detailed implementation of the algorithms tested in Chapter 8.

7.2 Heuristic 1: "Split the first two variables from the rest"

Priority list creation and general description

Heuristic 1 is relies on a very straightforward idea, which consists in splitting the first two variables of each monomial S of size at least three from the rest of variables in S.

The list `PrioritySubterms` is not created explicitly in this algorithm, but there is an implicit priority, which is the order in which monomials are given in $\mathcal{S}$. More precisely, the first element in `PrioritySubterms` consists of the first two variables of the first monomial in $\mathcal{S}$, the second element in the list consists of the first two variables of the second monomial, and so on. As an implementation choice, and to obtain a simplest possible algorithm, point 2 in building brick 4 has not been applied in this case.

Pseudocode of Heuristic 1

Step 1: Create list `TermsToSplit` containing monomials with degree at least three

Step 2: Process list `TermsToSplit`

```
1. Retrieve the first element $S$ of TermsToSplit
    (a) Let $S_0$ be the set consisting of the first two variables of monomial $S$. Set $A(S) = S_0$, $B(S) = S \setminus S_0$.
    (b) If $|B(S)| \geq 2$, then add $B(S)$ to TermsToSplit.
2. If TermsToSplit is empty, then STOP. Else, iterate 1 and 2.
```

7.3 Heuristic 2: "Most popular intersections first"

Priority list creation and general description

Heuristic 2 is based on the intuition that a small pairwise cover can be defined by giving priority to substituting "popular" subterms first, where popular subterms are those appearing frequently as subterms of monomials in the input $\mathcal{S}$.

In Heuristic 2, `PrioritySubterms` is created by first computing all intersections between pairs of monomials and then counting the number of times where a certain subterm appears as an intersection. Therefore, the first element of `PrioritySubterms` is the subterm that occurs most often as an intersection, the second element is the subterm that occurs second most often as an intersection, and so on. Building bricks 1 to 6 are used in the implementation of this algorithm.

Illustrative example

We present a small example illustrating the main idea of Heuristic 2.

Example 1. *If* $\mathcal{S} = \{\{1234\}, \{1256\}, \{1278\}\}$*, the pairwise cover defined by*

- $A(\{1234\}) = \{12\}$*,* $B(\{1234\}) = \{34\}$
- $A(\{1256\}) = \{12\}$*,* $B(\{1256\}) = \{56\}$
- $A(\{1278\}) = \{12\}$*,* $B(\{1278\}) = \{78\}$

consists of four elements, namely $\{\{12\}, \{34\}, \{56\}, \{78\}\}$*, while another pairwise cover such as*

- $A(\{1234\}) = \{13\}$*,* $B(\{1234\}) = \{24\}$
- $A(\{1256\}) = \{15\}$*,* $B(\{1256\}) = \{26\}$
- $A(\{1278\}) = \{17\}$*,* $B(\{1278\}) = \{28\}$

consists of six elements, namely $\{\{13\}, \{24\}, \{15\}, \{26\}, \{17\}, \{28\}\}$*. In the first pairwise cover, we took into account that* $\{12\}$ *appears as an intersection of pairs of monomials three times, hence it is at the top of the priority list.*

Pseudocode of Heuristic 2

Step 1: Create the priority list `PrioritySubterms`

1. Compute all two-by-two intersections of monomials in $\mathcal{S}$ and count the number of times that a set of variables appears as a full intersection of two monomials in $\mathcal{S}$.
2. Create `PrioritySubterms` as follows: the first subterm is the set with most occurrences as a full intersection of two monomials in $\mathcal{S}$, the second subterm is the set with second most occurrences as a full intersection of two monomials in $\mathcal{S}$, and so on.

Step 2: Create list `TermsToSplit` containing monomials with degree at least three

Step 3: Process list `TermsToSplit` following the order of `PrioritySubterms`

1. Retrieve the first set R of `PrioritySubterms`
2. Create an empty list `SupersetsR`
3. For every S in list `TermsToSplit` such that $S \supset R$, add S to `SupersetsR`
4. For every S in list `SupersetsR`
   (a) Set $A(S) = R$ and $B(S) = S \setminus R$
   (b) Remove S from `TermsToSplit`
   (c) If $|A(S)| \geq 3$, then add $A(S)$ to `TermsToSplit`
   (d) If $|B(S)| \geq 2$, then add $B(S)$ at the end of `PrioritySubterms`
   (e) If $|B(S)| \geq 3$, then add $B(S)$ to `TermsToSplit`
5. If `PrioritySubterms` is empty or `TermsToSplit` is empty, then STOP. Else, iterate 1, 2, 3, 4.

Step 4: Post-processing If `TermsToSplit` is not empty, then Step 2 of Heuristic 1 is applied as a post-processing.

7.4 Heuristic 3: "Most popular pairs first"

Priority list creation and general description

Heuristic 3 is based on the same intuitive idea as Heuristic 2, where we tried to substitute "popular" subsets of variables first. Heuristic 2 gave priority to substituting popular intersections of pairs of monomials. However, it might well be possible that subsets of variables that are not full intersections but only strict subsets of intersections appear more frequently among the input set of monomials $\mathcal{S}$.

Heuristic 3 creates `PrioritySubterms` by counting the number of times that a pair of variables appears as a subterm of a monomial in the input. Shortly, we first generate all subsets of size two of $\{x_1, \ldots, x_n\}$ and then we count, for each pair, the number of times where it appears as a subterm of a monomial in the input $\mathcal{S}$.

This idea is essentially the same as the one used by Buchheim and Rinaldi to make an instance reducible [30]. The main difference is that Buchheim and Rinaldi use a separation algorithm afterwards, while we implement the quadratization described in Theorem 18 and solve the corresponding quadratic problem.

Steps 2, 3 and 4 in the pseudocode of Heuristic 3 are exactly the same as in Heuristic 2 and building bricks 1 to 6 are used in the implementation of this algorithm.

Illustrative example

The following example illustrates the difference between Heuristic 2 and Heuristic 3, and describes the motivation of Heuristic 3.

Example 2. *If* $\mathcal{S} = \{\{12348\}, \{12359\}, \{126789\}\}$*, all two-by-two intersections are given by* $\mathcal{I} = \{\{123\}, \{128\}, \{129\}\}$*, and each one occurs exactly one time. No matter which of the three sets in* $\mathcal{I}$ *we substitute first, we will cover the monomials in* $\mathcal{S}$ *using five elements. For example, if we start by substituting* $\{123\}$ *in every monomial of set* $\mathcal{S}$ *we would cover its elements as follows*

- $A(\{12348\}) = \{123\}$*,* $B(\{12348\}) = \{48\}$
- $A(\{12359\}) = \{123\}$*,* $B(\{12359\}) = \{59\}$
- $A(\{126789\}) = \{128\}$*,* $B(\{126789\}) = \{679\}$

However, the subset of variables $\{12\}$ *is common to every monomial in* $\mathcal{S}$*, but it is not a full intersection of any pair of monomials. If we substitute* $\{12\}$ *in every monomial of set* $\mathcal{S}$ *we would cover its elements using four subsets*

- $A(\{12348\}) = \{12\}$*,* $B(\{12348\}) = \{348\}$
- $A(\{12359\}) = \{12\}$*,* $B(\{12359\}) = \{359\}$
- $A(\{126789\}) = \{12\}$*,* $B(\{126789\}) = \{6789\}$*.*

Pseudocode of Heuristic 3

Step 1: Create the priority list `PrioritySubterms`

1. Generate all pairs (subsets of size two) of variables in $\{x_1, \ldots, x_n\}$ and count the number of times that pair appears as a subset of a monomial in $\mathcal{S}$.
2. Create list `PrioritySubterms` consisting of pairs of variables as follows: the first pair in the list is the pair with most occurrences as a subterm of a monomial in $\mathcal{S}$, the second pair is the pair with second most occurrences as a subterm of a monomial in $\mathcal{S}$, and so on.

Step 2: Create list `TermsToSplit` containing monomials with degree at least three

Step 3: Process list `TermsToSplit` following the order of `PrioritySubterms`

1. Retrieve the first set R of `PrioritySubterms`
2. Create an empty list `SupersetsR`

3. For every S in list `TermsToSplit` such that $S \supset R$, add S to `SupersetsR`
4. For every S in list `SupersetsR`
   (a) Set $A(S) = R$ and $B(S) = S \backslash R$
   (b) Remove S from `TermsToSplit`
   (c) If $|A(S)| \geq 3$, then add $A(S)$ to `TermsToSplit`
   (d) If $|B(S)| \geq 2$, then add $B(S)$ at the end of `PrioritySubterms`
   (e) If $|B(S)| \geq 3$, then add $B(S)$ to `TermsToSplit`
5. If `PrioritySubterms` is empty or `TermsToSplit` is empty, then STOP. Else, iterate 1, 2, 3, 4.

Step 4: Post-processing If `TermsToSplit` is not empty, then Step 2 of Heuristic 1 is applied as a post-processing.

Part III

Computational experiments

Chapter 8

Comparing linear and quadratic reformulations

This chapter presents the results of an extensive set of experiments evaluating computational aspects of some linearization and quadratization methods presented in Part I and Part II of this thesis. The structure of this chapter is as follows: Section 8.1 states the objectives and scope and provides some technical specifications, Section 8.2 describes the considered methods and instances, Section 8.3 provides some preliminary results using persistencies, Section 8.4 presents and analyzes the results of the experiments, and Section 8.5 presents some concluding remarks.

8.1 General considerations

Objectives The purpose of the experiments presented in this chapter is twofold.

A first objective is to compare the resolution times of several linear and quadratic reformulations of a nonlinear binary problem *when relying on a commercial solver* to solve the reformulated problems. The objective of these experiments is *not* to compete with the existing literature in terms of resolution times, but only to provide an indication of which reformulation method is most suitable when using generic commercial software.

A second objective is to check whether the behavior of the considered methods is consistent over different classes of instances or if, on the contrary, the structure of the instances influences the relative performance of the tested methods.

Dependence on the underlying solver We chose to solve all linear and quadratic reformulations using CPLEX 12.7, therefore the computational results of this chapter are strongly influenced by this choice. This dependence on the underlying solver complicates the task of providing a fair comparison of the reformulation methods themselves, and should be kept in mind at all times when interpreting the results. Hence, the results presented in this chapter are only considered preliminary; in order to obtain definitive results, one should run further tests with other linear and quadratic solvers and check whether our results are still valid. For example, it is very likely that customized convexification solvers, such as the one proposed by Billionnet, Elloumi and Lambert [13, 14] have a better performance when solving our quadratic reformulations than CPLEX 12.7. We consider this as a future research direction and collaboration possibility. We highlight here some drawbacks and advantages of using CPLEX 12.7 for linear and quadratic programs.

An important drawback is the fact that the linear solver of CPLEX 12.7 has been developed for many years, while its quadratic solver is more recent and probably less powerful than its linear solver. This could lead to a false impression that quadratizations perform worse than linearizations, while the actual reason could be the quadratic resolution method and not the reformulation itself. We tried to account for this drawback by carefully testing several parameter configurations related to the choice of method to solve quadratic problems. CPLEX 12.7 offers two possible resolution techniques: the first one consists in convexifying the quadratic problem by making the matrix of the quadratic terms positive-semidefinite, and the second one consists in applying the standard linearization to the quadratic problem. Bliek, Bonami and Lodi [15] describe the resolution algorithms for quadratic optimization problems in binary variables in CPLEX 12.6. To the best of our knowledge, the same approaches are used in CPLEX 12.7. We tried several parametrizations to control these two options, but using non-default parameter settings did not seem to improve the results. Therefore we let the default setting `let CPLEX choose` for all relevant parameters in the final set of experiments. The specific parameter settings that were tested are described in the paragraph "Technical specifications and parameter settings" hereunder.

A second disadvantage is the fact that we can only use CPLEX 12.7 as a "blackbox" solver, meaning that we do not have complete control over the algorithms but only over some parameters. Moreover, new versions of the software possibly contain additional parameters the fine-tuning of which could potentially result in a different computational behavior. Several remarks concerning this "blackbox" framework are made in Chapter 10.

A third important disadvantage is that the use of CPLEX 12.7 together with our current implementation does not allow us to exploit some interesting properties of the reformulations themselves. For quadratic reformulations, let us highlight pre-processing algorithms based on *persistencies*, a property allowing to fix a subset of variables in the quadratic problem to its optimal integer value by solving its continuous linear relaxation [65, 93, 95]. This approach has proven very useful in the computer vision literature [25, 52, 74, 79, 98]. We made a straightforward implementation of this idea which did not result in clear improvements in terms of computing times. A detailed analysis of these preliminary experiments is presented in Section 8.3.

Let us also mention some advantages. First, CPLEX 12.7 is a commercial and widely used solver, meaning that the public that could benefit from our analyses is potentially very large. In the second place, as explained by Bliek et al. [15], quadratic problems are in most cases solved by CPLEX 12.7 using the standard linearization, meaning that we end up comparing linearizations with linearized quadratizations. Finally, the fact of using the same solver allows us to fix the underlying software, meaning that we do not compare a commercial solver against for example a home-made implementation, which would produce hardly comparable results.

Technical specifications and parameter settings All experiments in this chapter were run on a PC with processor Intel(R) Core(TM) i7-4510U CPU @ 2GHz-2.60GHz, RAM memory of 8 GB, and a Windows 10 64-bit operating system, using CPLEX 12.7.

Monomials, polynomials and models were created in a self-made Java implementation and translated to CPLEX using the Java Concert Technology.

All problems have been solved using default pre-processing settings. Several parameter configurations were tested for the resolution of quadratic 0–1 problems. A short description of the parameters considered is presented here; a more detailed explanation can be found in the ILOG CPLEX Optimization Studio 12.7.0 website [72].

- `indefinite MIQP switch.` This switch has two values: 0 (off) and 1 (on, default). When this switch is on, CPLEX tries to make adjustments to the elements of the matrix of quadratic coefficients of the objective function to make it positive semi-definite or to improve numerical behavior, when the matrix is already positive semi-definite.

- `MIQCP strategy switch.` This switch selects how MIQCPs are solved and has three values: 0 (let CPLEX choose, default), 1 (solve a QCP node relaxation at each node) and 2 (solve an LP node relaxation at each node).

- `linearization switch for QP, MIQP.` This parameter controls whether quadratic terms of the objective function are linearized or not and can be set to three different values: -1 (let CPLEX choose, default), 0 (do not linearize quadratic terms) and 1 (force the linearization of quadratic terms).

An extensive set of preliminary experiments showed that the value of parameters `indefinite MIQP switch` and `MIQCP strategy switch` had little or no influence on computing times. Concerning parameter `linearization switch for QP, MIQP`, the default setting (-1) and the forced linearization (1) had similar computing times, while not allowing linearizations of quadratic terms (0) resulted in a large increase of computing times. As a conclusion all of the previous parameters were set to default values.

8.2 Description of the methods and the instances

This section describes the methods that have been implemented and compared in the computational experiments, and the classes of instances considered.

8.2.1 Methods

Linearizations

SL: standard linearization - branch & cut. The standard linearization of the original multilinear optimization problem is solved in binary variables using the branch & cut implementation of CPLEX 12.7, where the automatic cut generation mechanism is used, with default settings. The precise equations of the standard linearization that have been implemented are (2.5) and (2.6) in Chapter 2. This method will be referred to in the tables and figures as SL. (This method was called CPLEX CUTS in Chapter 4.)

SL-2L: standard linearization - branch & cut with 2-link inequalities. The standard linearization of the original multilinear optimization problem is solved in binary variables using the branch & cut implementation of CPLEX 12.7, with automatic cuts generation. In addition, the 2-link inequalities defined in Chapter 4 are added as a pool of user cuts. *All* non-redundant 2-link inequalities are included in the pool, where an inequality is not redundant when it is not implied by the standard linearization constraints. (See Section 4.1 of Chapter 4 for a precise description of the cases for which these inequalities are not redundant.) This method will be referred to in the tables and figures as SL-2L. (This method was called CPLEX & USER (C & U) CUTS in Chapter 4.)

SL-NoCuts: standard linearization - branch & bound. The standard linearization of the original multilinear optimization problem is solved in binary variables using plain branch & bound. This method is implemented by using the branch & cut algorithm of CPLEX 12.7 where all cut generation parameters are turned off. This method will be referred to in the tables and figures as SL-NoCuts. (This method was called NO CUTS in Chapter 4.)

SL-Only-2L: standard linearization - branch & cut with *only* 2-link inequalities. The standard linearization of the original multilinear optimization problem is solved in binary variables using branch & cut. However, the only inequalities considered in the branch & cut algorithm are the 2-link inequalities, which are added as a pool of user cuts. *All* non-redundant 2-link inequalities are included in the pool. The automatic cut generation mechanism of CPLEX 12.7 is turned off using the corresponding parameters. This method will be referred to in the tables and figures as SL-Only-2L. (This method was called USER CUTS in Chapter 4.)

Quadratizations: termwise

As explained in Chapter 6, in termwise quadratization procedures each monomial is reformulated as a quadratic expression independently, with a separate set of auxiliary variables for each monomial.

Our implementations of termwise quadratization methods differ only in the quadratic reformulation used for positive monomials of degree at least three. Monomials with degree one or two are not reformulated, and negative monomials $N_n(x) = -\prod_{i=1}^{n} x_i$ are always reformulated using Freedman and Drineas' quadratization.

$$N_n(x) = \min_{y \in \{0,1\}} (n-1)y - \sum_{i=1}^{n} x_i y. \tag{5.2}$$

The following quadratizations have been implemented for positive monomials $P_n(x) = \prod_{i=1}^n x_i$ of degree $n \geq 3$. (See also Chapters 5 and 6.)

Ishikawa's quadratization This method will be referred to in the tables and figures as Ishikawa. Positive monomials are reformulated using

$$P_n(x) = \min_{y \in \{0,1\}^m} \sum_{i=1}^{m} y_i(c_{i,n}(-|x| + 2i) - 1) + \frac{|x|\,(|x| - 1)}{2}, \tag{5.5}$$

where $|x| = \sum_{i=1}^n x_i$, $m = \lfloor \frac{n-1}{2} \rfloor$ and

$$c_{i,n} = \begin{cases} 1, \text{if } n \text{ is odd and } i = m, \\ 2, \text{otherwise.} \end{cases}$$

n/4 quadratization This method will be referred to in the tables and figures as n/4.
Positive monomials are reformulated using Theorem 15

$$P_n(x) = \min_{y \in \{0,1\}^m} \frac{1}{2}\,(|x| - Ny_1 - 2Y)\,(|x| - Ny_1 - 2Y - 1), \tag{6.20}$$

where we have fixed $m = \lceil \frac{n}{4} \rceil$ (in Theorem 15 m could take many values), $y \in \{0,1\}^m$, $Y = \sum_{j=2}^m y_j \leq m - 1$, and $N = n - 2m$.

logn-1 quadratization This method will be referred to in the tables and figures as logn-1.
Positive monomials are reformulated using Theorem 14

$$P_n(x) = \min_{y \in \{0,1\}^m} \frac{1}{2}(|x| + 2^l - n - \sum_{i=1}^{l-1} 2^i y_i)(|x| + 2^l - n - \sum_{i=1}^{l-1} 2^i y_i - 1) \tag{6.19}$$

where $l = \lceil \log(n) \rceil$ and $m = l - 1$, which is the smallest possible number of variables required to quadratize the positive monomial.

Quadratizations: pairwise covers

All quadratizations based on pairwise covers follow the general ideas and pseudocode described in Chapter 7.

PC1: Pairwise Covers Heuristic 1: "Split the first two variables from the rest" The pseudocode of this method is described in Section 7.2. This method will be referred to in the tables and figures as PC1.

PC2: Pairwise covers Heuristic 2: "Most popular intersections first" The pseudocode of this method is described in Section 7.3. This method will be referred to in the tables and figures as PC2.

PC3: Pairwise covers Heuristic 3: "Most popular pairs first" The pseudocode of this method is described in Section 7.4. This method will be referred to in the tables and figures as PC3.

8.2.2 Instances

The experiments presented this chapter were tested on four sets of instances, called Random same degree, Random high degree, Vision and Autocorrelated sequences. The instance generation method of Random same degree, Random high degree and Vision instances is the same as described in Chapter 4, with the difference that random instances were maximized in Chapter 4 whereas in this chapter they are minimized. Concerning the size of the instances, the results for Vision instances are given here for instances with image sizes 10×10, 15×15, 20×20, 25×25, and 30×30 while in Chapter 4 image sizes were 10×10, 10×15, and 15×15. For Random same degree and Random high degree only the easiest instances could be considered, because the quadratic solver of CPLEX 12.7 was too slow for the most difficult (denser) instances. We describe the instance generation methods of Chapter 4 again hereunder for completeness.

Random same degree and Random high degree instances

Random instances are generated as in [30]. In the experiments presented in Chapter 4, random instances were maximized, whereas in this chapter all instances are minimized. This is because Chapter 4 only considered the standard linearization, which can be applied to minimization and maximization problems, while in this chapter we consider also quadratization methods, the definition of which has been given *specifically* for minimization problems and should be adapted for the maximization case.

For Random same degree instances, the number of variables n, the number of monomials m and the degree d are given as input. Given a triplet (d, n, m), a polynomial is generated by randomly, uniformly and independently choosing the variables to include in each of the m monomials. All monomials have the same degree d. Their coefficients are drawn uniformly in the interval $[-10, 10]$. All instances in this class have small degree, namely, $d \in \{3, 4\}$.

For Random high degree instances, only n and m are given as an input. Each of the m monomials is generated as follows: first, the degree d of the monomial is chosen from the set $\{2, \ldots, n\}$ with probability 2^{1-d}. In this way, we capture the fact that a random polynomial is likely to have more monomials of lower degree than monomials of higher degree. Then, the variables and coefficient of the monomial are chosen as for Random same degree instances. These instances are of much higher degree than Random same degree instances. The degree of each instance will be specified in the result tables and figures.

$$\begin{array}{cccccccccc}
1&1&1&1&1&0&0&0&0&0\\
1&1&1&1&1&0&0&0&0&0\\
1&1&1&1&1&0&0&0&0&0\\
1&1&1&1&1&0&0&0&0&0\\
1&1&1&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0
\end{array}$$

(a) TOP LEFT RECTANGLE

$$\begin{array}{cccccccccc}
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&1&1&1&1&1&0&0&0\\
0&0&1&1&1&1&1&0&0&0\\
0&0&1&1&1&1&1&0&0&0\\
0&0&1&1&1&1&1&0&0&0\\
0&0&1&1&1&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0
\end{array}$$

(b) CENTRE RECTANGLE

$$\begin{array}{cccccccccc}
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&1&1&0&0&0&0\\
0&0&1&1&1&1&1&1&0&0\\
0&0&1&1&1&1&1&1&0&0\\
0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0
\end{array}$$

(c) CROSS

Figure 8.1: VISION: base images of size 10×10.

VISION instances

This class of instances is inspired from the image restoration problem, which is widely investigated in computer vision. The problem consists in taking a *blurred image* as an input and in reconstructing an original *sharp base image* based on this input. It is out of the scope of this thesis to work with real-life images: we will rely on a simplified version of the problem and on relatively small scale instances in order to generate structured instances and to compare the performance of our linear and quadratic reformulation methods. Accordingly, we do not focus on the quality of image restoration (as engineers would typically do), but we devote more attention to the generation of relatively hard instances.

Input image definition An *image* is a rectangle consisting of $l \times h$ pixels. We model it as a matrix of dimension $l \times h$, where each element represents a pixel which takes value 0 or 1. An input *blurred image* is constructed by considering a *base image* and by applying a perturbation to it, that is, by changing the value of each pixel with a given probability. A base image is denoted as I^{base} and its pixels by p_{ij}^{base}. A blurred image is denoted by I^{blur} and its pixels by p_{ij}^{blur}.

We consider three squared base images, namely, TOP LEFT RECTANGLE (TLR), CENTRE RECTANGLE (CR), and CROSS (CX) (see Figure 8.1), with sizes 10×10, 15×15, 20×20, 25×25 and 30×30.

We define three different types of perturbations that can be applied to a base image I^{base} in order to generate I^{blur}, namely:

- NONE: $p_{ij}^{blur} = p_{ij}^{base}$ with probability 1, $\forall (i, j) \in [l] \times [h]$.
- LOW: $p_{ij}^{blur} = p_{ij}^{base}$ with probability 0.95, $\forall (i, j) \in [l] \times [h]$.
- HIGH: $p_{ij}^{blur} = p_{ij}^{base}$ with probability 0.5, $\forall (i, j) \in [l] \times [h]$ with $p_{ij}^{base} = 0$.

Regarding the class HIGH, note that changing the value of every pixel with probability 0.5 would lead to blurred images that are totally unrelated to the original base image; that is why we only apply the perturbation to the "white" pixels (originally taking value $p_{ij}^{base} = 0$) in this case.

Image restoration model The *image restoration model* associated with a blurred image I^{blur} is defined as an objective function $f(x) = L(x) + P(x)$ that must be minimized. The variables x_{ij}, for all $(i, j) \in [l] \times [h]$, represent the value assigned to each pixel in the output image. $L(x)$ is the linear part and models *similarity* between the input blurred image I_{blur} and the output. $P(x)$ is the nonlinear polynomial part and emphasizes *smoothness*: it aims at taking into account the fact that images typically consist of distinct objects, with pixels inside each object having similar colors, while pixels outside the objects have a different color. Much has been studied on the complex statistics of natural images, but we use here a simplified model.

- *Similarity*: $L(x) = a_L \sum_{i\in[l],j\in[h]}(p_{ij}^{blur} - x_{ij})^2$ minimizes the difference between the value of a pixel in the input image and the value that is assigned to the pixel in the output. Since $x_{ij} \in \{0, 1\}$, $L(x)$ is indeed linear. The coefficient of $L(x)$ is chosen as $a_L = 25$.

- *Smoothness*: $P(x)$ is a polynomial defined by considering 2×2 pixel windows $W_{ij} = \{x_{ij}, x_{i,j+1}, x_{i+1,j}, x_{i+1,j+1}\}$, for $i = 1, \dots, l-1$, $j = 1, \dots, h-1$. Smoothness is imposed by penalizing the objective function with a nonlinear monomial for each window W_{ij}. The more the assignment of variables in the window W_{ij} looks like a checkerboard, the higher the coefficient of the monomial, thus giving preference to smoother assignments. Table 8.1 provides the penalties used for each of the 16 assignments of values to a 2×2 window. So for example, the assignment of values $x_{ij} = x_{i,j+1} = 1$, $x_{i+1,j} = x_{i+1,j+1} = 0$ (third row in Table 8.1) gives rise to the monomial $30x_{ij}x_{i,j+1}(1 - x_{i+1,j})(1 - x_{i+1,j+1})$ in the objective function. We made the implementation choice of developing expressions of the type $30x_{ij}x_{i,j+1}(1-x_{i+1,j})(1-x_{i+1,j+1})$ into a multilinear function. Notice that one could also make the choice of not developing them and consider the function defined on the set of variables x_i and their complements $\bar{x}_i = 1 - x_i$, which possibly represents the structure of the underlying problem in a more accurate way and might have an impact on the results. We describe this point in more detail in the conclusions of this chapter.

The choice of coefficients in Table 8.1 and of the linear coefficient a_L was made by running a series of preliminary calibration tests aimed at finding a good balance between the importance given to smoothness and to similarity, so that the resulting instances are not too easy to solve.

Instance definition For each image size in $\{10\times10, 15\times15, 20\times20, 25\times25, 30\times30\}$ and for each base image, we have generated five instances, namely: one sharp image (the base image with perturbation type NONE), two blurred images with perturbation type LOW, and two blurred images with perturbation type HIGH.

Notice that the difference between the five instances associated with a given size and a given base image is due to the input blurred image, which results from a random perturbation.

Table 8.1: VISION: variable assignments of 2×2 pixel windows and associated penalty coefficients.

Variable assignments								Coefficient
0 0	1 1							10
0 0	1 1							
0 0	0 0	0 1	1 0	1 1	1 1	1 0	0 1	20
0 1	1 0	0 0	0 0	1 0	0 1	1 1	1 1	
1 1	0 0	1 0	0 1					30
0 0	1 1	1 0	0 1					
1 0	0 1							40
0 1	1 0							

This only affects the similarity term $L(x)$, while the smoothness model $P(x)$ remains the same for all instances of a given size.

AUTOCORRELATED SEQUENCES **instances**

This class of instances is inspired from a problem in statistical mechanics, modeling the phenomenon of "dynamical glass transition", which is the abrupt increase of orders of magnitude of the viscosity of a supercooled liquid in such a way that it becomes solid almost instantly. The set of instances has been directly downloaded from `http://polip.zib.de/autocorrelated_sequences/`. These are instances from the model given by Liers et al. [86], which is based on the Ising model and on the principle of spin-spin energy minimization. The authors rely on branch and bound and simulated annealing algorithms for the minimization of the energy function.

It is out of the scope of this thesis to give a detailed rigorous description of the model, we only give here an intuitive explanation as to why this problem can be modeled as a minimization in binary variables. The objective function of the model is to minimize the energies between pairs of interacting spins in a material. Spins can have two positions, up or down, which are modeled using binary variables. The energy corresponding to a pair of spins is minimized whenever two spins take the same value.

Bernasconi [12] showed that finding spin configurations with low energy states is equivalent to the problem of finding *low autocorrelation binary sequences*. Thus, the Hamiltonian, or energy function, as given in [86] is

$$\mathcal{H}_D = \mathcal{N}_{N,R} \sum_{\vec{x} \in \Lambda} \left[\sum_{d=1}^{\max d \in \mathcal{R}(\vec{x})} \left(\sum_{\substack{\text{couples in } \mathcal{R}(\vec{x}) \\ \text{at distance } d}} \sigma_j \sigma_k \right)^2 \right], \tag{8.1}$$

where D is the dimension, $\Lambda \subset \mathbb{Z}^D$ is a finite volume of cardinality N, $\mathcal{R}(\vec{x})$ is a hypercube of

Table 8.2: Computational experiments: summary of methods and instances.

	Methods / Instances	Random same	Random high	Vision	Autocorr. seq.
Linear	SL	✓	✓	✓	✓
	SL-2L	✗	✗	✓	✓
	SL-NoCuts	✗	✗	✗	✓
	SL-Only-2L	✗	✗	✗	✓
Quadratic	Ishikawa	✓†	✓	✓*,†	✓†
	n/4	✓†	✓	✓*,†	✓†
	logn-1	✓†	✓	✓*,†	✓†
	PC1	✓	✓	✓	✓
	PC2	✓	✓	✓	✓
	PC3	✓	✓	✓	✓

size R^D centered around site $\vec{x}$ and $\mathcal{N}_{N,R}$ is a normalization constant. Factors $\left(\sum_{\substack{\text{couples in } \mathcal{R}(\vec{x}) \\ \text{at distance } d}} \sigma_j \sigma_k\right)^2$ represent autocorrelations between localized pairs of spins σ_j and σ_k.

All instances are of degree four, have small number of variables, ranging between 20 to 60, and very large number of terms, ranging between 207 and 52 575 (see Table 8.19 for a detailed description of the instances). As it will be shown in Section 8.4.4, the instances in this class are particularly difficult to solve for all of our methods. In fact, we are only able to provide some preliminary results for a very small subset of these instances. However, we find it worth mentioning them here, because of the very reason that they represent a challenge for our methods. Moreover, in the original paper their inherent difficulty is explained by stating that the models present *deep minima separated by extensive energy barriers, making the search for global minima (low autocorrelation binary sequences) a very difficult task* [86].

8.2.3 Short summary

As a summary of this section, four classes of instances and ten reformulation methods have been described. The four linearization methods use the standard linearization as a model, and they differ by the sets of valid inequalities added in a branch & cut framework. From the six quadratizations, three consider different reformulations for positive monomials in a termwise context, and the other three are based on pairwise covers.

Not all ten methods have been compared compared for every class of instances; Table 8.2 summarizes the experiments that have been carried out.

For Random same degree and Random high degree instances, among all linearization methods only the default resolution SL has been tested, because previous results in Chapter 4 and [41] comparing the remaining three linearizations were not concluding. Linearizations SL-NoCuts and SL-Only-2L were not tested for Vision instances, because Chapter 4 and [41] clearly show that these methods are prohibitively slow for these instances.

Remark 10. *Termwise methods have been marked with* † *for instances of degree at most four. This is because for positive monomials of degree four, the equations corresponding to Ishikawa's quadratization* (5.5)*, to our quadratization with* $\lceil \frac{n}{4} \rceil$ *variables* (6.20) *and to our quadratization with* $\lceil \log(n) \rceil - 1$ *variables* (6.19)*, are exactly the same. For positive monomials of degree three, our quadratization with* $\lceil \log(n) \rceil - 1$ *variables* (6.19) *is different from the other two.*

This means that for instances in the set RANDOM SAME DEGREE *of degree four, the three methods* ISHIKAWA*,* N/4 *and* LOGN-1 *are described by the exact same equations. For instances in the set* RANDOM SAME DEGREE *of degree three,* ISHIKAWA *and* N/4 *are described by the same equations, but* LOGN-1 *is not. Finally, for* VISION *and* AUTOCORRELATED SEQUENCES*, all monomials of degree three are negative, and thus quadratized using Freedman and Drineas* (5.2)*, which results in* ISHIKAWA*,* N/4 *and* LOGN-1 *being exactly the same methods again.*

However, we report the results for the three methods separately because, even though the methods are theoretically identical, in some cases the computation times strongly differ. This is a surprising and inconvenient result, given that for the same method, one would expect similar computing times on repeated resolutions. We blame these differences in computing times on small coding differences. Indeed, either the order of the terms or the order of the variables within the terms is implemented in a slighty different way for every termwise method, possibly altering the structure of the branch & bound tree. One should take this remark into account when interpreting the results of the experiments. As we mention more in detail in Chapter 10, this is one of the main issues that we would like to investigate further.

Moreover, a proper comparison of termwise quadratization methods between each other can only be done in view of the results of the set of instances RANDOM HIGH DEGREE*, which contain terms of degree higher than four.*

Finally, methods marked with * have not been tested for the largest instances in the VISION set, due to their poor performance for smaller instances.

8.3 Persistencies: preliminary experiments

As mentioned in Section 8.1, comparing the computational performance of linear and quadratic reformulations using CPLEX 12.7 can be detrimental for quadratizations, because of the many years of development of linear solvers with respect to quadratic solvers. Moreover, previous computational experience in quadratic binary optimization has shown that an important advantage of quadratizations is the possibility of using interesting bounds, such as the ones given by *roof duality*, and the *persistency* property, which allows fixing a subset of variables to their provable optimal values [25]. Such approaches have proved to be especially useful in the field of computer vision [52, 74, 79, 98], where persistencies allow to greatly reduce the size of problems that originally contain millions of variables.

This section presents the results of a set of preliminary experiments carried out with the objective of comparing computing times of the resolution of binary quadratic problems with and without persistencies, by relying on a rather naive self-made implementation. These experiments do not aim to compete with efficient existing implementations such as the one by Boros, Hammer, Sun and Tavares [25], but only try to provide an understanding of whether the use of persistencies could be usfeul when using a commercial solver like CPLEX 12.7 with our set of instances.

Let us formally describe our implementation and the underlying theoretical results. Given a quadratic problem in binary variables, the *Weak Persistency Theorem* allows us to determine the values of a subset of the variables in the optimal solution of the quadratic integer problem, by solving the continuous relaxation of its standard linearization. The Weak Persistency Theorem was first stated in [65, 93, 95]; we present it here as stated by Boros and Hammer [24].

