HAL open science

Reformulation techniques in Mathematical Programming

Leo Liberti

To cite this version:

Leo Liberti. Reformulation techniques in Mathematical Programming. Other [cs.OH]. Université Paris Dauphine, 2007. hal-00163563

HAL Id: hal-00163563

https://hal.science/hal-00163563v1

Submitted on 17 Mar 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL open science

Reformulation techniques in Mathematical Programming

Leo Liberti

To cite this version:

Leo Liberti. Reformulation techniques in Mathematical Programming. Other [cs.OH]. Université Paris Dauphine, 2007. hal-00163563

HAL Id: hal-00163563

https://hal.archives-ouvertes.fr/hal-00163563

Submitted on 17 Mar 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ PARIS IX
LAMSADE

Mémoire présenté en vue de l'obtention de l'Habilitation à Diriger des Recherches

Techniques de Reformulation en Programmation Mathématique

Leo Liberti
LIX, École Polytechnique, Palaiseau, 91128 France
liberti@lix.polytechnique.fr

November 19, 2007

Jury:

Philippe MICHELON (Rapporteur), Professeur à l'Université d'Avignon
Nelson MACULAN (Rapporteur), Professeur à l'Universidade Federal do Rio de Janeiro (Brazil)
Hanif SHERALI (Rapporteur), Professeur au Virginia Polytechnic Institute (USA)
Alain BILLIONNET (Examinateur), Professeur au Conservatoir National d'Arts et Métiers
Abdel LISSER (Examinateur), Professeur à l'Université de Paris XI (Sud)
Philippe BAPTISTE (Examinateur), Professeur à l'École Polytechnique
Tapio WESTERLUND (Examinateur), Professeur à l'Université de Abø (Finlande)
Vangelis PASCHOS (Coordinateur), Professeur à l'Université Paris IX (Dauphine)

Résumé

L'objet centrale de cette thèse est l'utilisation de techniques de reformulations en programmation mathématique. Les problèmes d'optimisation et de décision peuvent être décrits précisement par une formulation composée de: paramètres numériques, variables de decision (leur valeurs étant determinées grâce au resultat d'un procès algorithmique), une ou plusieurs fonctions objectives à optimiser, et plusieurs ensembles de contraintes; fonctions objectives et contraintes peuvent être exprimées explicitement comme functions de paramètres et variables, ou implicitement comme conditions sur les variables. Ces éléments, c'est à dire paramètres, variables, fonctions objectives et contraintes, forment un langage appélé programmation mathématique. Pour chaque problème donné d'optimisation ou décision, il y a d'habitude un nombre infini de différents formulations de programmation mathématique possibles. Selon l'algorithme utilisé pour les résoudre, formulations distinctes sont plus ou moins efficaces et/ou efficientes. En outre, plusieurs sous-problèmes ressortants de l'algorithme de solution peuvent eux-mêmes être formulés comme des problèmes de programmation mathématique (appélés problèmes auxiliaires). Cette thèse présente un étude approfondi des transformations symboliques qui mappent des formulations de programmation mathématique a leur formes équivalentes et autres formulations reliés, et de leur impacte sur les algorithmes de solution.

Abstract

This thesis concerns the use of reformulation techniques in mathematical programming. Optimization and decision problems can be cast into a formulation involving sets of known numerical parameters, decision variables whose value is to be determined as a result of an algorithmic process, one of more optional objective functions to be optimized and various sets of constraints, which can be either expressed explicitly as functions of the parameters and variables, or as implicit requirements on the variables. These elements, namely parameters, variables, objective(s) and constraints, form a language called mathematical programming. There are usually many different possible equivalent mathematical programming formulations for the same optimization or decision problem. Different formulations often perform differently according to the type of algorithm employed to solve the problem. Furthermore, related auxiliary problems which may be useful during the course of the algorithmic solution process may arise and be also cast as mathematical programming formulations. This thesis is an in-depth study of the symbolic transformations that map a mathematical programming formulation to its equivalent forms and to other useful related formulations, and of their relations to various solution algorithms.

Contents

1 Introduction **8**

2 General framework **14**

2.1 A data structure for mathematical programming formulations 14

2.1.1 Examples 16

2.1.1.1 A quadratic problem 16

2.1.1.2 Balanced graph bisection 16

2.1.1.3 The Kissing Number Problem 18

2.2 A data structure for mathematical expressions 20

2.2.1 Standard form 21

2.3 Theoretical results 22

2.4 Standard forms in mathematical programming 29

2.4.1 Linear Programming 30

2.4.2 Mixed Integer Linear Programming 30

2.4.3 Nonlinear Programming 30

2.4.4 Mixed Integer Nonlinear Programming 31

2.4.5 Separable problems 31

2.4.6 Factorable problems 32

2.4.7 D.C. problems 32

2.4.8 Linear Complementarity problems 33
2.4.9 Bilevel Programming problems 34
2.4.10 Semidefinite Programming problems 34

3 Reformulations 36

3.1 Elementary reformulations 36
3.1.1 Objective function direction 36
3.1.2 Constraint sense 37
3.1.3 Liftings, restrictions and projections 37
3.1.3.1 Lifting 37
3.1.3.2 Restriction 37
3.1.3.3 Projection 38
3.1.4 Equations to inequalities 38
3.1.5 Inequalities to equations 39
3.1.6 Absolute value terms 40
3.1.7 Product of exponential terms 40
3.1.8 Binary to continuous variables 40
3.1.9 Integer to binary variables 41
3.1.9.1 Assignment variables 41
3.1.9.2 Binary representation 42
3.1.10 Feasibility to optimization problems 42
3.2 Exact linearizations 44
3.2.1 Piecewise linear objective functions 44
3.2.2 Product of binary variables 44
3.2.3 Product of binary and continuous variables 45

3.2.4 Complementarity constraints 45
3.2.5 Minimization of absolute values 46
3.2.6 Linear fractional terms 47
3.3 Advanced reformulations 47
3.3.1 Hansen's Fixing Criterion 48
3.3.2 Compact linearization of binary quadratic problems 48
3.3.3 Reduction Constraints 49
3.4 Advanced examples 50
3.4.1 The Hyperplane Clustering Problem 50
3.4.2 Selection of software components 53

4 Relaxations 58

4.1 Definitions 59
4.2 Elementary relaxations 59
4.2.1 Outer approximation 60
4.2.2 αBB convex relaxation 61
4.2.3 Branch-and-Contract convex relaxation 62
4.2.4 Symbolic reformulation based convex relaxation 62
4.2.5 BARON's convex relaxation 63
4.3 Advanced relaxations 63
4.3.1 Lagrangian relaxation 64
4.3.2 Semidefinite relaxation 65
4.3.3 Reformulation-Linearization Technique 66
4.3.3.1 Basic RLT 66
4.3.3.2 RLT Hierarchy 68

4.3.4 Signomial programming relaxations 69

4.4 Valid cuts 70

4.4.1 Valid cuts for MILPs 71

4.4.2 Valid cuts for NLPs 72

4.4.3 Valid cuts for MINLPs 74

5 Conclusion **75**

Bibliography **76**

List of Figures

2.1 The graphs e_1 (above) and e_2 (below) from Example 2.1.1.1. 17

2.2 The BGBP instance in Example 2.1.1.1. 18

2.3 The graph e_1' from Example 2.1.1.2. $L'_{ij} = L_{ij} + L_{ji}$ for all i, j. 19

2.4 The Kissing Number Problem in 3D. A configuration with 12 balls found by a Variable Neighbourhood Search global optimization solver. 20

2.5 Plots of $\sin(x)$ and $\frac{1}{2}x + \sin(x)$. 25

4.1 Piecewise linear underestimating approximations for concave (left) and convex (right) univariate functions. 70

4.2 A γ-valid cut. 73

Chapter 1

Introduction

Optimization and decision problems are usually defined by their input and a mathematical description of the required output: a mathematical entity with an associated value, or whether a given entity has a specified mathematical property or not. These mathematical entities and properties are expressed in the language of set theory, most often in the ZFC axiomatic system [55] (for clarity, a natural language such as English is usually employed in practice). The scope of set theory language in ZFC is to describe all possible mathematical entities, and its limits are given by Gödel's incompleteness theorem.

Optimization and decision problems are special in the sense that they are closely linked to a particular algorithmic process designed to solve them: more precisely, although the algorithm is not directly mentioned in the problem definition, the main reason why problems are cast is that a solution to the problem is desired. In this respect the usual set theoretical language, with all its expressive powers, falls short of this requirement: specifically, no algorithm is so "generic" that it can solve all problems formulated in terms of set theory. Just to make an example, all Linear Programming (LP) problems can be expressed in a language involving real numbers, variables, a linear form to be minimized, a system of linear equations to be satisfied, and a set of non-negativity constraints on the variables. This particular language used for describing LPs has much stricter limits than the set-theoretical language used in ZFC, of course. On the other hand there exists an algorithm, namely the simplex algorithm [24], which is generic enough to solve any LP problem, and which performs well in practice.

In its most general terms, a decision problem can be expressed as follows: given a set W and a subset $D \subseteq W$, decide whether a given $x \in W$ belongs to D or not. Even supposing that W has finite cardinality (so that the problem is certainly decidable), the only algorithm which is generic enough to solve this problem is complete enumeration, whose low efficiency

renders it practically useless. Informally speaking, when discussing decidable problems and solution algorithms, there is a trade-off between how powerful is the language used to express the problems, and how efficient the associated solution algorithm is.

Mathematical programming can be seen as a language which is powerful enough to express almost all practically interesting optimization and decision problems. Mathematical programming formulations can be categorized according to various properties, and rather efficient solution algorithms exist for many of the categories. The semantic scope of mathematical programming is to define optimization and decision problems: as this scope is narrower than that of the set theoretical language of ZFC, according to the trade-off principle mentioned above, the associated generic algorithms are more efficient.

As in most languages, the same concept can be expressed in many ways. More precisely, there are many equivalent formulations for each given problem (what the term "equivalent" means in this context will be defined later). Furthermore, solution algorithms for mathematical programming formulations often rely on solving a sequence of different problems (often termed *auxiliary problems*) related to the original one: although these are usually not equivalent to the original problem, they may be relaxations, projections, liftings, decompositions (among others). The relations between the original and auxiliary problems are expressed in the literature by means of logical, algebraic and/or transcendental expressions which draw on the same familiar ZFC language. As long as theoretical statements are being made, there is nothing wrong with this, for people are usually able to understand that language. There is, however, a big gap between understanding the logical/algebraic relations among sets of optimization problems, and being able to implement algorithms using these problems in various algorithmic steps. Existing data structures and code libraries usually offer numerical rather than symbolic facilities. Symbolic algorithms and libraries exist, but they are not purpose-built to deal with optimization and decision problems.

We shall illustrate what we mean by way of an example. Consider the Kissing Number Problem (KNP) in D dimensions [54], i.e. the determination of the maximum number of unit D-dimensional spheres that can be arranged around a central unit D-dimensional sphere. As all optimization problems, this can be cast (by using a bisection argument) as a sequence of decision problems on the cardinality of the current spheres configuration. Namely, given the positive integers D (dimension of Euclidean space) and N, is there a configuration of N unit spheres around the central one? For any fixed D, the answer will be affirmative or negative depending on the value of N. The highest N such that the answer is affirmative is the kissing number. Now, the decision problem version of the KNP can be cast as a nonconvex Nonlinear

Programming (NLP) feasibility problem as follows. For all $i \leq N$, let $x_i = (x_{i1}, \ldots, x_{iD}) \in \mathbb{R}^D$ be the center of the i-th sphere. We look for a set of vectors $\{x_i \mid i \leq N\}$ satisfying the following constraints:

$$\begin{aligned} \forall\, i \leq N \quad ||x_i|| &= 2 \\ \forall\, i < j \leq N \quad ||x_i - x_j|| &\geq 2 \\ \forall\, i \leq N \quad -2 \leq x_i &\leq 2. \end{aligned}$$

It turns out that this problem is numerically quite difficult to solve, as it is very unlikely that the local NLP solution algorithm will be able to compute a valid feasible starting solution straight away. Failing to find an initial feasible solution means that the solver will immediately abort without having made any progress. Most researchers with some experience in NLP solvers (such as e.g. SNOPT [36]), however, will immediately reformulate this problem into a more computationally amenable form by squaring the norms to get rid of a potentially problematic square root, and treating the reverse convex constraints $||x_i - x_j|| \geq 2$ as soft constraints by multiplying the right hand sides by a non-negative scaling variable α, which is then maximized:

$$\max \alpha \tag{1.1}$$

$$\forall\, i \leq N \quad ||x_i||^2 = 4 \tag{1.2}$$

$$\forall\, i < j \leq N \quad ||x_i - x_j||^2 \geq 4\alpha. \tag{1.3}$$

$$\forall\, i \leq N \quad -2 \leq x_i \leq 2 \tag{1.4}$$

$$\alpha \geq 0. \tag{1.5}$$

In this form, finding an initial feasible solution is trivial; for example, $x_i = (2, 0, \ldots, 0)$ for all $i \leq N$ will do. Subsequent solver iteration will likely be able to provide a solution. Should the computed value of α be ≥ 1, the solution would be feasible in the hard constraints, too. Currently, we are aware of no optimization language environment that is able to perform the described reformulation automatically. Whilst this is not a huge limitation for NLP experts, people who simply wish to model a problem and get its solution will fail to obtain one, and may even be led into thinking that the formulation itself is infeasible.