Theorem 19 (*Weak Persistency*, Theorem 9 in [24])**.** *Let $f(x) = c_0+\sum_{j=1}^n c_jx_j+\sum_{i=1}^n \sum_{j=i+1}^n c_{ij}x_ix_j$, be a quadratic pseudo-Boolean function on n variables, and let $\tilde{x}$ be an optimal solution of the continuous relaxation of the standard linearization of f*

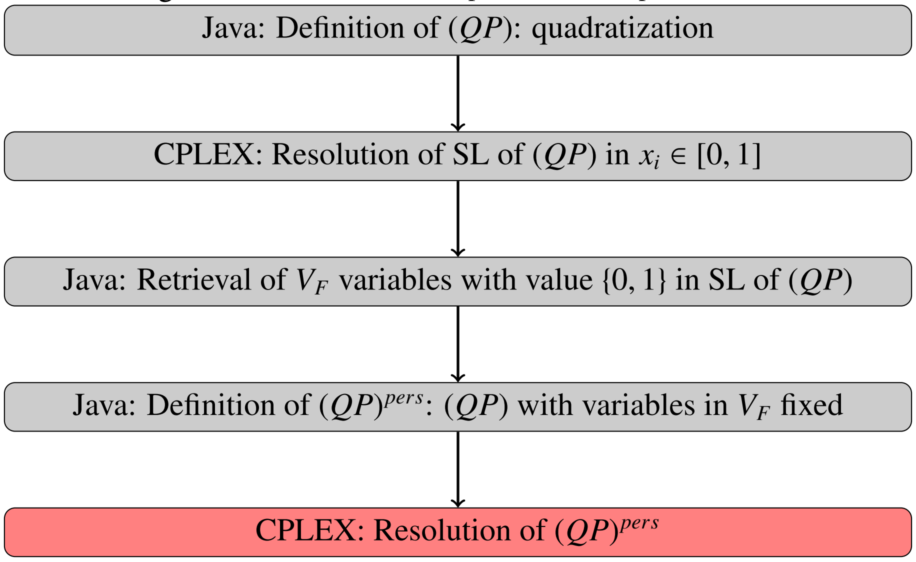
$$\begin{aligned}
\min\ & c_0 + \sum_{j=1}^n c_jx_j + \sum_{i=1}^n \sum_{j=i+1}^n c_{ij}y_{ij} & \\
s.\ t.\ & y_{ij} \geq x_i + x_j - 1 & 1 \leq i < j \leq n,\ c_{ij} > 0,\\
& y_{ij} \geq 0 & 1 \leq i < j \leq n,\ c_{ij} > 0,\\
& y_{ij} \leq x_i & 1 \leq i < j \leq n,\ c_{ij} < 0,\\
& y_{ij} \leq x_j & 1 \leq i < j \leq n,\ c_{ij} < 0,\\
& x_j \in \{0,1\}^n & 1 \leq j \leq n
\end{aligned}$$

for which $\tilde{x}_j = 1$ for $j \in O$, and $\tilde{x}_j = 0$ for $j \in Z$ (where O and Z are disjoint subsets of the indices). Then, for any minimizing vector $x^* \in argmin(f)$ switching the components to $x_j^* = 1$ for $j \in O$ and $x_j^* = 0$ for $j \in Z$ will also yield a minimum of f.

Our implementation of the resolution of a quadratic problem using persistencies is schematized as a flowchart in Figure 8.2. In the first box, we assume that we have defined a quadratization of the initial polynomial problem, which is either PC1, PC2, PC3, Ishikawa, n/4 or logn-1. The corresponding unconstrained quadratic problem in binary variables is denoted by (QP). In the second box, the standard linearization of (QP) is solved in continuous variables using CPLEX. The third box represents the retrieval, implemented in Java, of the subset of variables V_F taking integer values in the optimal solution of the standard linearization of (QP). V_F is the union of subsets Z and O in the notations of Theorem 19. In the fourth box, a new quadratic problem $(QP)^{pers}$ is defined. $(QP)^{pers}$ is the original problem (QP) with some additional constraints, imposing that the variables in V_F should take values either zero or one, depending on their values in the second box. Finally, the fifth box is the resolution of problem $(QP)^{pers}$ using CPLEX.

The remainder of this section presents the results of the corresponding computational experiments. For the selected instance sets, we present the percentages of variables that are fixed using persistencies, that is, the relative size of subset V_F with respect to the total number of variables in the quadratization. We also compare the computing times of the resolution of

Figure 8.2: Code flow of our persistencies implementation.


Java: Definition of (QP): quadratization
CPLEX: Resolution of SL of (QP) in $x_i \in [0, 1]$
Java: Retrieval of V_F variables with value $\{0, 1\}$ in SL of (QP)
Java: Definition of $(QP)^{pers}$: (QP) with variables in V_F fixed
CPLEX: Resolution of $(QP)^{pers}$

problems (QP) and $(QP)^{pers}$. It is important to notice that *we only present the computing times of the resolution of $(QP)^{pers}$, without including the overhead required for the pre-processing steps* represented in Figure 8.2 by the first four boxes.

8.3.1 Vision instances

We start by presenting the results for Vision instances with images of size 15×15; the results for images of other sizes are very similar and are described in Appendix A.

Table 8.3 shows the total number of variables of the quadratic problems corresponding to each quadratization method and Table 8.4 shows the percentage of variables of the quadratization that are fixed to an integer value using Theorem 19. The percentages of fixed variables are almost the same for each instance of size 15×15, because the difference in the instances comes only from the base image and the perturbation, which only has an impact on the linear part of the functions. A second important observation was made in Remark 10, pointing out that all termwise quadratization methods lead to the same model for Vision instances. Thus, the last three columns of Table 8.4 and Table 8.3 actually correspond to the same reformulation method. For termwise quadratizations, the linearized quadratic problems have an optimum where all variables have fractional values, therefore no variable can be fixed to its provable optimal value in (QP). On the other hand, pairwise covers fix between $20 - 40\%$ of the variables. It is not explicitly written in Table 8.4, but all variables are fixed to the value zero.

Table 8.5 shows the execution times required to solve the quadratic programs $(QP)^{pers}$ where variables in V_F have been fixed. (Only results for quadratizations based on pairwise covers are shown because for termwise quadratizations $(QP)^{pers}$ and (QP) are equal.) A bold font highlights the cases for which the quadratic program $(QP)^{pers}$ has a faster resolution time than (QP). This is the case for slightly more than half of the cases. However the improvement is of less than one second in almost all cases. Moreover, in the rest of the cases, the resolution time of $(QP)^{pers}$ is even slower than the resolution time of (QP).

The tables in Appendix A show similar results for images of sizes 10×10, 20×20, 25×25 and 30×30. The number of cases for which using persistencies is helpful is roughly half of the cases overall. However, for larger image sizes, the number of cases for which the improvement is noticeable seems to increase. We have considered that a significant improvement was at least five seconds faster, and corresponding figures are underlined in the tables.

All in all, even when improvements are observed, they remain small in relative value and may be due to random causes only. This, together with the fact that the reported computing times do not account for pre-processing times leads us to the conclusion that the effort to calculate persistencies is not worth with our implementation for Vision instances using CPLEX 12.7.

Table 8.3: Vɪsɪᴏɴ 15×15 $(n = 225, m = 1598)$: total number of variables in problem (QP).

	Pairwise covers			Termwise quadratizations		
	PC1	PC2	PC3	Iѕʜɪᴋ	ɴ/4	ʟᴏɢɴ-1
# variables QP	1024	1001	1001	1205	1205	1205

Table 8.4: Vɪsɪᴏɴ 15×15 $(n = 225, m = 1598)$: percentage of variables fixed using persistencies.

Instance (15×15)		Fixed variables (%)					
		Pairwise covers			Termwise quadratizations		
Base	Quality	PC1	PC2	PC3	Iѕʜɪᴋ	ɴ/4	ʟᴏɢɴ-1
ᴛᴏᴘ ʟᴇꜰᴛ ʀᴇᴄᴛ	ѕʜᴀʀᴘ	19.24	39.16	39.16	0.00	0.00	0.00
ᴛᴏᴘ ʟᴇꜰᴛ ʀᴇᴄᴛ	ʟᴏᴡ	19.24	37.86	37.86	0.00	0.00	0.00
ᴛᴏᴘ ʟᴇꜰᴛ ʀᴇᴄᴛ	ʟᴏᴡ	19.24	39.16	39.16	0.00	0.00	0.00
ᴛᴏᴘ ʟᴇꜰᴛ ʀᴇᴄᴛ	ʜɪɢʜ	19.24	39.16	39.16	0.00	0.00	0.00
ᴛᴏᴘ ʟᴇꜰᴛ ʀᴇᴄᴛ	ʜɪɢʜ	19.24	39.16	39.16	0.00	0.00	0.00
ᴄᴇɴᴛʀᴇ ʀᴇᴄᴛ	ѕʜᴀʀᴘ	19.24	38.26	38.26	0.00	0.00	0.00
ᴄᴇɴᴛʀᴇ ʀᴇᴄᴛ	ʟᴏᴡ	19.24	39.16	39.16	0.00	0.00	0.00
ᴄᴇɴᴛʀᴇ ʀᴇᴄᴛ	ʟᴏᴡ	19.24	39.16	39.16	0.00	0.00	0.00
ᴄᴇɴᴛʀᴇ ʀᴇᴄᴛ	ʜɪɢʜ	19.24	39.16	39.16	0.00	0.00	0.00
ᴄᴇɴᴛʀᴇ ʀᴇᴄᴛ	ʜɪɢʜ	19.24	39.16	39.16	0.00	0.00	0.00
ᴄʀᴏѕѕ	ѕʜᴀʀᴘ	19.24	39.16	39.16	0.00	0.00	0.00
ᴄʀᴏѕѕ	ʟᴏᴡ	19.24	39.16	39.16	0.00	0.00	0.00
ᴄʀᴏѕѕ	ʟᴏᴡ	19.24	39.16	39.16	0.00	0.00	0.00
ᴄʀᴏѕѕ	ʜɪɢʜ	19.24	39.16	39.16	0.00	0.00	0.00
ᴄʀᴏѕѕ	ʜɪɢʜ	19.24	39.16	39.16	0.00	0.00	0.00

Table 8.5: VISION 15 × 15 ($n = 225, m = 1598$): resolution times using persistencies.

Instance (15 × 15)		Exec. times (secs)					
		Pairwise covers					
Base	Quality	PC1	PC1-pers	PC2	PC2-pers	PC3	PC3-pers
TOP LEFT RECT	SHARP	16.86	**16.75**	12.30	**12.17**	12.39	12.48
TOP LEFT RECT	LOW	16.88	**16.78**	13.36	**13.25**	13.14	13.20
TOP LEFT RECT	LOW	18.22	**17.70**	15.63	**15.42**	15.52	15.69
TOP LEFT RECT	HIGH	50.53	**50.41**	22.33	22.64	22.70	**22.39**
TOP LEFT RECT	HIGH	78.00	**77.13**	35.75	**34.55**	34.08	34.58
CENTRE RECT	SHARP	20.63	21.16	12.59	**12.45**	12.59	12.67
CENTRE RECT	LOW	18.17	**18.03**	14.72	14.80	14.81	15.08
CENTRE RECT	LOW	25.55	**24.66**	14.98	15.31	15.13	15.33
CENTRE RECT	HIGH	48.94	**48.86**	25.03	25.28	24.78	25.00
CENTRE RECT	HIGH	40.59	**40.39**	36.31	**35.41**	35.55	36.11
CROSS	SHARP	20.95	**20.25**	12.81	**12.75**	12.91	**12.69**
CROSS	LOW	22.49	24.11	12.19	12.59	12.77	12.92
CROSS	LOW	20.08	**19.28**	12.69	12.89	13.13	13.81
CROSS	HIGH	52.16	**51.03**	32.41	35.44	32.59	**32.45**
CROSS	HIGH	50.72	**48.14**	25.13	**23.95**	24.30	25.83

8.3.2 Random high degree instances

Vision instances are all of degree four and have a very special structure. Therefore, we consider here a different set of instances, in order to obtain more representative results for the preliminary experiments on persistencies. We choose to use the set of instances Random high degree, because they are are highly unstructured and because they contain polynomials of large degrees, ranging between 7 and 16.

Table 8.6 reports the total number of variables of the quadratic problems, and the percentages of variables taking integer values in the optimal solution of the corresponding linearized quadratic problems. The original polynomials always contain $n = 200$ variables, and a number of terms varying from $m = 250$ to 400. Previous experimental results show that denser instances, that is, instances with a higher ratio m/n are more difficult to solve (see [30, 41] and Chapter 4). Interestingly, Table 8.6 shows that pairwise cover methods tend to fix larger percentages of the variables for denser instances, while termwise quadratizations behave in the opposite way. For example, for the least dense instances with $m = 250$ terms pairwise cover methods show percentages between $10.58 - 13.11\%$ of fixed variables while termwise methods $8.85 - 14.24\%$. However for the most dense instances with $m = 400$, pairwise covers fix $12.24 - 15.35\%$ while termwise quadratizations only $0.91 - 2.36\%$.

Table 8.7 shows the percentages of variables fixed to the values zero or one. It is interesting to notice that, as for Vision instances, pairwise cover methods fix almost all variables to zero. On the other hand, termwise quadratizations tend to fix more variables to one.

Table 8.8 shows the computing times of the resolutions of (QP) and $(QP)^{pers}$ for pairwise cover quadratizations. A bold font highlights the cases for which $(QP)^{pers}$ is solved faster than (QP). The results are similar as for Vision instances: there is an improvement for roughly half of the cases, but this improvement is very small. Moreover, as before, the pre-processing time used to compute the set of variables to fix is not accounted for.

Table 8.9 shows the computing times of the resolutions of (QP) and $(QP)^{pers}$ for termwise quadratizations. As before, a bold font highlights the cases for which $(QP)^{pers}$ is solved faster than (QP). Moreover, cases for which the improvement is important (more than five seconds) are also underlined. Interestingly, for some combinations of instance and quadratization method, there are remarkable improvements of hundreds or even a thousand of seconds. These improvements would be certainly useful even when accounting for the pre-processing time and are worth further investigation. However, such remarkable improvements only occur 12 times out of 60, and are not consistent among the methods or the instances. Moreover, there are also examples for which $(QP)^{pers}$ requires more than one hundred extra seconds of resolution time than (QP), and several other cases where $(QP)^{pers}$ is slower even if not by such a high computing time. All in all, we were not able to identify a consistent improvement in resolution times of quadratic problems when using persistencies together with CPLEX 12.7.

Table 8.6: Random high degree (n = 200): percentage of variables fixed using persistencies.

Instance				# variables QP						Fixed variables (%)					
				Pairwise covers			Termwise			Pairwise covers			Termwise		
n	m	id	deg	PC1	PC2	PC3	Ishik	n/4	logn-1	PC1	PC2	PC3	Ishik	n/4	logn-1
200	250	1	9	572	561	561	344	341	341	11.71	11.41	11.59	9.88	9.97	9.97
200	250	2	9	562	548	549	342	339	339	12.81	12.96	13.11	11.99	11.80	12.09
200	250	3	9	582	567	567	339	334	334	11.34	11.46	11.29	8.85	8.68	8.98
200	250	4	9	567	558	558	340	337	337	12.70	12.90	11.29	13.53	14.24	13.65
200	250	5	7	555	548	548	340	338	338	11.17	11.13	10.58	10.59	10.65	10.65
200	300	1	8	663	644	644	369	367	367	11.31	11.34	11.34	8.13	7.63	8.17
200	300	2	9	645	623	626	390	385	385	9.92	10.75	10.54	6.67	6.49	6.49
200	300	3	15	660	638	638	372	363	362	12.12	12.38	13.17	6.72	6.61	6.91
200	300	4	13	705	681	682	378	370	369	10.92	11.31	11.29	5.82	5.68	4.88
200	300	5	12	636	625	625	373	369	369	12.58	12.80	12.64	4.83	6.50	6.50
200	350	1	10	731	699	700	399	395	395	12.04	12.45	12.43	4.51	4.56	4.56
200	350	2	10	724	700	700	408	398	398	9.53	9.29	10.00	3.19	3.02	3.02
200	350	3	13	727	701	703	423	416	416	11.42	11.84	11.81	2.84	2.88	2.88
200	350	4	9	706	682	683	409	406	406	11.19	11.58	11.42	3.67	3.69	3.69
200	350	5	11	710	682	684	407	402	402	10.99	11.14	11.55	5.65	5.97	6.22
200	400	1	14	791	766	766	435	425	424	13.02	13.45	12.92	2.30	2.35	2.36
200	400	2	11	778	752	749	409	405	405	14.65	15.29	15.35	1.47	1.48	1.48
200	400	3	11	817	791	792	432	422	422	12.24	12.77	13.01	1.39	1.42	1.42
200	400	4	16	815	779	781	441	429	428	13.50	14.12	13.96	0.91	0.93	0.93
200	400	5	12	817	794	798	438	431	431	13.83	14.36	14.54	1.60	1.62	1.62

Table 8.7: Random high degree (n = 200): percentage of variables fixed to 0 and to 1 using persistencies.

Instance				Fixed variables (%)											
				PC1		PC2		PC3		Ishikawa		n/4		logn-1	
n	m	id	deg	%0	%1	%0	%1	%0	%1	%0	%1	%0	%1	%0	%1
200	250	1	9	11.71	0.00	11.41	0.00	11.59	0.00	2.62	7.27	2.64	7.33	2.64	7.33
200	250	2	9	12.81	0.00	12.96	0.00	13.11	0.00	3.22	8.77	3.24	8.55	3.24	8.85
200	250	3	9	11.34	0.00	11.46	0.00	11.29	0.00	2.65	6.19	2.69	5.99	2.69	6.29
200	250	4	9	12.70	0.00	12.90	0.00	11.29	0.00	3.53	10.00	3.56	10.68	3.26	10.39
200	250	5	7	11.17	0.00	11.13	0.00	10.58	0.00	2.06	8.53	2.07	8.58	2.07	8.58
200	300	1	8	11.31	0.00	11.34	0.00	11.34	0.00	1.08	7.05	1.09	6.54	1.09	7.08
200	300	2	9	9.92	0.00	10.75	0.00	10.54	0.00	1.03	5.64	1.04	5.45	1.04	5.45
200	300	3	15	12.12	0.00	12.23	0.16	13.17	0.00	1.88	4.84	1.93	4.68	1.93	4.97
200	300	4	13	10.92	0.00	11.31	0.00	11.29	0.00	0.26	5.56	0.27	5.41	0.27	4.61
200	300	5	12	12.58	0.00	12.80	0.00	12.64	0.00	1.61	3.22	1.63	4.88	1.63	4.88
200	350	1	10	12.04	0.00	12.45	0.00	12.43	0.00	0.50	4.01	0.51	4.05	0.51	4.05
200	350	2	10	9.53	0.00	9.29	0.00	10.00	0.00	0.25	2.94	0.25	2.76	0.25	2.76
200	350	3	13	11.42	0.00	11.84	0.00	11.81	0.00	0.47	2.36	0.48	2.40	0.48	2.40
200	350	4	9	11.19	0.00	11.58	0.00	11.42	0.15	0.73	2.93	0.74	2.96	0.74	2.96
200	350	5	11	10.99	0.00	11.14	0.00	11.55	0.00	0.74	4.91	1.00	4.98	1.00	5.22
200	400	1	14	13.02	0.00	13.45	0.00	12.92	0.00	0.46	1.84	0.47	1.88	0.47	1.89
200	400	2	11	14.65	0.00	15.29	0.00	15.35	0.00	0.49	0.98	0.49	0.99	0.49	0.99
200	400	3	11	12.24	0.00	12.77	0.00	13.01	0.00	0.00	1.39	0.00	1.42	0.00	1.42
200	400	4	16	13.50	0.00	14.12	0.00	13.96	0.00	0.23	0.68	0.23	0.70	0.23	0.70
200	400	5	12	13.83	0.00	14.36	0.00	14.54	0.00	0.23	1.37	0.23	1.39	0.23	1.39

Table 8.8: RANDOM HIGH DEGREE (n = 200): resolution times of pairwise covers using persistencies.

Instance				Exec. times (secs)					
				Pairwise covers					
n	m	id	deg	PC1	PC1-pers	PC2	PC2-pers	PC3	PC3-pers
200	250	1	9	1.19	1.81	1.39	2.7	1.36	3.31
200	250	2	9	2.14	**1.94**	1.95	2.25	1.98	**1.49**
200	250	3	9	2.28	**2.02**	2.01	**1.95**	1.97	2.17
200	250	4	9	1.81	**1.56**	1.16	1.16	1.28	1.36
200	250	5	7	1.41	**1.33**	1.53	1.55	1.81	1.84
200	300	1	8	5.67	**5.17**	5.69	8.23	7.2	9.27
200	300	2	9	11.91	**10.97**	10.56	**9.97**	10.3	10.36
200	300	3	15	4.08	**3.66**	4.56	4.63	3.55	3.56
200	300	4	13	11.97	14.03	10.69	13.77	8.56	8.75
200	300	5	12	6.14	**5.42**	3.78	**3.59**	3.69	**3.44**
200	350	1	10	12.44	**11.47**	6.31	**5.58**	6.41	**5.73**
200	350	2	10	14.66	15.88	10.66	13.51	11.97	**10.97**
200	350	3	13	16.88	**14.84**	8.41	**7.92**	9.95	10.05
200	350	4	9	6.92	**6.61**	4.97	**4.69**	4.83	**4.81**
200	350	5	11	4.11	4.47	3.83	3.89	3.00	3.06
200	400	1	14	11.25	**10.5**	9.92	**9.61**	9.91	9.91
200	400	2	11	16.36	16.86	12.41	12.58	16.2	17.95
200	400	3	11	22.88	23.83	21.31	**21.08**	25.13	25.41
200	400	4	16	29	**28.89**	20.76	22.31	20.94	21.19
200	400	5	12	12.41	15.88	16.33	22.99	16.56	**15.34**

Table 8.9: Random high degree (n = 200): resolution times of termwise quadratizations using persistencies.

Instance				Exec. times (secs) - tl means >3600 secs (gap %)					
				Termwise quadratizations					
n	m	id	deg	Ishikawa	Ishikawa-pers	n/4	n/4-pers	logn-1	logn-1-pers
200	250	1	9	5.17	7.03	8.7	14.19	7	11.89
200	250	2	9	13.81	14.67	21.98	29.56	19.5	**19.19**
200	250	3	9	18.13	**16.86**	50.61	**46.7**	24.23	**23.28**
200	250	4	9	13.75	**10.44**	25.06	**19.05**	14.73	15.19
200	250	5	7	13.33	13.39	26.34	**26.03**	12.94	13.66
200	300	1	8	19.31	**18.72**	74.59	**73.66**	22.34	22.81
200	300	2	9	48.72	49.94	221.53	230.86	101.69	107.03
200	300	3	15	53.92	**51.64**	463.01	**411.83**	101.19	106.34
200	300	4	13	275.28	**233.23**	2354.78	**1350.06**	481.75	500.13
200	300	5	12	36.41	36.91	81.92	82.08	53.45	59.77
200	350	1	10	67.86	74.13	241.45	305.88	73.13	76.08
200	350	2	10	358.70	474.72	2270.76	**2025.63**	669.49	**655.73**
200	350	3	13	136.97	**134.02**	865.66	878.20	370.31	**287.61**
200	350	4	9	32.31	32.50	63.94	**63.72**	45.36	45.72
200	350	5	11	37.13	**28.31**	66.26	**55.47**	48.11	**41.98**
200	400	1	14	693.34	**564.42**	tl (2.41 %)	tl (1.52 %)	tl (1.47 %)	tl (1.40 %)
200	400	2	11	33.55	33.63	61.77	**60.91**	42.86	44.03
200	400	3	11	tl (4.95 %)	tl (4.97 %)	tl (19.57 %)	tl (19.57 %)	tl (13.65 %)	tl (13.71 %)
200	400	4	16	tl (2.94 %)	tl (2.91 %)	tl (16.95 %)	tl (17.04 %)	tl (9.00 %)	tl (9.30 %)
200	400	5	12	350.09	**348.53**	850.36	**795.61**	665.42	**605.39**

8.3.3 Conclusions on the use of persistencies

In view of the previous results we decide *not* to use our implementation of persistencies to solve quadratic reformulations with CPLEX 12.7. While the use of persistencies results in faster computing times for roughly half of the instance-method combinations, these improvements are in general not very significant, even without taking pre-processing times into account. In the other half of the cases the use of persistencies is even detrimental.

All relevant improvements come from termwise quadratizations when applied to Random high degree instances. Interestingly, termwise methods tend to fix more variables to the value one than to the value zero. For pairwise covers, all variables are fixed to zero in most cases and the speed-up is not very significant. A possible explanation for this lack of significant improvements could be that CPLEX 12.7 fixes a similar set of variables in a pre-processing step, meaning that the fixed variables using persistencies are "easy" guesses for CPLEX. Moreover, even if termwise quadratization procedures show important speed-ups with the use of persistencies in some cases, the improved computing times are still much slower than those of pairwise covers quadratizations.

As a future improvement, our current implementation of persistencies should be revised and pre-processing times should be accounted for. In particular, Boros et al. [25] provided a network-flow based implementation resulting in the largest possible set of variables fixed using persistencies, which is additionally very efficient to calculate and gives very good results in terms of tighter bounds and faster resolution times. Moreover, their results apply to a large and diverse set of instances. A question of future research would be to use a similar pre-processing algorithm in our context.

Another interesting idea is to consider different bounding procedures for branch & bound algorithms. In [25] the authors mention bounding procedures based on the *roof dual* and the *iterated roof dual* bounds. Such an approach has not been considered in our experiments and is a question for future research.

8.4 Comparing linearizations and quadratizations: results

This section analyzes results of the computational experiments comparing the methods presented in Section 8.2.1. We present the results for each class of instances in a separate section. In all figures, missing data points represent resolution times larger than one hour, which is the time limit of all experiments.

As a general remark, whenever the automatic cut generation of CPLEX 12.7 was used in the branch & cut to solve linear reformulations, *zero-half cuts* were clearly the most frequently used type of cuts. Moreover, when user cuts were included, they were added before CPLEX automatic cuts, meaning that zero-half cuts might have also been generated from the pool of 2-link inequalities. For Vision and Autocorrelated sequences instances, many *clique cuts* were also generated, whereas they were not used for Random instances. Other classes of cuts that appeared are *lift-and-project cuts* and *Gomory fractional cuts*, although in much smaller number in general. We do not have further details on when exactly each type of cut was used nor on the interaction of CPLEX cuts and user cuts. However, we chose to include the previous remarks for potential future reference.

8.4.1 Random same degree instances

This section presents the results for the set of instances Random same degree, consisting of polynomials having either all terms of degree three or all terms of degree four (see Section 8.2.2 for a detailed description of the instances).

The experiments have been carried out for instances with $n = 200$, 400 and 600 variables. We present in detail the case $n = 200$; instances with $n = 400$ and 600 have a similar behavior and corresponding figures can be found in Appendix E. All computing times are reported in Tables B.1 and B.2 in Appendix B.

For this class of instances, method SL-2L has not been tested because it is not clear whether the 2-link inequalities are useful for this class of instances (see Chapter 4 and [41]).

Figures 8.3, 8.4, 8.5 and 8.6 present the instance identifier on the x-axis and the y-axis displays the execution times (in seconds) of the methods listed in the corresponding legend. Instance identifiers are of the form *n-m-d-id*, where n is the number of variables of the polynomial, m is the number of monomials, d is the degree and *id* is simply an identifier for the polynomial. For example, instance 200-400-3-1 has 200 variables, 400 terms all of which have degree 3 and it is the first polynomial with these parameters.

As a first general remark, Figures 8.3 and 8.4 show that the standard linearization SL performs much better than all quadratizations. Moreover, it should be noted that in comparison to the instance set considered in Chapter 4 and [41], we were here only able to test the easiest instances, given that for all quadratization methods, many of the hardest instances were not solved within an hour. (Remember that previous results show that the easiest instances are those with smaller ratios $\frac{m}{n}$ – see [30, 41]).

Let us now focus on Figure 8.3 and Figure 8.4, which present execution times for instances of degree three and degree four, respectively. Note that when comparing pairwise covers and termwise quadratizations it is not clear which of the two classes is more efficient for degree three instances, while for degree four instances pairwise covers are clearly faster.

In Figure 8.5 we only consider pairwise covers; the first ten data points correspond to instances of degree three and the last ten correspond to instances of degree four. We can see that instances of degree three are at least as difficult as instances of degree four for these methods. Figure 8.6 makes the same comparison for termwise quadratizations, and it is clear that instances of degree four are harder than instances of degree three, which seems more natural intuitively.

By Remark 10, for instances with all monomials of degree three, Ishikawa and n/4 are exactly the same quadratizations, while logn-1 is different. This is however not reflected in the figures, where the plots of termwise methods are almost undistinguishable. For degree four instances, all termwise quadratization methods have the same equations, and their computational behavior is also similar, as expected.

Let us now analyze the number of variables and positive quadratic terms required for each quadratization method. Table 8.10 reports the number of auxiliary variables and Table 8.11

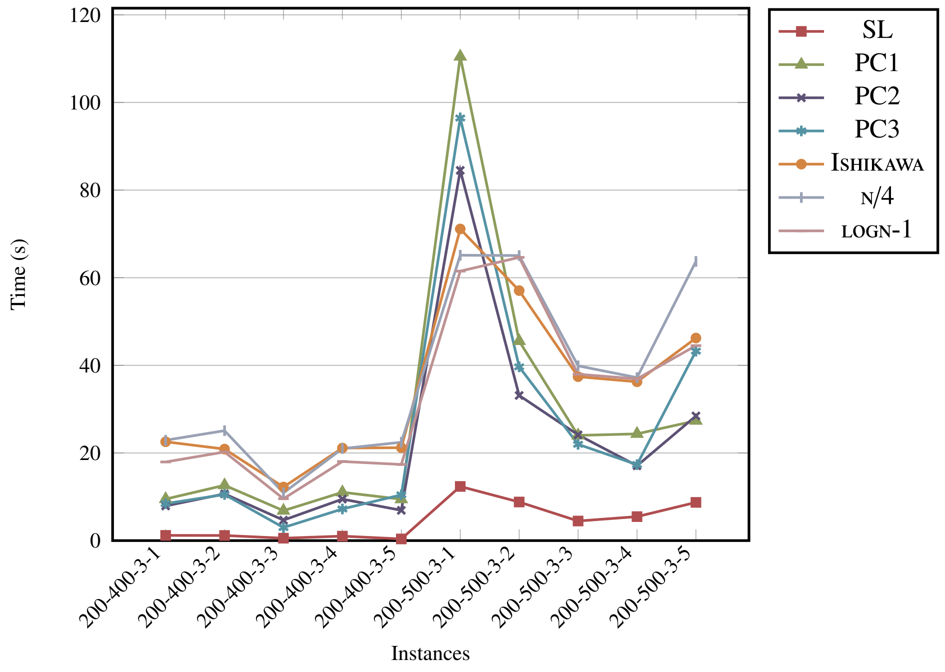


Figure 8.3: Random same degree (n = 200, deg = 3): linearizations and quadratizations computing times.

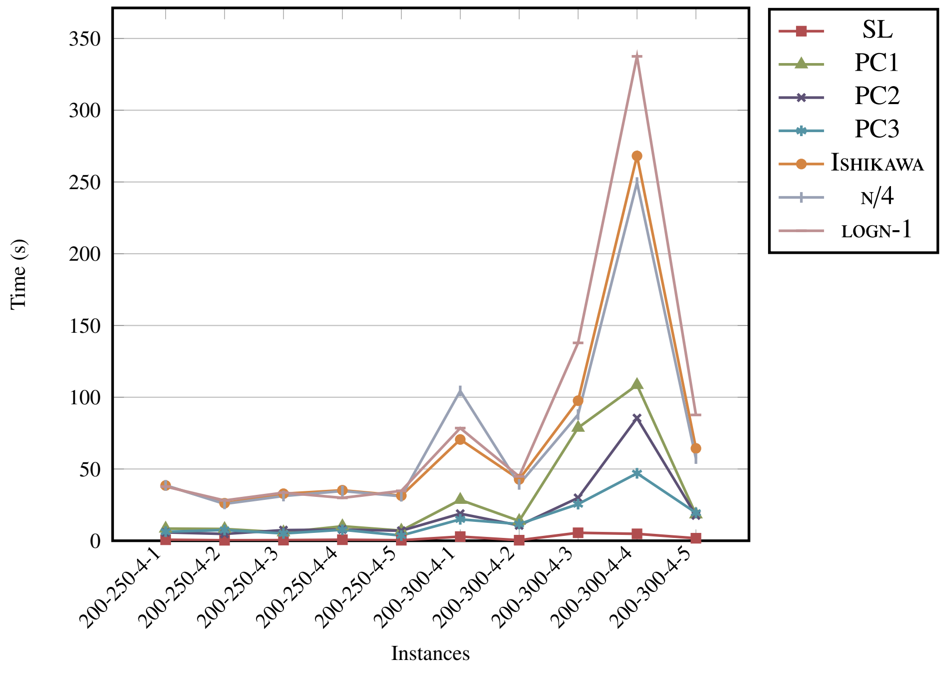


Figure 8.4: Random same degree (n = 200, deg = 4): linearizations and quadratizations computing times.

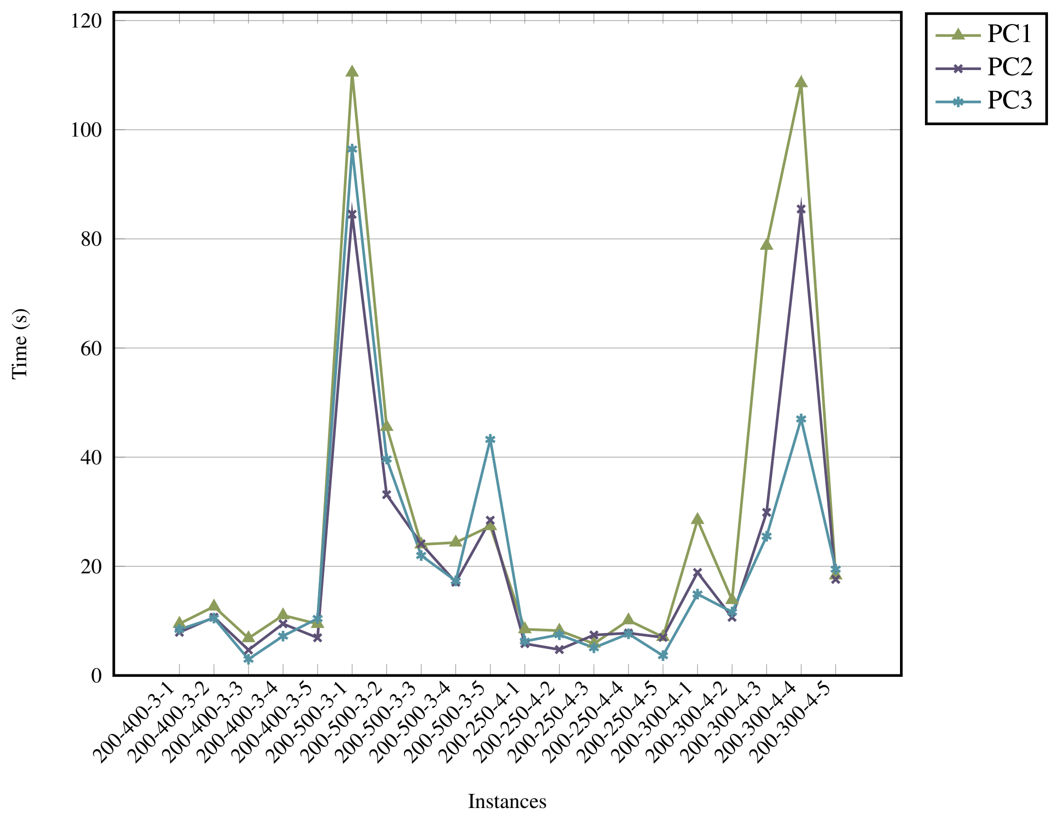


Figure 8.5: RANDOM SAME DEGREE ($n = 200$): pairwise covers computing times by degree.

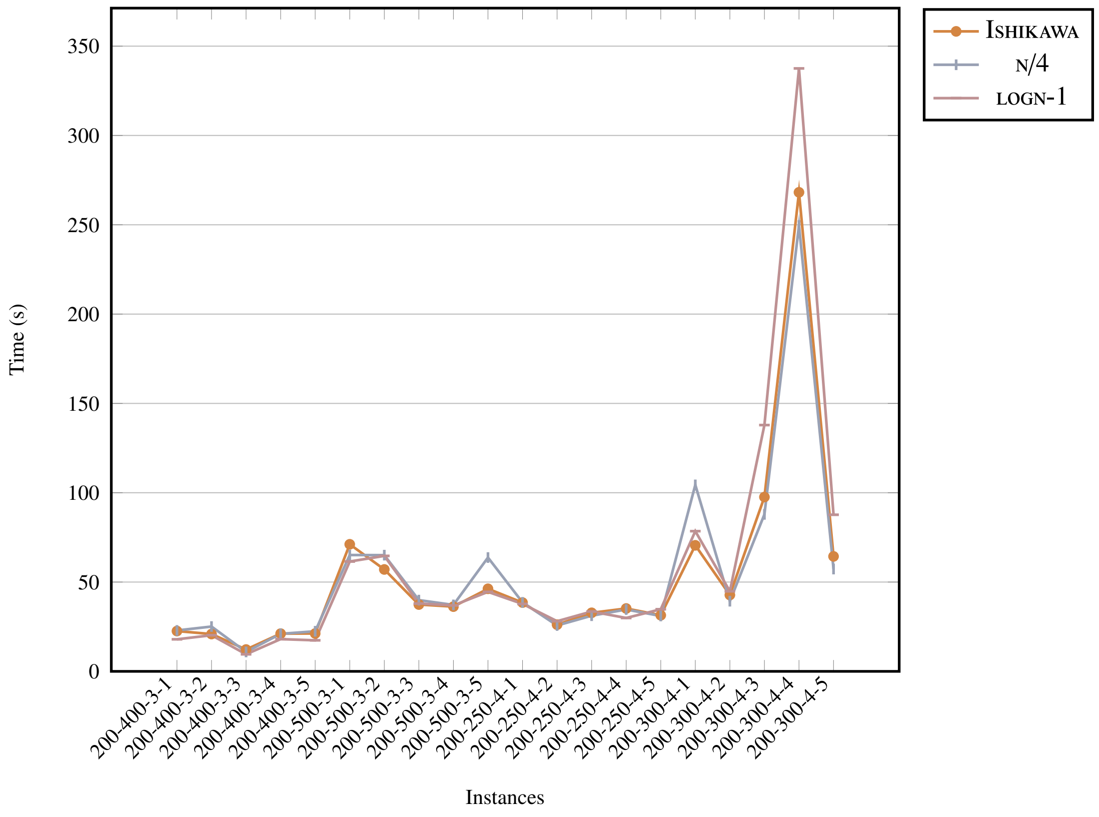


Figure 8.6: RANDOM SAME DEGREE ($n = 200$): termwise quadratizations computing times by degree.

Table 8.10: Random same degree (n = 200, deg = 3 and 4): number of y variables in quadratizations.

Instance				# y variables					
				Pairwise covers			Termwise quadratizations		
n	m	deg	id	PC1	PC2	PC3	Ishikawa	n/4	logn-1
200	400	3	1	390	361	361	400	400	400
200	400	3	2	398	375	375	400	400	500
200	400	3	3	398	371	371	400	400	400
200	400	3	4	394	364	364	400	400	400
200	400	3	5	398	365	365	400	400	400
200	500	3	1	491	449	449	500	500	500
200	500	3	2	493	458	458	500	500	500
200	500	3	3	494	455	455	500	500	500
200	500	3	4	494	446	446	500	500	500
200	500	3	5	494	445	445	500	500	500
200	250	4	1	490	455	455	250	250	250
200	250	4	2	493	459	459	250	250	250
200	250	4	3	486	462	462	250	250	250
200	250	4	4	491	463	463	250	250	250
200	250	4	5	493	453	453	250	250	250
200	300	4	1	590	545	546	300	300	300
200	300	4	2	584	548	548	300	300	300
200	300	4	3	588	545	545	300	300	300
200	300	4	4	593	552	552	300	300	300
200	300	4	5	591	545	547	300	300	300

reports the number of positive quadratic terms introduced by each quadratization for instances with n = 200. (See Appendix B for instances with n = 400 and 600, which present the same behavior.)

For instances of degree three, termwise quadratizations and pairwise covers based quadratizations introduce approximately the same number of variables and of positive quadratic terms. Figure 8.3 shows that both types of quadratizations have very similar computing times in this case, which is coherent with this observation.

However, for degree four instances, termwise quadratizations require about half of the auxiliary variables of pairwise covers quadratizations, while they introduce hundreds of positive quadratic terms more than pairwise covers. Moreover, Figure 8.4 shows that for instances of degree four, pairwise covers are always faster than termwise quadratizations, which suggests that the fact of introducing a small number of positive quadratic terms plays a more important role than the fact of having a small number of auxiliary variables.

Finally, in order to compare the relative performance of quadratization methods with

Table 8.11: Random same degree ($n = 200, \deg = 3$ and 4): number of positive quadratic terms.

Instance				# positive quadratic terms					
				Pairwise covers			Termwise quadratizations		
n	m	deg	id	PC1	PC2	PC3	Ishikawa	n/4	logn-1
200	400	3	1	584	555	555	582	582	582
200	400	3	2	588	565	565	570	570	570
200	400	3	3	588	561	561	570	570	570
200	400	3	4	588	558	558	582	582	582
200	400	3	5	619	586	586	663	663	663
200	500	3	1	738	696	696	741	741	741
200	500	3	2	744	709	709	753	753	753
200	500	3	3	729	690	690	705	705	705
200	500	3	4	742	694	694	744	744	744
200	500	3	5	731	682	682	711	711	711
200	250	4	1	627	592	592	822	822	822
200	250	4	2	615	581	581	732	732	732
200	250	4	3	611	587	587	750	750	750
200	250	4	4	610	582	582	714	714	714
200	250	4	5	613	573	573	720	720	720
200	300	4	1	743	698	699	918	918	918
200	300	4	2	730	694	694	876	876	876
200	300	4	3	737	694	694	894	894	894
200	300	4	4	755	714	714	972	972	972
200	300	4	5	745	699	701	924	924	924

Table 8.12: RANDOM SAME DEGREE all instances: quadratizations are on average × times slower than SL.

Method	× slower than SL (average)
PC1	17.15
PC2	14.58
PC3	15.16
ISHIKAWA	58.62
N/4	62.13
LOGN-1	64.57

SL, Table 8.12 shows a calculation of the overall factor by which quadratization methods are slower than SL. These factors have been computed by dividing, for each instance, the resolution time of the quadratization by the resolution time of SL, and by taking the average of all these ratios afterwards.

8.4.2 RANDOM HIGH DEGREE instances

This section presents the results for the set of instances RANDOM HIGH DEGREE, consisting of polynomials of degrees varying between 7 and 16, where the degree of each monomial is generated using an exponential distribution (see Section 8.2.2 for a detailed description of the instances).

The are instances with $n = 200$, 400 and 600 variables. As in the previous section, we present here only figures for $n = 200$; instances with $n = 400$ and 600 have a similar behavior and the corresponding figures are given in Appendix F. All computing times are detailed in Tables C.1, C.2 and C.3 in Appendix C.

As in the previous section, method SL-2L has not been tested because it is not clear whether the 2-link inequalities are useful for RANDOM HIGH DEGREE instances (see Chapter 4 and [41]).

Figures 8.7 and 8.8 present the instance identifier on the x-axis and the y-axis reports the execution times (in seconds) of the methods listed in the corresponding legend. Instance identifiers are of the form n-m-d-id, where n is the number of variables of the polynomial, m is the number of monomials, d is the degree of the polynomial and id is simply an id for the polynomial. For example, instance 200-250-9-2 has 200 variables, 250 terms, the degree of the polynomial is 9 and it is the second polynomial with these parameters. Only parameters n and m are fixed for this class of instances; the degree of the polynomial is determined randomly.

As for RANDOM SAME DEGREE instances, we were only able to test the easiest instances in comparison to the experiments presented in Chapter 4 and [41] because all quadratizations were too slow for the hardest examples.

Figure 8.7 presents computing times of all methods, while Figure 8.8 focuses on the fastest ones. A first general conclusion is that we can clearly divide all reformulations in three classes according to their computational performance. The fastest method is SL, solving ev-

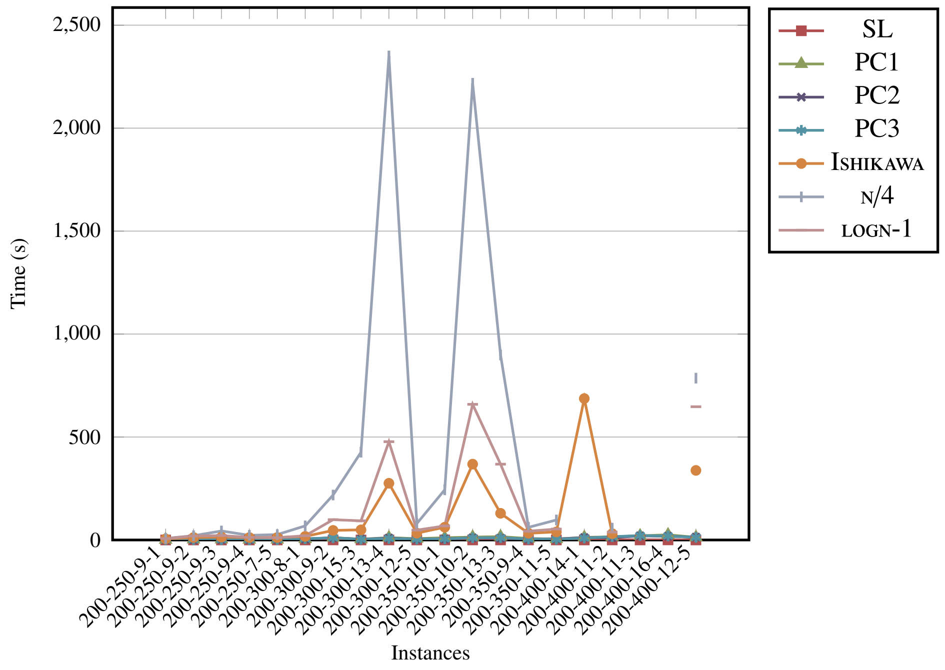


Figure 8.7: Random high degree ($n = 200$): linearizations and quadratizations computing times.

ery instance immediately. The second fastest methods are pairwise covers while termwise are the slowest, with high percentages of unsolved instances in one hour, as shown in Table 8.13. This table also reports the average factor by which quadratization methods are slower than SL. If we compare pairwise covers against each other, PC2 and PC3 have a similar behavior and are faster than PC1. If we compare termwise quadratizations against each other, Ishikawa is the fastest method, having a low percentage of unsolved instances in one hour, while n/4 is clearly the worst.

Table 8.14 and 8.15 report the number of auxiliary variables and the number of positive quadratic terms introduced by each quadratization, respectively, for polynomials with $n = 200$ (see Appendix C for $n = 400$ and 600). Independently of n, termwise quadratizations

Table 8.13: Random high degree all instances: quadratizations are on average $\times$ times slower than SL.

Method	$\times$ slower than SL (average)	% unsolved in 1h
PC1	80.71	0
PC2	70.87	0
PC3	70.37	0
Ishikawa	4726.26	8
n/4	8913.73	38
logn-1	4258.53	26

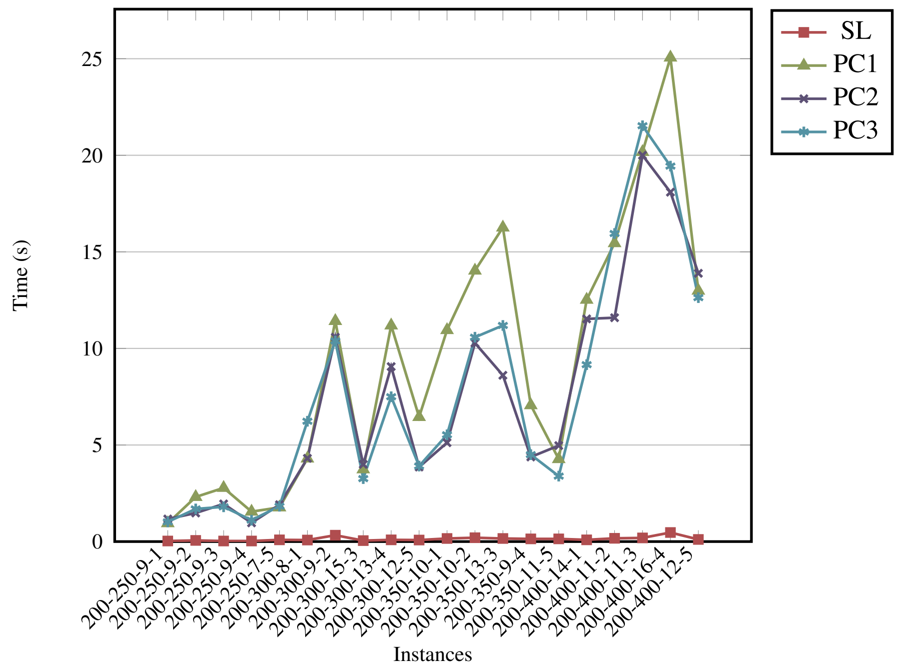


Figure 8.8: Random high degree ($n = 200$): linearizations and quadratizations fastest computing times.

introduce on average less than half of the number of variables required by pairwise covers. Concerning positive quadratic terms, termwise quadratizations have, in most cases, a few hundreds of terms more than pairwise covers, especially for denser instances and instances with higher degree. Among termwise methods, Ishikawa has in general less positive quadratic terms than n/4 and logn-1, which is probably more obvious for larger instances with $n = 400$ and 600. These observations together with the computing times seem to suggest that the fact of introducing a small number of positive quadratic terms plays again an important role.

8.4.3 Vision instances

This section presents the results for the set of instances Vision, consisting of polynomials of degree four inspired from the image restoration problem in computer vision (see Section 8.2.2 for a detailed description).

The experiments have been carried out for instances with base images of sizes 10×10, 15×15, 20×20, 25×25 and 30×30, using n = 100, 225, 400, 625 and 900 variables, respectively. We present here only figures for images of size 15 × 15. Figures for the remaining instances are given in Appendix G. All computing times are reported in Appendix D.

Figures 8.9 and 8.10 present the instance identifier on the x-axis and the y-axis displays the execution times (in seconds) of the methods listed in the legends. Instance identifiers are of the form `base-perturbation-id`, where `base` can be equal to `tl` (top left rectangle), `cr` (centre rectangle) or `cx` (cross), `perturbation` can be equal to `s` (sharp, no perturbation),

Table 8.14: Random high degree (n = 200): number of y variables in quadratizations.

Instance				# y variables					
				Pairwise covers			Termwise quadratizations		
n	m	id	deg	PC1	PC2	PC3	Ishikawa	n/4	logn-1
200	250	1	9	372	361	361	144	141	141
200	250	2	9	362	348	349	142	139	139
200	250	3	9	382	367	367	139	134	134
200	250	4	9	367	358	358	140	137	137
200	250	5	7	355	348	348	140	138	138
200	300	1	8	463	444	444	169	167	167
200	300	2	9	445	423	426	190	185	185
200	300	3	15	460	438	438	172	163	162
200	300	4	13	505	481	482	178	170	169
200	300	5	12	436	425	425	173	169	169
200	350	1	10	531	499	500	199	195	195
200	350	2	10	524	500	500	208	198	198
200	350	3	13	527	501	503	223	216	216
200	350	4	9	506	482	483	209	206	206
200	350	5	11	510	482	484	207	202	202
200	400	1	14	591	566	566	235	225	224
200	400	2	11	578	552	549	209	205	206
200	400	3	11	617	591	592	232	222	222
200	400	4	16	615	579	581	241	229	228
200	400	5	12	617	594	598	238	231	231

Table 8.15: RANDOM HIGH DEGREE (n = 200): number of positive quadratic terms.

Instance				# positive quadratic terms					
				Pairwise covers			Termwise quadratizations		
n	m	id	deg	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
200	250	1	9	437	426	426	468	483	483
200	250	2	9	421	407	408	465	482	482
200	250	3	9	440	425	425	480	495	495
200	250	4	9	432	423	423	444	456	456
200	250	5	7	423	416	416	439	451	451
200	300	1	8	525	506	506	477	495	495
200	300	2	9	519	497	500	561	583	583
200	300	3	15	531	509	509	728	759	756
200	300	4	13	580	556	557	733	763	760
200	300	5	12	517	506	506	614	632	632
200	350	1	10	620	588	589	663	685	685
200	350	2	10	614	590	590	769	802	802
200	350	3	13	629	603	605	777	804	804
200	350	4	9	591	567	568	615	636	636
200	350	5	11	602	574	576	703	727	727
200	400	1	14	689	664	664	927	963	960
200	400	2	11	664	638	635	623	639	639
200	400	3	11	716	690	691	902	939	939
200	400	4	16	715	679	681	999	1034	1031
200	400	5	12	714	691	695	851	882	882

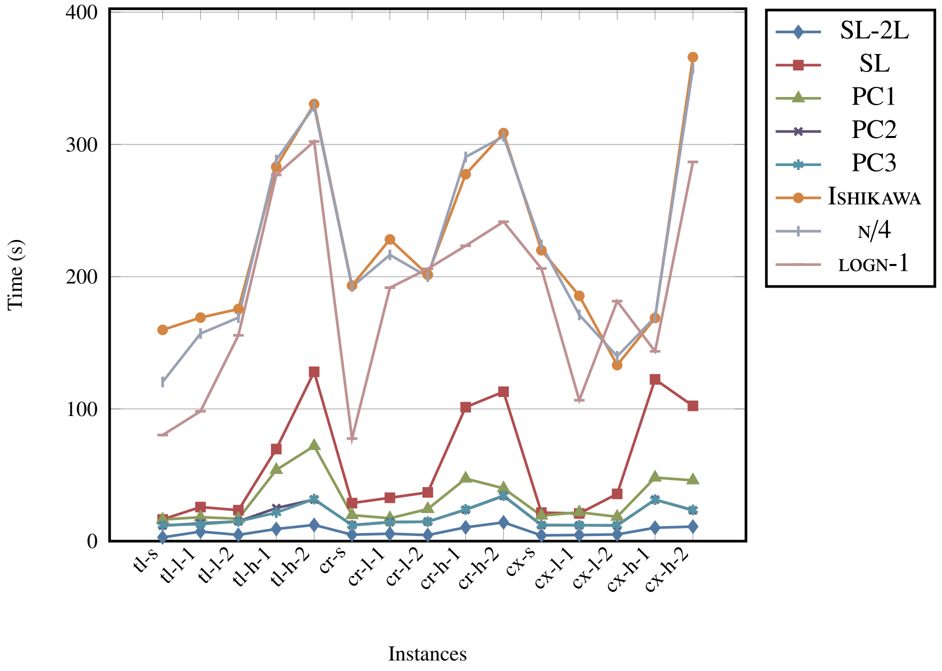


Figure 8.9: Vision 15 × 15 ($n = 225, m = 1598$): linearizations and quadratizations computing times.

`l` (low perturbation) or `h` (high perturbation) and `id` can be equal to 1 or 2. Only instances with a low or high perturbation use an `id`, because for these perturbation settings we have generated two samples, while there is only one sharp image.

Figure 8.9 reports computing times for all methods, while Figure 8.10 focuses on the fastest ones. For instances with images of sizes 20 × 20, 25 × 25 and 30 × 30 termwise quadratizations were not tested, because of their poor performance on smaller instances.

The main conclusion for Vision experiments is that no matter the size of the instance, there are clearly four classes of methods when according to computational performance. The fastest method is always SL-2L and is able to solve all instances up to 30 × 30 images in at most 4 minutes.

Table 8.16 shows a calculation of the average factor by which all methods are slower than SL-2L. For termwise methods, the averages only take into account the smallest classes of instances (10 × 10 and 15 × 15), and SL was not tested for 35 × 35 images. The second class of fastest methods are pairwise cover quadratizations, which are on average four our five times slower than SL-2L. The third best method is SL, with eight times slower computing times than SL-2L on average. Termwise quadratizations are the slowest, with times that are up to nine times slower than pairwise covers and up to 28 times slower than SL-2L.

Table 8.17 presents the number of auxiliary variables introduced by each quadratization and Table 8.18 reports the number of positive quadratic terms. These values only depend on the image size, because the definition of higher-degree monomials does not depend on the base image or the perturbation. Unlike random instances, pairwise covers require less auxil-

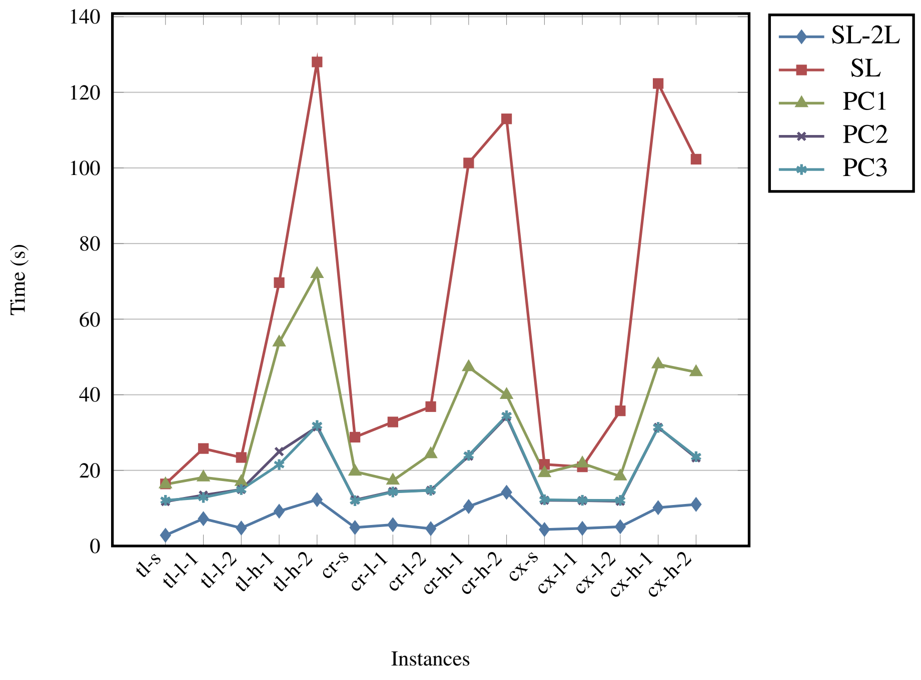


Figure 8.10: VISION 15×15 ($n = 225, m = 1598$): linearizations and quadratizations fastest computing times.

Table 8.16: VISION all instances: SL and quadratizations are on average $\times$ times slower than SL-2L.

Method	$\times$ slower than SL-2L (average)	% unsolved in 1h	% not tested
SL	8.73	0	16.7
PC1	4.47	12	0
PC2	3.72	0	0
PC3	3.74	0	0
ISHIKAWA	28.54	0	66.7
N/4	27.77	0	66.7
LOGN-1	25.22	0	66.7

Table 8.17: VISION: number of y variables in quadratizations.

Instance			# y variables					
			Pairwise covers			Termwise quadratizations		
Image size	n	m	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
10×10	100	668	334	321	321	405	405	405
15×15	225	1598	799	776	776	980	980	980
20×20	400	2928	1464	1431	1431	-	-	-
25×25	625	4658	2329	2286	2286	-	-	-
30×30	900	6788	3394	3341	3341	-	-	-
35×35	1525	9318	4659	4596	4596	-	-	-

Table 8.18: VISION: number of positive quadratic terms.

Instance			# positive quadratic terms					
			Pairwise covers			Termwise quadratizations		
Image size	n	m	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
10×10	100	668	416	403	403	648	648	648
15×15	225	1598	996	973	973	1568	1568	1568
20×20	400	2928	1826	1793	1793	-	-	-
25×25	625	4658	2906	2863	2863	-	-	-
30×30	900	6788	4236	4183	4183	-	-	-
35×35	1525	9318	5816	5753	5753	-	-	-

iary variables than termwise quadratizations. Pairwise covers also introduce a few hundreds less positive quadratic terms. Both factors explain the better computational performance of pairwise covers.

Focusing now on pairwise cover quadratizations, PC1 is slower than PC2 and PC3 for almost every instance. This makes sense intuitively, since methods PC2 and PC3 model interactions between monomials more accurately. Moreover, PC1 requires more auxiliary variables and positive quadratic terms.

Finally, although the three termwise quadratization methods are theoretically the same (see Remark 10), it seems that LOGN-1 has a different behavior than the other two termwise methods, being slightly faster on average. As mentioned in Remark 10 we blame this inconvenient and unexpected behavior on differences in coding the quadratizations. A possible explanation is that different orders of the terms and of the variables lead to different branch & cut trees, resulting in better computing times for LOGN-1 in many cases. This issue should however be analyzed in more detail, for example by coding identical equations for all termwise methods, in the sense of using the same order of variables and terms.

Large instances We ran some additional tests on instances of size 35×35, containing $n = 1225$ variables and $m = 9318$ monomials. SL-2L was able to solve all 35×35 instances within less than ten minutes, while PC1, PC2 and PC3 quadratizations were slower, but still solved all problems in less than one hour.

However, the time used to create models in Java started to play a role for these instances. For smaller instances, model creation times were not registered because they were small compared to the problem resolution time. For 35×35 instances, model creation times in Java are shown in detail in Table D.7 and total times (model creation plus resolution) are shown in Table D.8, of Appendix D. We display here corresponding figures. Figure 8.11 shows resolution times of the different methods; clearly SL-2L is the fastest. Figure 8.12 shows model creation times. In this case, PC1 is the fastest method while SL-2L is very slow. Finally, Figure 8.13 shows total times (model creation plus resolution). SL-2L is *not* the best method anymore for several instances. Indeed, long model creation times are not compensated by a faster resolution time of the SL-2L model.

The experiments on large Vision instances highlight a clear drawback of SL-2L as it is currently implemented. As a reminder, 2-link inequalities are added as a pool of user cuts to CPLEX 12.7, and *all* the 2-link inequalities are added to the pool. One can at most generate $m(m-1)$ 2-link inequalities, where m is the number of terms of the instance. However, we do not know exactly how the inequalities are handled by CPLEX during the branch and cut algorithm. Moreover, this approach works well for relatively small instances but for large instances model generation takes too much time. Other experiments have shown that for even larger instances, the memory consumption required to add all 2-links to the pool of cuts might become too high. A possible solution that we have not explored but that is considered as a future research question is the definition of a separation algorithm for the 2-link inequalities. Even though the number of 2-link inequalities is only quadratic in the number of terms of the original multilinear function, a separation procedure could be interesting to generate effective inequalities in a customized way. One possibility would be to dynamically generate only the most violated inequalities, or inequalities that we believe to be more relevant. Such a procedure could potentially avoid memory problems and reduce computing times in practice. An attempt to generate a subset of relevant inequalities in a pre-processing step has been implemented and is described in Chapter 9, where we decide heuristically which 2-link inequalities should be added to the pool of cuts.

8.4.4 Autocorrelated sequences instances: first experiments

This section presents the results for the set of instances Autocorrelated sequences, which consist of polynomials of degree four, with small number of variables and very large number of terms. These instances arise in the field of statistichal mechanics, modeling problems such as the transition of a supercooled liquid to glass. The model, which is closely related to the Ising model, aims at minimizing the autocorrelation between pairs of spins of the considered material (see Section 8.2.2 for a detailed description of the instances).

Autocorrelated sequences are the most difficult instances by far. We only report here results for eight instances out of forty, because none of our methods could solve the remaining

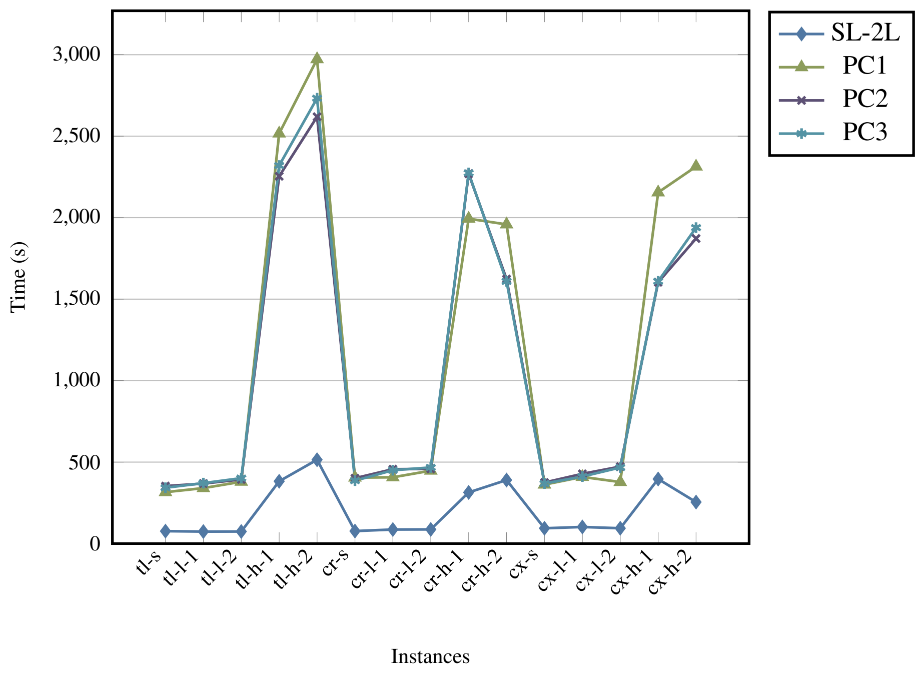


Figure 8.11: VISION 35×35 ($n = 1225, m = 9318$): linearizations and quadratizations computing times.

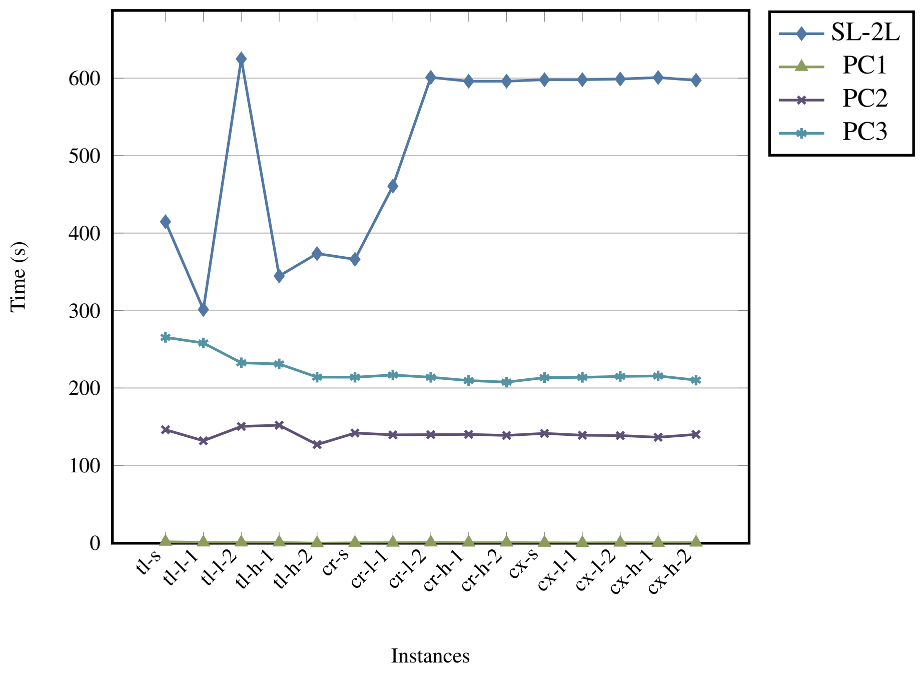


Figure 8.12: VISION 35×35 ($n = 1225, m = 9318$): Java model creation times.

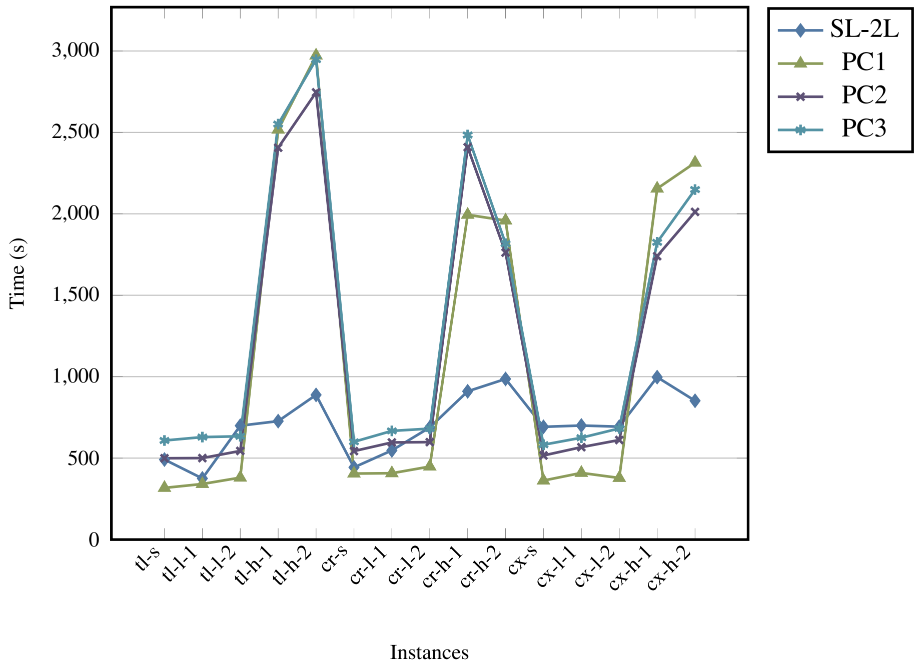


Figure 8.13: Vision 35×35 ($n = 1225, m = 9318$): total times (model creation + resolution).

instances in reasonable time. Table 8.19 describes the number of variables n and the number of terms m of the complete set of Autocorrelated sequences instances. Again, it seems that the difficulty comes from having large ratios $\frac{m}{n}$. Indeed, the largest instances have only $n = 60$ variables, but the number of terms is of the order of thousands for the majority of examples.

The work on Autocorrelated sequences is still ongoing. The instances highlighted in bold, for which we report computational results here, are such that at least one linear or quadratic reformulation was able to reach the optimum in less than an hour. For the remaining instances, either none of the methods finished within an hour or we run into memory problems.

It should be noted that not all instances were tested. For example, for instance 25_19, with $n = 25$ variables and $m = 3040$ terms, we run into memory and time limit problems. Then, we did not test denser instances on 25 variables. For the smallest instance with 45 variables we also had time and memory problems, so we did not test instances with more variables than 45. An open research direction is to extend these tests, possibly allowing larger time limits or using a more powerful machine. Even if the experiments on this test bed is far from complete, we find it worth to mention these first results because they show the challenging nature of the Autocorrelated sequences instances. Furthermore, `http://polip.zib.de/autocorrelated_sequences/` states that for several of these instances the optimal value is still unknown, and that no instance can be solved with a general-purpose solver within an hour.

Concerning our results, Table 8.20 reports the results of the four linearization methods

Table 8.19: AUTOCORRELATED SEQUENCES: variables (n) and terms (m) in each instance.

Id	n	m	Id	n	m
20_5	**20**	**207**	45_5	45	507
20_10	**20**	**833**	45_11	45	2 813
25_6	**25**	**407**	45_23	45	10 776
25_13	**25**	**1 782**	45_34	45	18 348
25_19	25	3 040	45_45	45	21 993
25_25	25	3 677	50_6	50	882
30_4	**30**	**223**	50_13	50	4 457
30_8	**30**	**926**	50_25	50	14 412
30_15	30	2 944	50_38	50	25 446
30_23	30	5 376	50_50	50	30 271
30_30	30	6 412	55_6	55	977
35_4	**35**	**263**	55_14	55	5 790
35_9	35	1 381	55_28	55	19 897
35_18	35	5 002	55_41	55	33 318
35_26	35	8 347	55_55	55	40 402
40_5	**40**	**447**	60_8	60	2 036
40_10	40	2 053	60_15	60	7 294
40_20	40	7 243	60_30	60	25 230
40_30	40	12 690	60_45	60	43 689
40_40	40	15 384	60_60	60	52 575

Table 8.20: Autocorrelated sequences: linearizations computing times.

Instance			Resolution time (secs)			
			Linearizations			
Id	n	m	SL-2L	SL	SL-2L-Only	SL-NoCuts
20_5	20	207	6.94	11.48	1.06	6.13
20_10	20	833	112.27	91.75	47	65.89
25_6	25	407	65.36	137.38	44.52	321.77
30_4	30	223	4.24	15.7	13.47	349.84
35_4	35	263	11.92	36.86	69.03	3104.45
25_13	25	1782	1645.2	2567.17	685.74	2408.69
30_8	30	926	2743.51	> 3600	> 3600	> 3600
40_5	40	447	1321.61	> 3600	> 3600	> 3600

Table 8.21: Autocorrelated sequences: quadratizations computing times.

Instance			Resolution time (secs)					
			Pairwise covers			Termwise quadratizations		
ID	n	m	PC1	PC2	PC3	Ishikawa	n/4	logn-1
20_5	20	207	10.58	5.05	4.27	37.47	34.95	35.34
20_10	20	833	90.28	159.47	137.69	417.72	419.53	365.47
25_6	25	407	106.67	80.17	121.03	629.66	631.77	466.92
30_4	30	223	13.52	7.17	7.03	29.67	31.02	36.08
35_4	35	263	24.13	13.25	11.2	49.77	49.92	54.14
25_13	25	1782	2311.09	> 3600	> 3600	> 3600	> 3600	> 3600
30_8	30	926	> 3600	> 3600	> 3600	> 3600	> 3600	> 3600
40_5	40	447	> 3600	914.27	2053.97	> 3600	> 3600	> 3600

SL-2L, SL, SL-2L-Only and SL-NoCuts and Table 8.21 reports the results of pairwise covers and termwise quadratizations.