Another insightful example of the types of limitations we refer to can be drawn from the KNP. We might wish to impose ordering constraints on some of the spheres to reduce the number of symmetric solutions. Ordering spheres packed on a spherical surface is hard to do in Euclidean coordinates, but it can be done rather easily in spherical coordinates, by simply stating that the value of a spherical coordinate of the i-th sphere must be smaller than the corresponding value in the j-th sphere. We can transform a Euclidean coordinate vector $x = (x_1, \ldots, x_D)$ in

D-spherical coordinates $(\rho, \vartheta_1, \ldots, \vartheta_{D-1})$ such that $\rho = ||x||$ and $\vartheta \in [0, 2\pi]^{D-1}$ by means of the following equations:

$$\rho = ||x|| \tag{1.6}$$

$$\forall k \leq D \quad x_k = \rho \sin \vartheta_{k-1} \prod_{h=k}^{D-1} \cos \vartheta_h \tag{1.7}$$

(this yields another NLP formulation of the KNP). Applying the D-spherical transformation is simply a matter of symbolic term replacement and algebraic simplification, and yet no optimization language environment offers such capabilities. By carrying things further, we might wish to devise an algorithm that dynamically inserts or removes constraints expressed in either Euclidean or spherical coordinates depending on the status of the current solution, and re-solves the (automatically) reformulated problem at each iteration. This may currently be done (up to a point) by optimization language environments such as AMPL [34], provided all constraints are part of a pre-specified family of parametric constraints. Creating new constraints by symbolic term replacement, however, is not a task that can currently be carried out automatically.

The limitations emphasized in the KNP example illustrate a practical need for very sophisticated software including numerical as well as symbolic algorithms, both applied to the unique goal of solving optimization problems cast as mathematical programming formulations. The current state of affairs is that there are many numerical optimization solvers and many Computer Algebra Systems (CAS) — such as Maple or Mathematica — whose efficiency is severely hampered by the full generality of their capabilities. In short, we would ideally need (small) parts of the symbolic kernels driving the existing CASes to be combined with the existing optimization algorithms, plus a number of super-algorithms capable of making automated, dynamic decisions on the type of reformulations that are needed to improve the current search process.

Although the above paradigm might seem far-fetched, it does in fact already exist in the form of the hugely successful CPLEX [47] solver targeted at solving Mixed-Integer Linear Programming (MINLP) problems. The initial formulation provided by the user is automatically simplified and improved with a considerable number of pre-processing steps which attempt to reduce the number of variables and constraints. Thereafter, at each node of the Branch-and-Bound algorithm, the formulation may be tightened as needed by inserting and removing additional valid constraints, in the hope that the current relaxed solution of the (automatically obtained) linear relaxation is improved. Advanced users may of course decide to tune many parameters controlling this process, but practitioners who simply need a practical answer can simply use default parameters and to let CPLEX decide what is best. Naturally, the task carried out by CPLEX is greatly simplified by the assumption that both objective function and constraints are

linear forms, which is obviously not the case in a general nonlinear setting.

In this thesis we attempt to move some steps in the direction of endowing general mathematical programming with the same degree of algorithmic automation enjoyed by linear programming. We propose: (a) a theoretical framework in which mathematical programming reformulations can be formalized in a unified way, and (b) a literature review of the most successful existing reformulation and relaxation techniques in mathematical programming. Since an all-comprehensive literature review in reformulation techniques would extend this thesis to possibly several hundreds (thousands?) pages, only a partial review has been provided. In this sense, this should be seen as "work in progress" towards laying the foundations to a computer software which is capable of reformulating mathematical programming formulations automatically. Note also that for this reason, the usual mathematical notations have been translated to a data structure framework that is designed to facilitate computer implementation. Most importantly, "functions" — which as mathematical entities are interpreted as maps between sets — are represented by expression trees: what is meant by the expression $x + y$, for example, is really a directed binary tree on the vertices $\{+, x, y\}$ with arcs $\{(+, x), (+, y)\}$. For clarity purposes, however, we also provide the usual mathematical languages.

One last (but not least) remark is that reformulations can be seen as a new way of expressing a known problem. Reformulations are syntactical operations that may add or remove variables or constraints, whilst keeping the fundamental structure of the problem optima invariant. When some new variables are added and some of the old ones are removed, we can usually try to re-interpret the reformulated problem and assign a meaning to the new variables, thus gaining new insights to the problem. One example of this is given in Sect. 3.4.2. One other area in mathematical programming that provides a similarly clear relationship between mathematical syntax and semantics is LP duality with the interpretation of reduced costs. This is important insofar as it offers alternative interpretations to known problems, which gains new and useful insights.

The rest of this thesis is organized as follows. In Chapter 2 we propose a general theoretical framework of definitions allowing a unified formalization of mathematical programming reformulations. The definitions allow a consistent treatment of the most common variable and constraint manipulations in mathematical programming formulations. In Chapter 3 we present a systematic study of a set of well known reformulations. Most reformulations are listed as symbolic algorithms acting on the problem structure, although the equivalent transformation in mathematical terms is given for clarity purposes. In Chapter 4 we present a systematic study of a set of well known relaxations. Again, relaxations are listed as symbolic algorithms acting

on the problem structure whenever possible, the equivalent mathematical transformation being given for clarity.

Chapter 2

General framework

In Sect. 2.1 we formally define what a mathematical programming formulation is. In Sect. 2.2 we discuss the expression tree function representation. In Sect. 2.3 we discuss some types of reformulations and establish some links between them. Sect. 2.4 lists the most common standard forms in mathematical programming.

2.1 A data structure for mathematical programming formulations

In this Chapter we give a formal definition of a mathematical programming formulation in such terms that can be easily implemented on a computer. We then give several examples to illustrate the generality of our definition. We refer to a mathematical programming problem in the most general form:

$$\left.\begin{array}{rrcl} \min & f(x) & & \\ & g(x) & \lesseqgtr & b \\ & x & \in & X, \end{array}\right\} \tag{2.1}$$

where f, g are function sequences of various sizes, b is an appropriately-sized real vector, and X is a cartesian product of continuous and discrete intervals.

The precise definition of a mathematical programming formulation lists the different formulation elements: parameters, variables having types and bounds, expressions depending on the parameters and variables, objective functions and constraints depending on the expressions. We let $\mathbb{P}$ be the set of all mathematical programming formulations, and $\mathbb{M}$ be the set of all matrices. This is used in Defn. 2.1.1 to define leaf nodes in mathematical expression trees, so that

the concept of a formulation can also accommodate multilevel and semidefinite programming problems.

2.1.1 Definition
Given an alphabet $\mathcal{L}$ consisting of countably many alphanumeric names $N_{\mathcal{L}}$ and operator symbols $O_{\mathcal{L}}$, a mathematical programming formulation P is a 7-tuple $(\mathcal{P}, \mathcal{V}, \mathcal{E}, \mathcal{O}, \mathcal{C}, \mathcal{B}, \mathcal{T})$, where:

- *$\mathcal{P} \subseteq N_{\mathcal{L}}$ is the sequence of parameter symbols: each element $p \in \mathcal{P}$ is a parameter name;*
- *$\mathcal{V} \subseteq N_{\mathcal{L}}$ is the sequence of variable symbols: each element $v \in \mathcal{V}$ is a variable name;*
- *$\mathcal{E}$ is the set of expressions: each element $e \in \mathcal{E}$ is a Directed Acyclic Graph (DAG) $e = (V_e, A_e)$ such that:*
  - *(a) $V_e \subseteq \mathcal{L}$ is a finite set*
  - *(b) there is a unique vertex $r_e \in V_e$ such that $\delta^-(r_e) = \emptyset$ (such a vertex is called the root vertex)*
  - *(c) vertices $v \in V_e$ such that $\delta^+(v) = \emptyset$ are called leaf vertices and their set is denoted by $\lambda(e)$; all leaf vertices v are such that $v \in \mathcal{P} \cup \mathcal{V} \cup \mathbb{R} \cup \mathbb{P} \cup \mathbb{M}$*
  - *(d) for all $v \in V_e$ such that $\delta^+(v) \neq \emptyset$, $v \in O_{\mathcal{L}}$*
  - *(e) two weightings $\chi, \xi : V_e \to \mathbb{R}$ are defined on V_e: $\chi(v)$ is the node coefficient and $\xi(v)$ is the node exponent of the node v; for any vertex $v \in V_e$, we let $\tau(v)$ be the symbolic term of v: namely, $v = \chi(v)\tau(v)^{\xi(v)}$.*

  elements of $\mathcal{E}$ are sometimes called expression trees; nodes $v \in O_{\mathcal{L}}$ represent an operation on the nodes in $\delta^+(v)$, denoted by $v(\delta^+(v))$, with output in $\mathbb{R}$;
- *$\mathcal{O} \subseteq \{-1, 1\} \times \mathcal{E}$ is the sequence of objective functions; each objective function $o \in \mathcal{O}$ has the form (d_o, f_o) where $d_o \in \{-1, 1\}$ is the optimization direction (-1 stands for minimization, $+1$ for maximization) and $f_o \in E$;*
- *$\mathcal{C} \subseteq \mathcal{E} \times \mathcal{S} \times \mathbb{R}$ (where $\mathcal{S} = \{-1, 0, 1\}$) is the sequence of constraints c of the form (e_c, s_c, b_c) with $e_c \in E, s_c \in S, b_c \in \mathbb{R}$:*

$$c \equiv \begin{cases} e_c \leq b_c & \text{if} \quad s_c = -1 \\ e_c = b_c & \text{if} \quad s_c = 0 \\ e_c \geq b_c & \text{if} \quad s_c = 1; \end{cases}$$

- *$\mathcal{B} \subseteq \mathbb{R}^{|\mathcal{V}|} \times \mathbb{R}^{|\mathcal{V}|}$ is the sequence of variable bounds: for all $v \in \mathcal{V}$ let $\mathcal{B}(v) = [L_v, U_v]$ with $L_v, U_v \in \mathbb{R}$;*
- *$\mathcal{T} \subseteq \{0, 1, 2\}^{|\mathcal{V}|}$ is the sequence of variable types: for all $v \in \mathcal{V}$, v is called a continuous variable if $\mathcal{T}(v) = 0$, an integer variable if $\mathcal{T}(v) = 1$ and a binary variable if $\mathcal{T}(v) = 2$.*

We remark that for a sequence of variables $z \subseteq \mathcal{V}$ we write $\mathcal{T}(z)$ and respectively $\mathcal{B}(z)$ to mean the corresponding sequences of types and respectively bound intervals of the variables in z. Given a formulation $P = (\mathcal{P}, \mathcal{V}, \mathcal{E}, \mathcal{O}, \mathcal{C}, \mathcal{B}, \mathcal{T})$, the *cardinality* of P is $|P| = |\mathcal{V}|$. We sometimes refer to a formulation by calling it an *optimization problem* or simply a *problem*.

2.1.2 Definition
Any formulation Q that can be obtained by P by a finite sequence of symbolic operations carried out on the data structure is called a problem transformation.

2.1.1 Examples

In this section we provide some explicitly worked out examples that illustrate Defn. 2.1.1.

2.1.1.1 A quadratic problem

Consider the problem of minimizing the quadratic form $3x_1^2 + 2x_2^2 + 2x_3^2 + 3x_4^2 + 2x_5^2 + 2x_6^2 - 2x_1x_2 - 2x_1x_3 - 2x_1x_4 - 2x_2x_3 - 2x_4x_5 - 2x_4x_6 - 2x_5x_6$ subject to $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 0$ and $x_i \in \{-1, 1\}$ for all $i \leq 6$. For this problem,

- $\mathcal{P} = \emptyset$;
- $\mathcal{V} = (x_1, x_2, x_3, x_4, x_5, x_6)$;
- $\mathcal{E} = (e_1, e_2)$ where e_1, e_2 are the graphs shown in Fig. 2.1;
- $\mathcal{O} = (-1, e_1)$;
- $\mathcal{C} = ((e_2, 0, 0))$;
- $\mathcal{B} = ([-1, 1], [-1, 1], [-1, 1], [-1, 1], [-1, 1], [-1, 1])$;
- $\mathcal{T} = (2, 2, 2, 2, 2, 2)$.

2.1.1.2 Balanced graph bisection

Example 2.1.1.1 is a (scaled) mathematical programming formulation of a balanced graph bisection problem instance. This problem is defined as follows.

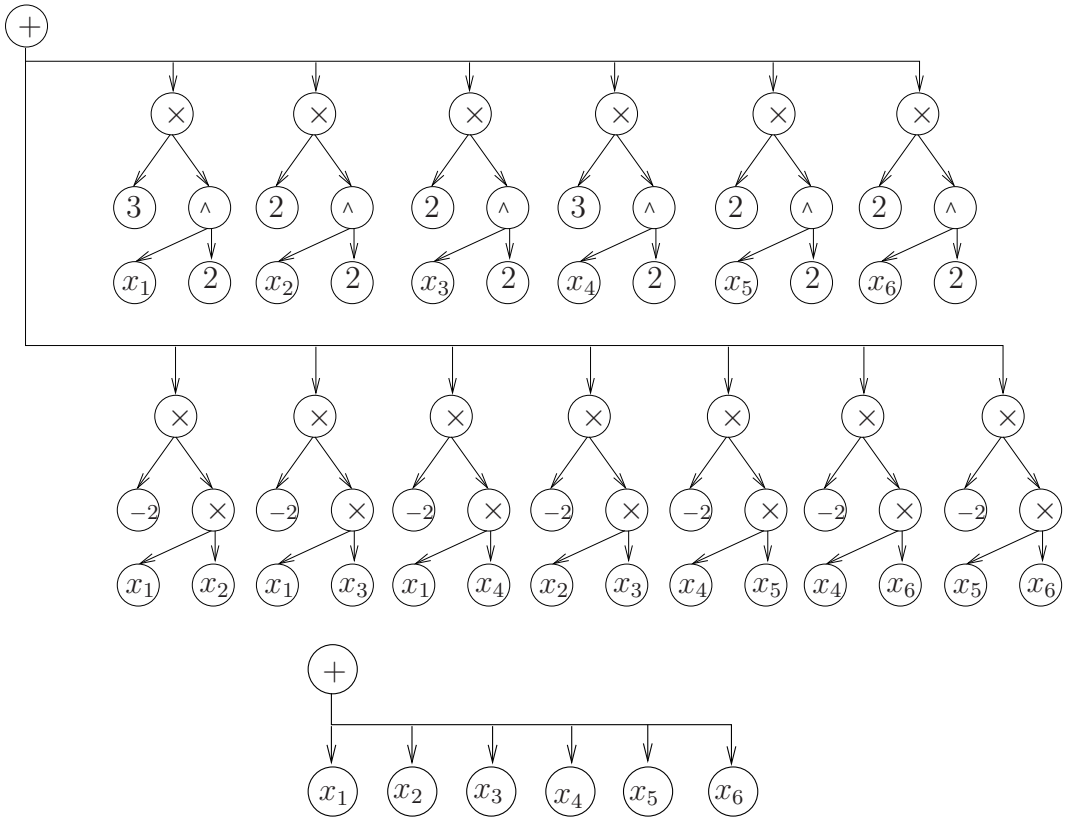

Figure 2.1: The graphs e_1 (above) and e_2 (below) from Example 2.1.1.1.

BALANCED GRAPH BISECTION PROBLEM (BGBP). Given an undirected graph $G = (V, E)$ without loops or parallel edges such that $|V|$ is even, find a subset $U \subset V$ such that $|U| = \frac{|V|}{2}$ and the set of edges $C = \{\{u, v\} \in E \mid u \in U, v \notin U\}$ is as small as possible.

The problem instance considered in Example 2.1.1.1 is shown in Fig. 2.2. To all vertices $i \in V$ we associate variables $x_i = \begin{cases} 1 & i \in U \\ 0 & i \notin U \end{cases}$. The number of edges in C is counted by $\frac{1}{4} \sum_{\{i,j\} \in E} (x_i - x_j)^2$. The fact that $|U| = \frac{|V|}{2}$ is expressed by requiring an equal number of variables at 1 and -1, i.e. $\sum_{i=1}^{6} x_i = 0$.

We can also express the problem in Example 2.1.1.1 as a particular case of the more general optimization problem:

$$\left.\begin{array}{lrcl} \min_x & x^\top L x & & \\ \text{s.t.} & x\mathbf{1} & = & 0 \\ & x & \in & \{-1, 1\}^6, \end{array}\right\}$$

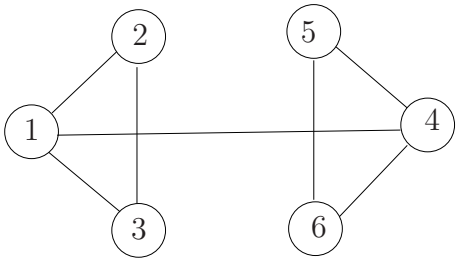

Figure 2.2: The BGBP instance in Example 2.1.1.1.

where

$$L = \begin{pmatrix} 3 & -1 & -1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 & 0 \\ -1 & 0 & 0 & 3 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{pmatrix}$$

and $\mathbf{1} = (1, 1, 1, 1, 1, 1)^\top$. We represent this class of problems by the following mathematical programming formulation:

- $\mathcal{P} = (L_{ij} \mid 1 \leq i, j \leq 6)$;
- $\mathcal{V} = (x_1, x_2, x_3, x_4, x_5, x_6)$;
- $\mathcal{E} = (e_1', e_2)$ where e_1' is shown in Fig. 2.3 and e_2 is shown in Fig. 2.1 (below);
- $\mathcal{O} = (-1, e_1')$;
- $\mathcal{C} = ((e_2, 0, 0))$;
- $\mathcal{B} = ([-1, 1], [-1, 1], [-1, 1], [-1, 1], [-1, 1], [-1, 1])$;
- $\mathcal{T} = (2, 2, 2, 2, 2, 2)$.

2.1.1.3 The Kissing Number Problem

The kissing number problem formulation (1.1)-(1.5) is as follows:

- $\mathcal{P} = (N, D)$;

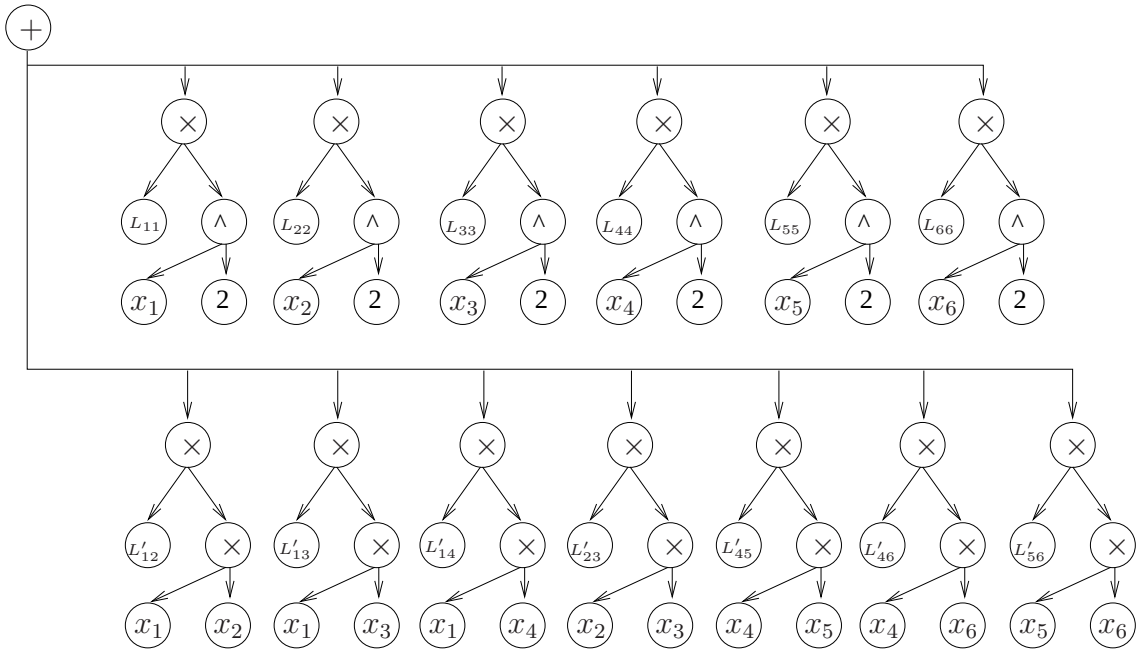

Figure 2.3: The graph e'_1 from Example 2.1.1.2. $L'_{ij} = L_{ij} + L_{ji}$ for all i, j.

- $\mathcal{V} = (x_{ik} \mid 1 \leq i \leq N \wedge 1 \leq k \leq D)$;
- $\mathcal{E} = (f, h_j, g_{ij} \mid 1 \leq i < j \leq N)$, where f is the expression tree for α, h_j is the expression tree for $||x_j||^2$ for all $j \leq N$, and g_{ij} is the expression tree for $||x_i - x_j||^2 - 4\alpha$ for all $i < j \leq N$;
- $\mathcal{O} = (1, f)$;
- $\mathcal{C} = ((h_i, 0, 4) \mid i \leq N) \cup ((g_{ij}, 1, 0) \mid i < j \leq N)$;
- $\mathcal{B} = [-2, 2]^{ND}$;
- $\mathcal{T} = \{0\}^{ND}$.

As mentioned in Chapter 1, the kissing number problem is defined as follows.

> KISSING NUMBER PROBLEM (KNP). Find the largest number N of non-overlapping unit spheres in $\mathbb{R}^D$ that are adjacent to a given unit sphere.

The formulation of Example 2.1.1.3 refers to the decision version of the problem: given integers N and D, is there an arrangement of N non-overlapping unit spheres in $\mathbb{R}^D$ adjacent to a given unit sphere? An example for $N = 12$ and $D = 3$ is shown in Fig. 2.4.

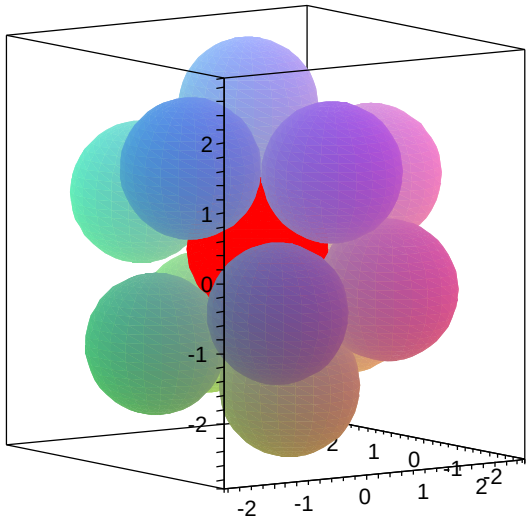

Figure 2.4: The Kissing Number Problem in 3D. A configuration with 12 balls found by a Variable Neighbourhood Search global optimization solver.

2.2 A data structure for mathematical expressions

Given an expression tree DAG $e = (V, A)$ with root node $r(e)$ and whose leaf nodes are elements of $\mathbb{R}$ or of $\mathbb{M}$ (the set of all matrices), the *evaluation* of e is the (numerical) output of the operation represented by the operator in node r applied to all the subnodes of r (i.e. the nodes adjacent to r); in symbols, we denote the output of this operation by $r(\delta^+(r))$. Naturally, the arguments of the operator must be consistent with the operator meaning. We remark that for leaf nodes belonging to $\mathbb{P}$ (the set of all formulations), the evaluation is not defined; the problem in the leaf node must first be solved and a relevant optimal value (e.g. an optimal variable value, as is the case with multilevel programming problems) must replace the leaf node.

For any $e \in E$, the *evaluation tree* of $\bar{e}$ is a DAG $\bar{e} = (\bar{V}, A)$ where $\bar{V} = \{v \in V \mid |\delta^+(v)| > 0 \vee v \in \mathbb{R}\} \cup \{x(v) \mid |\delta^+(v)| = 0 \wedge v \in \mathcal{V}\}$ (in short, the same as V with every variable leaf node replaced by the corresponding value $x(v)$). Evaluation trees are evaluated by Alg. 1. We can now naturally extend the definition of evaluation of e at a point x to expression trees whose leaf nodes are either in $\mathcal{V}$ or $\mathbb{R}$.

2.2.1 Definition
Given an expression $e \in E$ with root node r and a point x, the evaluation $e(x)$ of e at x is the evaluation $r(\delta^+(r))$ of the evaluation tree $\bar{e}$.

Algorithm 1 The evaluation algorithm for expression trees.

```
double Eval(node v) {
  double ρ;
  if (v ∈ O_L) {
    // v is an operator
    array α;
    ∀ u ∈ δ⁺(v) {
      α(u) =Eval(u);
    }
    ρ = χ(v)v(α)^ξ(v);
  } else {
    // v is a constant value
    ρ = χ(v)v^ξ(v);
  }
  return ρ;
}
```

We consider a sufficiently rich operator set O_L including at least $+, \times$, power, exponential, logarithm, trigonometric and inverse trigonometric functions (for real arguments) and inner product (for matrix arguments). Note that since any term t is weighted by a multiplier coefficient $\chi(t)$ there is no need to employ a $-$ operator, for it suffices to multiply $\chi(t)$ by -1 in the appropriate term(s) t; a division u/v is expressed by multiplying u by v raised to the power -1. Depending on the problem form, it may sometimes be useful to enrich O_L with other (more complex) terms. In general, we view an operator in O_L as an atomic operation on a set of variables with cardinality at least 1.

2.2.1 Standard form

Since in general there is more than one way to write a mathematical expression, it is useful to adopt a standard form; whilst this does not resolve all ambiguities, it nonetheless facilitates the task of writing symbolic computation algorithms acting on the expression trees. For any expression node t in an expression tree $e = (V, A)$:

- if t is a sum:

1. $|\delta^+(t)| \geq 2$
2. no subnode of t may be a sum (sum associativity);
3. no pair of subnodes $u, v \in \delta^+(t)$ must be such that $\tau(u) = \tau(v)$ (i.e. like terms must be collected); as a consequence, each sum only has one monomial term for each monomial type
4. a natural (partial) order is defined on $\delta^+(t)$: for $u, v \in \delta^+(t)$, if u, v are monomials, u, v are ordered by degree and lexicographically

- if t is a product:
  1. $|\delta^+(t)| \geq 2$
  2. no subnode of t may be a product (product associativity);
  3. no pair of subnodes $u, v \in \delta^+(t)$ must be such that $\tau(u) = \tau(v)$ (i.e. like terms must be collected and expressed as a power)
- if t is a power:
  1. $|\delta^+(t)| = 2$
  2. the exponent may not be a constant (constant exponents are expressed by setting the exponent coefficient $\xi(t)$ of a term t)
  3. the natural order on $\delta^+(t)$ lists the base first and the exponent later.