Figures 8.14, 8.15 and 8.16 present, for the easiest examples specified on the x-axis, corresponding computing times on the y-axis. Figure 8.14 compares linearization methods only, and shows that none of the methods is clearly better than others, except maybe for SL-NoCuts, which is clearly much slower than the rest for two of the instances. Figure 8.15 compares quadratizations only. In this case the relative behavior of the quadratizations is similar to the other instance sets: termwise quadratizations seem to perform worse than paiwise covers. Figure 8.16 compares both linearization and quadratization methods and shows that there is no clear distinction between the performance of the different methods.

We remark that these results are not representative, because of the very small number of instances that we were able to test. The fact that several rather small instances, starting from 25_19 (n = 25, m = 3040), present memory problems for the standard linearization

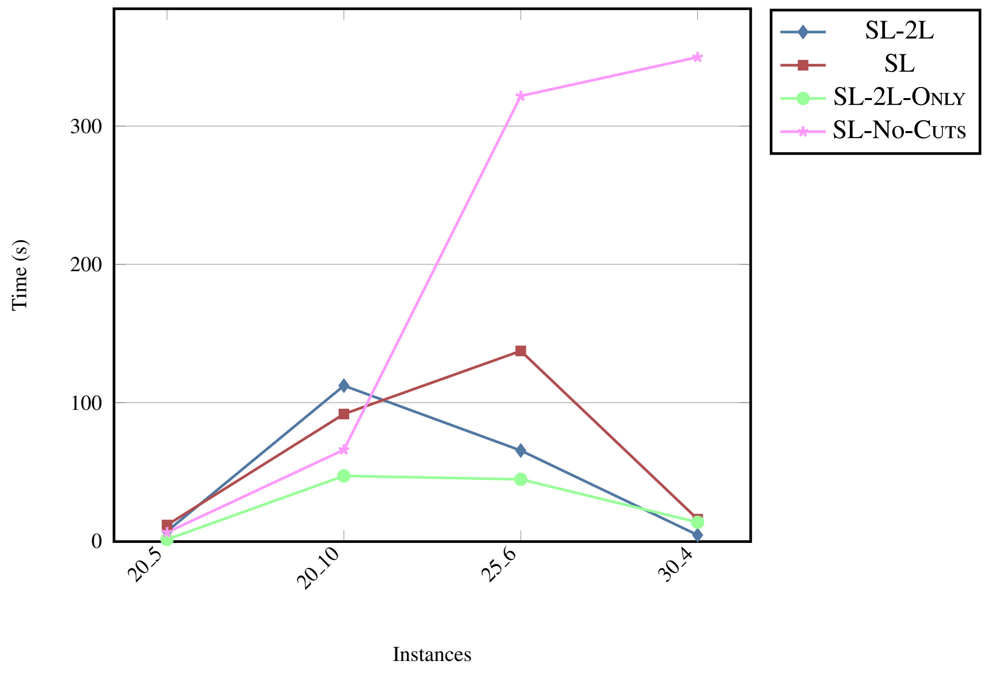


Figure 8.14: Autocorrelated sequences easiest instances: linearizations computing times.

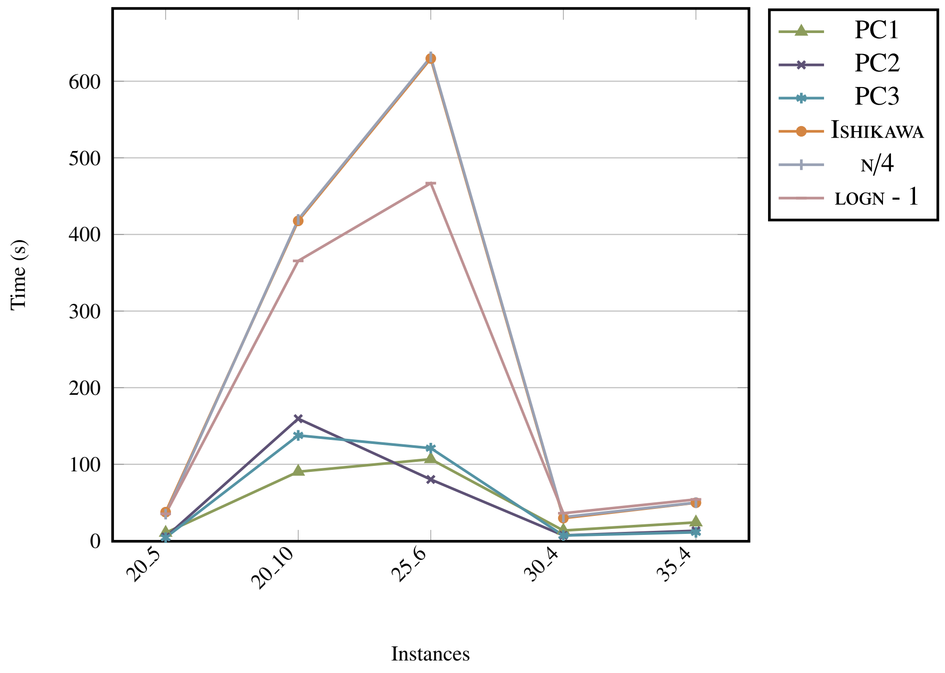


Figure 8.15: Autocorrelated sequences: quadratizations computing times.

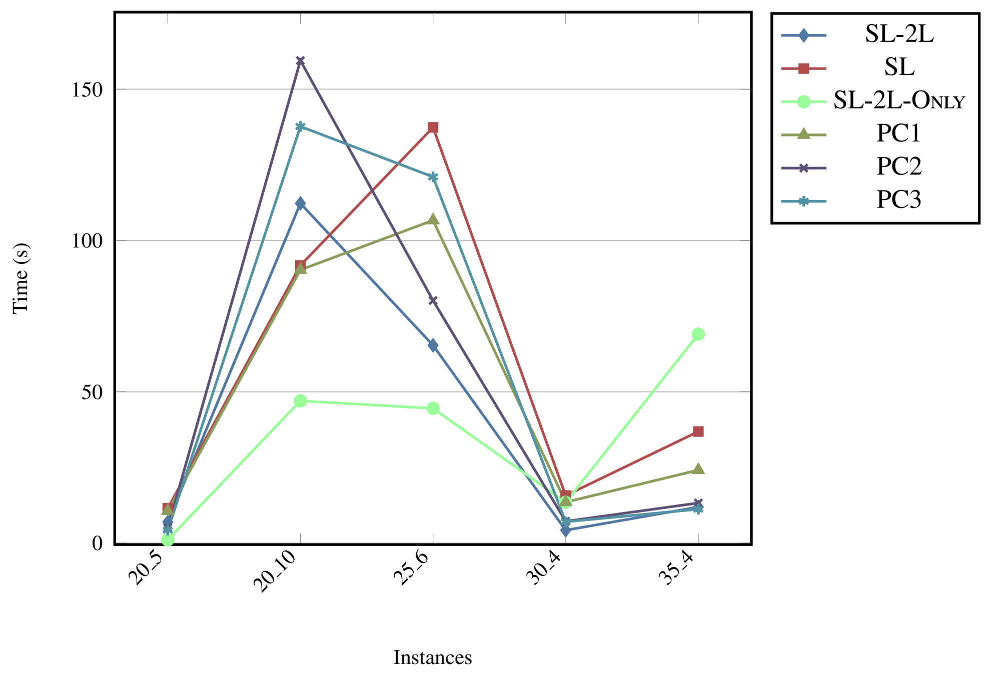


Figure 8.16: Autocorrelated sequences: linearizations and quadratizations computing times.

inequalities is again a motivation for the algorithm presented in Chapter 9, where only a subset of the 2-link inequalities is chosen in a heuristic way.

Concerning auxiliary variables and positive quadratic terms, similarly as to Vision instances, pairwise covers quadratizations introduce less auxiliary variables and less positive quadratic terms, which might again explain better computing times for pairwise covers (see Tables 8.22 and 8.23).

8.5 Conclusions

A first general conclusion is that linearization methods seem to be faster than quadratization methods for almost all classes of instances. Let us highlight again that this behavior heavily relies on the fact that we use CPLEX 12.7 to solve both linear and quadratic problems, which might be detrimental for quadratizations. Section 8.3 presented a thorough study aimed at improving the resolution times of quadratic reformulations by using persistencies in CPLEX 12.7, but it did not lead to significant results. Since the influence of the underlying linear and quadratic solver remains a crucial factor when comparing the performance of the reformulations, an essential future research direction would be to make a careful comparison of our current results with the same experiments using a different solver. Chapter 10 presents in detail several related ideas to explore.

Another important conclusion is that there might be some hidden factors influencing the behavior of the resolution methods due to the "blackbox" nature of CPLEX 12.7. For ex-

Table 8.22: AUTOCORRELATED SEQUENCES: number of y variables in quadratizations.

Instance			# y variables					
			Pairwise covers			Termwise quadratizations		
ID	n	m	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
20_5	20	207	70	70	70	117	117	117
20_10	20	833	135	135	135	678	678	678
25_6	25	407	110	110	110	272	272	272
30_4	30	223	84	84	84	109	109	109
35_4	35	263	99	99	99	129	129	129
25_13	25	1782	222	222	222	1535	1535	1535
30_8	30	926	182	182	182	714	714	714
40_5	40	447	150	150	150	257	257	257

Table 8.23: AUTOCORRELATED SEQUENCES: number of positive quadratic terms.

Instance			# positive quadratic terms					
			Pairwise covers			Termwise quadratizations		
ID	n	m	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
20_5	20	207	103	103	103	268	268	268
20_10	20	833	345	345	345	1395	1395	1395
25_6	25	407	193	193	193	608	608	608
30_4	30	223	111	111	111	246	246	246
35_4	35	263	131	131	131	291	291	291
25_13	25	1782	707	707	707	3132	3132	3132
30_8	30	926	402	402	402	1502	1502	1502
40_5	40	447	223	223	223	588	588	588

ample, we highlighted the fact that for some degree four instances, even though all termwise methods have the same equations, there exist relevant differences in computing times. The difference was particularly striking for Vision instances, where logn-1 behaved in a very different way than n/4 and Ishikawa in spite of having the exact same mathematical expressions. The correctness of the implementation was carefully checked; the methods only presented differences in the order in which terms and variables were added to the objective function in CPLEX 12.7, which we believe results in a different branch & bound tree thus leading to different computing times. This makes us insist on the fact that one cannot derive final conclusions from such computational experiments unless there is a consistent behavior over large sets of instances. This was the case for example for Vision instances, for which the three groups of methods tested (termwise quadratizations, pairwise covers quadratizations and linearization) consistently showed the same relative performances over the whole set of instances.

Concerning the classes of instances, it is clear that the relative performance of the different methods strongly depends on the nature of the instances. For example, for Vision instances, pairwise cover quadratizations performed better than the standard linearization alone. This is an interesting and rather unexpected outcome, contradicting our hypothesis that CPLEX 12.7 is better suited to solve linear than quadratic problems for this particular class of instances.

Another interesting result was obtained for Random high degree instances, where we compared the relative performance of the different termwise quadratization methods. For these instances, Ishikawa was more efficient than n/4 and logn-1, which is surprising considering the fact that Ishikawa requires a higher number of auxiliary variables. These results confirm an idea previously considered in the literature, namely, the fact that a small number of variables should not be the only criterion to evaluate the quality of a quadratization, but one should also consider properties like the number of positive quadratic terms as a vague measure of distance from submodularity.

Finally, when comparing quadratic reformulations of a different nature, it seems that pairwise covers are more efficient than termwise quadratizations in general. This behavior is consistent for almost all classes of instances. Also, for non-random instances, the standard linearization with 2-links is more efficient than the standard linearization alone. A common feature of pairwise covers and 2-link inequalities is that both types of methods take into account the interactions between monomials, by considering their intersections. This seems to confirm that a good property of a reformulation is the fact of better exploiting the structure of the original polynomial problem; a property that we did not consider at a theoretical level when designing the reformulations, but that is clearly brought out by the experiments.

In the same spirit, let us remark that in our implementation of the nonlinear model for Vision instances, we might already be overlooking some structural properties of the original problem. Consider a 2×2 window consisting of pixels $x_{11}, x_{12}, x_{21}, x_{22}$ and consider the assignment $x_{11} = 0, x_{12} = 0, x_{21} = 1, x_{22} = 1$, which is modeled as

$$30(1 - x_{11})(1 - x_{12})x_{21}x_{22}, \tag{8.2}$$

where the coefficient 30 represents the penalty specified in Table 8.1. In our implementation, equation (8.2) is developed into a multilinear expression

$$30\left(x_{21}x_{22} - x_{21}x_{22}x_{11} - x_{21}x_{22}x_{12} + x_{21}x_{22}x_{11}x_{12}\right).$$

The same procedure is applied to all assignments of variables $x_{11}, x_{12}, x_{21}, x_{22}$ and the multilinear terms of all expressions are added together afterwards. However, one could think of implementing these models *without* developing expression (8.2), but directly defining a quadratization or a linearization, by considering it as a monomial defined on the original x_i variables and on their complements $\bar{x}_i = 1 - x_i$. For example, the standard linearization inequalities for (8.2) would be $y \leq 1 - x_{11}$, $y \leq 1 - x_{12}$, $y \leq x_{21}$, $y \leq x_{22}$ and $y \geq 1 - x_{11} + 1 - x_{12} + x_{21} + x_{22} - 3$. The reasoning behind this idea is that the fact of not developing expression (8.2) leads to a more accurate model of the original application, which probably better captures structural properties like interactions between pixels. Moreover, we will require less auxiliary variables when reformulating expression (8.2) directly.

Chapter 9

A heuristic algorithm to select 2-link inequalities

This chapter presents a heuristic method to add a subset of relevant 2-link inequalities to the standard linearization instead of the whole set, as was done in the previous chapter. This idea is motivated by the fact that, even if the number of 2-link inequalities is only quadratic in the number of monomials of the instance, adding all inequalities might not be necessary to obtain a good computational performance. We present some preliminary results for Vision and Autocorrelated sequences instances.

9.1 Motivation

The computational experiments presented in Chapter 8 showed that adding 2-link inequalities to the standard linearization can yield great improvements for the resolution of some classes of problems, especially for Vision instances, and potentially for Autocorrelated sequences instances. The method presented in Chapter 8, denoted by SL-2L, consisted in adding *all* 2-link inequalities as a pool of user cuts to the branch & cut framework of CPLEX 12.7. This approach seems reasonable because one can generate at most $m(m-1)$ inequalities, where m is the number of monomials of a polynomial. Moreover, 2-links are only added to the pool for pairs of monomials having an intersection of at least two variables, because otherwise they are redundant. Since several instances contain many pairs of terms with an intersection of size smaller than two, the number of effectively added 2-links is usually much smaller than $m(m-1)$.

However, Section 8.4.3 of Chapter 8 showed that, for the largest Vision experiments with 9 318 terms, the fact of adding all 2-links resulted in large model creation times, which were not compensated by the fast resolution of the corresponding integer programs. Furthermore, the use of all 2-link inequalities for rather small instances in the Autocorrelated sequences set led to memory problems. Both time and memory issues could be addressed by using algorithms that do not require the addition of the whole set of 2-link inequalities to the standard linearization model, but only a subset of them.

We present here an approach that is motivated by the good performance shown by the quadratizations based on small pairwise covers, especially by PC2 and PC3 which accounted for the structure of the original problem by modeling subsets of variables that appear often in the original monomial set using the same auxiliary variable at each occurence. The intuitive idea behind 2-link inequalities is very similar, in the sense that they strengthen the formulation by accounting for interactions between pairs of monomials. Therefore, we will use the same idea of counting "popular" intersections and generate the 2-links corresponding only to subsets of variables that appear often as an intersection of two monomials. The same technical specifications and parameter settings of Chapter 8 apply here.

9.2 Description of the heuristic: "Most popular 2-links"

We describe here the heuristic algorithm "Most popular 2-links" in high level pseudocode. Given an integer N^{2L}, representing the number of inequalities that the user wants to generate, the algorithm will compute the most relevant N^{2L} inequalities, where the relevancy of an inequality is defined in the sense of the criterion "Most popular intersections first", described in Section 7.3. This set of inequalities, which is generally smaller than the whole set of 2-links, will be added as a pool of user cuts to the standard linearization model.

Let $\mathcal{S}$ be the set of subsets of variables associated with a multilinear polynomial. The first step of the pseudocode is the same as the first step of PC2 heuristic. The second step verifies that the number of inequalities requested by the user does not exceed the maximum number of 2-links that an instance admits. The last step adds the selected inequalities to the pool of cuts.

Step 1: Create the priority list `PrioritySubterms`

1. Compute all two-by-two intersections of monomials in $\mathcal{S}$ and count the number of times that a set of variables appears as a full intersection of two monomials in $\mathcal{S}$.

2. Create `PrioritySubterms` as follows: the first subterm is the set with most occurrences as a full intersection of two monomials in $\mathcal{S}$, the second subterm is the set with second most occurrences as a full intersection of two monomials in $\mathcal{S}$, and so on.

Step 2: Compute the effective number of 2-links to be added Let N^{eff} be the effective number of 2-links to be added to the formulation.

1. Compute $N^{\cap}$, the number of intersections of pairs of monomials with at least two variables.

2. If $N^{\cap} < N^{2L}$ set $N^{eff} = N^{\cap}$. Else set $N^{eff} = N^{2L}$.

Step 3: Generate N^{eff} 2-link inequalities following the order of `PrioritySubterms`

1. For each element V in `PrioritySubterms`
    - For each pair of monomials $S, T \in \mathcal{S}$ with intersection $S \cap T = V$, if the pool of cuts contains less than N^{eff} inequalities, then add the 2-link inequalities corresponding to S and T to the pool, else STOP.

9.3 Results

This section presents the computing times obtained by solving Vision and Autocorrelated sequences instances using the heuristic "Most popular 2-links". We chose not to test the heuristic on Random same degree and Random high degree instances because it was not clear whether adding all 2-link inequalities was helpful with respect to the plain standard linearization (see [41] and Chapter 4). However this case could be further investigated, since one could make the hypothesis that adding the whole set of 2-links is too expensive for Random same degree and Random high degree instances, but adding only a subset of the inequalities might be useful.

In all figures, missing data points represent resolution times larger than one hour, which is the time limit that was set for all experiments.

9.3.1 Vision instances

Let us consider the set of Vision instances. First, in order to have an idea of the total number of 2-links for each instance, Table 9.1 shows the number of 2-links that can be generated for the Vision instances. As a reminder, 2-links are only generated for pairs of monomials having at least two variables in common. Since the polynomials of the Vision instances have many terms of degree 1 or 2, there are far less than $m(m-1)$ inequalities. Notice that the number of 2-links only depends on the size of the image, because the choice of the base image and the perturbation do not impact the nonlinear part of the model.

Table 9.1: VISION: maximum number of 2-links per image size.

Image size	n	m	Total # 2-links
10x10	100	668	5 184
15x15	225	1 598	12 824
20x20	400	2 928	23 864
25x25	625	4 658	38 304
30x30	900	6 788	56 144
35x35	1 225	9 318	77 384

Table 9.2: VISION: number of 2-links per level.

Instance	# 2-links per level (% of total)					
Image size	Level 1	Level 2	Level 3	Level 4	Level 5	Total # 2-links
10x10	1000 (19%)	2000 (39%)	3000 (58%)	4000 (77%)	5000 (96%)	5 184
15x15	2500 (19%)	5000 (39%)	7500 (58%)	10000 (78%)	12500 (97%)	12 824
20x20	4700 (20%)	9400 (39%)	14100 (59%)	18800 (79%)	23500 (98%)	23 864
25x25	7600 (20%)	15200 (40%)	22800 (60%)	30400 (79%)	38000 (99%)	38 304
30x30	11200 (20%)	22400 (40%)	33600 (60%)	44800 (80%)	56000 (99%)	56 144

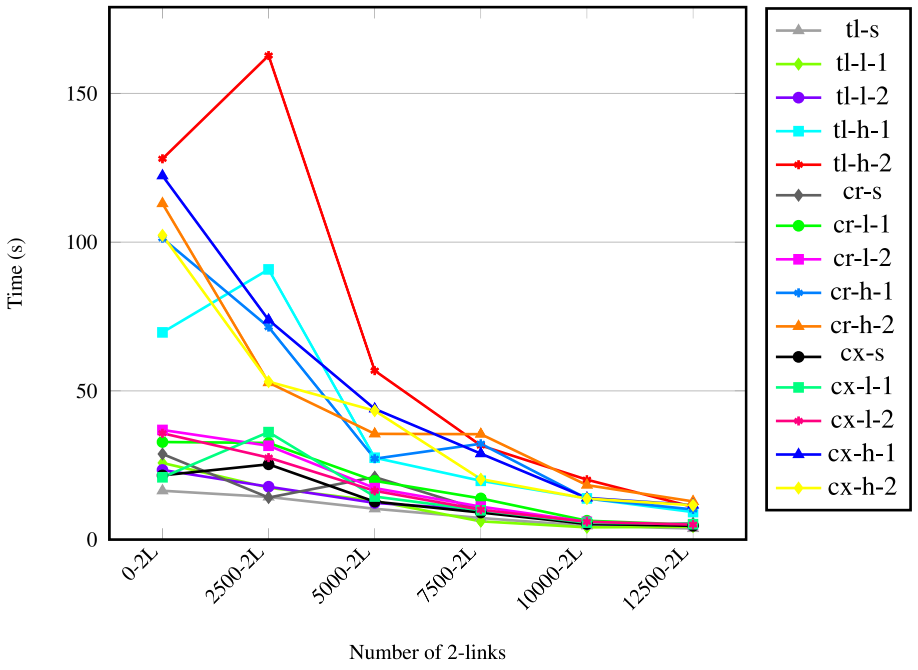


Figure 9.1: Vision 15×15 ($n = 225, m = 1598$): 2-links heuristic computing times.

In the computational experiments we have defined five *levels* of 2-link inequalities to add to the formulation, which are roughly 20%, 40%, 60%, 80% and 100% of the total number of 2-links. Table 9.2 presents the exact numbers of 2-links defined in each level.

The behavior of the Vision instances is very similar for all image sizes, thus we only present here Figure 9.1 for 15×15 images, while the rest of figures can be found in Appendix G. The tables corresponding to all instances are in Appendix D. The x-axis of Figure 9.1 has a mark for each level of 2-links and the y-axis gives the resolution times in seconds of the instances in the legend. The first data point in the x-axis corresponds to the resolution times of SL (without 2-links) reported in Chapter 8.

A general trend that can be observed is that the higher the number of 2-links added to the problem, the smaller the computing times become. Figure 9.1 and the figures in Appendix D also highlight the fact that the hardest instances are those with high levels of perturbation (tl-h-1, tl-h-2, cr-h-1, cr-h-2, cx-h-1, cx-h-2), no matter the size of the image.

In some cases there is an increase in computing times when going from lower to higher levels of 2-links. For many instances, adding only 20% of 2-links seems to be worse than adding no 2-links at all. This observation should however be taken carefully, because the tests with no 2-links were run as a separate batch and in another period of time than the remaining five levels, meaning that the performance of the machine could have slightly changed due factors out of our control. In order to have a fair comparison the tests should be re-lauched with the six levels, including the plain SL, in a same batch.

The fluctuations between time measurements of the first levels containing 0%, 20% and

40% of 2-links seem to increase with the size of the instances. For larger instances, it seems that starting from a certain level of 2-links added, the time measurements stabilize, at least for instances with a high perturbation. For the hardest instances with images of size 30×30 in particular, there is a dramatic difference between adding 60% and 80% of the 2-links (see Figure G.9), which indicates that many inequalities are added to obtain satisfactory computing times.

9.3.2 Autocorrelated sequences instances

This section presents the results of the 2-links heuristic for the set of Autocorrelated sequences instances. Table 9.3 shows the number of 2-links that can be generated for the Autocorrelated sequences instances. Notice that, compared to Vision instances with a similar number of terms, Autocorrelated sequences instances potentially admit a much higher number of 2-link inequalities. One reason for this is that Autocorrelated sequences instances have less terms of degree 1 or 2. On the other hand Autocorrelated sequences instances are much denser, meaning that they have a larger ratio $\frac{m}{n}$. For example, Vision instances of size 20x20 have 2 928 monomials and use 400 variables, while Autocorrelated sequences instance 30_15 has a similar number of terms (2 944) but only 30 variables. Intuitively, the denser instances Autocorrelated sequences will contain more pairs of monomials with an intersection of at least two variables, because there are less choices of variables to be included in each term. As mentioned in Chapter 8, the higher density of terms in Autocorrelated sequences instances might explain their intrinsic difficulty.

For Autocorrelated sequences instances, each instance has a different number of 2-links. As for Vision instances, five levels of 2-links have been defined, and each of these levels contains approximately 20%, 40%, 60%, 80% and 100% of the total number of 2-links. Table 9.4 presents the exact number of 2-links corresponding to each level.

Table 9.5 presents the detailed computing times for the standard linearization together with the specified number of 2-links, Figure 9.2 gives a graphical representation for the easiest instances and Figure 9.3 presents the results of the most difficult instances.

The results in Figure 9.2 are not as significant as the results for Vision instances, due to the small number of instances that we were able to test. There seems to be a trend of decreasing computing times for higher percentages of 2-links, but this trend is not confirmed for instance 20_10, for which adding 100% presents the worst performance. Thus, no relevant conclusions can be drawn from these experiments. The results in Figure 9.3 are even less informative, due to the high quantity of missing data points, which represent problems requiring more than one hour of resolution.

As a final experiment for the Autocorrelated sequences instances, we fixed a very large number of 2-link inequalities (100 000) to be added to the standard linearization and let CPLEX 12.7 solve the reformulation for one hour, in order to retrieve the final gap of the branch & cut procedure. The inequalities to add are chosen using the heuristic described in Section 9.2. Note that even if 100 000 inequalities are a very large number compared to Vision instances (which could admit 77 384 2-links for the largest instances), they do not even represent 0.5% of the total number of inequalities for the largest Autocorrelated sequences

Table 9.3: AUTOCORRELATED SEQUENCES: maximum number of 2-links per instance.

Id	n	m	# 2-links	Id	n	m	# 2-links
20_5	**20**	**207**	**3 136**	45_5	45	507	8 086
20_10	**20**	**833**	**54 704**	45_11	45	2 813	235 064
25_6	**25**	**407**	**10 646**	45_23	45	10 776	2 257 348
25_13	**25**	**1 782**	**172 878**	45_34	45	18 348	5 288 906
25_19	25	3 040	426 302	45_45	45	21 993	7 012 544
25_25	25	3 677	581 028	50_6	50	882	24 046
30_4	**30**	**223**	**2 076**	50_13	50	4 457	479 378
30_8	**30**	**926**	**45 038**	50_25	50	14 412	3 363 356
30_15	30	2 944	353 084	50_38	50	25 446	8 309 036
30_23	30	5 376	958 212	50_50	50	30 271	10 872 808
30_30	30	6 412	1 270 298	55_6	55	977	26 726
35_4	**35**	**263**	**2 466**	55_14	55	5 790	696 466
35_9	35	1 381	81 314	55_28	55	19 897	5 303 858
35_18	35	5 002	766 112	55_41	55	33 318	11 986 060
35_26	35	8 347	1 765 148	55_55	55	40 402	16 141 280
40_5	**40**	**447**	**7 096**	60_8	60	2 036	103 238
40_10	40	2 053	149 384	60_15	60	7 294	960 764
40_20	40	7 243	1 281 696	60_30	60	25 230	7 321 614
40_30	40	12 690	3 169 270	60_45	60	43 689	17 394 212
40_40	40	15 384	4 285 626	60_60	60	52 575	23 126 262

Table 9.4: AUTOCORRELATED SEQUENCES: number of 2-links per level.

Instance			Number of 2-links added for each %				
Id	n	m	20%	40%	60%	80%	100%
20_5	20	207	600	1 200	1 800	2 400	3 136
20_10	20	833	11 000	22 000	33 000	44 000	54 704
25_6	25	407	2 100	4 200	6 300	8 400	10 646
30_4	30	223	400	800	1 200	1 600	2 076
35_4	35	263	500	1 000	1 500	2 000	2 466
25_13	25	1782	34 600	69 200	103 800	138 400	172 878
30_8	30	926	9 000	18 000	27 000	36 000	45 038
40_5	40	447	1 400	2 800	4 200	5 600	7 096

Table 9.5: AUTOCORRELATED SEQUENCES: 2-links heuristic computing times.

Instance			Resolution time (secs) for a given % of 2-links				
Id	n	m	20%	40%	60%	80%	100%
20_5	20	207	7.13	4.24	6.22	6.08	6.94
20_10	20	833	66.55	95.23	80.09	76.88	112.27
25_6	25	407	125.98	124.44	70.97	85.25	65.36
30_4	30	223	7.59	8.83	8.00	5.02	4.24
35_4	35	263	23.09	22.61	19.16	9.94	11.92
25_13	25	1782	2675.13	> 3600	1971.77	> 3600	1645.20
30_8	30	926	> 3600	> 3600	> 3600	> 3600	2743.51
40_5	40	447	1492.31	2275.99	1235.26	1197.86	1321.61

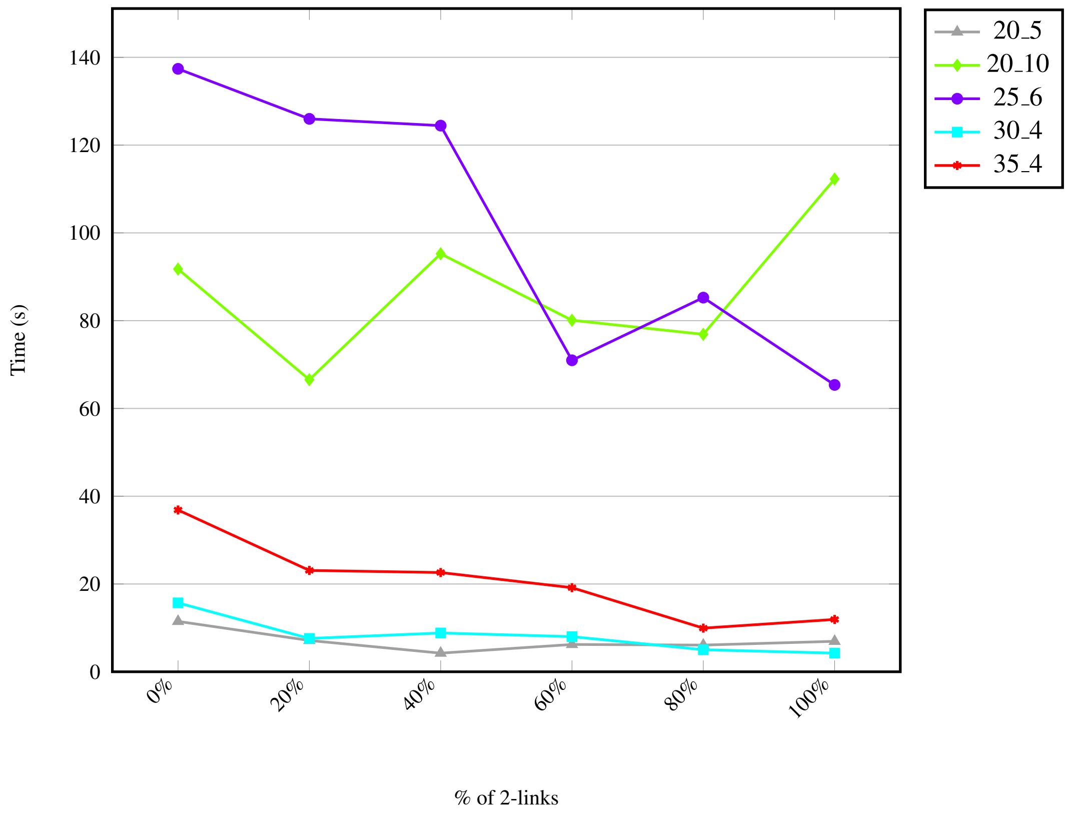


Figure 9.2: AUTOCORRELATED SEQUENCES easiest instances: 2-links heuristic computing times.

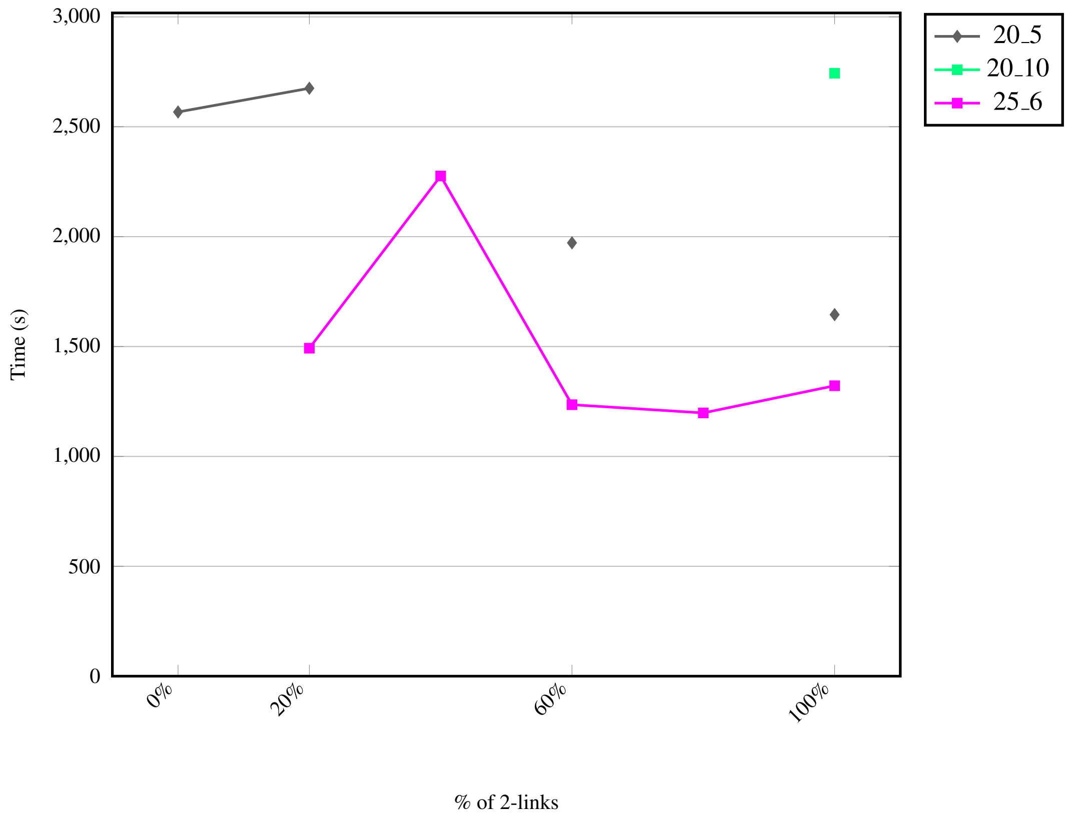


Figure 9.3: Autocorrelated sequences difficult instances: 2-links heuristic computing times.

Table 9.6: AUTOCORRELATED SEQUENCES: gap after 3600s running SL + 100 000 2-links.

Id	n	m	Gap (%)	Id	n	m	Gap (%)
20_5	20	207	0	45_5	45	507	9.28
20_10	20	833	0	45_11	45	2813	215.33
25_6	25	407	0	45_23	45	10776	551.37
25_13	25	1782	0	45_34	45	18348	894.76
25_19	25	3040	184.77	50_6	50	882	85.97
25_25	25	3677	177.66	50_13	50	4457	298.25
30_4	30	223	0	50_25	50	14412	763.54
30_8	30	926	36.4	55_6	55	977	95.05
30_15	30	2944	183.65	55_14	55	5790	416.71
30_23	30	5376	229.87	55_28	55	19897	842.24
30_30	30	6412	338.22	60_8	60	2036	147.1
35_4	35	263	0	60_15	60	7294	479.26
35_9	35	1381	89.66	60_30	60	25230	993.9
35_18	35	5002	321.73	45_45	45	21993	$\geq$ 1000
35_26	35	8347	439.51	50_38	50	25446	$\geq$ 1000
40_5	40	447	0	50_50	50	30271	$\geq$ 1000
40_10	40	2053	145.69	55_41	55	33318	$\geq$ 1000
40_20	40	7243	369.03	55_55	55	40402	$\geq$ 1000
40_30	40	12690	776.19	60_45	60	43689	$\geq$ 1000
40_40	40	15384	860.37	60_60	60	52575	$\geq$ 1000

instances, which can admit up to 23 126 262 inequalities.

Table 9.6 presents the final gaps for every instance. For the last seven listed instances, the CPLEX log did not even show a gap after one hour, which means that it is larger than 1000%.

Figure 9.4 presents the optimality gaps for those instances with a gap smaller than 1000%. It can be appreciated from both Figure 9.4 and Table 9.6 that for a vast majority of the instances, the gaps are still very large after one hour of execution, which indicates that the standard linearization is not a suitable method to solve these instances, even when it is possibly improved with 2-link inequalities. This confirms that AUTOCORRELATED SEQUENCES instances remain very challenging, as it is shown in `http://polip.zib.de/autocorrelated_sequences/`.

9.4 Conclusions

As a general conclusion, the experiments on the 2-links heuristic show that for VISION instances resolution times tend to be shorter for larger pools of user cuts. For the hardest instances with a high perturbation (tl-h-1, tl-h-2, cr-h-1, cr-h-2, cx-h-1, cx-h-2), computing times are only significantly reduced for very large pools of user cuts containing around 80% or almost all inequalities. (See for example figures for 25×25 and 30×30 images in Appendix G.) Resolution times of easier instances already drop for smaller pools; for example, for 10×10 and 15×15 images, adding only $20 - 40\%$ of 2-links reduces computing times

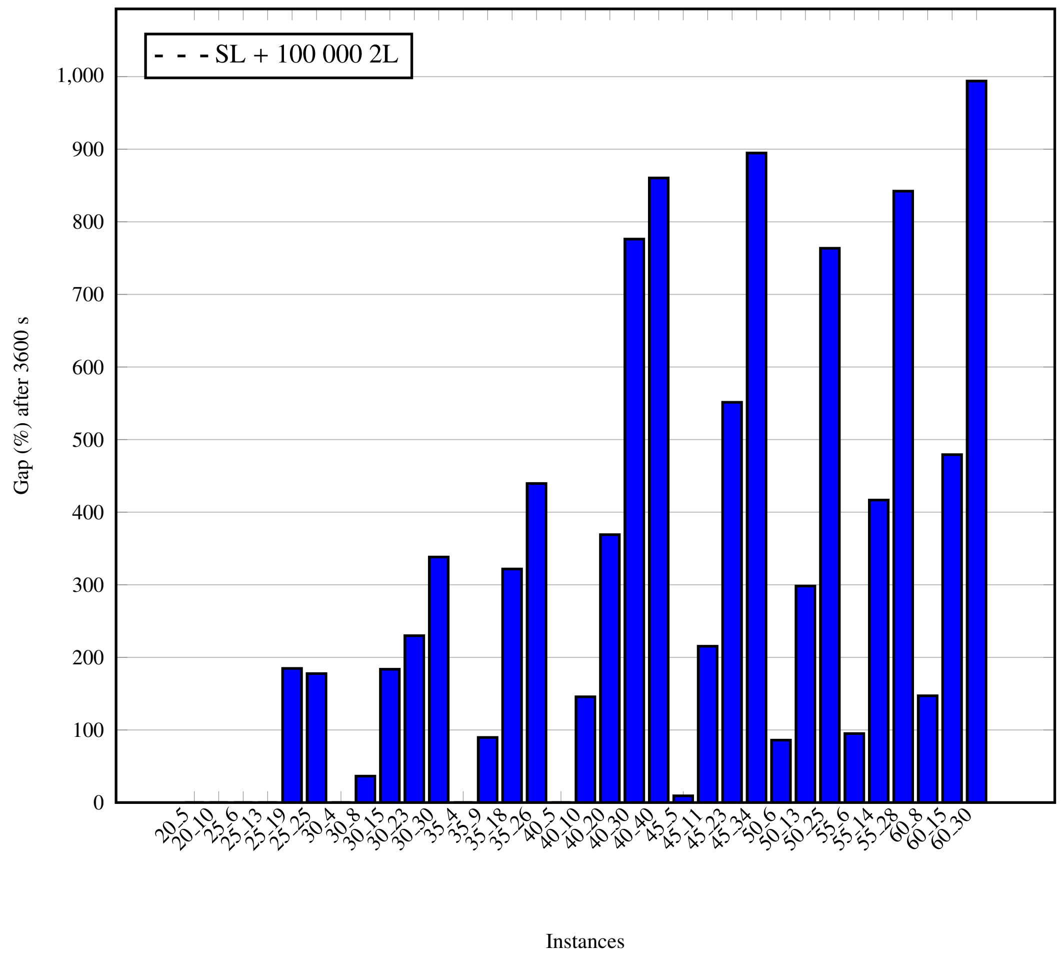


Figure 9.4: Autocorrelated sequences: gap after 3600s running SL + 100 000 2-links.

by half in many cases.

For Autocorrelated sequences the results are not conclusive, and even with a very large number of 2-links added heuristically, the gaps after one hour of resolution are very large, which seems to indicate that linearization methods are not well-suited for these particularly challenging instances.

A possible extension of the “Most popular 2-links” heuristic is the definition of alternative criteria to choose the most relevant 2-links. For example, one could select the 2-links randomly, or give priority to inequalities corresponding to subsets of terms that do *not* appear very often in the original monomial set, as opposed to our current implementation. This criterion might seem counter-intuitive at first, but it relies on the idea that the branch and cut implementation of CPLEX might be already taking care of variables that appear most frequently by branching, thus the addition of cuts corresponding to “unpopular” sets of variables could help by adding complementary information to the algorithm. As mentioned in Section 8.4.3, another interesting approach would be the definition of a separation algorithm for the 2-link inequalities, which could lead to a better practical performance, in terms of time and memory consumption, by dynamically selecting the inequalities to add in the branch and cut algorithm. Finally, the heuristic should also be tested on the sets of instances Random same degree and Random high degree. For these instances, the results of Chapter 4 showed that the addition of the whole set of 2-links did not improve resolution times, however it is possible that times were slow because the set of 2-links was too large, a hypothesis that could be tested using the heuristic method presented in this chapter.

Chapter 10

Conclusions

This thesis considered the problem of minimizing a pseudo-Boolean function, that is, a real-valued function f in binary variables $x \in \{0, 1\}^n$, without additional constraints. We examined resolution approaches based on the idea of reformulating the original nonlinear problem into a linear or a quadratic one, by introducing auxiliary variables. A crucial assumption made throughout the thesis is that pseudo-Boolean functions are represented as multilinear polynomials; this assumption relying in turn on the well-known fact that a pseudo-Boolean function can be associated to a unique multilinear expression.

Part I focused on linearizations, more precisely on the *standard linearization*, a techinque that consists in introducing a set of auxiliary variables, each one representing a nonlinear monomial, where the correspondence between variables and monomials is enforced using linear constraints. Chapter 3 characterized multilinear functions for which the standard linearization inequalities completely describe the convex hull of their integer feasible points. This characterization was given in terms of the balancedness of the matrix defining the standard linearization constraints and in terms of the acyclicity of the hypergraph associated with a multilinear polynomial, and was derived from a more general result considering the signs of the coefficients. Chapter 4 defined the class of 2-link inequalities, modeling interactions between pairs of monomials with a non-empty intersection. For the case of functions with exactly two higher-degree monomials, the standard linearization together with the 2-link inequalities is a perfect formulation. Moreover, computational experience shows that the 2-link inequalities strengthen the standard linearization formulation and can be very useful to solve non-random instances.

Part II focused on quadratic reformulations. Given a pseudo-Boolean function f, a quadratization is a quadratic function g depending on the original variables and on a set of auxiliary variables such that when g is minimized over the auxiliary variables, f is recovered point by point. This is a broad definition encompassing many quadratizations with different properties. Chapter 6 focused on the question of defining quadratizations for pseudo-Boolean functions using a small number of auxiliary variables. Among other results, we defined a quadratization for the positive monomial, using only a logarithmic number of variables, improving previously published upper bounds by orders of magnitude. A matching lower bound was also provided, proving that our upper bound is best possible. This result is especially remarkable because a quadratization for any pseudo-Boolean function can be defined by separately quadratizing the monomials of its multilinear representation. The result for the positive monomial was derived from a more general result for exact k-out-of-n and at least k-out-of-n functions, for which logarithmic bounds on the number of auxiliary variables significantly improve previously published results. Bounds for other classes of functions, such as symmetric functions or functions with many zeros were also established. Chapter 7 considered a quadratization method of a different nature, which splits each monomial into two subterms that are associated with an auxiliary variable each. Defining such a quadratization with smallest number of auxiliary variables is an NP-hard problem, which we addressed by defining heuristic algorithms based on the idea of substituting sets of variables appearing in more than one monomial of the original mutlilinear expression by the same auxiliary variable.

Part III focused on computational experiments. Chapter 8 presented the results of an extensive set of experiments aimed at comparing the resolution times of several linear and quadratic reformulations when using CPLEX 12.7. A first conclusion of this chapter is that linear reformulations are solved for most instances much faster than quadratic reformulations when using CPLEX 12.7. This is in a sense natural, because advances in quadratic programming are generally much more recent than in linear programming. However, this also highlights the fact that our results are heavily influenced by the choice of underlying linear and quadratic solvers, and cannot be considered definitive without repeating the experiments using other linear and quadratic resolution techniques. As a second important conclusion, let us remark that the performance of the tested methods strongly depends on the set of instances. For example, for non-random instances, methods better exploiting the structure of the underlying application are much more efficient than the rest, to the extent that quadratic reformulations with this property are solved faster than the standard linearization. Chapter 9 presented a heuristic algorithm to select a subset of interesting 2-link inequalities, with the objective of dealing with long model creation times and memory problems encountered when adding of the whole set of 2-link inequalities to the standard linearization.

Further research on linearizations

Several future research directions emerge in the context of linear reformulations. Let us first remark that we restricted ourselves to the standard linearization, which is a very precise linearization method. Part I could be generalized by considering other linearization techniques. Nevertheless, several interesting points can be considered in our setting.

A first natural question is whether the 2-link inequalities can be generalized. Del Pia and Khajavirad provided a partial answer to this question by defining the class of *flower inequalities*, generalizing the 2-links to the case of several monomials such that a particular monomial has an intersection of at least two variables with the rest of monomials. The flower inequalities, together with the standard linearization define a complete description for certain acyclic hypergraphs [47]. To the best of our knowledge it is still an open question to characterize all cases for which the 2-link inequalities together with the standard linearization provide a perfect formulation. Moreover, perfect formulations for functions with exactly three or four higher-degree monomials have not been specifically addressed in the literature.

From a computational point of view, an interesting idea that has not been implemented is the definition of a separation algorithm to avoid the addition of the whole set of 2-link inequalities to the reformulation. Indeed, even if the 2-links are only in quadratic number in the size of the input, the addition of all inequalities becomes impractical for certain instances of reasonably large size. The main objective of implementing a separation procedure would be to improve computing times and memory consumption in practice, for example by dynamically generating effective inequalities during the execution of the branch and cut algorithm, such as most violated inequalities or most relevant ones.

Further research on quadratizations

Several research directions concerning quadratizations arise from the results presented in Part II. From a theoretical perspective, our contributions mainly concerned the question of minimizing the number of auxiliary variables required to define quadratizations in termwise procedures. Most lower and upper bounds presented in Chapter 6 are tight in the sense of having the same order of magnitude. However, it would be nice to close the unit gap remaining for exact k-out-of-n functions and the slightly larger gap for at least k-out-of-n functions. Another interesting question would be to consider further classes of pseudo-Boolean functions to derive similar results.

A more general aspect worth investigating is the definition of other desirable properties for quadratizations, such as those mentioned in Chapter 5. We only briefly analyzed the number of positive quadratic terms in the experimental results of Chapter 8, but a thorough analysis of the influence of this magnitude on computing times would be worth considering.

One of the main reasons as to why quadratic reformulations are widely used by the computer vision community is the use of *persistencies*, which we briefly described in Chapter 8. Given a quadratic binary optimization problem, persistencies allow to fix a subset of the variables to their provable optimal values by solving its continuous standard linearization. This approach is very useful in computer vision where models contain millions of variables and persistencies can substantially reduce the size of these problems. Moreover, the resolution of the continuous standard linearization of the quadratic problem provides the *roof dual* bound [65], which has proven very useful for image restoration and also for other problems [25]. We made a naive implementation to calculate persistencies, by directly solving the corresponding linear problem using CPLEX 12.7. Vision instances did not show significant improvements in computing times, and the use of persistencies seemed to be even detrimental with our implementation. However, for a subset of the Random high degree instances, the use of persistencies greatly improved computing times. The set of instances showing this behavior was not significant enough to use persistencies in all experiments, but it would be interesting to investigate further why such improvements were obtained for that precise set of instances. Moreover, several related ideas remain to be explored, such as the use of roof duality bounds as a bounding procedure in a the branch & bound framework of CPLEX (or another solver).

Boros et al. [25] proposed a very efficient algorithm based on network flows which returns the largest possible subset of variables to fix; this algorithm has not been tested on our instances, and this is a question to address in the near future. Finally, one could also think of generalizing the persistency result to higher-degree binary problems.

General considerations on computational experiments

Let us highlight some ideas specific to the computational experiments presented in Part III.

A major drawback of the computational experiments presented in Chapter 8 is that the reported resolution times heavily depend on the choice of the software used to solve the linear and quadratic reformulations. We chose CPLEX 12.7 because it is an extensively used commercial solver. However, it might not be able to exploit interesting properties of quadratic problems, leading to a false impression that linear reformulations are always better than quadratic ones. Several ideas can be implemented in order to draw a more complete picture of which reformulations are best. A straightfoward idea would be to run the same set of experiments using different commercial solvers for linear and quadratic problems. Another possible approach is to use algorithms specifically designed to handle the quadratic case, such as a convexification-based resolution method [13, 14]. An important challenge of this approach is the interaction between both implementations. We consider this as a subject of investigation in the near future, in the context of a collaboration with the authors.

Furthermore, another disadvantage of using a "blackbox" solver is that little control of the algorithms is left to the user, despite the large set of parameters to tune. Indeed, we observed in Section 8.4.1 that all termwise quadratizations have the same mathematical expression for Vision instances, but that in some cases the behavior of one of the methods was very different from the other two. We believe that this inconvenient result is a consequence of small differences in the order of the terms and variables when coding the equations, which can result in different structures of the branch & bound tree. A related aspect is the fact that in a "blackbox" solver, the user might not be aware of some random decisions that are taken by CPLEX during the resolution of an integer program. For example, when deciding the next branch to explore in a branch & bound algorithm, ties might be broken with a random draw. As mentioned in the release notes of CPLEX 12.7, the solver offers now the possibility of running a model several times and compiling statistical tests on the average runtimes and standard deviations of each execution. Such tests have not been carried out and could be considered to further clarify the reason of unsatisfactory results as the above.

One could also think of several improvements concerning particular classes of instances. Regarding Vision instances, all models are degree four polynomials, because interactions of pixels are taken into account for windows of size 2×2. This model is very limited, and a direct extension of the existing implementation would be to consider larger windows, or even different choices for pixel neighborhoods, like crosses. Another important improvement in the way that Vision instances are handled concerns the way in which the polynomials themselves are defined. As mentioned in the conclusions of Chapter 8, the fact of developing the expressions corresponding to the assignments of values to pixel windows into a multilinear polynomial might already result in a less accurate model of the underlying problem.

Finally, the preliminary experiments on the set of Autocorrelated sequences instances did not lead to firm conclusions. Autocorrelated sequences is clearly the most difficult set of instances. We included the results of these first experiments in this thesis to remark the challenge that they represent to our reformulation methods, but it is an open question to understand why these instances are so difficult to solve.

As a final future research question let us highlight again that throughout the thesis we assumed that a pseudo-Booelan function was represented by its unique multilinear polynomial. A key future research direction would be to study reformulation methods that do *not* rely on this assumption. This is an interesting question both from a theoretical and from a computational perspective. A motivation to further investigate this point is the application of joint supply chain management design and inventory management presented in Chapter 1. This problem is modeled as a nonlinear optimization problem in binary variables containing square roots in the objective function, and is particularly interesting because it is a real-world application representing a significant challenge for the reformulation methods presented in this thesis.

Final word

There exists a current trend towards a better understanding of nonlinear problems in several fields such as mathematical programming, optimization and operations research and also in more distant areas like computer vision. We believe that the interest in nonlinear problems will continue to grow in the near future because of their significant modeling possibilities.

This thesis aimed at contributing our grain of sand in the context of unconstrained nonlinear binary optimization. Of course, linear and quadratic reformulation techniques should be considered among many other methods such as reformulation-linearization techniques, semi-definite programming, or branch & bound algorithms, and it is not clear whether one of these methods is clearly better than the rest. Nonetheless, this thesis shows that there are many interesting questions to investigate in the context of reformulations.

We explored theoretical questions, such as valid inequalities, perfect formulations, and properties to define interesting reformulations such as minimizing the number of auxiliary variables. From a computational point of view, the results presented in this thesis show that many of the considered methods can be very competitive and solve large problems in reasonable time. An important point illustrated by our experiments is that it is becoming increasingly difficult to properly compare the computational performance of different methods by relying on commercial solvers, because they are becoming more and more complex and it is difficult to identify the features of the "blackbox" influencing the results. This thesis also showed that it is fundamental to consider both theoretical and computational aspects to gain a better understanding of a problem. Finally, the previous literature on which we built part of our results comes from the computer vision community, showing the importance of collaboration between fields, not only for applied but also for methodological advances.

Appendix A

Tables: Persistencies for Vision instances

The tables in this appendix present the total number of variables of problem (QP), the percentages of fixed variables using persistencies and the computing times of quadratization methods with and without using persistencies for Vision instances with images of sizes 10×10, 20×20, 25×25 and 30×30. The conclusions of these experiments are analogous to those described in Section 8.3 for 15×15 images.

Table A.1: Vision 10×10 ($n = 100, m = 668$): total number of variables in problem (QP).

	Pairwise covers			Termwise quadratizations		
	PC1	PC2	PC3	Ishik	n/4	logn-1
# variables QP	434	421	421	505	505	505

Table A.2: Vision 10×10 ($n = 100, m = 668$): percentage of variables fixed using persistencies.

Instance (10×10)		Fixed variables (%)					
		Pairwise covers			Termwise quadratizations		
Base	Quality	PC1	PC2	PC3	Ishik	n/4	logn-1
top left rect	sharp	18.89	38.48	38.48	0.00	0.00	0.00
top left rect	low	18.89	38.48	38.48	0.00	0.00	0.00
top left rect	low	18.89	38.48	38.48	0.00	0.00	0.00
top left rect	high	18.89	38.48	38.48	0.00	0.00	0.00
top left rect	high	18.89	38.48	38.48	0.00	0.00	0.00
centre rect	sharp	18.89	38.48	38.48	0.00	0.00	0.00
centre rect	low	18.89	38.48	38.48	0.00	0.00	0.00
centre rect	low	18.89	38.48	38.48	0.00	0.00	0.00
centre rect	high	18.89	38.48	38.48	0.00	0.00	0.00
centre rect	high	18.89	38.48	38.48	0.00	0.00	0.00
cross	sharp	18.89	38.48	38.48	0.00	0.00	0.00
cross	low	18.89	38.48	38.48	0.00	0.00	0.00
cross	low	18.89	38.48	38.48	0.00	0.00	0.00
cross	high	18.89	38.48	38.48	0.00	0.00	0.00
cross	high	18.89	38.48	38.48	0.00	0.00	0.00

Table A.3: VISION 10×10 ($n = 100, m = 668$): resolution times using persistencies.

Instance (10×10)		Exec. times (secs)					
		Pairwise covers					
Base	Quality	PC1	PC1-pers	PC2	PC2-pers	PC3	PC3-pers
TOP LEFT RECT	SHARP	2.89	5.44	3.02	5.28	3.08	6.33
TOP LEFT RECT	LOW	3.58	3.66	4.13	**3.89**	3.89	**3.73**
TOP LEFT RECT	LOW	3.72	3.47	4.03	4.09	3.41	3.69
TOP LEFT RECT	HIGH	8.27	8.27	7.16	**7.05**	7.25	7.53
TOP LEFT RECT	HIGH	9.55	9.7	8.63	8.94	8.72	**8.45**
CENTRE RECT	SHARP	3.75	**3.69**	3.64	3.76	3.61	3.98
CENTRE RECT	LOW	4.06	**4.01**	3.38	3.50	3.50	3.52
CENTRE RECT	LOW	3.70	3.77	3.53	3.69	3.61	3.76
CENTRE RECT	HIGH	6.45	6.50	7.94	8.34	7.25	**6.77**
CENTRE RECT	HIGH	5.83	5.92	5.72	5.72	5.81	6.13
CROSS	SHARP	3.41	3.47	4.17	**4.09**	4.19	**4.05**
CROSS	LOW	3.94	**3.84**	3.80	**3.63**	3.70	3.72
CROSS	LOW	4.23	4.47	4.09	**4.00**	4.08	4.11
CROSS	HIGH	5.44	5.49	5.47	**5.45**	5.53	6.05
CROSS	HIGH	6.28	**6.26**	6.22	**6.14**	6.14	6.14

Table A.4: VISION 20×20 ($n = 400, m = 2928$): total number of variables in problem (QP).

	Pairwise covers			Termwise quadratizations		
	PC1	PC2	PC3	ISHIK	N/4	LOGN-1
# variables QP	1864	1831	1831	2205	2205	2205

Table A.5: VISION 20×20 ($n = 400, m = 2928$): percentage of variables fixed using persistencies.

Instance (20×20)		Fixed variables (%)					
		Pairwise covers			Termwise quadratizations		
Base	Quality	PC1	PC2	PC3	ISHIK	N/4	LOGN-1
TOP LEFT RECT	SHARP	19.42	39.43	39.43	0.00	0.00	0.00
TOP LEFT RECT	LOW	19.42	39.43	39.43	0.00	0.00	0.00
TOP LEFT RECT	LOW	19.42	39.43	39.43	0.00	0.00	0.00
TOP LEFT RECT	HIGH	19.42	39.43	39.43	0.00	0.00	0.00
TOP LEFT RECT	HIGH	19.42	39.43	39.43	0.00	0.00	0.00
CENTRE RECT	SHARP	19.42	39.43	39.43	0.00	0.00	0.00
CENTRE RECT	LOW	19.42	39.43	39.43	0.00	0.00	0.00
CENTRE RECT	LOW	19.42	39.43	39.43	0.00	0.00	0.00
CENTRE RECT	HIGH	19.42	39.43	39.43	0.00	0.00	0.00
CENTRE RECT	HIGH	19.42	38.07	38.07	0.00	0.00	0.00
CROSS	SHARP	19.42	39.43	39.43	0.00	0.00	0.00
CROSS	LOW	19.42	39.43	39.43	0.00	0.00	0.00
CROSS	LOW	19.42	39.43	39.43	0.00	0.00	0.00
CROSS	HIGH	19.42	39.43	39.43	0.00	0.00	0.00
CROSS	HIGH	19.42	39.43	39.43	0.00	0.00	0.00

Table A.6: Vision 20×20 ($n = 400, m = 2928$): resolution times using persistencies.

Instance (20×20)		Exec. times (secs)					
		Pairwise covers					
Base	Quality	PC1	PC1-pers	PC2	PC2-pers	PC3	PC3-pers
top left rect	sharp	45.38	47.42	41.09	**40.66**	40.06	**39.69**
top left rect	low	48.61	50.97	42.94	**42.44**	42.58	**42.53**
top left rect	low	50.78	**48.66**	45.24	46.95	45.75	**45.02**
top left rect	high	153.77	**153.08**	123.55	125.34	126.64	127.06
top left rect	high	169.81	$\underline{\mathbf{158.99}}$	137.98	139.77	136.84	140.33
centre rect	sharp	51.00	**48.50**	43.89	45.38	43.55	45.47
centre rect	low	48.75	49.52	48.88	49.59	48.88	49.75
centre rect	low	55.06	55.19	51.16	**49.70**	49.17	50.30
centre rect	high	190.58	**189.45**	147.25	158.38	149.92	**145.33**
centre rect	high	147.34	**146.61**	133.81	**130.66**	131.50	134.84
cross	sharp	53.14	**52.84**	39.80	41.00	40.22	40.89
cross	low	53.91	55.52	47.14	47.39	48.20	**48.11**
cross	low	54.81	**54.44**	43.33	43.41	43.26	43.86
cross	high	163.03	163.16	132.16	**128.61**	127.89	132.11
cross	high	155.39	**153.58**	122.72	$\underline{\mathbf{117.06}}$	117.13	120.28

Table A.7: VISION 25×25 ($n = 625, m = 4658$): total number of variables in problem (QP).

	Pairwise covers			Termwise quadratizations		
	PC1	PC2	PC3	ISHIK	N/4	LOGN-1
# variables QP	2954	2911	2911	3505	3505	3505

Table A.8: VISION 25×25 ($n = 625, m = 4658$): percentage of variables fixed using persistencies.

Instance (25×25)		Fixed variables (%)					
		Pairwise covers					
Base	Quality	PC1	PC2	PC3	ISHIK	N/4	LOGN-1
TOP LEFT RECT	SHARP	19.53	39.57	39.57	0.00	0.00	0.00
TOP LEFT RECT	LOW	19.53	38.85	38.85	0.00	0.00	0.00
TOP LEFT RECT	LOW	19.53	39.57	39.57	0.00	0.00	0.00
TOP LEFT RECT	HIGH	19.53	39.57	39.57	0.00	0.00	0.00
TOP LEFT RECT	HIGH	19.53	39.57	39.57	0.00	0.00	0.00
CENTRE RECT	SHARP	19.53	39.57	39.57	0.00	0.00	0.00
CENTRE RECT	LOW	19.53	39.57	39.57	0.00	0.00	0.00
CENTRE RECT	LOW	19.53	39.57	39.57	0.00	0.00	0.00
CENTRE RECT	HIGH	19.53	39.57	39.57	0.00	0.00	0.00
CENTRE RECT	HIGH	19.53	39.57	39.57	0.00	0.00	0.00
CROSS	SHARP	19.53	39.57	39.57	0.00	0.00	0.00
CROSS	LOW	19.53	39.57	39.57	0.00	0.00	0.00
CROSS	LOW	19.53	39.57	39.57	0.00	0.00	0.00
CROSS	HIGH	19.53	39.57	39.57	0.00	0.00	0.00
CROSS	HIGH	19.53	39.57	39.57	0.00	0.00	0.00

Table A.9: VISION 25 × 25 ($n = 625, m = 4658$): resolution times using persistencies.

Instance (25 × 25)		Exec. times (secs)					
		Pairwise covers			Termwise quadratizations		
Base	Quality	PC1	PC1-pers	PC2	PC2-pers	PC3	PC3-pers
TOP LEFT RECT	SHARP	124.41	**121.58**	117.44	**114.77**	113.33	**112.64**
TOP LEFT RECT	LOW	143.48	**135.83**	104.52	**102.16**	104.61	**103.13**
TOP LEFT RECT	LOW	141.06	143.67	111.77	**111.33**	112.13	113.31
TOP LEFT RECT	HIGH	453.98	455.28	466.64	506.30	458.78	478.83
TOP LEFT RECT	HIGH	493.22	**484.03**	501.86	505.59	512.13	**499.94**
CENTRE RECT	SHARP	135.67	139.33	96.95	97.61	98.42	**97.91**
CENTRE RECT	LOW	125.73	**125.16**	128.94	**127.45**	127.41	127.64
CENTRE RECT	LOW	143.94	146.52	109.11	109.33	111.22	112.33
CENTRE RECT	HIGH	427.88	**415.73**	467.61	**459.30**	461.13	466.50
CENTRE RECT	HIGH	484.22	493.56	490.69	494.58	503.78	**495.89**
CROSS	SHARP	131.52	**129.89**	116.38	118.52	115.95	119.89
CROSS	LOW	155.83	156.27	109.31	112.28	109.41	**107.16**
CROSS	LOW	141.61	142.91	120.59	**119.47**	120.95	122.58
CROSS	HIGH	455.13	459.95	409.61	414.97	432.33	**406.51**
CROSS	HIGH	415.02	416.08	524.76	529.00	539.27	**536.22**

Table A.10: Vision 30 × 30 ($n = 900, m = 6788$): total number of variables in problem (QP).

	Pairwise covers			Termwise quadratizations		
	PC1	PC2	PC3	Ishik	n/4	logn-1
# variables QP	4294	4241	4241	5105	5105	5105

Table A.11: Vision 30 × 30 ($n = 900, m = 6788$): percentage of variables fixed using persistencies.

Instance (30 × 30)		Fixed variables (%)					
		Pairwise covers			Termwise quadratizations		
Base	Quality	PC1	PC2	PC3	Ishik	n/4	logn-1
top left rect	sharp	19.61	39.66	39.66	0.00	0.00	0.00
top left rect	low	19.61	39.66	39.66	0.00	0.00	0.00
top left rect	low	19.61	39.66	39.66	0.00	0.00	0.00
top left rect	high	19.61	39.66	39.66	0.00	0.00	0.00
top left rect	high	19.61	39.66	39.66	0.00	0.00	0.00
centre rect	sharp	19.61	39.66	39.66	0.00	0.00	0.00
centre rect	low	19.61	39.66	39.66	0.00	0.00	0.00
centre rect	low	19.61	39.66	39.66	0.00	0.00	0.00
centre rect	high	19.61	37.40	37.40	0.00	0.00	0.00
centre rect	high	19.61	38.48	38.48	0.00	0.00	0.00
cross	sharp	19.61	39.66	39.66	0.00	0.00	0.00
cross	low	19.61	38.46	38.46	0.00	0.00	0.00
cross	low	19.61	39.66	39.66	0.00	0.00	0.00
cross	high	19.61	39.66	39.66	0.00	0.00	0.00
cross	high	19.61	39.66	39.66	0.00	0.00	0.00

Table A.12: VISION 30×30 ($n = 900, m = 6788$): resolution times using persistencies.

Instance (30×30)		Exec. times (secs)					
		Pairwise covers					
Base	Quality	PC1	PC1-pers	PC2	PC2-pers	PC3	PC3-pers
TOP LEFT RECT	SHARP	227.27	235.31	173.48	182.28	172.73	176.92
TOP LEFT RECT	LOW	237.86	**233.67**	174.72	179.89	176.67	178.59
TOP LEFT RECT	LOW	227.41	230.03	192.38	**190.01**	193.01	**188.56**
TOP LEFT RECT	HIGH	1250.88	**1229.86**	682.78	**677.58**	690.20	**676.00**
TOP LEFT RECT	HIGH	1085.67	1087.61	705.70	731.36	699.38	716.36
CENTRE RECT	SHARP	243.13	244.14	175.52	**175.28**	173.64	178.06
CENTRE RECT	LOW	257.39	**251.53**	178.99	180.24	178.36	178.55
CENTRE RECT	LOW	244.25	250.55	184.20	188.06	194.16	**188.94**
CENTRE RECT	HIGH	889.16	**880.36**	726.19	**725.20**	755.75	**730.95**
CENTRE RECT	HIGH	1187.81	**1182.78**	821.73	**787.30**	774.13	808.88
CROSS	SHARP	220.13	223.39	173.56	177.45	173.72	176.63
CROSS	LOW	275.16	**267.77**	181.14	185.78	184.36	**180.91**
CROSS	LOW	266.39	267.16	182.92	188.47	182.13	186.06
CROSS	HIGH	1321.61	1350.50	764.63	776.59	773.03	783.16
CROSS	HIGH	1360.53	**1312.69**	946.42	970.22	973.34	**956.69**

Appendix B

Tables: Random same degree instances

B.1 Computing times of linearizations and quadratizations

This section contains the tables presenting execution times of the experiments described in Section 8.4.1 for Random same degree instances. Tables B.1 and B.2 presents results for instances of degree three and degree four, respectively. Corresponding figures can be found in Section 8.4.1 and Appendix E.

Table B.1: Random same degree (deg = 3): linearizations and quadratizations computing times.

Instance				Resolution time (secs)						
				Linearizations	Pairwise covers			Termwise quadratizations		
n	m	deg	id	SL	PC1	PC2	PC3	Ishikawa	n/4	logn-1
200	400	3	1	1.19	9.48	7.91	8.44	22.56	22.91	17.95
200	400	3	2	1.17	12.63	10.67	10.52	20.89	25.09	20.23
200	400	3	3	0.55	6.83	4.67	2.98	12.23	10.89	9.56
200	400	3	4	1.02	11.03	9.47	7.22	21.13	21	18.05
200	400	3	5	0.38	9.44	6.91	10.44	21.2	22.45	17.38
200	500	3	1	12.36	110.47	84.5	96.45	71.13	65.13	61.5
200	500	3	2	8.81	45.55	33.13	39.66	57.06	65.06	64.64
200	500	3	3	4.47	24.01	24.13	21.98	37.41	39.89	37.97
200	500	3	4	5.47	24.36	17.08	17.33	36.25	37.2	36.91
200	500	3	5	8.72	27.36	28.45	43.25	46.23	63.72	44.5
400	700	3	1	3.13	16.11	18.33	19.03	35.53	39.42	37.13
400	700	3	2	0.5	13.59	6.36	10.94	21.64	22.16	22.73
400	700	3	3	0.83	18.69	23.41	16.88	34.63	41.75	33.39
400	700	3	4	1.51	18.94	21.44	15.83	48.83	46.08	41.74
400	700	3	5	1.67	22.89	20.41	19.69	43.26	46.56	38.77
400	800	3	1	12.64	106.36	68.47	80.39	101.64	98.08	91.73
400	800	3	2	10.78	63.05	47.8	72.77	95.59	95.84	76.47
400	800	3	3	6.23	34.09	27.31	35.77	71.81	80.61	68.88
400	800	3	4	16.36	205.97	161.19	350.69	144.42	245.83	142.41
400	800	3	5	11.64	121.67	118.8	75.11	125.5	127	87.34
600	1100	3	1	9.8	54.19	66.59	52.7	85.42	103.23	120.27
600	1100	3	2	17.17	149.64	119.45	435.08	155.69	184.5	149.97
600	1100	3	3	11	113.45	106.2	99.66	158.41	160.03	180.7
600	1100	3	4	21.31	432.36	477.49	412.06	348.51	223.98	301.14
600	1100	3	5	2.5	41.8	52.89	54.63	76.56	103.11	102.44
600	1200	3	1	26.53	971.09	784.16	850.52	387.48	474.3	349.53
600	1200	3	2	14.51	202.67	126.58	203.08	300.53	234.81	239.97
600	1200	3	3	38.5	720.25	625.34	685.55	237.67	372.72	340.55
600	1200	3	4	68.83	782.66	803.36	1345.23	949.25	605.55	1094.74
600	1200	3	5	56.84	856.23	643.47	781.69	517.05	448.95	383.5

Table B.2: RANDOM SAME DEGREE (deg = 4): linearizations and quadratizations computing times.

Instance				Resolution time (secs)						
				Linearizations	Pairwise covers			Termwise quadratizations		
n	m	deg	id	SL	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
200	250	4	1	0.7	8.47	5.83	6.28	38.5	38.61	37.76
200	250	4	2	0.34	8.22	4.75	7.47	26.13	25.68	28.08
200	250	4	3	0.44	5.83	7.42	5.08	32.78	31.16	33.5
200	250	4	4	0.72	10.11	7.77	7.64	35.2	34.64	29.89
200	250	4	5	0.36	7.06	6.98	3.64	31.42	31	34.77
200	300	4	1	2.89	28.49	18.88	14.92	70.56	104.39	78.49
200	300	4	2	0.38	13.83	10.66	11.63	42.76	39.28	44.81
200	300	4	3	5.55	78.73	29.89	25.52	97.59	87.89	137.88
200	300	4	4	4.81	108.52	85.47	47	268.17	249.56	337.53
200	300	4	5	1.75	18.34	17.61	19.47	64.36	57.28	87.67
400	450	4	1	0.42	15.17	17.84	12.64	59.92	72.41	71.86
400	450	4	2	0.58	12.03	11.8	11.64	72.8	75.95	65.94
400	450	4	3	0.77	18.06	18.64	16.09	106.81	84.63	95.89
400	450	4	4	0.81	15.7	12.13	18.13	98.83	86.67	96.3
400	450	4	5	0.61	14.38	11.69	12.28	88.67	104	96.78
400	500	4	1	6.36	95.89	63.89	65.13	725.01	440.92	981.39
400	500	4	2	5.14	58.55	61.69	51.67	188.41	191.05	298.38
400	500	4	3	4.13	79.89	42.59	42	340.92	187.2	196.63
400	500	4	4	4.58	103.01	55.52	59.41	293	573.33	603.13
400	500	4	5	7.42	80.74	59.14	78	266.14	227.94	362.89
600	650	4	1	2.69	41.83	38.92	39.72	182.14	180.48	235.73
600	650	4	2	2.31	54.95	42.14	42.77	241.88	331.3	246.22
600	650	4	3	0.66	23.3	23.61	25.7	139.75	146.59	160.95
600	650	4	4	0.84	21.73	27.86	26.7	153.94	158.73	204.86
600	650	4	5	0.41	26.03	21.63	21.47	158.03	132.58	156.63
600	700	4	1	14.38	586.13	250.3	237.52	1256.36	2357.16	1484.92
600	700	4	2	2.66	49.58	37.94	44.66	224.23	266.27	262.88
600	700	4	3	6.66	68.05	62.08	77.44	424.97	427.03	421.55
600	700	4	4	5.48	64.86	73.11	60.27	292.26	433.91	463.67
600	700	4	5	0.98	28.81	28.36	27.72	161.17	192.47	198.11

B.2 Number of auxiliary variables and positive quadratic terms

Tables B.3 and B.4 present the number of auxiliary variables required for each quadratization method for RANDOM SAME DEGREE instances of $n = 400$ and $n = 600$ variables, of degrees three and four, respectively. Tables B.5 and B.6 present the number positive quadratic terms of each quadratization for the same instances.