The usual mathematical nomenclature (linear forms, polynomials, and so on) applies to expression trees.

2.3 Theoretical results

Consider a mathematical programming formulation $P = (\mathcal{P}, \mathcal{V}, \mathcal{E}, \mathcal{O}, \mathcal{C}, \mathcal{B}, \mathcal{T})$ and a function $x : \mathcal{V} \to \mathbb{R}^{|\mathcal{V}|}$ (called *point*) which assigns values to the variables.

2.3.1 Definition
A point x is type feasible if:

$$x(v) \in \begin{cases} \mathbb{R} & \textit{if} \quad \mathcal{T}(v) = 0 \\ \mathbb{Z} & \textit{if} \quad \mathcal{T}(v) = 1 \\ \{L_v, U_v\} & \textit{if} \quad \mathcal{T}(v) = 2 \end{cases}$$

for all $v \in \mathcal{V}$; x is bound feasible if $x(v) \in \mathcal{B}(v)$ for all $v \in \mathcal{V}$; x is constraint feasible if for all $c \in \mathcal{C}$ we have: $e_c(x) \leq b_c$ if $s_c = -1$, $e_c(x) = b_c$ if $s_c = 0$, and $e_c(x) \geq b_c$ if $s_c = 1$. A point x is feasible in P if it is type, bound and constraint feasible.

A point x feasible in P is also called a *feasible solution* of P. A point which is not feasible is called *infeasible*. Denote by $\mathcal{F}(P)$ the feasible points of P.

2.3.2 Definition
A feasible point x is a local optimum of P with respect to the objective $o \in \mathcal{O}$ if there is a non-empty neighbourhood N of x such that for all feasible points $y \neq x$ in N we have $d_o f_o(x) \geq d_o f_o(y)$. A local optimum is strict if $d_o f_o(x) > d_o f_o(y)$. A feasible point x is a global optimum of P with respect to the objective $o \in \mathcal{O}$ if $d_o f_o(x) \geq d_o f_o(y)$ for all feasible points $y \neq x$. A global optimum is strict if $d_o f_o(x) > d_o f_o(y)$.

Denote the set of local optima of P by $\mathcal{L}(P)$ and the set of global optima of P by $\mathcal{G}(P)$. If $\mathcal{O}(P) = \emptyset$, we define $\mathcal{L}(P) = \mathcal{G}(P) = \mathcal{F}(P)$.

2.3.3 Example
The point $x = (-1, -1, -1, 1, 1, 1)$ is a strict global minimum of the problem in Example 2.1.1.1 and $|\mathcal{G}| = 1$ as $U = \{1, 2, 3\}$ and $V \setminus U = \{4, 5, 6\}$ is the only balanced partition of V leading to a cutset size of 1.

It appears from the existing literature that the term "reformulation" is almost never formally defined in the context of mathematical programming. The general consensus seems to be that given a formulation of an optimization problem, a reformulation is a different formulation having the same set of optima. Various authors make use of this definition without actually making it explicit, among which [98, 103, 116, 72, 30, 38, 18, 87, 48, 35]. Many of the proposed reformulations, however, stretch this implicit definition somewhat. Liftings, for example (which consist in adding variables to the problem formulation), usually yield reformulations where an optimum in the original problem is mapped to a set of optima in the reformulated problem (see Sect. 3.1.3.1). Furthermore, it is sometimes noted how a reformulation in this sense is overkill because the reformulation only needs to hold at global optimality [1]. Furthermore, reformulations sometimes really refer to a change of variables, as is the case in [82]. Throughout the rest of this section we give various definitions for the concept of reformulation, and we explore the relations between them. We consider two problems

$$\begin{aligned} P &= (\mathcal{P}(P), \mathcal{V}(P), \mathcal{E}(P), \mathcal{O}(P), \mathcal{C}(P), \mathcal{B}(P), \mathcal{T}(P)) \\ Q &= (\mathcal{P}(Q), \mathcal{V}(Q), \mathcal{E}(Q), \mathcal{O}(Q), \mathcal{C}(Q), \mathcal{B}(Q), \mathcal{T}(Q)). \end{aligned}$$

Reformulations have been formally defined in the context of *optimization problems* (which are defined as decision problems with an added objective function). As was noted in Ch. 1, we

see mathematical programming as a language used to describe and eventually solve optimization problems, so the difference is slim. The following definition is found in [12].

2.3.4 Definition
Let P_A and P_B be two optimization problems. A reformulation $B(\cdot)$ of P_A as P_B is a mapping from P_A to P_B such that, given any instance A of P_A and an optimal solution of $B(A)$, an optimal solution of A can be obtained within a polynomial amount of time.

This definition is directly inspired to complexity theory and **NP**-completeness proofs. In the more practical and implementation oriented context of this thesis, Defn. 2.3.4 has one weak point, namely that of polynomial time. In practice, depending on the problem and on the instance, a polynomial time reformulation may just be too slow; on the other hand, Defn. 2.3.4 may bar a non-polynomial time reformulation which might be actually carried out within a practically reasonable amount of time. Furthermore, a reformulation in the sense of Defn. 2.3.4 does not necessarily preserve local optimality, which might in some cases be a desirable reformulation feature. It should be mentioned that Defn. 2.3.4 was proposed in a paper that was more theoretical in nature, using an algorithmic equivalence between problems in order to attempt to rank equivalent **NP**-hard problems by their solution difficulty.

The following definition was proposed by H. Sherali [91].

2.3.5 Definition
A problem Q is a reformulation of P if:

- *there is a bijection $\sigma : \mathcal{F}(P) \to \mathcal{F}(Q)$;*
- $|\mathcal{O}(P)| = |\mathcal{O}(Q)|$;
- *for all $p = (e_p, d_p) \in \mathcal{O}(P)$, there is a $q = (e_q, d_q) \in \mathcal{O}(Q)$ such that $e_q = f(e_p)$ where f is a monotonic univariate function.*

Defn. 2.3.5 imposes a very strict condition, namely the bijection between feasible regions of the original and reformulated problems. Although this is too strict for many useful transformations to be classified as reformulations, under some regularity conditions on σ it presents some added benefits, such as e.g. allowing easy correspondences between partitioned subspaces of the feasible regions and mapping sensitivity analysis results from reformulated to original problem.

In the rest of the section we discuss alternative definitions which only make use of the concept of optimum. These encompass a larger range of transformations as they do not require a bijection between the feasible regions, the way Defn. 2.3.5 does.

2.3.6 Definition
Q is a local reformulation of P if there is a function $\varphi : \mathcal{F}(Q) \to \mathcal{F}(P)$ such that (a) $\varphi(y) \in \mathcal{L}(P)$ for all $y \in \mathcal{L}(Q)$, (b) φ restricted to $\mathcal{L}(Q)$ is surjective. This relation is denoted by $P \prec_\varphi Q$.

Informally, a local reformulation transforms all (local) optima of the original problem into optima of the reformulated problem, although more than one reformulated optimum may correspond to the same original optimum. A local reformulation does not lose any local optimality information and makes it possible to map reformulated optima back to the original ones; on the other hand, a local reformulation does not keep track of globality: some global optima in the original problem may be mapped to local optima in the reformulated problem, or vice-versa (see Example 2.3.7).

2.3.7 Example
Consider the problem $P \equiv \min_{x\in[-2\pi,2\pi]} \sin(x)$ and $Q \equiv \min_{x\in[-2\pi,2\pi]} \frac{1}{2}x + \sin(x)$. It is easy to verify that there is a bijection between the local optima of Q and those of P (see Fig. 2.5). However, although Q has a unique global optimum, every local optimum in P is global (hence no mapping cannot be surjective).

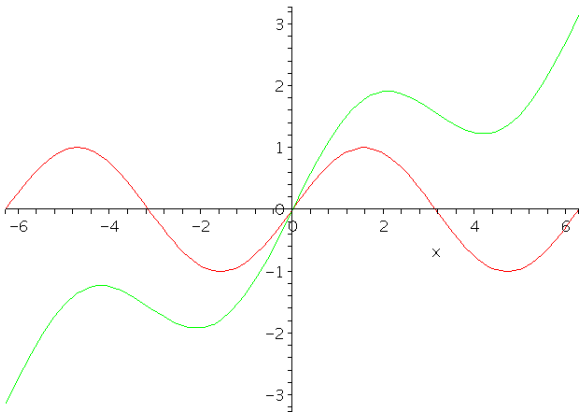

Figure 2.5: Plots of $\sin(x)$ and $\frac{1}{2}x + \sin(x)$.

2.3.8 Definition
Q is a global reformulation of P if there is a function $\varphi : \mathcal{F}(Q) \to \mathcal{F}(P)$ such that (a) $\varphi(y) \in \mathcal{G}(P)$ for all $y \in \mathcal{G}(Q)$, (b) φ restricted to $\mathcal{G}(Q)$ is surjective. This relation is denoted by $P \lhd_\varphi Q$.

Informally, a global reformulation transforms all global optima of the original problem into global optima of the reformulated problem, although more than one reformulated global optimum may correspond to the same original global optimum. Global reformulations are desirable,

in the sense that they make it possible to retain the useful information about the global optima whilst ignoring local optimality. At best, given a difficult problem P with many local minima, we would like to find a global reformulation Q where $\mathcal{L}(Q) = \mathcal{G}(Q)$.

2.3.9 Example
Consider a problem P with $\mathcal{O}(P) = \{f\}$. Let Q be a problem such that $\mathcal{O}(Q) = \{\breve{f}\}$ and $\mathcal{F}(Q) = \text{conv}(\mathcal{F}(P))$, where conv$(\mathcal{F}(P))$ is the convex hull of the points of $\mathcal{F}(P)$ and $\breve{f}$ is the convex envelope of f over the convex hull of $\mathcal{F}(P)$ (in other words, f is the greatest convex function underestimating f on $\mathcal{F}(P)$). Since the set of global optima of P is contained in the set of global optima of Q [44], the convex envelope is a global reformulation.

Unfortunately, finding convex envelopes in explicit form is not easy. A considerable amount of work exists in this area: e.g. for bilinear terms [80, 6], trilinear terms [81], fractional terms [108], monomials of odd degree [71, 59] the envelope is known in explicit form (this list is not exhaustive). See [106] for recent theoretical results and a rich bibliography.

2.3.10 Definition
Q is an opt-reformulation of P (denoted by $P < Q$) if there is a function $\varphi : \mathcal{F}(Q) \to \mathcal{F}(P)$ such that $P \prec_\varphi Q$ and $P \lhd_\varphi Q$.

This type of reformulation preserves both local and global optimality information, which makes it very attractive. Even so, Defn. 2.3.10 fails to encompass those problem transformations that eliminate some global optima whilst ensuring that at least one global optimum is left. Such transformations are specially useful in Integer Programming problems having a lot of symmetric optimal solutions: restricting the set of global optima in such cases may be beneficial. One such example is the pruning of Branch-and-Bound regions based on the symmetry group of the problem presented in [78]: the set of cuts generated by the procedure fails in general to be a global reformulation in the sense of Defn. 2.3.8 because the number of global optima in the reformulated problem is smaller than that of the original problem.

2.3.11 Lemma
The relations $\prec, \lhd, <$ are reflexive and transitive, but in general not symmetric.

Proof. For reflexivity, simply take φ as the identity. For transitivity, let $P \prec Q \prec R$ with functions $\varphi : \mathcal{F}(Q) \to \mathcal{F}(P)$ and $\psi : \mathcal{F}(R) \to \mathcal{F}(Q)$. Then $\vartheta = \varphi \circ \psi$ has the desired properties. In order to show that $\prec$ is not symmetric, consider a problem P with variables x and a unique minimum x^* and a problem Q which is exactly like P but has one added variable $w \in [0, 1]$. It is easy to show that $P \prec Q$ (take φ as the projection of (x, w) on x). However,

since for all $w \in [0,1]$ (x^*, w) is an optimum of Q, there is no function of a singleton to a continuously infinite set that is surjective. □

Given a pair of problems P, Q where $\prec, \lhd, <$ are symmetric on the pair, we call Q a *symmetric reformulation* of P. We remark also that by Lemma (2.3.11) we can compose elementary reformulations together to create chained reformulations (see Sect. 3.4 for examples).

Continuous reformulations are of an altogether different type. These are based on a continuous map τ (invertible on the variable domains) acting on the continuous relaxation of the feasible space of the two problems.

2.3.12 Definition
For P, Q having the following properties:

(a) $|P| = n, |Q| = m$,

(b) $\mathcal{V}(P) = x, \mathcal{V}(Q) = y$,

(c) $\mathcal{O}(P) = (f, d), \mathcal{O}(Q) = (f', d')$ *where f is a sequence of expressions in $\mathcal{E}(P)$ and d is a vector with elements in $\{-1, 1\}$ (and similarly for f', d'),*

(d) $\mathcal{C}(P) = (g, -\mathbf{1}, \mathbf{0}), \mathcal{C}(Q) = (g', -\mathbf{1}, \mathbf{0})$ *where g is a sequence of expressions in $\mathcal{E}(P)$, $\mathbf{0}$ (resp. $\mathbf{1}$) is a vector of 0s (resp. 1s) of appropriate size (and similarly for g'),*

(e) f, f' *are continuous functions and g, g' are sequences of continuous functions,*

Q is a continuous reformulation of P with respect to a reformulating bijection τ (denoted by $P \approx_\tau Q$) if $\tau : \mathbb{R}^n \to \mathbb{R}^m$ is a continuous map, invertible on the variable domains $\prod_{x_i \in x} \mathcal{B}(x_i)$, such that $f' \circ \tau = f$, $g' \circ \tau = g$ and $\mathcal{B}(y) = \tau(\mathcal{B}(x))$, and such that τ^{-1} is also continuous.

It is easy to show that τ is an invertible map $\mathcal{F}(P) \to \mathcal{F}(Q)$. Change of variables usually provide a continuous reformulations. For example, (1.6)-(1.7) yield a continuous invertible map τ that provides a continuous reformulation of the KNP in polar coordinates. Continuous reformulations are in some sense similar to reformulations in the sense of Defn. 2.3.5: they are stronger, in that they require the invertible mapping to be continuous; and they are weaker, in that they impose no additional condition on the way the objective functions are reformulated.

2.3.13 Lemma
$\approx_\tau$ is an equivalence relation.

Proof. Taking τ as the identity shows reflexivity, and the fact that τ is a bijection shows symmetry. Transitivity follows easily by composition of reformulating bijections. □

In the next results, we underline some relations between different reformulation types.

2.3.14 Lemma
If $P \approx_\tau Q$ with $|P| = n, |Q| = m$, for all $x \in \mathbb{R}^n$ which is bound and constraint feasible in P, $\tau(x)$ is bound and constraint feasible in Q.

Proof. Suppose without loss of generality that the constraints and bounds for P can be expressed as $g(x) \leq 0$ for $x \in \mathbb{R}^n$ and those for Q can be expressed as $g'(y) \leq 0$ for $y \in \mathbb{R}^m$. Then $g'(y) = g'(\tau(x)) = (g' \circ \tau)(x) = g(x) \leq 0$. □

2.3.15 Proposition
If $P \approx_\tau Q$ with $\mathcal{V}(P) = x, \mathcal{V}(Q) = y$, $|P| = n, |Q| = m$, $|\mathcal{O}(P)| = |\mathcal{O}(Q)| = 1$ such that (f, d) is the objective function of P and (f', d') is that of Q, $d = d'$, $\mathcal{T}(x) = 0, \mathcal{T}(y) = 0$, then τ is a bijection $\mathcal{L}(P) \to \mathcal{L}(Q)$ and $\mathcal{G}(P) \to \mathcal{G}(Q)$.

Proof. Let $x \in \mathcal{L}(P)$. Then there is a neigbourhood $N(P)$ of x such that for all $x' \in N(P)$ with $x' \in \mathcal{F}(P)$ we have $df(x') \leq df(x)$. Since τ is a continuous invertible map, $N(Q) = \tau(N(P))$ is a neighbourhood of $y = \tau(x)$ (so $\tau^{-1}(N(Q)) = N(P)$). For all $y' \in \mathcal{F}(Q)$, by Lemma 2.3.14 and because all problem variable are continuous, $\tau^{-1}(y') \in \mathcal{F}(P)$. Hence for all $y' \in N(Q) \cap \mathcal{F}(Q)$, $x' = \tau^{-1}(y') \in N(P) \cap \mathcal{F}(P)$. Thus, $d'f'(y') = df'(\tau(x')) = d(f' \circ \tau)(x') = df(x') \leq df(x) = d(f \circ \tau^{-1})(y) = d'f'(y)$. Thus for all $x \in \mathcal{L}(P)$, $\tau(x) \in \mathcal{L}(Q)$. The same argument applied to τ^{-1} shows that for all $y \in \mathcal{L}(Q)$, $\tau^{-1}(y) \in \mathcal{L}(P)$; so τ restricted to $\mathcal{L}(P)$ is a bijection. As concerns global optima, let $x^* \in \mathcal{G}(P)$ and $y^* = \tau(x^*)$; then for all $y \in \mathcal{F}(Q)$ with $y = \tau(x)$, we have $d'f'(y) = d'f'(\tau(x)) = d(f \circ \tau)(x) = df(x) \leq df(x^*) = d'(f \circ \tau^{-1})(y^*) = d'f'(y^*)$, which shows that $y^* \in \mathcal{G}(Q)$. The same argument applied to τ^{-1} shows that τ restricted to $\mathcal{G}(P)$ is a bijection. □

2.3.16 Theorem
If $P \approx_\tau Q$ with $\mathcal{V}(P) = x, \mathcal{V}(Q) = y$, $|P| = n, |Q| = m$, $|\mathcal{O}(P)| = |\mathcal{O}(Q)| = 1$ such that (f, d) is the objective function of P and (f', d') is that of Q, $d = d'$, $\mathcal{T}(x) = 0, \mathcal{T}(y) = 0$, then $P < Q$ and $Q < P$.

Proof. The fact that $P < Q$ follows from Prop. 2.3.15. The reverse follows by considering τ^{-1}. □

2.3.17 Proposition
Let P, Q be two problems with $\mathcal{V}(P) = x, \mathcal{V}(Q) = y$, $|P| = n, |Q| = m$, $|\mathcal{O}(P)| = |\mathcal{O}(Q)| = 1$ such that (f, d) is the objective function of P and (f', d') is that of Q, $d = d'$, $\mathcal{L}(P)$ and $\mathcal{L}(Q)$ both consist of isolated points in the respective Euclidean topologies, and assume $P \prec Q$ and $Q \prec P$. Then there is a continuous invertible map $\tau : \mathcal{F}(P) \to \mathcal{F}(Q)$.

Proof. Since $P \prec Q$ there is a surjective function $\varphi : \mathcal{L}(Q) \to \mathcal{L}(P)$, which implies $|\mathcal{L}(Q)| \geq |\mathcal{L}(P)|$. Likewise, since $Q \prec P$ there is a surjective function $\psi : \mathcal{L}(P) \to \mathcal{L}(Q)$, which implies $|\mathcal{L}(P)| \geq |\mathcal{L}(Q)|$. This yields $|\mathcal{L}(P)| = |\mathcal{L}(Q)|$, which means that there is a bijection $\tau : \mathcal{L}(P) \to \mathcal{L}(Q)$. Because $\mathcal{L}(P) \subseteq \mathbb{R}^n$ and $\mathcal{L}(Q) \subseteq \mathbb{R}^m$ only contain isolated points, there is a way to extend τ to $\mathbb{R}^n$ so that it is continuous and invertible on the x variable domains, and so that τ^{-1} enjoys the same properties (define τ in the natural way on the segments between pairs of points in $\mathcal{L}(P)$ and "fill in the gaps"). □

In summary, continuous reformulations of continuous problems are symmetric reformulations, whereas symmetric reformulations may not necessarily be continuous reformulations. Furthermore, continuous reformulations applied to discrete problems may fail to be opt-reformulations. This happens because integrality constraints do not transform with the map τ along with the rest of the problem constraints.

2.3.18 Definition
Any problem Q that is related to a given problem P by a formula $f(Q, P) = 0$ where f is a computable function is called an auxiliary problem with respect to P.

Deriving the formulation of an auxiliary problem may be a hard task, depending on f. The most useful auxiliary problems are those whose formulation can be derived algorithmically in time polynomial in $|P|$.

2.4 Standard forms in mathematical programming

Solution algorithms for mathematical programming problems read a formulation as input and attempt to compute an optimal feasible solution as output. Naturally, algorithms which exploit problem structure are usually more efficient than those that do not. In order to be able to exploit the structure of the problem, solution algorithms solve problems that are cast in a *standard form* that emphasizes the useful structure. We remark that casting a problem in a standard form is an opt-reformulation. A good reformulation framework should be aware of the available solution

algorithms and attempt to reformulate given problems into the most appropriate standard form. In this section we review the most common standard forms.

2.4.1 Linear Programming

A mathematical programming problem P is a Linear Programming (LP) problem if (a) $|\mathcal{O}| = 1$ (i.e. the problem only has a single objective function); (b) e is a linear form for all $e \in \mathcal{E}$; and (c) $\mathcal{T}(v) = 0$ (i.e. v is a continuous variable) for all $v \in \mathcal{V}$.

An LP is in standard form if (a) $s_c = 0$ for all constraints $c \in \mathcal{C}$ (i.e. all constraints are equality constraints) and (b) $\mathcal{B}(v) = [0, +\infty]$ for all $v \in \mathcal{V}$. LPs are expressed in standard form whenever a solution is computed by means of the simplex method [24]. By constrast, if all constraints are inequality constraints, the LP is known to be in *canonical form*.

2.4.2 Mixed Integer Linear Programming

A mathematical programming problem P is a Mixed Integer Linear Programming (MILP) problem if (a) $|\mathcal{O}| = 1$; and (b) e is a linear form for all $e \in \mathcal{E}$.

A MILP is in standard form if $s_c = 0$ for all constraints $c \in \mathcal{C}$ and if $\mathcal{B}(v) = [0, +\infty]$ for all $v \in \mathcal{V}$. The most common solution algorithms employed for solving MILPs are Branch-and-Bound (BB) type algorithms [47]. These algorithms rely on recursively partitioning the search domain in a tree-like fashion, and evaluating lower and upper bounds at each search tree node to attempt to implicitly exclude some subdomains from consideration. BB algorithms usually employ the simplex method as a sub-algorithm acting on an auxiliary problem, so they enforce the same standard form on MILPs as for LPs. As for LPs, a MILP where all constraints are inequalities is in *canonical form*.

2.4.3 Nonlinear Programming

A mathematical programming problem P is a Nonlinear Programming (NLP) problem if (a) $|\mathcal{O}| = 1$ and (b) $\mathcal{T}(v) = 0$ for all $v \in \mathcal{V}$.

Many fundamentally different solution algorithms are available for solving NLPs, and most of them require different standard forms. One of the most widely used is Sequential Quadratic Programming (SQP) [36], which requires problem constraints $c \in \mathcal{C}$ to be expressed in the form

$l_c \leq c \leq u_c$ with $l_c, u_c \in \mathbb{R} \cup \{-\infty, +\infty\}$. More precisely, an NLP is in SQP standard form if for all $c \in \mathcal{C}$ (a) $s_c \neq 0$ and (b) there is $c' \in \mathcal{C}$ such that $e_c = e_{c'}$ and $s_c = -s_{c'}$.

2.4.4 Mixed Integer Nonlinear Programming

A mathematical programming problem P is a Mixed Integer Nonlinear Programming (MINLP) problem if $|\mathcal{O}| = 1$. The situation as regards MINLP standard forms is generally the same as for NLPs, save that a few more works have appeared in the literature about standard forms for MINLPs [102, 103, 85, 64]. In particular, the Smith standard form [103] is purposefully constructed so as to make symbolic manipulation algorithms easy to carry out on the formulation. A MINLP is in Smith standard form if:

- $\mathcal{O} = \{d_o, e_o\}$ where e_o is a linear form;
- $\mathcal{C}$ can be partitioned into two sets of constraints $\mathcal{C}_1, \mathcal{C}_2$ such that c is a linear form for all $c \in \mathcal{C}_1$ and $c = (e_c, 0, 0)$ for $c \in \mathcal{C}_2$ where e_c is as follows:
  1. $r(e_c)$ is the sum operator
  2. $\delta^+(r(e_c)) = \{\otimes, v\}$ where (a) $\otimes$ is a nonlinear operator where all subnodes are leaf nodes, (b) $\chi(v) = -1$ and (c) $\tau(v) \in \mathcal{V}$.

Essentially, the Smith standard form consists of a linear part comprising objective functions and a set of constraints; the rest of the constraints have a special form $\otimes(x, y) - v = 0$, with $v, x, y \in \mathcal{V}(P)$ and $\otimes$ a nonlinear operator in $O_{\mathcal{L}}$. By grouping all nonlinearities in a set of equality constraints of the form "variable = operator(variables)" (called *defining constraints*) the Smith standard form makes it easy to construct auxiliary problems. The Smith standard form can be constructed by recursing on the expression trees of a given MINLP [101] and is an opt-reformulation.

Solution algorithms for solving MINLPs are usually extensions of BB type algorithms [103, 64, 61, 111, 84].

2.4.5 Separable problems

A problem P is in separable form if (a) $\mathcal{O}(P) = \{(d_o, e_o)\}$, (b) $\mathcal{C}(P) = \emptyset$ and (c) e_o is such that:

- $r(e_o)$ is the sum operator
- for all distinct $u, v \in \delta^+(r(e_o))$, $\lambda(u) \cap \lambda(v) \cap \mathcal{V}(P) = \emptyset$.

The separable form is a standard form by itself. It is useful because it allows a very easy problem decomposition: for all $u \in \delta^+(r(e_o))$ it suffices to solve the smaller problems Q_u with $\mathcal{V}(Q) = \lambda(v) \cap \mathcal{V}(P)$, $\mathcal{O}(Q) = \{(d_o, u)\}$ and $\mathcal{B}(Q) = \{\mathcal{B}(P)(v) \mid v \in \mathcal{V}(Q)\}$. Then $\bigcup_{u \in \delta^+(r(e_o))} x(\mathcal{V}(Q_u))$ is a solution for P.