Table B.3: Random same degree (deg = 3): number of y variables in quadratizations.

Instance				# y variables					
				Pairwise covers			Termwise quadratizations		
n	m	deg	id	PC1	PC2	PC3	Ishikawa	n/4	logn-1
400	700	3	1	697	680	680	700	700	700
400	700	3	2	696	668	668	700	700	700
400	700	3	3	695	677	677	700	700	700
400	700	3	4	693	677	677	700	700	700
400	700	3	5	694	677	677	700	700	700
400	800	3	1	796	761	761	800	800	800
400	800	3	2	796	767	767	800	800	800
400	800	3	3	795	763	763	800	800	800
400	800	3	4	797	767	767	800	800	800
400	800	3	5	794	774	774	800	800	800
600	1100	3	1	1092	1072	1072	1100	1100	1100
600	1100	3	2	1097	1072	1072	1100	1100	1100
600	1100	3	3	1096	1071	1071	1100	1100	1100
600	1100	3	4	1094	1076	1076	1100	1100	1100
600	1100	3	5	1093	1073	1073	1100	1100	1100
600	1200	3	1	1195	1165	1165	1200	1200	1200
600	1200	3	2	1196	1171	1171	1200	1200	1200
600	1200	3	3	1192	1162	1162	1200	1200	1200
600	1200	3	4	1195	1170	1170	1200	1200	1200
600	1200	3	5	1198	1169	1169	1200	1200	1200

Table B.4: RANDOM SAME DEGREE (deg = 4): number of y variables in quadratizations.

Instance				# y variables					
				Pairwise covers			Termwise quadratizations		
n	m	deg	id	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
400	450	4	1	893	867	867	450	450	450
400	450	4	2	894	860	860	450	450	450
400	450	4	3	898	860	861	450	450	450
400	450	4	4	895	857	857	450	450	450
400	450	4	5	895	857	857	450	450	450
400	500	4	1	990	953	953	500	500	500
400	500	4	2	993	957	958	500	500	500
400	500	4	3	991	949	949	500	500	500
400	500	4	4	989	957	957	500	500	500
400	500	4	5	991	944	945	500	500	500
600	650	4	1	1295	1258	1259	650	650	650
600	650	4	2	1293	1260	1260	650	650	650
600	650	4	3	1294	1263	1263	650	650	650
600	650	4	4	1292	1264	1264	650	650	650
600	650	4	5	1292	1265	1265	650	650	650
600	700	4	1	1395	1357	1357	700	700	700
600	700	4	2	1393	1350	1350	700	700	700
600	700	4	3	1395	1352	1352	700	700	700
600	700	4	4	1395	1348	1348	700	700	700
600	700	4	5	1392	1352	1352	700	700	700

Table B.5: Random same degree (deg = 3): number of positive quadratic terms.

Instance				# positive quadratic terms					
				Pairwise covers			Termwise quadratizations		
n	m	deg	id	PC1	PC2	PC3	Ishikawa	n/4	logn-1
400	700	3	1	1037	1020	1020	1020	1020	1020
400	700	3	2	1016	988	988	960	960	960
400	700	3	3	1036	1018	1018	1023	1023	1023
400	700	3	4	1051	1035	1035	1074	1074	1074
400	700	3	5	1037	1020	1020	1029	1029	1029
400	800	3	1	1203	1168	1168	1221	1221	1221
400	800	3	2	1192	1163	1163	1188	1188	1188
400	800	3	3	1206	1174	1174	1233	1233	1233
400	800	3	4	1182	1152	1152	1155	1155	1155
400	800	3	5	1185	1165	1165	1173	1173	1173
600	1100	3	1	1628	1608	1608	1608	1608	1608
600	1100	3	2	1646	1621	1621	1647	1647	1647
600	1100	3	3	1659	1634	1634	1689	1689	1689
600	1100	3	4	1674	1656	1656	1740	1740	1740
600	1100	3	5	1638	1618	1618	1635	1635	1635
600	1200	3	1	1802	1772	1772	1821	1821	1821
600	1200	3	2	1781	1756	1756	1755	1755	1755
600	1200	3	3	1779	1749	1749	1761	1761	1761
600	1200	3	4	1777	1752	1752	1746	1746	1746
600	1200	3	5	1751	1722	1722	1659	1659	1659

Table B.6: Random same degree (deg = 4): number of positive quadratic terms.

Instance				# positive quadratic terms					
				Pairwise covers			Termwise quadratizations		
n	m	deg	id	PC1	PC2	PC3	Ishikawa	n/4	logn-1
400	450	4	1	1100	1074	1074	1242	1242	1242
400	450	4	2	1113	1079	1079	1314	1314	1314
400	450	4	3	1142	1104	1105	1464	1464	1464
400	450	4	4	1119	1081	1081	1344	1344	1344
400	450	4	5	1135	1097	1097	1440	1440	1440
400	500	4	1	1257	1220	1220	1602	1602	1602
400	500	4	2	1241	1205	1206	1488	1488	1488
400	500	4	3	1218	1176	1176	1362	1362	1362
400	500	4	4	1243	1211	1211	1524	1524	1524
400	500	4	5	1218	1171	1172	1362	1362	1362
600	650	4	1	1604	1567	1568	1854	1854	1854
600	650	4	2	1609	1576	1576	1896	1896	1896
600	650	4	3	1631	1600	1600	2022	2022	2022
600	650	4	4	1634	1606	1606	2052	2052	2052
600	650	4	5	1621	1594	1594	1974	1974	1974
600	700	4	1	1749	1711	1711	2124	2124	2124
600	700	4	2	1737	1694	1694	2064	2064	2064
600	700	4	3	1732	1689	1689	2022	2022	2022
600	700	4	4	1751	1704	1704	2136	2136	2136
600	700	4	5	1727	1687	1687	2010	2010	2010

Appendix C

Tables: Random high degree instances

C.1 Computing times of linearizations and quadratizations

This section contains the tables presenting execution times of the experiments described in Section 8.4.2 for Random high degree instances. Tables C.1, C.2 and C.3 present results for instances with n = 200, 400 and 600 variables, respectively. Corresponding figures can be found in Section 8.4.2 and Appendix F.

Table C.1: Random high degree (n = 200): linearizations and quadratizations computing times.

Instance				Resolution time (secs)						
				Linearizations	Pairwise covers			Termwise quadratizations		
n	m	id	deg	SL	PC1	PC2	PC3	Ishikawa	n/4	logn-1
200	250	1	9	0.03	0.95	1.17	1.02	5.38	7.72	6.97
200	250	2	9	0.06	2.31	1.48	1.69	12.08	20.72	19.23
200	250	3	9	0.03	2.78	1.95	1.8	14.16	43.78	21.44
200	250	4	9	0.03	1.55	0.97	1.11	11.91	22.59	12.58
200	250	5	7	0.09	1.77	1.91	1.83	12.06	26.3	11.08
200	300	1	8	0.08	4.3	4.31	6.23	17.36	68.47	20.59
200	300	2	9	0.33	11.42	10.58	10.39	46.5	218.3	98.63
200	300	3	15	0.05	3.74	4	3.26	49.22	426.19	93.03
200	300	4	13	0.09	11.19	9.06	7.5	275.42	2350.66	477.39
200	300	5	12	0.08	6.45	3.86	3.91	33.2	79.77	47.88
200	350	1	10	0.16	10.95	5.12	5.53	61.98	243.77	68.3
200	350	2	10	0.2	14.03	10.28	10.58	368.33	2218	658.72
200	350	3	13	0.16	16.26	8.61	11.19	129.56	898.66	368.16
200	350	4	9	0.14	7.06	4.38	4.5	32.75	61.41	42.72
200	350	5	11	0.14	4.27	4.97	3.39	39.02	97.22	53.03
200	400	1	14	0.09	12.52	11.53	9.16	686.83	> 3600	> 3600
200	400	2	11	0.17	15.45	11.59	15.95	30.72	57.58	38.26
200	400	3	11	0.19	20.17	20	21.53	> 3600	> 3600	> 3600
200	400	4	16	0.47	25.06	18.09	19.44	> 3600	> 3600	> 3600
200	400	5	12	0.11	12.99	13.89	12.63	337.84	785.8	647.08

Table C.2: Random high degree (n = 400): linearizations and quadratizations computing times.

Instance				Resolution time (secs)						
				Linearizations	Pairwise covers			Termwise quadratizations		
n	m	id	deg	SL	PC1	PC2	PC3	Ishikawa	n/4	logn-1
400	450	1	16	0.06	3.69	3.25	4.08	224.2	2837.99	592.56
400	450	2	10	0.06	4.05	3.91	2.94	363.81	> 3600	509.19
400	450	3	11	0.28	8.56	4.39	3.42	46.72	778.34	221.41
400	450	4	11	0.06	5.2	2.84	3.64	98.61	1269.49	368.45
400	450	5	17	0.09	4.45	5.16	4.8	117.97	1300.09	385.63
400	500	1	10	0.09	8.05	6.7	7.03	48.38	254.53	86.39
400	500	2	12	0.06	7.08	7.41	5.06	1686.19	> 3600	> 3600
400	500	3	11	0.13	6.02	5.75	6.05	415.33	> 3600	> 3600
400	500	4	12	0.08	13.05	14.16	11.55	579.28	> 3600	1647.22
400	500	5	12	0.11	7.08	7.22	7.51	142.09	1256.81	218.33
400	550	1	13	0.11	8.28	8.97	7.53	182.09	1761.45	650.53
400	550	2	12	0.13	5.67	5.55	9.47	112.34	866.19	314.05
400	550	3	10	0.16	12.03	11.3	9.8	1458.97	> 3600	> 3600
400	550	4	12	0.09	14.89	15.03	10.61	105.72	1065.45	181.59
400	550	5	14	0.09	16.69	10.03	10.34	1745.39	> 3600	> 3600

Table C.3: Random high degree (n = 600): linearizations and quadratizations computing times.

Instance				Resolution time (secs)						
				Linearizations	Pairwise covers			Termwise quadratizations		
n	m	id	deg	SL	PC1	PC2	PC3	Ishikawa	n/4	logn-1
600	650	1	12	0.06	12.81	10.25	12.84	477.95	1611.5	671.23
600	650	2	11	0.11	4.84	6.11	5.95	2432.58	> 3600	> 3600
600	650	3	11	0.11	9.51	7.49	8.58	267.99	1955.31	630.14
600	650	4	10	0.11	4.41	4.16	4.59	25.03	80.2	33
600	650	5	12	0.09	7.66	7.11	7.84	3261.55	> 3600	> 3600
600	700	1	9	0.11	8.08	7.24	8.33	202.8	> 3600	576.14
600	700	2	9	0.19	9.05	8.06	9.22	454.5	3542.8	592.77
600	700	3	9	0.16	14.55	16.42	16.26	2243.06	> 3600	> 3600
600	700	4	10	0.13	9.84	13.24	11.3	720.55	> 3600	1448.69
600	700	5	10	0.11	8.98	8.16	9.19	445.14	> 3600	696.52
600	750	1	14	0.22	25.13	16.99	19.36	> 3600	> 3600	> 3600
600	750	2	11	0.19	20.36	19.28	16.23	724.81	2043.56	766.44
600	750	3	11	0.14	15.74	13.45	14.5	> 3600	> 3600	> 3600
600	750	4	11	0.13	5.49	5.33	5.73	810.03	> 3600	> 3600
600	750	5	11	0.14	9.55	9.05	8.55	616.41	> 3600	3371.05

C.2 Number of auxiliary variables and positive quadratic terms

Tables C.4 and C.5 present the number of auxiliary variables required for each quadratization for RANDOM HIGH DEGREE instances with $n = 400$ and $n = 600$ variables, respectively. Tables C.6 and C.7 present the number of positive quadratic terms of each quadratization for the same instances.

Table C.4: Random high degree (n = 400): number of y variables in quadratizations.

Instance				# y variables					
				Pairwise covers			Termwise quadratizations		
n	m	id	deg	PC1	PC2	PC3	Ishikawa	n/4	logn-1
400	450	1	16	671	662	663	276	267	266
400	450	2	10	663	653	651	268	258	258
400	450	3	11	684	673	671	277	271	271
400	450	4	11	684	666	670	281	274	274
400	450	5	17	693	678	677	272	266	266
400	500	1	10	766	754	755	302	297	297
400	500	2	12	752	736	735	308	301	301
400	500	3	11	743	728	727	291	281	281
400	500	4	12	774	746	747	297	286	286
400	500	5	12	757	742	742	302	292	292
400	550	1	13	797	785	785	324	317	316
400	550	2	12	775	769	769	294	286	286
400	550	3	10	814	798	796	318	310	310
400	550	4	12	849	823	822	336	329	329
400	550	5	14	843	818	817	339	330	329

Table C.5: RANDOM HIGH DEGREE (n = 600): number of y variables in quadratizations.

Instance				# y variables					
				Pairwise covers			Termwise quadratizations		
n	m	id	deg	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
600	650	1	12	1024	1010	1010	397	386	386
600	650	2	11	949	937	936	359	345	345
600	650	3	11	951	943	943	373	365	365
600	650	4	10	919	909	908	341	339	339
600	650	5	12	992	977	978	372	358	358
600	700	1	9	1046	1029	1029	413	406	406
600	700	2	9	1023	1011	1011	380	369	369
600	700	3	9	1053	1045	1045	417	405	405
600	700	4	10	1084	1074	1074	418	406	406
600	700	5	10	1051	1032	1033	406	394	394
600	750	1	14	1109	1091	1090	425	413	413
600	750	2	11	1151	1136	1136	423	417	417
600	750	3	11	1157	1142	1142	441	426	426
600	750	4	11	1047	1037	1036	392	376	376
600	750	5	11	1091	1082	1080	422	413	413

Table C.6: RANDOM HIGH DEGREE (n = 400): number of positive quadratic terms.

Instance				# positive quadratic terms					
				Pairwise covers			Termwise quadratizations		
n	m	id	deg	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
400	450	1	16	789	789	790	1080	1120	1117
400	450	2	10	777	767	765	940	975	975
400	450	3	11	810	799	797	950	982	982
400	450	4	11	810	792	796	987	1024	1024
400	450	5	17	811	796	795	929	965	965
400	500	1	10	902	890	891	968	1000	1000
400	500	2	12	890	874	873	1101	1146	1146
400	500	3	11	867	852	851	1024	1058	1058
400	500	4	12	901	873	874	1014	1047	1047
400	500	5	12	883	868	868	997	1033	1033
400	550	1	13	938	926	926	1108	1154	1151
400	550	2	12	892	886	886	985	1017	1017
400	550	3	10	954	938	936	1119	1161	1161
400	550	4	12	1007	981	980	1213	1264	1264
400	550	5	14	982	957	956	1110	1148	1145

Table C.7: Random high degree (n = 600): number of positive quadratic terms.

Instance				# positive quadratic terms					
				Pairwise covers			Termwise quadratizations		
n	m	id	deg	PC1	PC2	PC3	Ishikawa	n/4	logn-1
600	650	1	12	1190	1176	1176	1284	1323	1323
600	650	2	11	1097	1085	1084	1277	1325	1325
600	650	3	11	1105	1097	1097	1191	1236	1236
600	650	4	10	1071	1061	1060	1022	1055	1055
600	650	5	12	1139	1124	1125	1362	1412	1412
600	700	1	9	1228	1211	1211	1311	1358	1358
600	700	2	9	1199	1187	1187	1336	1373	1373
600	700	3	9	1231	1223	1223	1383	1424	1424
600	700	4	10	1271	1261	1261	1484	1541	1541
600	700	5	10	1224	1205	1206	1357	1402	1402
600	750	1	14	1291	1273	1272	1402	1448	1448
600	750	2	11	1334	1319	1319	1355	1399	1399
600	750	3	11	1350	1335	1335	1685	1751	1751
600	750	4	11	1212	1202	1201	1438	1485	1485
600	750	5	11	1277	1268	1266	1445	1497	1497

Appendix D

Tables: Vision instances

D.1 Computing times of linearizations and quadratizations

This section contains the tables presenting execution times of the experiments described in Section 8.4.3 for Vision instances. Tables D.1, D.2, D.3, D.4, D.5 and D.6 present the resolution times of linear and quadratic reformulations of problems with image sizes 10×10, 15×15, 20×20, 25×25, 30×30 and 35×35, respectively. Table D.7 and D.8 present, respectively, model creation times in Java and total times (model creation plus resolution) for 35×35 images. Corresponding figures can be found in Section 8.4.3 and Appendix G.

Table D.1: Vision 10 × 10 (n = 100, m = 668): linearizations and quadratizations computing times.

Instance (10 × 10)		Resolution time (secs)							
		Linearizations		Pairwise covers			Termwise quadratizations		
Base	Quality	SL-2L	SL	PC1	PC2	PC3	Ishikawa	n/4	logn-1
top left rect	sharp	0.76	2.83	2.69	2.77	2.81	11.86	10.73	15.67
top left rect	low	0.84	3.81	4.2	3.91	4.55	20.53	18.31	26.55
top left rect	low	1.06	3.44	3.06	3.09	3.05	18.67	17.69	22.78
top left rect	high	2.09	12.47	7.72	6.58	6.63	46.27	46.88	41.69
top left rect	high	2.84	11.19	8.67	7.97	7.97	57.13	58.42	40.06
centre rect	sharp	1.02	4.72	3.28	3.33	3.34	29.42	29.72	33.28
centre rect	low	1.2	3.92	3.58	3.14	3.25	30.73	31	31.66
centre rect	low	0.95	5.36	3.44	3.7	3.31	25.86	25.23	33.28
centre rect	high	1.64	10.86	6	6.22	6.17	36.22	36.38	25.97
centre rect	high	1.8	14.08	5.44	5.51	5.38	27.16	27.19	28.81
cross	sharp	1.09	4.3	3.19	3.78	3.75	26.3	26.64	34.89
cross	low	1.2	3.72	3.45	3.44	3.47	31.14	30.77	28.28
cross	low	1.08	6.19	4.56	4.23	4.67	37.63	39.31	32.73
cross	high	1.75	13.95	5.2	5.2	5.22	39.75	38.55	22.95
cross	high	2.44	12.88	6.11	5.92	5.83	40.73	41.03	37.53

Table D.2: VISION 15 × 15 ($n = 225, m = 1598$): linearizations and quadratizations computing times.

Instance (15 × 15)		Resolution time (secs)							
		Linearizations		Pairwise covers			Termwise quadratizations		
Base	Quality	SL-2L	SL	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
TOP LEFT RECT	SHARP	2.84	16.42	16.31	11.75	12.08	159.75	120.42	80.22
TOP LEFT RECT	LOW	7.23	25.78	18.14	13.45	12.83	169.05	156.98	98.22
TOP LEFT RECT	LOW	4.75	23.41	16.94	14.95	14.92	175.52	169.03	155.66
TOP LEFT RECT	HIGH	9.2	69.66	53.84	24.97	21.53	283.11	288.2	276.94
TOP LEFT RECT	HIGH	12.26	128.03	71.94	31.5	31.86	330.53	328.67	302.22
CENTRE RECT	SHARP	4.88	28.78	19.67	12.17	11.98	193.27	192.38	77.59
CENTRE RECT	LOW	5.63	32.8	17.3	14.42	14.28	228.16	216.47	191.67
CENTRE RECT	LOW	4.59	36.86	24.28	14.73	14.72	201.44	199.98	205.91
CENTRE RECT	HIGH	10.44	101.31	47.3	23.76	23.97	277.47	290.52	223.36
CENTRE RECT	HIGH	14.2	112.98	39.95	34.19	34.45	308.58	306.05	241.59
CROSS	SHARP	4.36	21.59	19.28	12.09	12.23	219.95	224.05	206.19
CROSS	LOW	4.66	20.94	21.84	12	12.09	185.52	171.11	106.56
CROSS	LOW	5.09	35.75	18.41	11.83	12.06	133.14	139.8	181.48
CROSS	HIGH	10.14	122.33	48.05	31.38	31.38	168.75	169.38	143.53
CROSS	HIGH	11	102.31	45.98	23.3	23.63	366.03	357.67	286.78

Table D.3: Vision 20×20 ($n = 400, m = 2928$): linearizations and quadratizations computing times.

Instance (20×20)		Resolution time (secs)							
		Linearizations		Pairwise covers			Termwise quadratizations		
Base	Quality	SL-2L	SL	PC1	PC2	PC3	Ishikawa	n/4	logn-1
top left rect	sharp	10.78	66.22	46.03	40.36	41.14	-	-	-
top left rect	low	14.25	74.7	48.38	42.88	42.38	-	-	-
top left rect	low	13.25	79.88	48.77	45.78	46.08	-	-	-
top left rect	high	39.45	269.48	151.05	120.14	119.88	-	-	-
top left rect	high	36.7	490.05	161.09	142.39	144.2	-	-	-
centre rect	sharp	14.73	82.05	49.36	43.75	43.33	-	-	-
centre rect	low	15.89	99.42	48.55	48.55	47.55	-	-	-
centre rect	low	16.95	105.09	55.78	47.11	47.45	-	-	-
centre rect	high	44.48	433.42	182.89	145.25	151.42	-	-	-
centre rect	high	31.88	581.25	157.36	129.56	127.53	-	-	-
cross	sharp	13.58	77.69	53.22	39.91	40.59	-	-	-
cross	low	16.22	84.59	53.38	48.58	48.17	-	-	-
cross	low	15.89	110.91	56.25	42.83	43.91	-	-	-
cross	high	34.76	235.41	158.33	127.72	125.7	-	-	-
cross	high	43.55	945.25	154.83	118.2	118.72	-	-	-

Table D.4: VISION 25 × 25 ($n = 625, m = 4658$): linearizations and quadratizations computing times.

Instance (25 × 25)		Resolution time (secs)							
		Linearizations		Pairwise covers			Termwise quadratizations		
Base	Quality	SL-2L	SL	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
TOP LEFT RECT	SHARP	28.83	135.19	122.11	112.23	112.72	-	-	-
TOP LEFT RECT	LOW	29.03	206.14	148.55	101.01	102.48	-	-	-
TOP LEFT RECT	LOW	29.67	259.64	138.19	108.23	108.23	-	-	-
TOP LEFT RECT	HIGH	68.53	> 3600	550.63	455.19	470.83	-	-	-
TOP LEFT RECT	HIGH	87.23	1300.77	465.44	490.83	500.44	-	-	-
CENTRE RECT	SHARP	32	308.36	135.63	95.63	94.31	-	-	-
CENTRE RECT	LOW	30.09	174.39	120.52	125.67	130.67	-	-	-
CENTRE RECT	LOW	34.7	259.63	146.5	106.42	110.14	-	-	-
CENTRE RECT	HIGH	93	> 3600	440.59	451.03	454.61	-	-	-
CENTRE RECT	HIGH	72.27	> 3600	548.92	499.13	482.33	-	-	-
CROSS	SHARP	34.92	308.02	126.09	119.3	114.61	-	-	-
CROSS	LOW	35.16	452.16	152.14	106.06	105.02	-	-	-
CROSS	LOW	37.23	211.66	142.55	116.8	120.31	-	-	-
CROSS	HIGH	72.28	> 3600	464.3	406.02	402.42	-	-	-
CROSS	HIGH	75.44	3141.48	431.56	524	521.11	-	-	-

Table D.5: VISION 30×30 ($n = 900, m = 6788$): linearizations and quadratizations computing times.

Instance (30×30)		Resolution time (secs)							
		Linearizations		Pairwise covers			Termwise quadratizations		
Base	Quality	SL-2L	SL	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
TOP LEFT RECT	SHARP	45.05	471.17	226.97	168.09	169.44	-	-	-
TOP LEFT RECT	LOW	45.27	413.52	227.72	171.58	179.19	-	-	-
TOP LEFT RECT	LOW	49.11	613.26	222.95	186.44	186.13	-	-	-
TOP LEFT RECT	HIGH	181.84	> 3600	1212.14	690.25	671.41	-	-	-
TOP LEFT RECT	HIGH	202.05	> 3600	1091.8	740.05	745.22	-	-	-
CENTRE RECT	SHARP	57.28	692.39	243.89	171.83	170.67	-	-	-
CENTRE RECT	LOW	55.31	837.78	248.83	177.33	173.58	-	-	-
CENTRE RECT	LOW	57.49	434.86	266.89	182.69	176.78	-	-	-
CENTRE RECT	HIGH	230.51	2406.17	853.19	766.95	744.86	-	-	-
CENTRE RECT	HIGH	210.06	> 3600	1252.03	766.48	776.72	-	-	-
CROSS	SHARP	63.3	594.19	217.5	173.19	172.05	-	-	-
CROSS	LOW	57.84	774.3	267.69	177.64	178.72	-	-	-
CROSS	LOW	55.67	681.16	272.36	178.05	181.17	-	-	-
CROSS	HIGH	184.67	> 3600	1326.47	762.67	766.61	-	-	-
CROSS	HIGH	244.08	> 3600	1295.44	899.19	927.45	-	-	-

Table D.6: Vision 35 × 35 ($n = 1225, m = 9318$): linearizations and quadratizations computing times.

Instance (35 × 35)		Resolution time (secs)							
		Linearizations		Pairwise covers			Termwise quadratizations		
Base	Quality	SL-2L	SL	PC1	PC2	PC3	Ishikawa	n/4	logn-1
top left rect	sharp	76.31	-	315.58	352.89	342.61	-	-	-
top left rect	low	73.66	-	340.47	368.16	370.92	-	-	-
top left rect	low	74.08	-	379.45	393.61	401.48	-	-	-
top left rect	high	382.38	-	2516.42	2254.06	2319.98	-	-	-
top left rect	high	514.47	-	2972.42	2619.05	2732.97	-	-	-
centre rect	sharp	76.83	-	404.94	401.17	386.08	-	-	-
centre rect	low	86.34	-	406.74	456.48	450.25	-	-	-
centre rect	low	86.94	-	447.44	458.27	467.14	-	-	-
centre rect	high	313.95	-	1993.61	2268.91	2274.39	-	-	-
centre rect	high	389.92	-	1958.56	1623.27	1609.41	-	-	-
cross	sharp	93.92	-	361.91	374.63	368.63	-	-	-
cross	low	101.94	-	409.05	428.02	411.23	-	-	-
cross	low	94.09	-	377.7	472.42	467.03	-	-	-
cross	high	396.16	-	2154.95	1603.64	1609.47	-	-	-
cross	high	254.72	-	2313.64	1871.97	1938.81	-	-	-

Table D.7: VISION 35×35 ($n = 1225, m = 9318$): Java model creation times.

Instance (35×35)		Model creation time (Java) (secs)							
		Linearizations		Pairwise covers			Termwise quadratizations		
Base	Quality	SL-2L	SL	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
TOP LEFT RECT	SHARP	414.69	-	1.42	146.11	265.39	-	-	-
TOP LEFT RECT	LOW	301.34	-	0.53	131.84	258.08	-	-	-
TOP LEFT RECT	LOW	624.92	-	0.55	150.39	232.52	-	-	-
TOP LEFT RECT	HIGH	344.62	-	0.58	151.94	231.02	-	-	-
TOP LEFT RECT	HIGH	373.53	-	-0.42	126.95	214.03	-	-	-
CENTRE RECT	SHARP	366.17	-	0.06	141.83	213.92	-	-	-
CENTRE RECT	LOW	460.66	-	0.26	139.52	216.75	-	-	-
CENTRE RECT	LOW	601.06	-	0.56	139.73	213.86	-	-	-
CENTRE RECT	HIGH	596.05	-	0.39	140.09	209.61	-	-	-
CENTRE RECT	HIGH	596.08	-	0.44	138.73	207.59	-	-	-
CROSS	SHARP	598.08	-	0.09	141.37	213.37	-	-	-
CROSS	LOW	598.06	-	-0.05	138.98	213.77	-	-	-
CROSS	LOW	598.91	-	0.3	138.58	214.97	-	-	-
CROSS	HIGH	600.84	-	0.05	136.36	215.53	-	-	-
CROSS	HIGH	597.28	-	0.36	140.03	210.19	-	-	-

Table D.8: VISION 35×35 ($n = 1225, m = 9318$): total times (model creation + resolution).

Instance (35×35)		Model + resolution time (secs)							
		Linearizations		Pairwise covers			Termwise quadratizations		
Base	Quality	SL-2L	SL	PC1	PC2	PC3	ISHIKAWA	N/4	LOGN-1
TOP LEFT RECT	SHARP	491	-	317	499	608	-	-	-
TOP LEFT RECT	LOW	375	-	341	500	629	-	-	-
TOP LEFT RECT	LOW	699	-	380	544	634	-	-	-
TOP LEFT RECT	HIGH	727	-	2517	2406	2551	-	-	-
TOP LEFT RECT	HIGH	888	-	2972	2746	2947	-	-	-
CENTRE RECT	SHARP	443	-	405	543	600	-	-	-
CENTRE RECT	LOW	547	-	407	596	667	-	-	-
CENTRE RECT	LOW	688	-	448	598	681	-	-	-
CENTRE RECT	HIGH	910	-	1994	2409	2484	-	-	-
CENTRE RECT	HIGH	986	-	1959	1762	1817	-	-	-
CROSS	SHARP	692	-	362	516	582	-	-	-
CROSS	LOW	700	-	409	567	625	-	-	-
CROSS	LOW	693	-	378	611	682	-	-	-
CROSS	HIGH	997	-	2155	1740	1825	-	-	-
CROSS	HIGH	852	-	2314	2012	2149	-	-	-

D.2 2-links heuristic

Tables D.9, D.10, D.11, D.12 and D.13 present the execution times of the 2-links heuristic for image sizes 10×10, 15×15, 20×20, 25×25, 30×30 and 35×35, respectively. Some interpretations of these figures have been made in Section 9.3.1.

Table D.9: VISION 10×10 ($n = 100, m = 668$): 2-links heuristic computing times.

Instance (10×10)		Resolution time (secs) for a given number of 2-links (2L)					
Base	Quality	1000 2L	2000 2L	3000 2L	4000 2L	5000 2L	5200 2L
TOP LEFT RECT	SHARP	3.03	2.25	1.98	1.02	0.94	0.88
TOP LEFT RECT	LOW	3.92	2.81	1.69	1	0.94	1.13
TOP LEFT RECT	LOW	3.23	2.63	2.41	0.92	0.75	0.94
TOP LEFT RECT	HIGH	11.39	11.31	4.02	2.84	3.27	3.23
TOP LEFT RECT	HIGH	7.06	5.14	4.14	2.27	3.48	3.78
CENTRE RECT	SHARP	3.39	2.58	1.34	1.31	1.05	1.03
CENTRE RECT	LOW	6.36	4.53	2.36	1.47	1.52	1.44
CENTRE RECT	LOW	3.8	3.09	1.84	1.11	1.14	1.16
CENTRE RECT	HIGH	6.11	5.27	3.8	2.2	1.97	1.7
CENTRE RECT	HIGH	10.5	8.51	3.36	2.95	2.36	2.25
CROSS	SHARP	4.36	2.84	1.78	0.97	0.97	1.08
CROSS	LOW	3.97	3.69	1.8	1.16	1.14	1.22
CROSS	LOW	3.02	2.27	2	1.03	1.13	1.06
CROSS	HIGH	11.42	7.53	3.5	1.86	2.11	2.55
CROSS	HIGH	12.95	7.53	8.28	2.44	1.73	1.76

Table D.10: Vision 15 × 15 (n = 225, m = 1598): 2-links heuristic computing times.

Instance (15 × 15)		Resolution time (secs) for a given number of 2-links (2L)					
Base	Quality	2500 2L	5000 2L	7500 2L	10000 2L	12500 2L	12900 2L
top left rect	sharp	14.33	10.36	7.25	4.63	3.73	3.7
top left rect	low	17.44	13.02	6.13	4.11	4.34	4.45
top left rect	low	17.78	12.33	9.98	5.16	4.78	4.59
top left rect	high	90.84	27.52	19.76	13.84	9.25	9.34
top left rect	high	162.74	56.8	31.64	20.13	10.99	12.02
centre rect	sharp	14.13	21.02	10	5.14	5.42	5.59
centre rect	low	32.5	19.75	13.84	6.34	5	5.11
centre rect	low	31.55	17.31	11.08	5.98	4.92	4.98
centre rect	high	71.45	27.19	32.22	13.78	10.13	9.41
centre rect	high	52.78	35.55	35.44	18.31	12.83	10.33
cross	sharp	25.31	12.75	9.09	5.28	4.63	5.09
cross	low	36.13	14.47	9.84	5.7	5.17	5.31
cross	low	27.56	16.39	10.02	5.97	5.03	5.22
cross	high	73.84	43.94	28.89	14	11.63	12.27
cross	high	53.11	43.33	20.33	13.78	11.55	11.83

Table D.11: VISION 20 × 20 ($n = 400, m = 2928$): 2-links heuristic computing times.

Instance (20 × 20)		Resolution time (secs) for a given number of 2-links (2L)					
Base	Quality	4700 2L	9400 2L	14100 2L	18800 2L	23500 2L	23900 2L
TOP LEFT RECT	SHARP	60.78	46.09	24.09	14.33	12.7	12.44
TOP LEFT RECT	LOW	51.67	55.05	29.48	14.8	14.84	15.13
TOP LEFT RECT	LOW	72.63	64.66	36.33	18.17	14.13	14.06
TOP LEFT RECT	HIGH	238.06	220.41	128.67	54.7	42.3	45.14
TOP LEFT RECT	HIGH	395.41	483.01	96.2	40.92	47.58	41.42
CENTRE RECT	SHARP	72.39	40.08	40.08	16.59	15.61	15.22
CENTRE RECT	LOW	133.66	72.28	40.24	19.33	15.58	16.75
CENTRE RECT	LOW	97.55	46.56	30.98	17.56	17.95	18.47
CENTRE RECT	HIGH	720.5	241.14	178.22	76.69	42.58	44.22
CENTRE RECT	HIGH	157.08	146.59	116.45	59.47	36.91	34.39
CROSS	SHARP	74.25	50.17	27.13	15.31	15.56	16.59
CROSS	LOW	88.84	67.22	29.36	17.34	15.42	14.89
CROSS	LOW	64.52	71.67	35.09	16.19	14.47	15.89
CROSS	HIGH	705.86	292.05	166.75	53.45	37.83	37.98
CROSS	HIGH	258.69	222.28	276.94	62.08	41.55	44.38

Table D.12: Vision 25 × 25 ($n = 625, m = 4658$): 2-links heuristic computing times.

Instance (25 × 25)		Resolution time (secs) for a given number of 2-links (2L)					
Base	Quality	7600 2L	15200 2L	22800 2L	30400 2L	38000 2L	38400 2L
TOP LEFT RECT	SHARP	109.09	90.39	82.22	36.42	27.89	28.09
TOP LEFT RECT	LOW	143.61	130.63	75.44	41.03	29.06	29.69
TOP LEFT RECT	LOW	225.26	132.83	102.75	43.8	29.19	29.3
TOP LEFT RECT	HIGH	1984.75	586.14	908	121.53	87.95	87.73
TOP LEFT RECT	HIGH	> 3600	940.38	399.98	113	105.5	100.02
CENTRE RECT	SHARP	130.02	96.75	68	33.2	32.55	33.27
CENTRE RECT	LOW	208.94	190.55	87.06	35.05	29.44	30.44
CENTRE RECT	LOW	236.13	122.25	78.86	43.88	37.61	36.19
CENTRE RECT	HIGH	1252.16	1897.95	1053.2	133.7	100.41	97.75
CENTRE RECT	HIGH	1603.17	627.94	478.78	181.89	92.09	102.91
CROSS	SHARP	248.05	108.75	50.39	33.17	31.67	31.3
CROSS	LOW	404.95	191.33	70.19	40.75	35.83	35.24
CROSS	LOW	310.69	172.76	70.69	36.51	37.86	37.97
CROSS	HIGH	2660.97	1074.94	469.36	125.95	93.88	89.58
CROSS	HIGH	943.11	1642.26	471.48	157.02	69.14	70.2

Table D.13: VISION 30 × 30 ($n = 900, m = 6788$): 2-links heuristic computing times.