2.4.6 Factorable problems

A problem P is in factorable form [80, 117, 100, 111] if:

1. $\mathcal{O} = \{(d_o, e_o)\}$
2. $r(e_o) \in \mathcal{V}$ (consequently, the vertex set of e_o is simply $\{r(e_o)\}$)
3. for all $c \in \mathcal{C}$:
   - $s_c = 0$
   - $r(e_c)$ is the sum operator
   - for all $t \in \delta^+(r(e_c))$, either (a) t is a unary operator and $\delta^+(t) \in \lambda(e_c)$ (i.e. the only subnode of t is a leaf node) or (b) t is a product operator with $\delta^+(t) = \{u, v\}$ such that u, v are both unary operators with only one leaf subnodes.

The factorable form is a standard form by itself. Factorable forms are useful because it is easy to construct many auxiliary problems (including convex relaxations, [80, 4, 100]) from problems cast in this form. In particular, factorable problems can be reformulated to separable problems [80, 111, 84].

2.4.7 D.C. problems

The acronym "d.c." stands for "difference of convex". Given a set $\Omega \subseteq \mathbb{R}^n$, a function $f : \Omega \to \mathbb{R}$ is a *d.c. function* if it is a difference of convex functions, i.e. there exist convex functions $g, h : \Omega \to \mathbb{R}$ such that, for all $x \in \Omega$, we have $f(x) = g(x) - h(x)$. Let C, D be convex sets; then the set $C \backslash D$ is a *d.c. set*. An optimization problem is *d.c.* if the objective function is

d.c. and Ω is a d.c. set. In most of the d.c. literature, however [114, 105, 45], a mathematical programming problem is d.c. if:

- $\mathcal{O} = \{(d_o, e_o)\}$;
- e_o is a d.c. function;
- c is a linear form for all $c \in \mathcal{C}$.

D.C. programming problems have two fundamental properties. The first is that the space of all d.c. functions is dense in the space of all continuous functions. This implies that any continuous optimization problem can be approximated as closely as desired, in the uniform convergence topology, by a d.c. optimization problem [114, 45]. The second property is that it is possible to give explicit necessary and sufficient global optimality conditions for certain types of d.c. problems [114, 105]. Some formulations of these global optimality conditions [104] also exhibit a very useful algorithmic property: if at a feasible point x the optimality conditions do not hold, then the optimality conditions themselves can be used to construct an improved feasible point x'.

2.4.8 Linear Complementarity problems

Linear complementarity problems (LCP) are nonlinear feasibility problems with only one nonlinear constraint. A mathematical programming problem is defined as follows [35], p. 50:

- $\mathcal{O} = \emptyset$;
- there is a constraint $c' = (e, 0, 0) \in \mathcal{C}$ such that (a) $t = r(e)$ is a sum operator; (b) for all $u \in \delta^+(t)$, u is a product of two terms v, f such that $v \in \mathcal{V}$ and $(f, 1, 0) \in \mathcal{C}$;
- for all $c \in \mathcal{C} \setminus \{c'\}$, e_c is a linear form.

Essentially, an LCP is a feasibility problem of the form:

$$\left.\begin{array}{rcl} Ax & \geq & b \\ x & \geq & 0 \\ x^\top(Ax - b) & = & 0, \end{array}\right\}$$

where $x \in \mathbb{R}^n$, A is an $m \times n$ matrix and $b \in \mathbb{R}^m$.

Many types of mathematical programming problems (including MILPs with binary variables [35, 48]) can be recast as LCPs or small extensions of LCP problems [48]. Furthermore, some types of LCPs can be reformulated to LPs [75] and as separable bilinear programs [76]. Certain types of LCPs can be solved by an interior point method [52, 35].

2.4.9 Bilevel Programming problems

The bilevel programming (BLP) problem consists of two nested mathematical programming problems named the *leader* and the *follower* problem.

A mathematical programming problem P is a *bilevel programming problem* if there exist two programming problems L, F (the leader and follower problem) and a subset $\ell \neq \emptyset$ of all leaf nodes of $\mathcal{E}(L)$ such that any leaf node $v \in \ell$ has the form $(v, \mathcal{F})$ where $v \in \mathcal{V}(F)$.

The usual mathematical notation is as follows [28, 12]:

$$\left.\begin{array}{rl} \min_y & F(x(y), y) \\ & \min_x \quad f(x, y) \\ & \text{s.t.} \quad x \in X, \quad y \in Y, \end{array}\right\} \tag{2.2}$$

where X, Y are arbitrary sets. This type of problem arises in economic applications. The leader knows the cost function of the follower, who may or may not know that of the leader; but the follower knows the optimal strategy selected by the leader (i.e. the optimal values of the decision variables of L) and takes this into account to compute his/her own optimal strategy.

BLPs can be reformulated exactly to MILPs with binary variables and vice-versa [12], where the reformulation is as in Defn. 2.3.4. Furthermore, two typical Branch-and-Bound (BB) algorithms for the considered MILPs and BLPs have the property that the the MILP BB can be "embedded" in the BLP BB (this roughly means that the BB tree of the MILP is a subtree of the BB tree of the BLP); however, the contrary does not hold. This seems to hint at a practical solution difficulty ranking in problems with the same degree of worst-case complexity (both MILPs and BLPs are **NP**-hard).

2.4.10 Semidefinite Programming problems

Consider known symmetric $n \times n$ matrices C, A_k for $k \leq m$, a vector $b \in \mathbb{R}^m$ and a symmetric $n \times n$ matrix $X = (x_{ij})$ where x_{ij} is a problem variable for all $i, j \leq n$. The following is a

semidefinite programming problem (SDP) in primal form:

$$\left.\begin{array}{rrcl} \min_X & C \bullet X & & \\ \forall k \leq m & A_k \bullet X & = & b_i \\ & X & \succeq & 0, \end{array}\right\} \tag{2.3}$$

where $X \succeq 0$ is a constraint that indicates that X should be positive semidefinite. We also consider the SDP in dual form:

$$\left.\begin{array}{rrcl} \max_{y,S} & b^\top y & & \\ & \sum_{k \leq m} y_k A_k + S & = & C \\ & S & \succeq & 0, \end{array}\right\} \tag{2.4}$$

where S is a symmetric $n \times n$ matrix and $y \in \mathbb{R}^m$. Both forms of the SDP problem are convex NLPs, so the duality gap is zero. Both forms can be solved by a particular type of polynomial-time interior point method (IPM), which means that solving SDPs is practically efficient [7, 112]. SDPs are important because they provide tight relaxations to (nonconvex) quadratically constrained quadratic programming problems (QCQP), i.e. problems with a quadratic objective and quadratic constraints (see Sect. 4.3.2).

SDPs can be easily modelled with the data structure described in Defn. 2.1.1, for their expression trees are linear forms where each leaf node contains a symmetric matrix. There is no need to explicitly write the semidefinite constraints $X \succeq 0, S \succeq 0$ because the solution IPM algorithms will automatically find optimal X, S matrices that are semidefinite.

Chapter 3

Reformulations

In this chapter we give a systematic study of various types of elementary reformulations (Sect. 3.1) and exact linearizations (Sect. 3.2). Sect. 3.4 provides a few worked out examples. In this summary, we tried to focus on two types of reformulations: those that are in the literature, but may not be known to every optimization practitioner, and those that represent the "tricks of the trade" of most optimization researchers but have never (to the best of our knowledge) been formalized explicitly; so the main contributions of this chapter are systematic and didactic. Since the final aim of automatic reformulations is let the computer arrive at an alternative formulation which is easier to solve, we concentrated on those reformulations which simplified nonlinear terms into linear terms, or which reduced integer variables to continuous variables. By contrast, we did not cite important reformulations (such as the LP duality) which are fundamental in solution algorithms and alternative problem interpretation, but which do not significantly alter solution difficulty.

3.1 Elementary reformulations

In this section we introduce some elementary reformulations in the proposed framework.

3.1.1 Objective function direction

Given an optimization problem P, the optimization direction d_o of any objective function $o \in \mathcal{O}(P)$ can be changed by simply setting $d_o \leftarrow -d_o$. This is an opt-reformulation where φ is the identity, and it rests on the identity $\min f(x) = -\max -f(x)$. We denote the effect of this

reformulation carried out for all objective functions in a given set O by ObjDir(P, O).

3.1.2 Constraint sense

Changing constraint sense simply means to write a constraint c expressed as $e_c \leq b_c$ as $-e_c \geq -b_c$, or $e_c \geq b_c$ as $-e_c \leq -b_c$. This is sometimes useful to convert the problem formulation to a given standard form. This is an opt-reformulation where φ is the identity. It can be carried out on the formulation by setting $\chi(r(e_c)) \leftarrow -\chi(r(e_c))$, $s_c \leftarrow -s_c$ and $b_c = -b_c$. We denote the effect of this reformulation carried out for all constraints in a given set C by ConSense(P, C).

3.1.3 Liftings, restrictions and projections

We define here three important classes of auxiliary problems: liftings, restrictions and projections. Essentially, a lifting is the same as the original problem but with more variables. A restriction is the same as the original problem but with some of the variables replaced by either parameters or constants. A projection is the same as the original problem projected onto fewer variables. Whereas it is possible to give definitions of liftings and restrictions in terms of symbolic manipulations to the data structure given in Defn. 2.1.1, such a definition is in general not possible for projections. Projections and restrictions are in general not opt-reformulations nor reformulations in the sense of Defn. 2.3.5.

3.1.3.1 Lifting

A *lifting* Q of a problem P is a problem such that: $\mathcal{P}(Q) \supsetneq \mathcal{P}(P)$, $\mathcal{V}(Q) \supsetneq \mathcal{V}(P)$, $\mathcal{O}(Q) = \mathcal{O}(P)$, $\mathcal{E}(Q) \supsetneq \mathcal{E}(P)$, $\mathcal{C}(Q) = \mathcal{C}(P)$, $\mathcal{B}(Q) \supsetneq \mathcal{B}(P)$, $\mathcal{T}(Q) \supsetneq \mathcal{T}(P)$. This is an opt-reformulation where φ is a projection operator from $\mathcal{V}(Q)$ onto $\mathcal{V}(P)$: for $y \in \mathcal{F}(Q)$, let $\varphi(y) = (y(v) \mid v \in \mathcal{V}(P))$. We denote the lifting with respect to a new set of variables V by Lift(P, V).

Essentially, a lifting is obtained by adding new variables to an optimization problem.

3.1.3.2 Restriction

A *restriction* Q of a problem P is such that:

- $\mathcal{P}(Q) \supseteq \mathcal{P}(P)$
- $\mathcal{V}(Q) \subsetneq \mathcal{V}(P)$
- $|\mathcal{O}(Q)| = |\mathcal{O}(P)|$
- $|\mathcal{C}(Q)| = |\mathcal{C}(P)|$
- for each $e \in \mathcal{E}(P)$ there is $e' \in \mathcal{E}(Q)$ such that e' is the same as e with any leaf node $v \in \mathcal{V}(P) \setminus \mathcal{V}(Q)$ replaced by an element of $\mathcal{P}(Q) \cup \mathbb{R}$.

We denote the restriction with respect to a sequence of variable V with a corresponding sequence of values A by Restrict(P, V, A).

Essentially, a restriction is obtained by fixing some variables at corresponding given values.

3.1.3.3 Projection

A *projection* Q of a problem P is such that:

- $\mathcal{P}(Q) \supseteq \mathcal{P}(P)$
- $\mathcal{V}(Q) \subsetneq \mathcal{V}(P)$
- $\mathcal{E}, \mathcal{O}, \mathcal{C}, \mathcal{B}, \mathcal{T}(Q)$ are so that for all $y \in \mathcal{F}(Q)$ there is $x \in \mathcal{F}(P)$ such that $x(v) = y(v)$ for all $v \in \mathcal{V}(Q)$.

In general, symbolic algorithms to derive projections depend largely on the structure of the expression trees in E. If E consists entirely of linear forms, this is not difficult (see e.g. [14], Thm. 1.1). We denote the projection onto a set of variables $V = \mathcal{V}(Q)$ as Proj(P, V).

Essentially (and informally) $\mathcal{F}(Q) = \{y \mid \exists x\ (x, y) \in \mathcal{F}(P)\}$.

3.1.4 Equations to inequalities

Converting equality constraints to inequalities may be useful to conform to a given standard form. Suppose P has an equality constraint $c = (e_c, 0, b_c)$. This can be reformulated to a problem Q as follows:

- add two constraints $c_1 = (e_c, -1, b_c)$ and $c_2 = (e_c, 1, b_c)$ to $\mathcal{C}$;
- remove c from $\mathcal{C}$.

This is an opt-reformulation denoted by Eq2Ineq(P, c).

Essentially, we replace the constraint $e_c = b_c$ by the two constraints $e_c \leq b_c, e_c \geq b_c$.

3.1.5 Inequalities to equations

Converting inequalities to equality constraints is useful to convert problems to a given standard form: a very well known case is the standard form of a Linear Programming problem for use with the simplex method. Given a constraint c expressed as $e_c \leq b_c$, we can transform it into an equality constraint by means of a lifting operation and a simple symbolic manipulation on the expression tree e_c, namely:

- add a variable v_c to $\mathcal{V}(P)$ with interval bounds $\mathcal{B}(v_c) = [0, +\infty]$ (added to $\mathcal{B}(P)$) and type $\mathcal{T}(v_c) = 0$ (added to $\mathcal{T}(P)$);
- add a new root node r_0 corresponding to the operator $+$ (sum) to $e_c = (V, A)$, two arcs $(r_0, r(e_c))$, (r_0, v) to A, and we then set $r(e_c) \leftarrow r_0$;
- set $s_c \leftarrow 0$.

We denote this transformation carried out on the set of constraints C by Slack(P, C). Naturally, for original equality constraints, this transformation is defined as the identity.

Performing this transformation on any number of inequality constraints results into an opt-reformulation.

3.1.1 Proposition
Given a set of constraints $C \subseteq \mathcal{C}(P)$, the problem $Q =$ Slack(P, C) is an opt-reformulation of P.

Proof. We first remark that $\mathcal{V}(P) \subseteq \mathcal{V}(Q)$. Consider φ defined as follows: for each $y \in \mathcal{F}(Q)$ let $\varphi(y) = x = (y(v) \mid v \in \mathcal{V}(P))$. It is then easy to show that φ satisfies Defn. 2.3.10. □

3.1.6 Absolute value terms

Consider a problem P involving a term $e = (V, A) \in \mathcal{E}$ where $r(e)$ is the absolute value operator $|\cdot|$ (which is continuous but not differentiable everywhere); since this operator is unary, there is a single expression node f such that $(r(e), f) \in A$. This term can be reformulated so that it is differentiable, as follows:

- add two continuous variables t^+, t^- with bounds $[0, +\infty]$;
- replace e by $t^+ + t^-$;
- add constraints $(f - t^+ - t^-, 0, 0)$ and $(t^+t^-, 0, 0)$ to $\mathcal{C}$.

This is an opt-reformulation denoted by AbsDiff(P, e).

Essentially, we replace all terms $|f|$ in the problem by a sum $t^+ + t^-$, and then add the constraints $f = t^+ - t^-$ and $t^+t^- = 0$ to the problem.

3.1.7 Product of exponential terms

Consider a problem P involving a product $g = \prod_{i \le k} h_i$ of exponential terms, where $h_i = e^{f_i}$ for all $i \le k$. This term can be reformulated as follows:

- add a continuous variable w to $\mathcal{V}$ with $\mathcal{T}(w) = 0$ and bounds $\mathcal{B}(w) = [0, +\infty]$;
- add a constraint $c = (e_c, 0, 0)$ where $e_c = \sum_{i \le k} f_i - \log(w)$ to $\mathcal{C}$;
- replace g with w.

This is an opt-reformulation denoted by ProdExp(P, g). It is useful because many nonlinear terms (product and exponentials) have been the reduced to only one (the logarithm).

Essentially, we replace the product $\prod_i e^{f_i}$ by an added nonnegative continuous variable w and then add the constraint $\log(w) = \sum_i f_i$ to the problem.

3.1.8 Binary to continuous variables

Consider a problem P involving a binary variable $x \in \mathcal{V}$ with ($\mathcal{T}(x) = 2$). This can be reformulated as follows:

- add a constraint $c = (e_c, 0, 0)$ to $\mathcal{C}$ where $e_c = x^2 - x$;
- set $\mathcal{T}(x) = 0$.

This is an opt-reformulation denoted by Bin2Cont(P, x).. Since a binary variable $x \in \mathcal{V}$ can only take values in $\{0, 1\}$, any univariate equation in x that has exactly $x = 0$ and $x = 1$ as solutions can replace the binary constraint $x \in \{0, 1\}$. The most commonly used is the quadratic constraint $x^2 = x$.

In principle, this would reduce all binary problems to nonconvex quadratically constrained problems, which can be solved by a global optimization (GO) solver for nonconvex NLPs. In practice, GO solvers rely on an NLP subsolver to do most of the computationally intensive work, and NLP solvers are generally not very good in handling nonconvex/nonlinear equality constraints such as $x^2 = x$. This reformulation, however, is often used in conjunction with the relaxation of binary linear and quadratic problems (see Sect. 4.4.3).

3.1.9 Integer to binary variables

It is sometimes useful, for different reasons, to convert general integer variables to binary (0-1) variables. One example where this yields a crucial step into a complex linearization is given in Sect. 3.4.2. There are two established ways of doing this: one entails introducing binary assignment variables for each integer values that the variable can take; the other involves the binary representation of the integer variable value. Supposing the integer variable value is n, the first way employs $O(n)$ added binary variables, whereas the second way only employs $O(\log_2(n))$. The first way is sometimes used to linearize complex nonlinear expressions of integer variables by transforming them into a set of constants to choose from (see example in Sect. 3.4.2). The second is often used in an indirect way to try and break symmetries in 0-1 problems: by computing the integer values of the binary representation of two 0-1 vectors x_1, x_2 as integer problem variables v_1, v_2, we can impose ordering constraints such as $v_1 \leq v_2 + 1$ to exclude permutations of x_1, x_2 from the feasible solutions.

3.1.9.1 Assignment variables

Consider a problem P involving an integer variable $v \in \mathcal{V}$ with type $\mathcal{T}(v) = 1$ and bounds $\mathcal{B}(v) = [L_v, U_v]$ such that $U_v - L_v > 1$. Let $V = \{L_v, \ldots, U_v\}$ be the variable domain. Then P can be reformulated as follows:

- for all $j \in V$ add a binary variable w_j to $\mathcal{V}$ with $\mathcal{T}(w_j) = 2$ and $\mathcal{B}(w_j) = [0, 1]$;
- add a constraint $c = (e_c, 0, 1)$ where $e_c = \sum_{j \in V} w_j$ to $\mathcal{C}$;
- add an expression $e = \sum_{j \in V} j w_j$ to $\mathcal{E}$;
- replace all occurrences of v in the leaf nodes of expressions in $\mathcal{E}$ with e.

This is an opt-reformulation denoted by Int2Bin(P, v).

Essentially, we add assignment variables $w_j = 1$ if $v = j$ and 0 otherwise. We then add an assignment constraint $\sum_{j \in V} w_j = 1$ and replace v with $\sum_{j \in V} j w_j$ throughout the problem.

3.1.9.2 Binary representation

Consider a problem P involving an integer variable $v \in \mathcal{V}$ with type $\mathcal{T}(v) = 1$ and bounds $\mathcal{B}(v) = [L_v, U_v]$ such that $U_v - L_v > 1$. Let $V = \{L_v, \ldots, U_v\}$ be the variable domain. Then P can be reformulated as follows:

- let b be the minimum exponent such that $|V| \leq 2^b$;
- add b binary variables $w_1, \ldots, w_b$ to $\mathcal{V}$ such that $\mathcal{T}(w_j) = 2$ and $\mathcal{B}(w_j) = [0, 1]$ for all $j \leq b$;
- add an expression $e = L_v + \sum_{j \leq b} w_j 2^j$
- replace all occurrences of v in the leaf nodes of expressions in $\mathcal{E}$ with e.

This is an opt-reformulation denoted by BinaryRep(P, v).

Essentially, we write the binary representation of v as $L_v + \sum_{j \leq b} w_j 2^j$.

3.1.10 Feasibility to optimization problems

The difference between decision and optimization problems in computer science reflects in mathematical programming on the number of objective functions in the formulation. A formulation without objective functions models a feasibility problem; a formulation with one or more objective models an optimization problem. As was pointed out by the example in the introduction (see Ch. 1, p. 10), for computational reasons it is sometimes convenient to reformulate

a feasibility problem in an optimization problem by introducing constraint tolerances. Given a feasibility problem P with $\mathcal{O} = \emptyset$, we can reformulate it to an optimization problem Q as follows:

- add a large enough constant M to $\mathcal{P}(Q)$;
- add a continuous nonnegative variable ε to $\mathcal{V}(Q)$ with $\mathcal{T}(\epsilon) = 0$ and $\mathcal{B}(\epsilon) = [0, M]$;
- for each equality constraint $c = (e_c, 0, b_c) \in \mathcal{C}$, apply Eq2Ineq$(P, c)$;
- add the expression ε to $\mathcal{E}(Q)$;
- add the objective function $o = (\varepsilon, -1)$ to $\mathcal{O}(Q)$;
- for each constraint $c = (e_c, s_c, b_c) \in \mathcal{C}$ (we now have $s_c \neq 0$), let $e'_c = e_c + s_c\varepsilon$ and $c' = (e'_c, s_c, b_c)$; add c' to $\mathcal{C}(Q)$.

As the original problem has no objective function, the usual definitions of local and global optima do not hold. Instead, we define any point in $\mathcal{F}(P)$ to be both a local and a global optimum (see paragraph under Defn. 2.3.2). Provided the original problem is feasible, this is an opt-reformulation denoted by Feas2Opt(P).

3.1.2 Proposition
Provided $\mathcal{F}(P) \neq \emptyset$, the reformulation Feas2Opt(P) is an opt-reformulation.

Proof. Let F be the projection of $\mathcal{F}(Q)$ on the space spanned by the variables of $\mathcal{P}$ (i.e. all variables of $\mathcal{Q}$ but ε, see Sect. 3.1.3.3), and let π be the projection map. We then have $\mathcal{F}(P) \subseteq F$ (this is because the constraints of Q essentially define a constraint relaxation of P, see Sect. 4.1 and Defn. 4.1.3). Let $x' \in \mathcal{F}(P)$. We define $\psi : F \to \mathcal{F}(P)$ to be the identity on $\mathcal{F}(P)$ and trivially extend it to $\mathcal{F}(Q) \smallsetminus F$ by setting $\psi(z) = x'$ for all $z \in \mathcal{F}(Q) \smallsetminus F$. The function $\phi = \psi \circ \pi$ maps $\mathcal{F}(Q)$ to $\mathcal{F}(P)$, and preserves local minimality by construction, as per Defn. 2.3.6. Since ε is bounded below by zero, and the restriction (see Sect. 3.1.3.2) of Q to $\varepsilon = 0$ is exactly P, any $x^{\in}\mathcal{G}(Q)$ is also in $\mathcal{F}(P)$. Moreover, by definition $\mathcal{G}(P) = \mathcal{F}(P)$ as $\mathcal{O}(P) = \emptyset$, showing that the identity (projected on F) preserves global minimality in the sense of Defn. 2.3.8. $\square$

3.2 Exact linearizations

3.2.1 Definition
An exact linearization of a problem P is an opt-reformulation Q of P where all expressions $e \in \mathcal{E}(P)$ are linear forms.

Different nonlinear terms are linearized in different ways, so we sometimes speak of a linearization of a particular nonlinear term instead of a linearization of a given problem. The amount of work on exact linearizations is considerable, specially in the field of mixed 0/1 quadratic programming (see e.g. [39, 16, 42]). In this Section, we just list the elementary reformulations in this field.

3.2.1 Piecewise linear objective functions

Consider a problem P having an objective function $o = (d_o, e_o) \in \mathcal{O}(P)$ and a finite set of expressions e_k for $k \in K$ such that $e_o = d_o \min\limits_{k \in K} d_o e_k$ (this is a piecewise linear objective function of the form $\min \max_k e_k$ or $\max \min_k e_k$ depending on d_o). This can be linearized by adding one variable and $|K|$ constraints to the problem as follows:

- add a continuous variable t to $\mathcal{V}$ bounded in $[-\infty, +\infty]$;
- for all $k \in K$, add the constraint $c_k = (e_k - t, d_o, 0)$ to $\mathcal{C}$.

This is an opt-reformulation denoted by MinMax(P).

Essentially, we can reformulate an objective function $\min \max_{k \in K} e_k$ as $\min t$ by adding a continuous variable t and the constraints $\forall k \in K \; t \geq e_k$ to the problem.

3.2.2 Product of binary variables

Consider a problem P where one of the expressions $e \in \mathcal{E}(P)$ is $\prod\limits_{k \in \bar{K}} v_k$, where $v_k \in \mathcal{V}(P)$, $\mathcal{B}(v_k) = [0, 1]$ and $\mathcal{T}(v_k) = 2$ for all $k \in \bar{K}$ (i.e. v_k are binary 0-1 variables). This product can be linearized as follows:

- add a continuous variable w tp $\mathcal{V}$ bounded in $[0, 1]$;
- add the constraint $(\sum_{k \in \bar{K}} v_k - w, -1, |\bar{K}| - 1)$ to $\mathcal{C}$;

- for all $k \in \bar{K}$ add the constraint $(w - v_k, -1, 0)$ to $\mathcal{C}$.

This is an opt-reformulation denoted by Prod$(P, \bar{K})$.

Essentially, a product of binary variables $\prod_{k \in \bar{K}} v_k$ can be replaced by an added continuous variable $w \in [0, 1]$ and added constraints $\forall k \in \bar{K}\ w \leq v_k$ and $w \geq \sum_{k \in \bar{K}} v_k - |\bar{K}| + 1$.

As products of binary variables model the very common AND operation, linearizations of binary products are used very often. Hammer and Rudeanu [40] cite [33] as the first published appearance of the above linearization for cases where $|\bar{K}| = 2$. For problems P with products $v_i v_j$ for a given set of pairs $\{i, j\} \in K$ where v_i, v_j are all binary variables, the linearization consists of $|Q|$ applications of Prodbin$(P, \{i, j\})$ for each $\{i, j\} \in K$. Furthermore, we replace each squared binary variable v_i^2 by simply v_i (as $v_i^2 = v_i$ for binary variables v_i). We denote this linearization by ProdSet(P, K).