Instance (30 × 30)		Resolution time (secs) for a given number of 2-links (2L)					
Base	Quality	11200 2L	22400 2L	33600 2L	44800 2L	56000 2L	56200 2L
TOP LEFT RECT	SHARP	267.03	206.69	127.53	71.14	44.47	44.98
TOP LEFT RECT	LOW	401.22	339.09	202.16	97.38	54.11	55.39
TOP LEFT RECT	LOW	509.41	315.44	156.02	89.28	52.55	51.81
TOP LEFT RECT	HIGH	> 3600	> 3600	2880.06	190.42	179.64	179.55
TOP LEFT RECT	HIGH	> 3600	> 3600	> 3600	195.41	201.94	205.64
CENTRE RECT	SHARP	443.73	289.56	130.11	57.09	54.66	55.69
CENTRE RECT	LOW	535.81	333.64	204.02	59.08	58.33	58.3
CENTRE RECT	LOW	669.86	304.31	137.83	73.8	52.11	53.51
CENTRE RECT	HIGH	> 3600	> 3600	2275.77	485.51	232.66	235.66
CENTRE RECT	HIGH	> 3600	1923.78	2097.17	663.36	153.06	169.98
CROSS	SHARP	377.38	246.56	122.41	62.5	61.81	61.3
CROSS	LOW	359.13	326.58	137	65.31	50.06	51.63
CROSS	LOW	504.41	293.11	115.8	64.34	56.78	56.72
CROSS	HIGH	2985	> 3600	2138.06	428.61	140.97	174.88
CROSS	HIGH	> 3600	3229.01	2402.56	560.78	263.38	394.14

Appendix E

Figures: Random same degree instances

E.1 Computing times of linearizations and quadratizations

This appendix contains figures summarizing the results of the computational experiments comparing linearization and quadratization methods for Random same degree instances with $n = 400$ and 600 variables. The analysis of these figures is analogous to the analysis made in Section 8.4.1 for instances with $n = 200$.

As a reminder, Figures E.1, E.3, E.5, E.6, E.2, E.4, E.7 and E.8, show the instance identifier on the x-axis and the y-axis represents execution times of the methods listed in the legends. Instance identifiers are of the form *n-m-d-id*, where n is the number of variables of the polynomial, m is the number of monomials, d is the degree and *id* is simply an id for the polynomial. For example, instance 400-700-3-2 has 400 variables, 700 terms all of which have degree 4 and it is the second polynomial with these parameters. Tables corresponding to the figures in this appendix can be found in Appendix B.

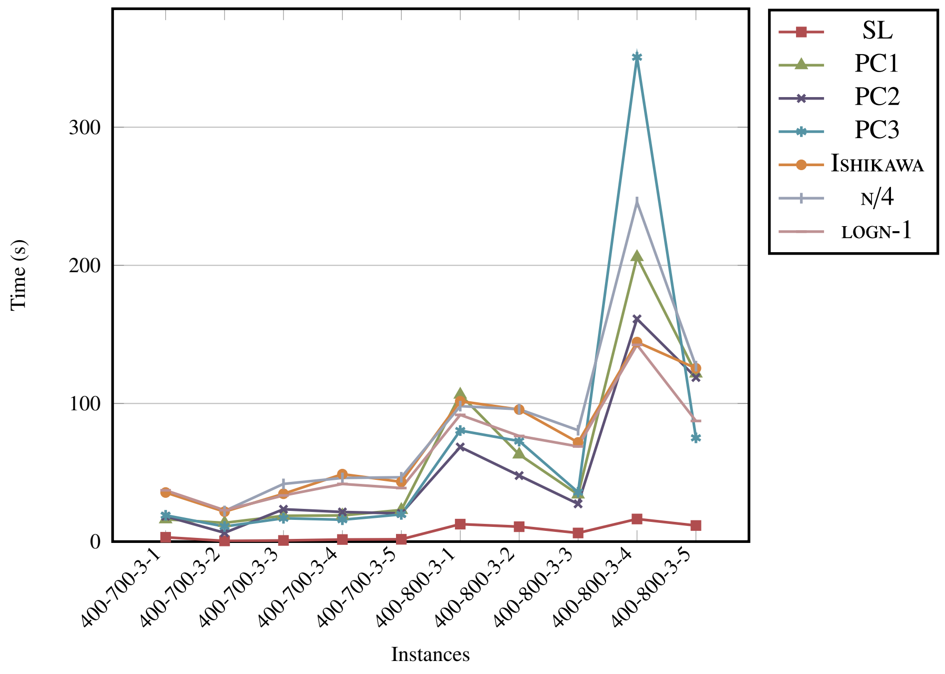


Figure E.1: Random same degree ($n = 400, \deg = 3$): linearizations and quadratizations computing times.

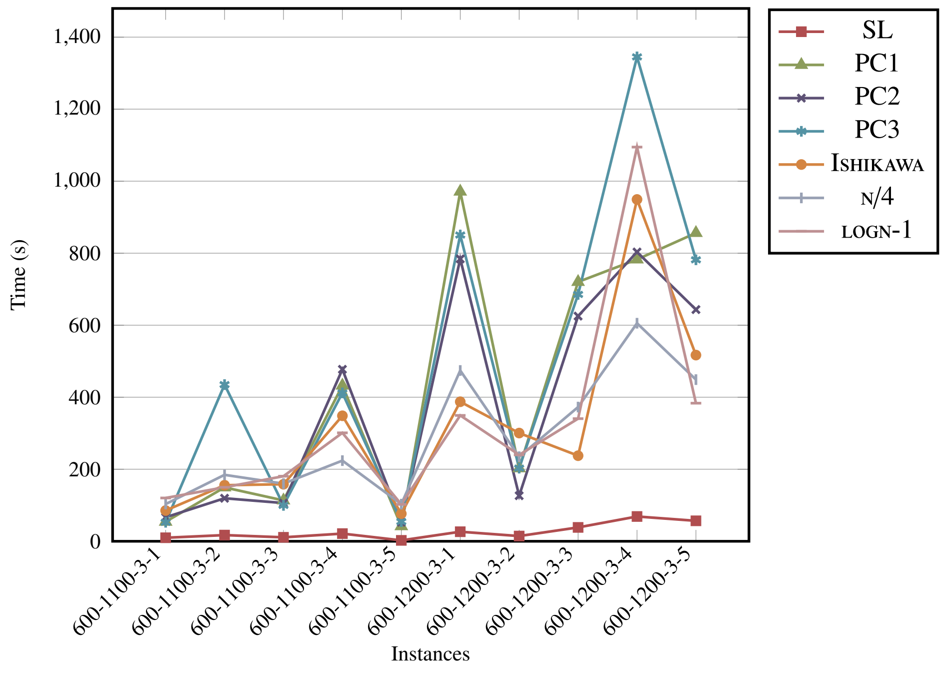


Figure E.2: Random same degree ($n = 600, \deg = 3$): linearizations and quadratizations computing times.

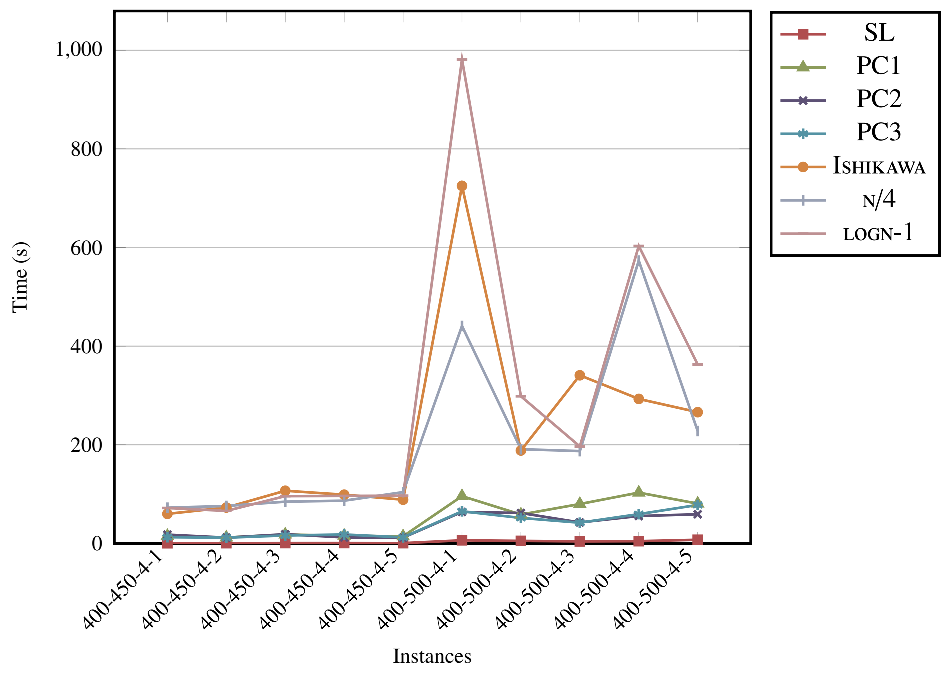


Figure E.3: Random same degree ($n = 400, \deg = 4$): linearizations and quadratizations computing times.

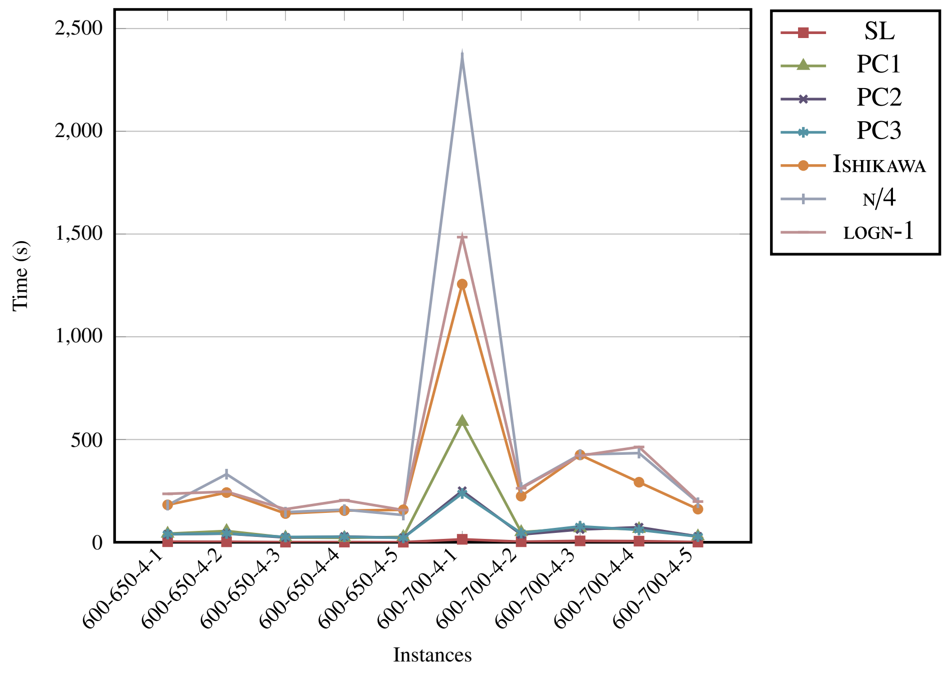


Figure E.4: Random same degree ($n = 600, \deg = 4$): linearizations and quadratizations computing times.

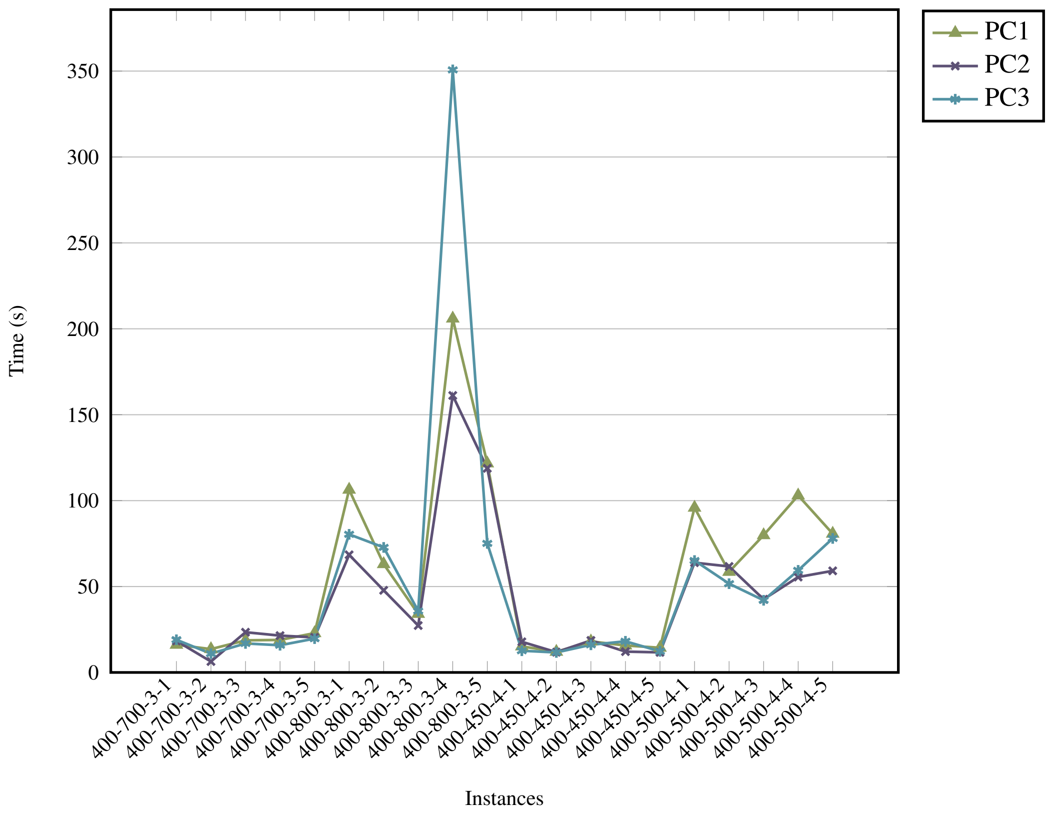


Figure E.5: Random same degree (n = 400): pairwise covers computing times by degree.

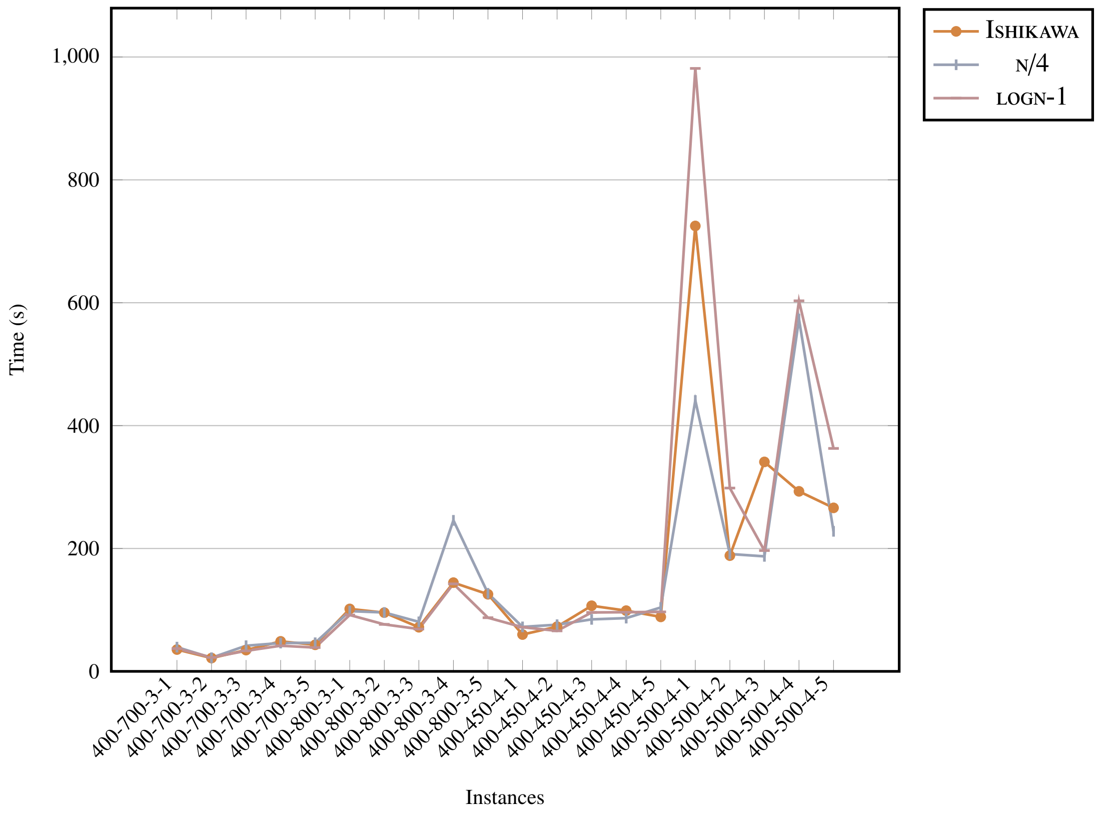


Figure E.6: Random same degree (n = 400): termwise quadratizations computing times by degree.

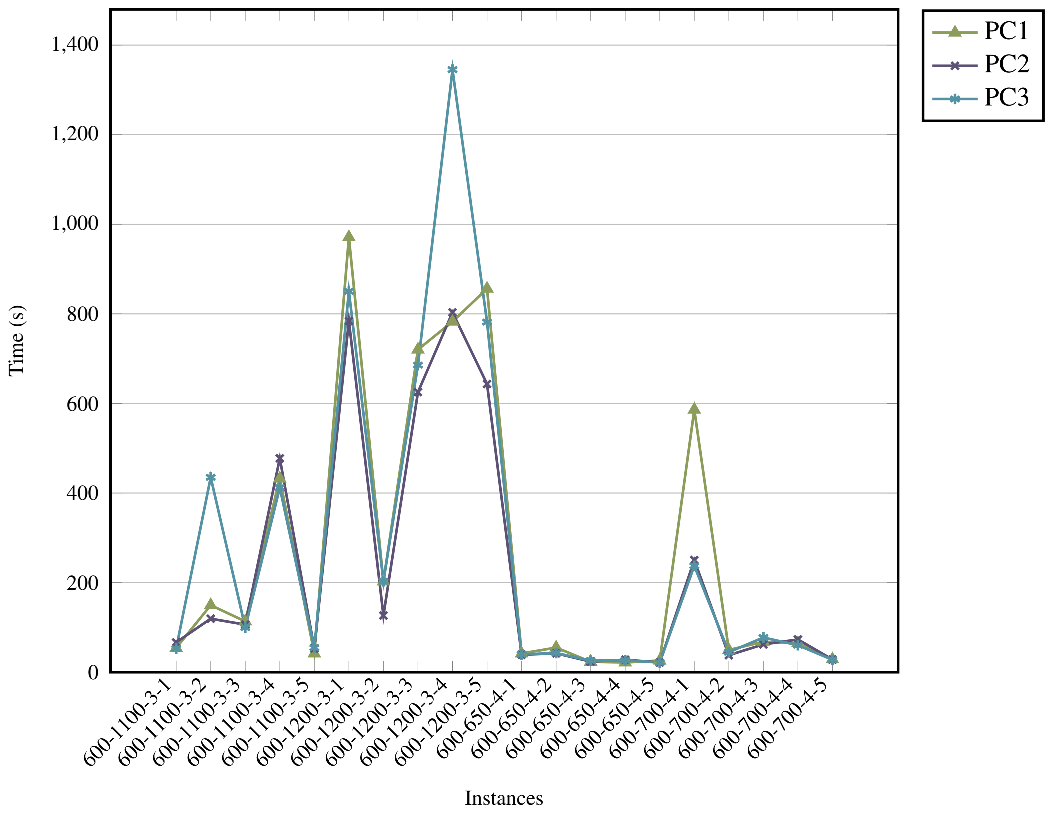


Figure E.7: Random same degree (n = 600): pairwise covers computing times by degree.

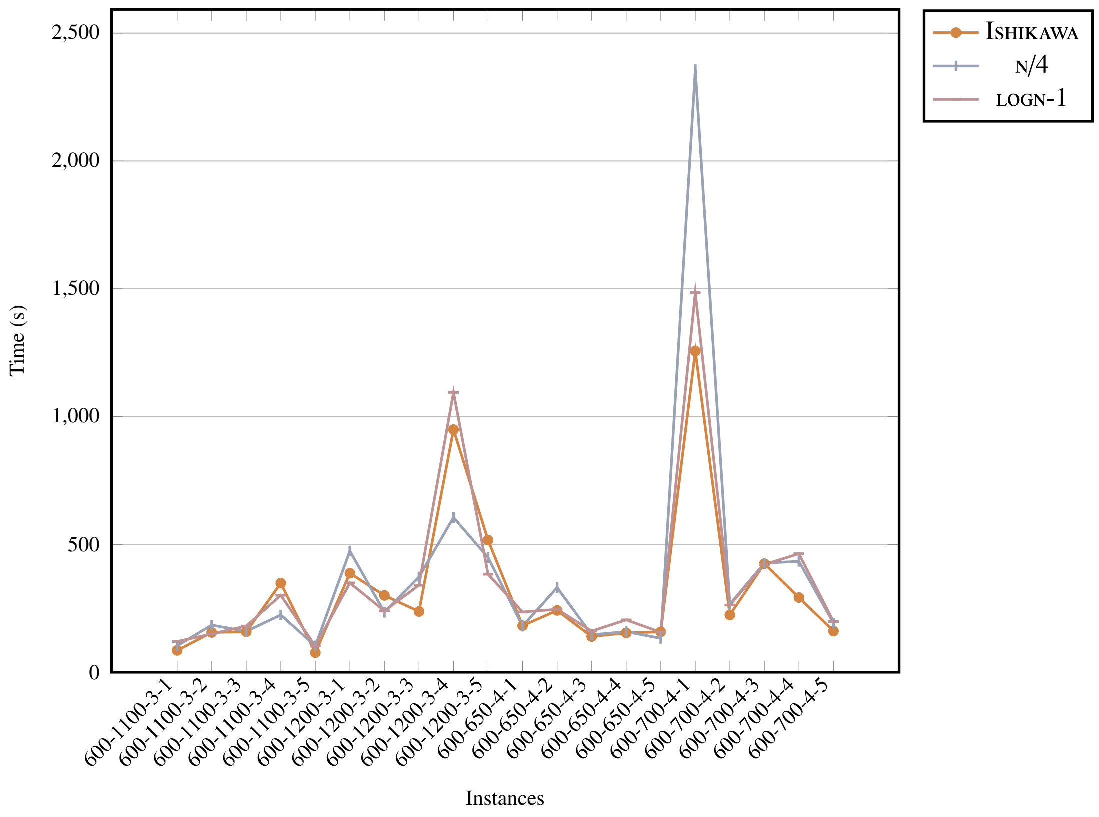


Figure E.8: Random same degree ($n = 600$): termwise quadratizations computing times by degree.

Appendix F

Figures: Random high degree instances

F.1 Computing times of linearizations and quadratizations

This appendix contains figures summarizing the results of the computational experiments comparing linearization and quadratization methods for Random high degree instances with $n = 400$ and 600 variables. The analysis of these figures is analogous to the analysis made in Section 8.4.2 for instances with $n = 200$.

Figures F.1, F.2, F.3 and F.4, show the instance identifier on the x-axis and the y-axis represents the execution times of the methods listed in the legends. Instance identifiers are of the form *n-m-d-id*, where n is the number of variables of the polynomial, m is the number of monomials, d is the degree of the polynomial and *id* is simply an id for the polynomial. For example, instance 400-450-11-3 has 400 variables, 450 terms, the degree of the polynomial is 11 and it is the third polynomial with these parameters. Only parameters n and m are fixed; the degree of the polyomial is determined randomly, as is the degree of each monomial. Tables corresponding to the figures in this appendix can be found in Appendix C.

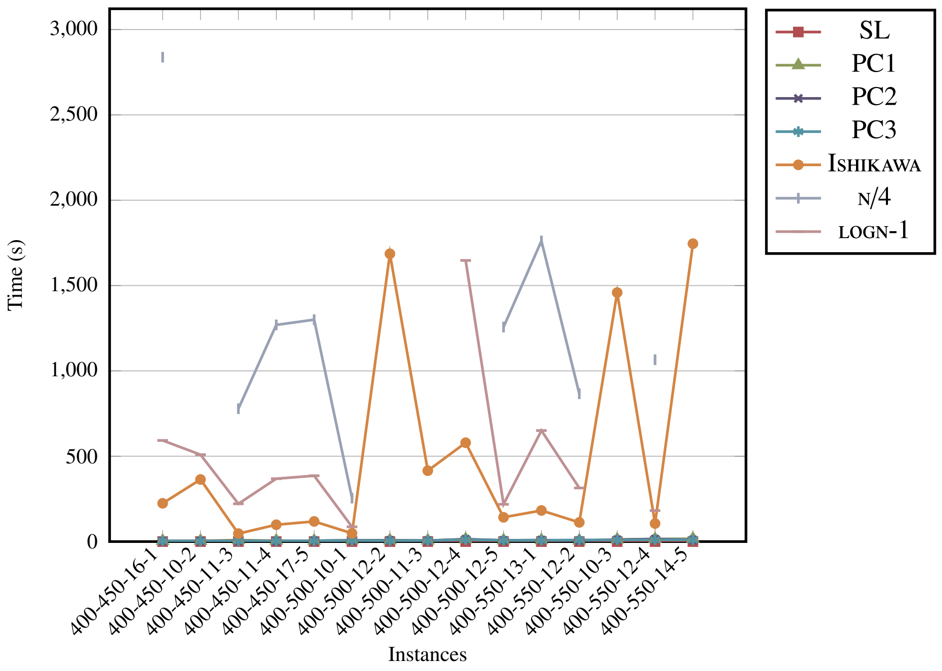


Figure F.1: Random high degree (n = 400): linearizations and quadratizations computing times.

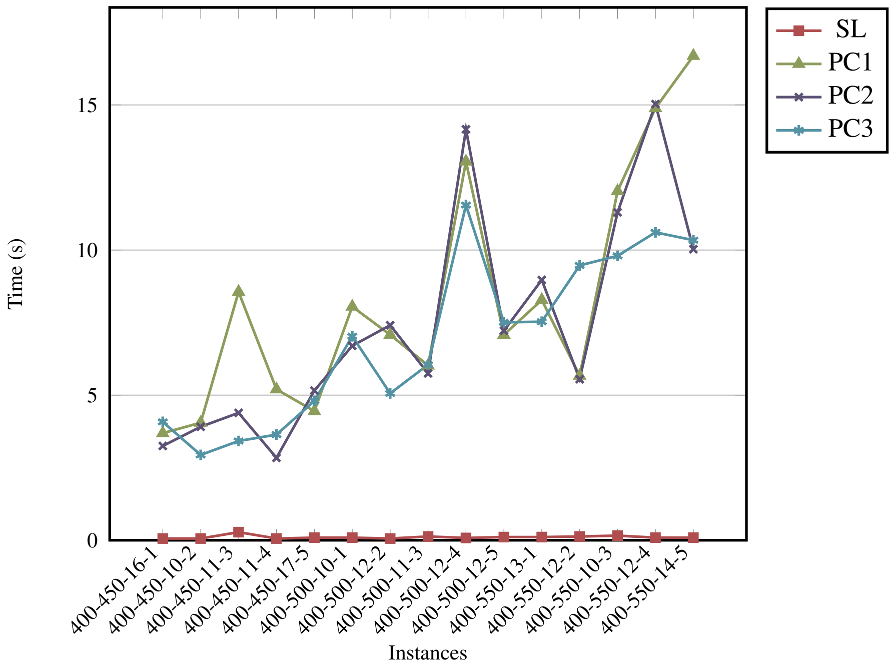


Figure F.2: Random high degree (n = 400): linearizations and quadratizations fastest computing times.

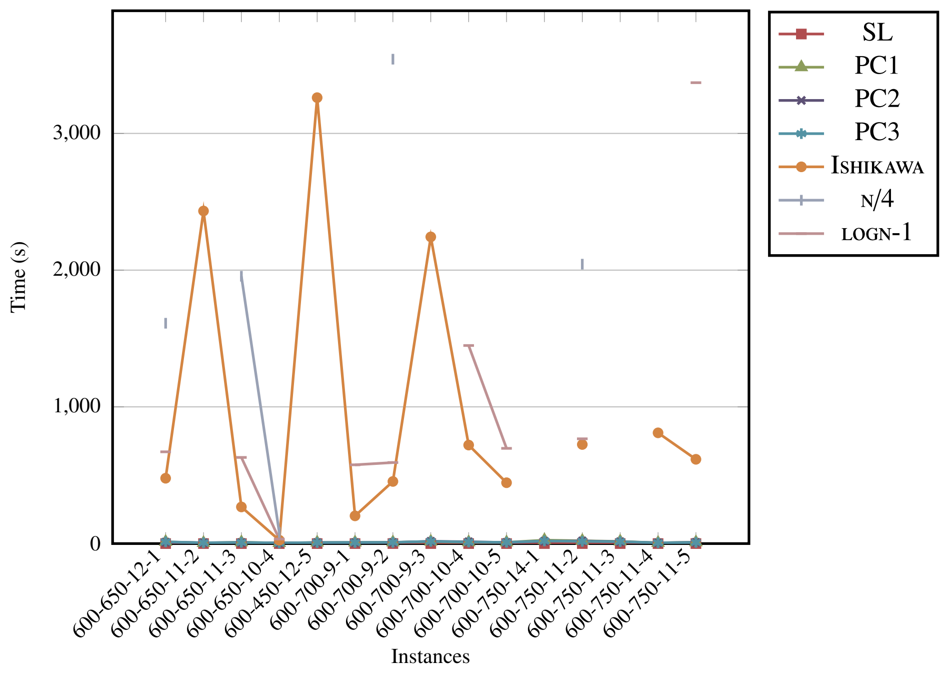


Figure F.3: Random high degree (n = 600): linearizations and quadratizations computing times.

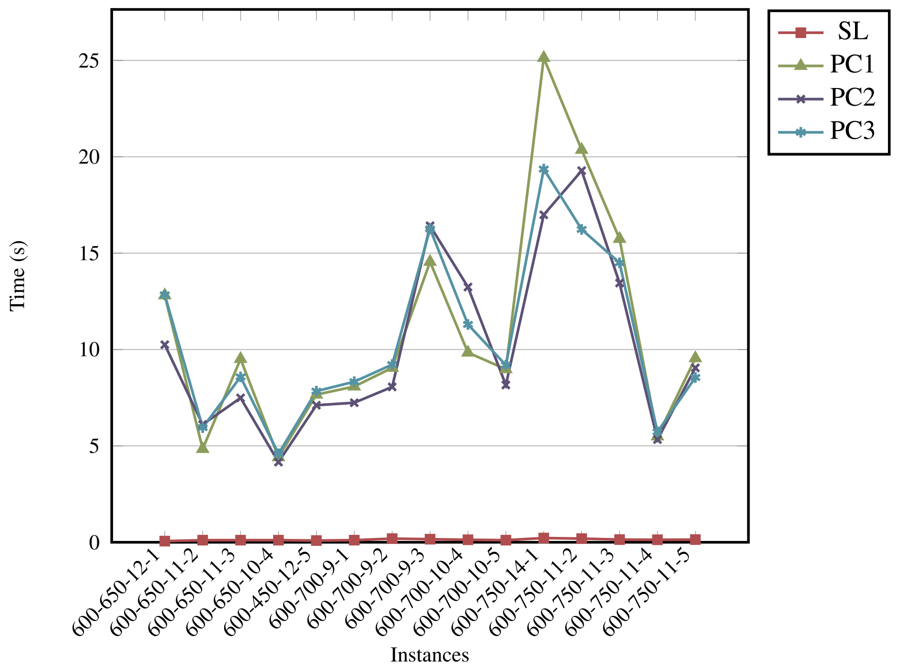


Figure F.4: Random high degree (n = 600): linearizations and quadratizations fastest computing times.

Appendix G

Figures: Vision instances

G.1 Computing times of linearizations and quadratizations

This section contains figures summarizing the results of the computational experiments comparing linearization and quadratization methods for Vision instances with base images of sizes 10×10, 20×20, 25×25 and 30×30, using $n = 100$, 400, 625 and 900 variables, respectively. The analysis of these figures is analogous to the analysis made in Section 8.4.3 for instances with image size 15×15 and $n = 225$.

Figures G.1, G.2, G.3, G.4 and G.5, present the instance identifier on the x-axis and the y-axis represents the execution times of the methods listed in the legends. Instance identifiers are of the form `base-perturbation-id`, where `base` can be equal to `tl` (top left rectangle), `cr` (centre rectangle) or `cx` (cross), `perturbation` can be equal to `s` (sharp, no perturbation), `l` (low perturbation) or `h` (high perturbation) and `id` can be equal to 1 or 2. Only instances with a low or high perturbation use an `id`, because for these perturbation settings we have generated two samples, while there is only one possible sharp image. Tables corresponding to the figures in this appendix can be found in Appendix D.

G.2 2-links heuristic

This section contains the figures of the results of computational experiments on the 2-links heuristic for Vision instances base images of sizes 10×10, 20×20, 25×25 and 30×30, using $n = 100$, 400, 625 and 900 variables, respectively. The analysis of these figures is analogous to the analysis made in Section 9.3.1 for instances with image size 15×15 and $n = 225$.

Figures G.6, G.7, G.8 and G.9, show on the x-axis the number of 2-links added to the pool of user cuts, and corresponding computing times are represented on the y-axis. There is a plot for each instance in the legend; instances are identified as in the previous section. Tables corresponding to the figures in this appendix can be found in Appendix D.

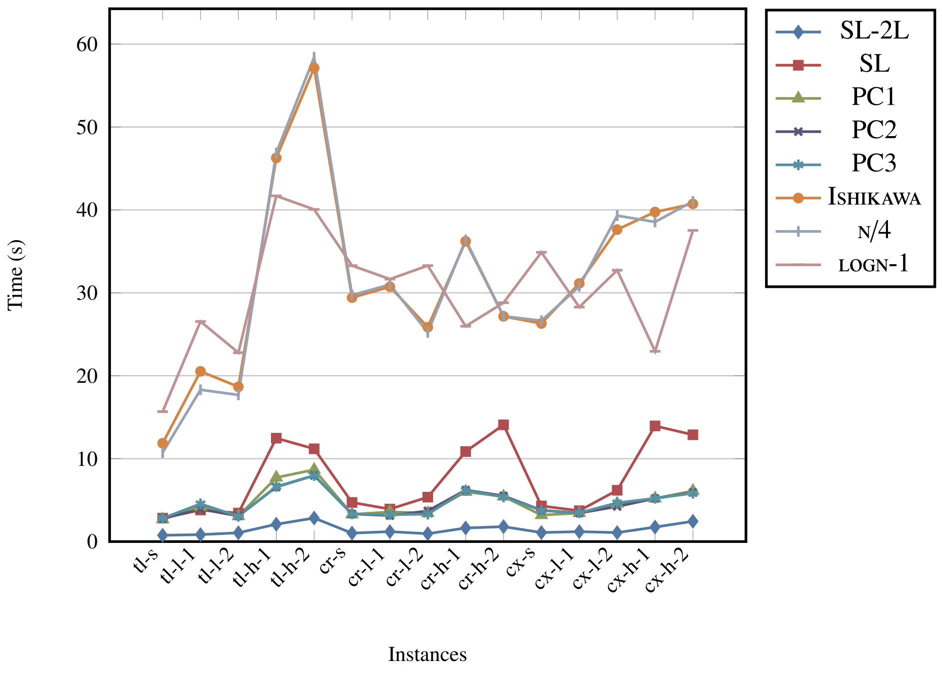


Figure G.1: VISION 10×10 ($n = 100, m = 668$): linearizations and quadratizations computing times.

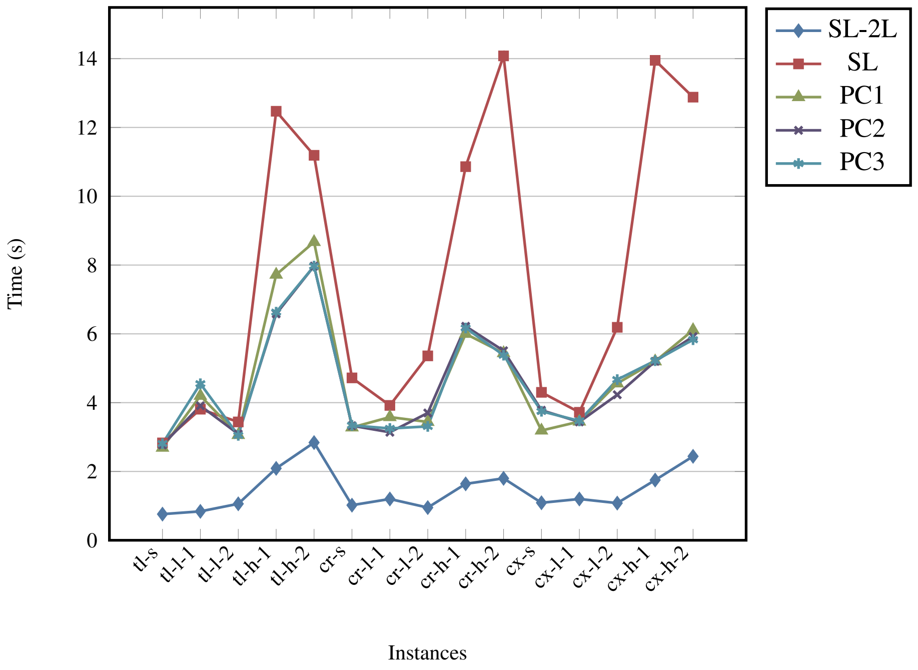


Figure G.2: VISION 10×10 ($n = 100, m = 668$): linearizations and quadratizations fastest computing times.

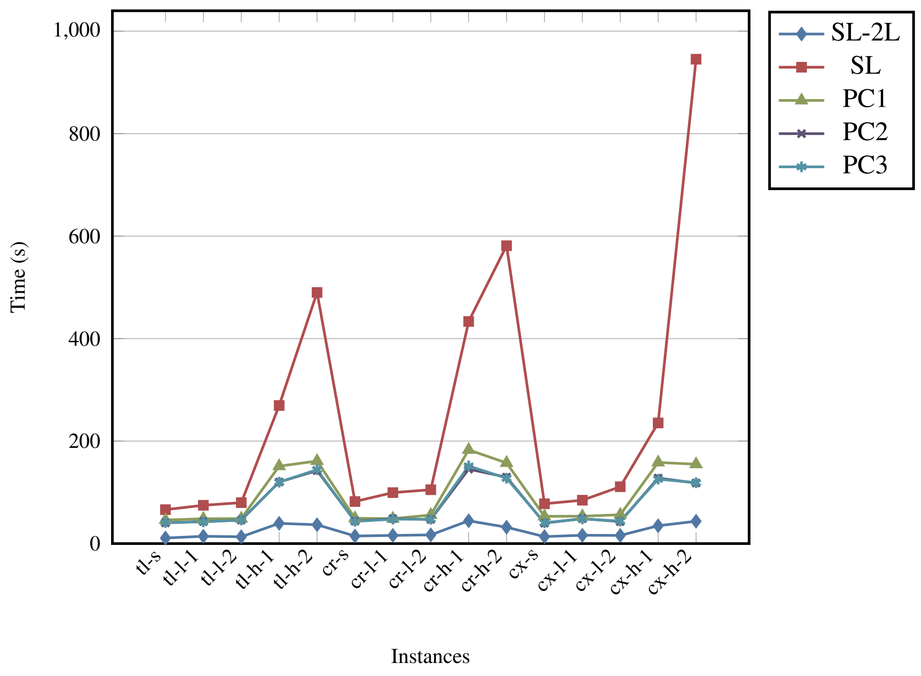


Figure G.3: Vision 20×20 ($n = 400, m = 2928$): linearizations and quadratizations computing times.

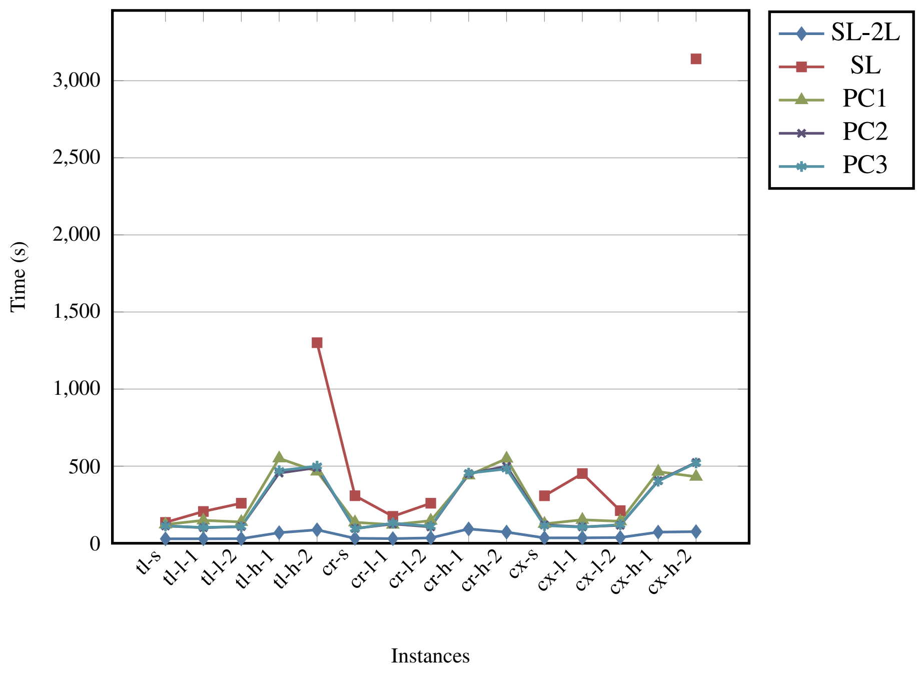


Figure G.4: Vision 25×25 ($n = 625, m = 4658$): linearizations and quadratizations computing times.

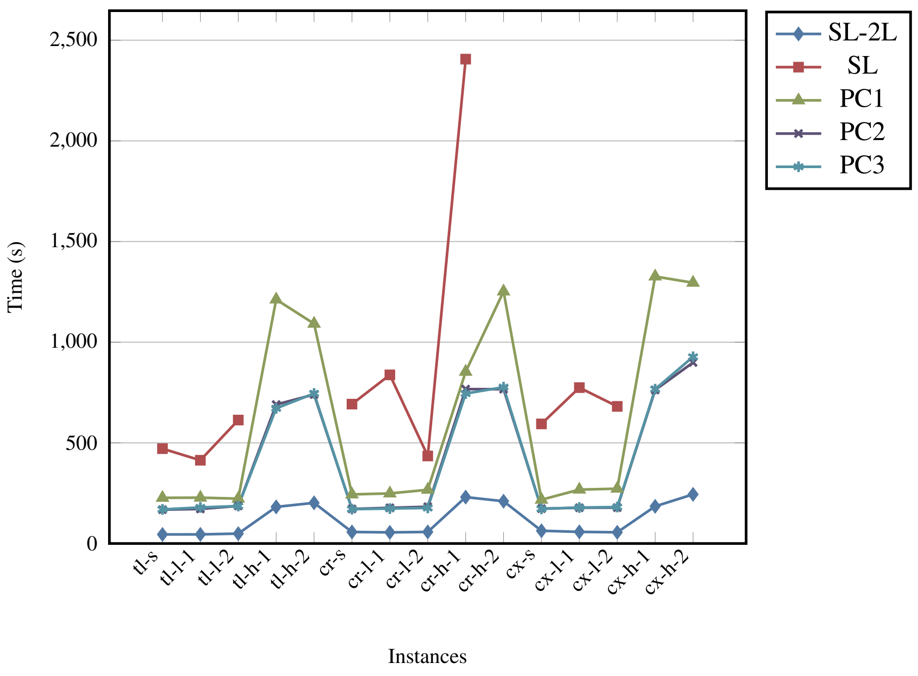


Figure G.5: VISION 30×30 ($n = 900, m = 6788$): linearizations and quadratizations computing times.

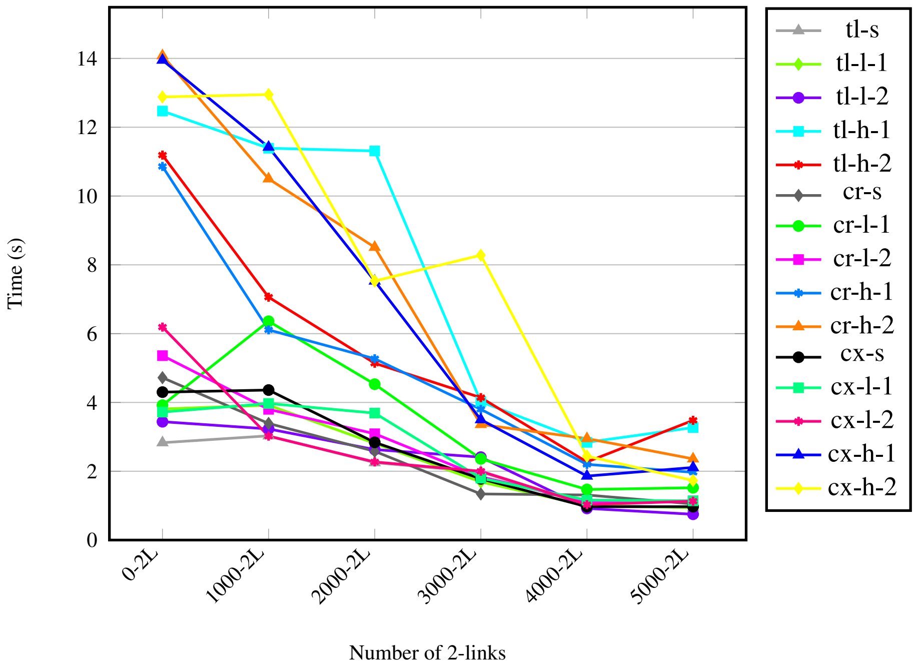


Figure G.6: VISION 10×10 ($n = 100, m = 668$): 2-links heuristic computing times.

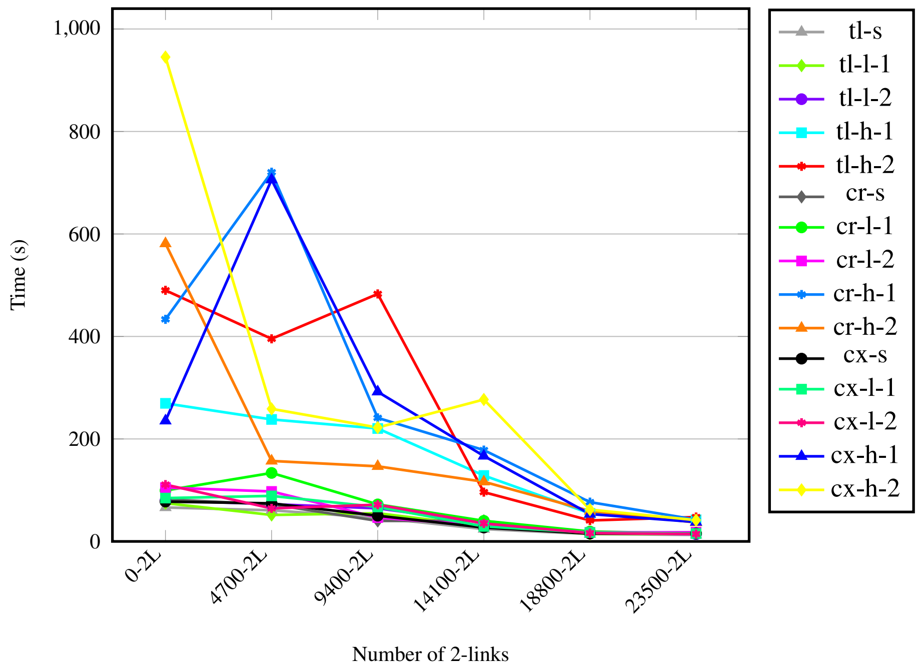


Figure G.7: Vision 20 × 20 ($n = 400, m = 2928$): 2-links heuristic computing times.

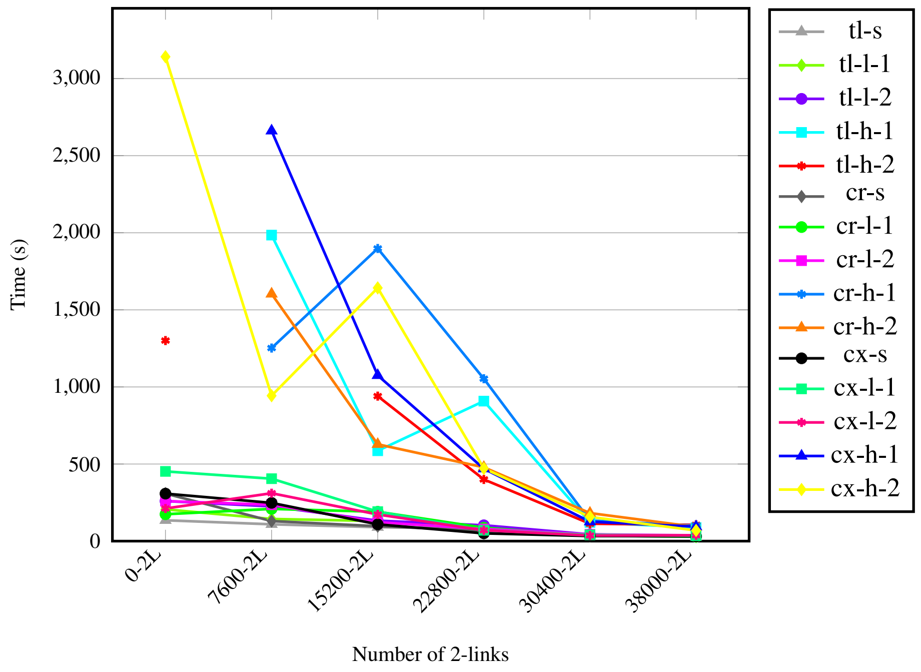


Figure G.8: Vision 25 × 25 ($n = 625, m = 4658$): 2-links heuristic computing times.

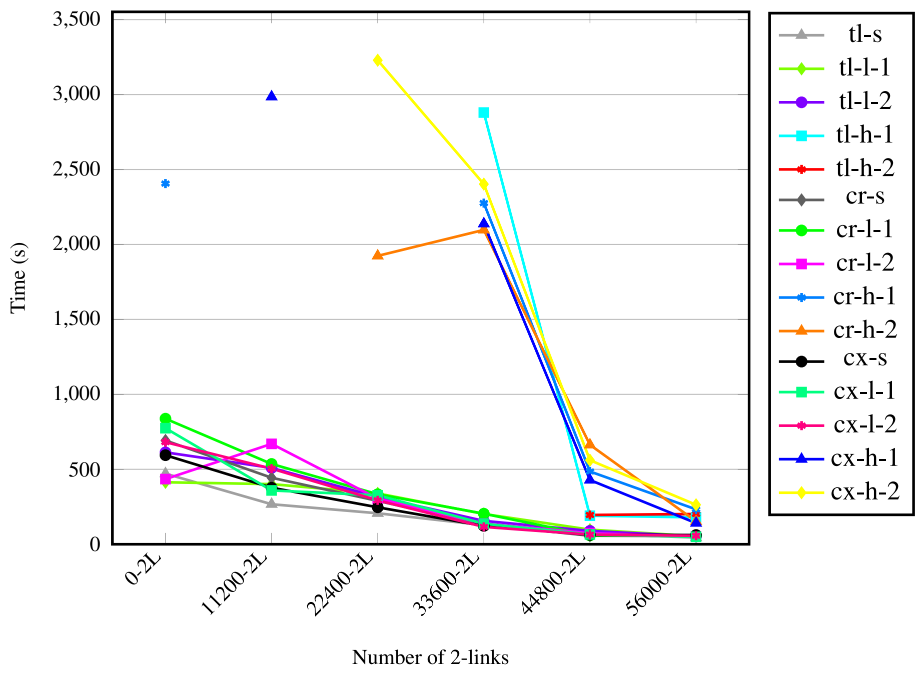


Figure G.9: VISION 30×30 ($n = 900, m = 6788$): 2-links heuristic computing times.

Bibliography

[1] F. A. Al-Khayyal and J. E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8(2):273–286, 1983. (Cited on page 16.)

[2] N. Alon and Z. Füredi. Covering the cube by affine hyperplanes. *European Journal of Combinatorics*, 14:79–83, 1993. (Cited on page 78.)

[3] R. Anstee. Hypergraphs with no special cycles. *Combinatorica*, 3(2):141–146, 1983. (Cited on page 21.)

[4] M. Anthony, E. Boros, Y. Crama, and A. Gruber. Quadratization of symmetric pseudo-Boolean functions. *Discrete Applied Mathematics*, 203:1 – 12, 2016. (Cited on pages 2, 11, 56, 60, 65, 67, 68, and 72.)

[5] M. Anthony, E. Boros, Y. Crama, and A. Gruber. Quadratic reformulations of nonlinear binary optimization problems. *Mathematical Programming*, 162(1-2):115–144, 2017. (Cited on pages 2, 7, 11, 12, 55, 59, 62, 63, 67, 68, 69, 72, 81, and 82.)

[6] E. Balas. Disjunctive programming: properties of the convex hull of feasible points. *GSIA Management Science Research Report MSRR 348, Carnegie Mellon University (1974).*, 1974. Published as invited paper in *Discrete Applied Mathematics* 89 (1998) 3–44. (Cited on pages 34 and 35.)

[7] E. Balas. Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. *SIAM Journal on Algebraic and Discrete Methods*, 6(3):466–486, 1985. (Cited on pages 34 and 35.)

[8] F. Barahona, M. Grötschel, and A. R. Mahjoub. Facets of the bipartite subgraph polytope. *Mathematics of Operations Research*, 10(2):340–358, 1985. (Cited on page 4.)

[9] F. Barahona and A. R. Mahjoub. On the cut polytope. *Mathematical Programming*, 36(2):157–173, 1986. (Cited on page 4.)

[10] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM (JACM)*, 30(3):479–513, 1983. (Cited on page 21.)

[11] C. Berge. *Graphs and Hypergraphs*. North-Holland Publishing Company, Amsterdam, 1976. (Cited on page 21.)

[12] J. Bernasconi. Low autocorrelation binary sequences: statistical mechanics and configuration space analysis. *J. Phys. France*, 48(4):559–567, 1987. (Cited on page 99.)

[13] A. Billionnet and S. Elloumi. Using a mixed integer quadratic programming solver for the unconstrained quadratic 0–1 problem. *Mathematical Programming*, 109(1):55–68, 2007. (Cited on pages 92 and 152.)

[14] A. Billionnet, S. Elloumi, and A. Lambert. Extending the QCR method to general mixed-integer programs. *Mathematical programming*, 131(1–2):381–401, 2012. (Cited on pages 92 and 152.)

[15] C. Bliek, P. Bonami, and A. Lodi. Solving mixed-integer quadratic programming problems with IBM-CPLEX: a progress report. In *Proceedings of the Twenty-Sixth RAMP Symposium*, pages 171–180, 2014. (Cited on page 92.)

[16] P. Bonami, M. Kilinç, and J. Linderoth. Algorithms and software for convex mixed integer nonlinear programs. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, pages 1–39, 2012. (Cited on page 2.)

[17] E. Boros, Y. Crama, and P. L. Hammer. Upper-bounds for quadratic 0–1 maximization. *Operations Research Letters*, 9(2):73–79, 1990. (Cited on page 4.)

[18] E. Boros, Y. Crama, and P. L. Hammer. Chvátal cuts and odd cycle inequalities in quadratic 0–1 optimization. *SIAM Journal on Discrete Mathematics*, 5(2):163–177, 1992. (Cited on page 4.)

[19] E. Boros, Y. Crama, and E. Rodríguez-Heck. Compact quadratizations for pseudo-Boolean functions. Submitted, 2018. (Cited on pages 11 and 65.)

[20] E. Boros, Y. Crama, and E. Rodríguez-Heck. Quadratizations of symmetric pseudo-Boolean functions: sub-linear bounds on the number of auxiliary variables. In *ISAIM*, Fort Lauderdale, 2018. ISAIM, International Symposium on Artificial Intelligence and Mathematics. Available at http://isaim2018.cs.virginia.edu/. (Cited on page 76.)

[21] E. Boros and A. Gruber. On quadratization of pseudo-Boolean functions. In *ISAIM*, Fort Lauderdale, 2012. ISAIM, International Symposium on Artificial Intelligence and Mathematics. Open Source: arXiv:1404.6538v1. (Cited on pages 56 and 60.)

[22] E. Boros and P. L. Hammer. The max-cut problem and quadratic 0-1 optimization; polyhedral aspects, relaxations and bounds. *Annals of Operations Research*, 33(3):151–180, 1991. (Cited on page 4.)

[23] E. Boros and P. L. Hammer. Cut-polytopes, Boolean quadric polytopes and non-negative quadratic pseudo-Boolean functions. *Mathematics of Operations Research*, 18(1):245–253, 1993. (Cited on page 4.)

[24] E. Boros and P. L. Hammer. Pseudo-Boolean optimization. *Discrete Applied Mathematics*, 123(1):155–225, 2002. (Cited on pages 1, 2, 3, 9, 62, 82, and 102.)

[25] E. Boros, P. L. Hammer, X. Sun, and G. Tavares. A max-flow approach to improved lower bounds for quadratic unconstrained binary optimization (QUBO). *Discrete Optimization*, 5(2):501–529, 2008. In Memory of George B. Dantzig. (Cited on pages 7, 92, 101, 102, 112, 151, and 152.)

[26] F. Boukouvala, R. Misener, and C. A. Floudas. Global optimization advances in mixed-integer nonlinear programming, MINLP, and constrained derivative-free optimization, CDFO. *European Journal of Operational Research*, 252(3):701–727, 2016. (Cited on page 2.)

[27] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001. (Cited on page 7.)

[28] C. Buchheim, Y. Crama, and E. Rodríguez-Heck. Berge-acyclic multilinear 0-1 optimization problems. Submitted, 2017. (Cited on pages 10, 17, and 19.)

[29] C. Buchheim and L. Klein. Combinatorial optimization with one quadratic term: spanning trees and forests. *Discrete Applied Mathematics*, 177:34–52, 2014. (Cited on pages 17 and 40.)

[30] C. Buchheim and G. Rinaldi. Efficient reduction of polynomial zero-one optimization to the quadratic case. *SIAM Journal on Optimization*, 18(4):1398–1413, 2007. (Cited on pages 2, 10, 11, 17, 40, 41, 42, 44, 61, 62, 63, 82, 86, 96, 107, and 113.)

[31] C. Buchheim and G. Rinaldi. Terse integer linear programs for Boolean optimization. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:121–139, 2009. (Cited on pages 2, 10, 11, and 61.)

[32] S. Burer and A. N. Letchford. Non-convex mixed-integer nonlinear programming: a survey. *Surveys in Operations Research and Management Science*, 17(2):97–106, 2012. (Cited on page 2.)

[33] M. Conforti and G. Cornuéjols. Balanced 0,±1-matrices, bicoloring and total dual integrality. *Mathematical Programming*, 71(3):249–258, 1995. (Cited on pages 23 and 24.)

[34] M. Conforti, G. Cornuéjols, A. Kapoor, and K. Vušković. Balanced 0,±1 matrices, Parts I-II. *Journal of Combinatorial Theory, Series B*, 81(2):243–274, 2001. (Cited on page 22.)

[35] M. Conforti, G. Cornuéjols, and K. Vušković. Balanced matrices. *Discrete Mathematics*, 306(19–20):2411–2437, 2006. (Cited on pages 19, 21, 23, and 24.)

[36] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer Programming*, volume 271 of *Graduate Texts in Mathematics*. Springer, Switzerland, 2014. (Cited on pages 16, 34, and 35.)

[37] M. C. Cooper and S. Živný. Hybrid Tractable Classes of Constraint Problems. In A. Krokhin and S. Živný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 113–135. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017. (Cited on page 10.)

[38] Y. Crama. Linearization techniques and concave extensions in nonlinear 0–1 optimization. Technical Report RUTCOR Research Report 32–88, Rutgers University, New Brunswick, NJ, 1988. (Cited on pages 17 and 23.)

[39] Y. Crama. Concave extensions for nonlinear 0–1 maximization problems. *Mathematical Programming*, 61(1):53–60, 1993. (Cited on pages 16, 17, 20, and 23.)

[40] Y. Crama and P. L. Hammer. *Boolean functions: Theory, algorithms, and applications*. Cambridge University Press, New York, N. Y., 2011. (Cited on pages 1, 2, and 9.)

[41] Y. Crama and E. Rodríguez-Heck. A class of valid inequalities for multilinear 0-1 optimization problems. *Discrete Optimization*, 25:28–47, 2017. (Cited on pages 11, 17, 29, 100, 107, 113, 119, and 139.)

[42] C. D'Ambrosio and A. Lodi. Mixed integer nonlinear programming tools: a practical overview. *4OR*, 9(4):329–349, 2011. (Cited on page 2.)

[43] C. De Simone. The cut polytope and the Boolean quadric polytope. *Discrete Mathematics*, 79(1):71–75, 1990. (Cited on pages 3 and 4.)

[44] P. M. Dearing, P. L. Hammer, and B. Simeone. Boolean and graph theoretic formulations of the Simple Plant Location Problem. *Transportation Science*, 26(2):138–148, 1992. (Cited on pages 4 and 5.)

[45] A. Del Pia and A. Khajavirad. On balanced matrices and polynomial solvability of multilinear programs. Personal communication, 2016. (Cited on pages 17 and 23.)

[46] A. Del Pia and A. Khajavirad. A polyhedral study of binary polynomial programs. *Mathematics of Operations Research*, 42(2):389–410, 2016. (Cited on pages 2, 10, 17, 31, and 34.)

[47] A. Del Pia and A. Khajavirad. The multilinear polytope for acyclic hypergraphs. *SIAM Journal on Optimization*, 28(2):1049–1076, 2018. (Cited on pages 2, 10, 17, 23, 52, 57, and 150.)

[48] A. Del Pia and A. Khajavirad. On decomposability of multilinear sets. *Mathematical Programming: Series A*, 170(2):387–415, 2018. (Cited on pages 10, 17, and 57.)

[49] F. Djeumou Fomeni, K. Kaparis, and A. Letchford. Cutting planes for RLT relaxations of mixed 0-1 polynomial programs. *Mathematical Programming*, 151(2):639–658, 2015. (Cited on pages 2, 10, and 17.)

[50] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM (JACM)*, 30(3):514–550, 1983. (Cited on page 21.)

[51] A. Fischer, F. Fischer, and S. T. McCormick. Matroid optimisation problems with nested non-linear monomials in the objective function. *Mathematical Programming*, 169(2):417–446, 2018. (Cited on pages 2, 10, 17, 30, and 32.)

[52] A. Fix, A. Gruber, E. Boros, and R. Zabih. A hypergraph-based reduction for higher-order binary Markov random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(7):1387–1395, 2015. (Cited on pages 6, 46, 56, 58, 62, 63, 82, 92, and 101.)

[53] C. A. Floudas and C. E. Gounaris. A review of recent advances in global optimization. *Journal of Global Optimization*, 45(1):3, 2008. (Cited on page 2.)

[54] R. Fortet. L'algèbre de Boole et ses applications en recherche opérationnelle. *Cahiers du Centre d'Études de Recherche Opérationnelle*, 4:5–36, 1959. (Cited on pages 10 and 16.)

[55] R. Fortet. Applications de l'algèbre de Boole en recherche opérationnelle. *Revue Française d'Automatique, Informatique et Recherche Opérationnelle*, 4(14):17–26, 1960. (Cited on pages 10 and 16.)

[56] D. Freedman and P. Drineas. Energy minimization via graph cuts: settling what is possible. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 939–946, June 2005. (Cited on pages 6, 58, and 59.)

[57] D. Gamarnik, D. Shah, and Y. Wei. Belief propagation for min-cost network flow: Convergence and correctness. *Operations Research*, 60(2):410–428, 2012. (Cited on page 57.)

[58] F. Glover and E. Woolsey. Further reduction of zero-one polynomial programming problems to zero-one linear programming problems. *Operations Research*, 21(1):156–161, 1973. (Cited on pages 10 and 16.)

[59] F. Glover and E. Woolsey. Technical note: converting the 0-1 polynomial programming problem to a 0-1 linear program. *Operations Research*, 22(1):180–182, 1974. (Cited on pages 10 and 16.)

[60] B. Goldengorin and D. Krushinsky. Complexity evaluation of benchmark instances for the p-median problem. *Mathematical and Computer Modelling*, 53(9):1719–1736, 2011. (Cited on page 5.)

[61] I. E. Grossmann. Review of nonlinear mixed-integer and disjunctive programming techniques. *Optimization and Engineering*, 3(3):227–252, 2002. (Cited on page 2.)

[62] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981. (Cited on page 56.)

[63] P. L. Hammer. Some network flow problems solved with pseudo-Boolean programming. *Operations Research*, 13(3):388–399, 1965. (Cited on pages 7 and 56.)

[64] P. L. Hammer. Plant location - A pseudo-Boolean approach. *Israel Journal of Technology*, 6:330–332, 1968. (Cited on page 4.)

[65] P. L. Hammer, P. Hansen, and B. Simeone. Roof duality, complementation and persistency in quadratic 0-1 optimization. *Mathematical Programming*, 28(2):121–155, 1984. (Cited on pages 7, 92, 102, and 151.)

[66] P. L. Hammer, I. Rosenberg, and S. Rudeanu. On the determination of the minima of pseudo-Boolean functions. *Studii si Cercetari Matematice*, 14:359–364, 1963. in Romanian. (Cited on page 1.)

[67] P. L. Hammer and S. Rudeanu. *Boolean methods in operations research and related areas*. Ökonometrie und Unternehmensforschung. Springer-Verlag, Berlin, 1968. (Cited on page 1.)

[68] P. Hansen, B. Jaumard, and V. Mathon. State-of-the-art survey: constrained nonlinear 0–1 programming. *ORSA Journal on Computing*, 5(2):97–119, 1993. (Cited on page 2.)

[69] P. Hansen and B. Simeone. Unimodular functions. *Discrete Applied Mathematics*, 14(3):269 – 281, 1986. (Cited on pages 17, 20, and 23.)

[70] F. Harary. On the notion of balance of a signed graph. *Michigan Mathematical Journal*, 2(2):143–146, 1953. (Cited on page 21.)

[71] R. Hemmecke, M. Köppe, J. Lee, and R. Weismantel. Nonlinear integer programming. In *50 Years of Integer Programming 1958-2008*, pages 561–618. Springer, 2010. (Cited on page 2.)

[72] IBM. IBM ILOG CPLEX Optimizer 12.6 and 12.7. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/. (Cited on pages 41 and 93.)

[73] H. Ishikawa. Higher-order clique reduction in binary graph cut. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2993–3000, June 2009. (Cited on pages 60 and 65.)

[74] H. Ishikawa. Transformation of general binary MRF minimization to the first-order case. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(6):1234–1249, June 2011. (Cited on pages 5, 6, 46, 56, 57, 58, 60, 65, 92, and 101.)

[75] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM (JACM)*, 48(4):761–777, July 2001. (Cited on page 56.)

[76] F. Kahl and P. Strandmark. Generalized roof duality for pseudo-Boolean optimization. In *IEEE International Conference on Computer Vision*, pages 255–262, Nov 2011. (Cited on page 63.)

[77] F. Kahl and P. Strandmark. Generalized roof duality. *Discrete Applied Mathematics*, 160(16):2419–2434, 2012. (Cited on page 63.)

[78] J. H. Kappes, B. Andres, F. A. Hamprecht, C. Schnörr, S. Nowozin, D. Batra, S. Kim, B. X. Kausler, T. Kröger, J. Lellmann, N. Komodakis, B. Savchynskyy, and C. Rother. A comparative study of modern inference techniques for structured discrete energy minimization problems. *International Journal of Computer Vision*, 115(2):155–184, 2015. (Cited on pages 7 and 46.)

[79] V. Kolmogorov and C. Rother. Minimizing nonsubmodular functions with graph cuts – a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7):1274–1279, July 2007. (Cited on pages 46, 92, and 101.)

[80] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, Feb 2004. (Cited on pages 6, 58, and 59.)

[81] A. Krokhin and S. Živný. The Complexity of Valued CSPs. In A. Krokhin and S. Živný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 233–266. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017. (Cited on page 10.)

[82] V. Lempitsky, S. Roth, and C. Rother. Fusionflow: Discrete-continuous optimization for optical flow estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008. (Cited on page 7.)

[83] V. Lempitsky, C. Rother, and A. Blake. Logcut - efficient graph cut optimization for markov random fields. In *IEEE International Conference on Computer Vision*, pages 1–8, Oct 2007. (Cited on page 7.)

[84] S. Leyffer, J. Linderoth, J. Luedtke, A. Miller, and T. Munson. Applications and algorithms for mixed integer nonlinear programming. *Journal of Physics: Conference Series*, 180(1):012–014, 2009. (Cited on page 2.)

[85] D. Li and X. Sun. *Nonlinear Integer Programming*. Springer-Verlag US, 2006. (Cited on page 2.)

[86] F. Liers, E. Marinari, U. Pagacz, F. Ricci-Tersenghi, and V. Schmitz. A non-disordered glassy model with a tunable interaction range. *Journal of Statistical Mechanics: Theory and Experiment*, (5):L05003, 2010. (Cited on pages 99 and 100.)

[87] N. Linial and J. Radhakrishnan. Essential covers of the cube by hyperplanes. *Journal of Combinatorial Theory, Series A*, 109(2):331–338, 2005. (Cited on page 78.)

[88] L. Lovász. Submodular functions and convexity. In A. Bachem, B. Korte, and M. Grötschel, editors, *Mathematical Programming The State of the Art*, pages 235–257. Springer Berlin Heidelberg, 1983. (Cited on page 56.)

[89] J. Luedtke, M. Namazifar, and J. Linderoth. Some results on the strength of relaxations of multilinear functions. *Mathematical Programming*, 136(2):325–351, 2012. (Cited on page 16.)

[90] G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I – convex underestimating problems. *Mathematical Programming*, 10(1):147–175, 1976. (Cited on page 16.)

[91] C. Michini. On the strength of the cycle relaxation for the Boolean quadric polytope. Manuscript, 2016. (Cited on pages 17 and 23.)

[92] C. C. Moallemi and B. V. Roy. Convergence of min-sum message passing for quadratic optimization. *IEEE Transactions on Information Theory*, 55(5):2413–2423, 2009. (Cited on page 57.)

[93] G. L. Nemhauser and L. E. Trotter. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975. (Cited on pages 92 and 102.)

[94] M. Padberg. The boolean quadric polytope: some characteristics, facets and relatives. *Mathematical Programming*, 45(1–3):139–172, 1989. (Cited on pages 4, 20, and 23.)

[95] J.-C. Picard and M. Queyranne. On the integer-valued variables in the linear vertex packing problem. *Mathematical Programming*, 12(1):97–101, 1977. (Cited on pages 92 and 102.)

[96] I. G. Rosenberg. Reduction of bivalent maximization to the quadratic case. *Cahiers du Centre d'Études de Recherche Opérationnelle*, 17:71–74, 1975. (Cited on pages 7, 11, 55, 58, 61, 63, and 82.)

[97] C. Rother, P. Kohli, W. Feng, and J. Jia. Minimizing sparse higher order energy functions of discrete variables. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1382–1389, June 2009. (Cited on pages 58, 59, and 60.)

[98] C. Rother, V. Kolmogorov, V. Lempitsky, and M. Szummer. Optimizing binary MRFs via extended roof duality. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007. (Cited on pages 92 and 101.)

[99] H. S. Ryoo and N. V. Sahinidis. Analysis of bounds for multilinear functions. *Journal of Global Optimization*, 19(4):403–424, 2001. (Cited on page 16.)

[100] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000. (Cited on page 56.)

[101] Z.-J. M. Shen, C. Coullard, and M. S. Daskin. A joint location-inventory model. *Transportation Science*, 37(1):40–55, 2003. (Cited on page 8.)

[102] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(6):1068–1080, 2008. (Cited on page 7.)

[103] S. Živný, D. A. Cohen, and P. G. Jeavons. The expressive power of binary submodular functions. *Discrete Applied Mathematics*, 157(15):3347–3358, 2009. (Cited on pages 56 and 57.)

[104] L. J. Watters. Reduction of integer polynomial programming problems to zero-one linear programming problems. *Operations Research*, 15:1171–1174, 1967. (Cited on pages 10 and 16.)

[105] F. You and I. E. Grossmann. Mixed-integer nonlinear programming models and algorithms for large-scale supply chain design with stochastic inventory management. *Industrial & Engineering Chemistry Research*, 47(20):7802–7817, 2008. (Cited on pages 8 and 9.)

[106] W. I. Zangwill. Media selection by decision programming. *Journal of Advertising Research*, 5:30–36, 1965. (Cited on pages 10 and 16.)

Index

List of Tables

List of Figures