3.2.3 Product of binary and continuous variables

Consider a problem P involving products $v_i v_j$ for a given set K of ordered variable index pairs (i, j) where v_i is a binary 0-1 variable and v_j is a continuous nonnegative variable with $\mathcal{B}(v_j) = [0, U_j]$. The problem can be linearized as follows:

- for all $(i, j) \in K$ add a continuous variable w_{ij} bounded by $[0, U_j]$ to $\mathcal{V}$;
- for all $(i, j) \in K$ replace the product terms $v_i v_j$ by the variable w_{ij};
- for all $(i, j) \in K$ add the constraints
  $(w_{ij} - U_j v_i, -1, 0), (w_{ij} - v_j, -1, 0), (v_j + U_j v_i - w_{ij}, -1, U_j)$ to $\mathcal{C}$.

This is an opt-reformulation denoted by ProdBinCont(P, K).

Essentially, a product of a binary variable v_i and a continuous nonnegative variable v_j bounded above by U_j can be replaced by an added variable w_{ij} and added constraints $w_{ij} \leq U_j v_i$, $w_{ij} \leq v_j$ and $w_{ij} \geq v_j + U_j v_i - U_j$.

3.2.4 Complementarity constraints

Consider a problem P involving constraints of the form $c = (e_c, 0, 0)$ where (a) $r(e_c)$ is the sum operator, (b) for each node e outgoing from $r(e_c)$, e is a product operator, (c) each of these

product nodes e has two outgoing nodes f, g such that $f \geq 0$ and $g \geq 0$. We can linearize such a constraint as follows:

- for each product operator node e outgoing from $r(e_c)$ and with outgoing nodes f, g:
  1. add a parameter $M > 0$ (as large as possible) to $\mathcal{P}$;
  2. add a binary variable w to $\mathcal{V}$ with $\mathcal{T}(v) = 2$ and $\mathcal{B} = [0, 1]$
  3. add the constraints $(f - Mw, -1, 0)$ and $(g + Mw, -1, M)$ to $\mathcal{C}$
- delete the constraint c.

Provided we set M as an upper bound to the maximum values attainable by f and g, this is an opt-reformulation which is also a linearization. We denote it by CCLin(P).

Essentially, we linearize complementarity constraints $\sum_{k \in K} f_k g_k = 0$ where $f_k, g_k \geq 0$ by eliminating the constraint, adding 0-1 variables w_k for all $k \in K$ and the linearization constraints $f_k \leq M w_k$ and $g_k \leq M(1 - w_k)$.

3.2.5 Minimization of absolute values

Consider a problem P with a single objective function $o = (d_o, e_o) \in \mathcal{O}$ where $e_o = (-d_o) \sum\limits_{k \in \bar{K}} e_k$ where the operator represented by the root node $r(e_k)$ of e_k is the absolute value $|\cdot|$ for all $k \in K \subseteq \bar{K}$. Since the absolute value operator is unary, $\delta^+(r(e_k))$ consists of the single element f_k. Provided f_k are linear forms, this problem can be linearized as follows. For each $k \in K$:

- add continuous variables t_k^+, t_k^- with bounds $[0, +\infty]$;
- replace e_k by $t_k^+ + t_k^-$;
- add constraints $(f_k - t_k^+ - t_k^-, 0, 0)$ to $\mathcal{C}$.

This is an opt-reformulation denoted by MinAbs(P, K).

Essentially, we can reformulate an objective function $\min \sum_{k \in \bar{K}} |f_k|$ as $\min \sum_{k \in \bar{K}} (t_k^+ + t_k^-)$ whilst adding constraints $\forall k \in \bar{K}\ f_k = t_k^+ - t_k^-$ to the problem. This reformulation is related to AbsDiff(P, e) (see Sect. 3.1.6), however the complementarity constraints $t_k^+ t_k^- = 0$

are not needed because of the objective function direction: at a global optimum, because of the minimization of $t_k^+ + t_k^-$, at least one of the variables will have value zero, thus implying the complementarity.

3.2.6 Linear fractional terms

Consider a problem P where an expression in $\mathcal{E}$ has a sub-expression e with a product operator and two subnodes e_1, e_2 where $\xi(e_1) = 1$, $\xi(e_2) = -1$, and e_1, e_2 are affine forms such that $e_1 = \sum_{i \in V} a_i v_i + b$ and $e_2 = \sum_{i \in V} c_i v_i + d$, where $v \subseteq \mathcal{V}$ and $\mathcal{T}(v_i) = 0$ for all $i \in V$ (in other words e is a linear fractional term $\frac{a^\top v + b}{c^\top v + d}$ on continuous variables v). Assume also that the variables v only appear in some linear constraints of the problem $Av = q$ (A is a matrix and q is a vector in $\mathcal{P}$). Then the problem can be linearized as follows:

- add continuous variables α_i, β to $\mathcal{V}$ (for $i \in V$) with $\mathcal{T}(\alpha_i) = \mathcal{T}(\beta) = 0$;
- replace e by $\sum_{i \in V} a_i \alpha_i + b\beta$;
- replace the constraints in $Av = q$ by $A\alpha - q\beta = 0$;
- add the constraint $\sum_{i \in V} c_i \alpha_i + d\beta = 1$;
- remove the variables v from $\mathcal{V}$.

This is an opt-reformulation denoted by LinFract(P, e).

Essentially, α_i plays the role of $\frac{v_i}{c^\top v + d}$, and β that of $\frac{1}{c^\top v + d}$. It is then easy to show that e can be re-written in terms of α, β as $a^\top \alpha + b\beta$, $Av = q$ can be re-written as $A\alpha = q\beta$, and that $c^\top \alpha + d\beta = 1$. Although the original variables v are removed from the problem, their values can be obtained by α, β after the problem solution, by computing $v_i = \frac{\alpha_i}{\beta}$ for all $i \in V$.

3.3 Advanced reformulations

In this section we review a few advanced reformulations in the literature.

3.3.1 Hansen's Fixing Criterion

This method applies to unconstrained quadratic 0-1 problems of the form $\min_{x \in \{0,1\}^n} x^\top Q x$ where Q is an $n \times n$ matrix [41], and relies on fixing some of the variables to values guaranteed to provide a global optimum.

Let P be a problem with $\mathcal{P} = \{n \in \mathbb{N}, \{q_{ij} \in \mathbb{R} \mid 1 \leq i, j \leq n\}\}$, $\mathcal{V} = \{x_i \mid 1 \leq i \leq n\}$, $\mathcal{E} = \{f = \sum_{i,j \leq n} q_{ij} x_i x_j\}$, $\mathcal{O} = \{(f, -1)\}$, $\mathcal{C} = \emptyset$, $\mathcal{B} = [0,1]^n$, $\mathcal{T} = \mathbf{2}$. This can be restricted (see Sect. 3.1.3.2) as follows:

- initialize two sequences $V = \emptyset, A = \emptyset$;
- for all $i \leq n$:
  1. if $q_{ii} + \sum_{j<i} \min(0, q_{ij}) + \sum_{j>i} \min(0, q_{ij}) > 0$ then append x_i to V and 0 to A;
  2. (else) if $q_{ii} + \sum_{j<i} \max(0, q_{ij}) + \sum_{j>i} \max(0, q_{ij}) < 0$ then append x_i to V and 1 to A;
- apply Restrict(P, V, A).

This opt-reformulation is denoted by FixQB(P).

Essentially, any time a binary variable consistently decreases the objective function value when fixed, independently of the values of other variables, it is fixed.

3.3.2 Compact linearization of binary quadratic problems

This reformulation concerns a problem P with the following properties:

- there is a subset of binary variables $x \subseteq \mathcal{V}$ with $|x| = n, \mathcal{T}(x) = 2, \mathcal{B}(x) = [0,1]^n$;
- there is a set $E = \{(i,j) \mid 1 \leq i \leq j \leq n\}$ in $\mathcal{P}$ such that the terms $x_i x_j$ appear as sub-expressions in the expressions $\mathcal{E}$ for all $(i,j) \in E$;
- there is an integer $K \leq n$ in $\mathcal{P}$ and a covering $\{I_k \mid k \leq K\}$ of $\{1, \ldots, n\}$ such that $(\sum_{i \in I_k} x_i, 0, 1)$ is in $\mathcal{C}$ for all $k \leq K$;
- there is a covering $\{J_k \mid k \leq K\}$ of $\{1, \ldots, n\}$ such that $I_k \subseteq J_k$ for all $k \leq K$ such that, letting $F = \{(i,j) \mid \exists k \leq K((i,j) \in I_k \times J_k \vee (i,j) \in J_k \times I_k)\}$, we have $E \subseteq F$.

Under these conditions, the problem P can be exactly linearized as follows:

- for all $(i,j) \in F$ add continuous variables w_{ij} with $\mathcal{T}(w_{ij}) = 0$ and $\mathcal{B}(w_{ij}) = [0,1]$;
- for all $(i,j) \in E$ replace sub-expression $x_i x_j$ with w_{ij} in the expressions $\mathcal{E}$;
- for all $k \leq K, j \in J_k$ add the constraint $(\sum_{i \in I_k} w_{ij} - x_j, 0, 0)$ to $\mathcal{C}$.
- for all $(i,j) \in F$ add the constraint $w_{ij} = w_{ji}$ to $\mathcal{C}$.

This opt-reformulation is denoted by RCLin(P, E). It was shown in [66] that this linearization is exact and has other desirable tightness properties. See [66] for examples.

3.3.3 Reduction Constraints

This reformulation concerns a problem P with the following properties:

- there is a subset $x \subseteq \mathcal{V}$ with $|x| = n$ and a set $E = \{(i,j) \mid 1 \leq i \leq j \leq n\}$ in $\mathcal{P}$ such that the terms $x_i x_j$ appear as sub-expressions in the expressions $\mathcal{E}$ for all $(i,j) \in E$;
- there is a number $m \leq n$, an $m \times n$ matrix $A = (a_{ij})$ and an m-vector b in $\mathcal{P}$ such that $(\sum_{j \leq n} a_{ij} x_j, 0, b_i) \in \mathcal{C}$ for all $i \leq m$.

Let $F = \{(i,j) \mid (i,j) \in E \vee \exists k \leq m (a_{kj} \neq 0\}$. Under these conditions, P can be reformulated as follows:

- for all $(i,j) \in F$ add continuous variables w_{ij} with $\mathcal{T}(w_{ij}) = 0$ and $\mathcal{B}(w_{ij}) = [-\infty, +\infty]$;
- for all $(i,j) \in E$ replace sub-expression $x_i x_j$ with w_{ij} in the expressions $\mathcal{E}$;
- for all $i \leq n, k \leq m$ add the constraints $(\sum_{j \leq n} a_{kj} w_{ij} - b_k x_i, 0, 0)$ to $\mathcal{C}$: we call this linear system the *Reduction Constraint System* (RCS) and $(\sum_{j \leq n} a_{kj} w_{ij}, 0, 0)$ the *companion* system;
- let $B = \{(i,j) \in F \mid w_{ij}$ is basic in the companion$\}$;
- let $N = \{(i,j) \in F \mid w_{ij}$ is non-basic in the companion$\}$;
- add the constraints $(w_{ij} - x_i x_j, 0, 0)$ for all $(i,j) \in N$.

This opt-reformulation is denoted by RedCon(P), and its validity was shown in [63]. It is important because it effectively reduces the number of quadratic terms in the problem (only those corresponding to the set N are added). This reformulation can be extended to work with sparse sets E [72], namely sets E whose cardiality is small with respect to $\frac{1}{2}n(n+1)$.

Essentially, the constraints $w_{ij} = x_i x_j$ for $(i,j) \in B$ are replaced by the RCS $\forall i \leq n\ (Aw_i = x_i)$, where $w_i = (w_{i1}, \ldots, w_{in})$.

3.4 Advanced examples

We give in this section a few advanced examples that illustrate the power of the elementary reformulations given above.

3.4.1 The Hyperplane Clustering Problem

As an example of what can be attained by combining these simple reformulations presented in this chapter, we give a MINLP formulation to the

> HYPERPLANE CLUSTERING PROBLEM (HCP) [26, 22]. Given a set of points $p = \{p_i \mid 1 \leq i \leq m\}$ in $\mathbb{R}^d$ we want to find a set of n hyperplanes $w = \{w_{j1}x_1 + \ldots + w_{jd} = w_j^0 \mid 1 \leq j \leq n\}$ in $\mathbb{R}^d$ and an assignment of points to hyperplanes such that the distances from the hyperplanes to their assigned points are minimized.

We then derive a MILP reformulation. For clarity, we employ the usual mathematical notation instead of the notation given Defn. 2.1.1.

The problem P can be modelled as follows:

- *Parameters*. The set of parameters is given by $p \in \mathbb{R}^{m \times d}, m, n, d \in \mathbb{N}$.
- *Variables*. We consider the hyperplane coefficient variables $w \in \mathbb{R}^{n \times d}$, the hyperplane constants $w^0 \in \mathbb{R}^n$, and the 0-1 assignment variables $x \in \{0,1\}^{m \times n}$.
- *Objective function*. We minimize the total distance, weighted by the assignment variable:

$$\min \sum_{i \leq m} \sum_{j \leq n} |w_j p_i - w_j^0| x_{ij}.$$

- *Constraints*. We consider assignment constraints: each point must be assigned to exactly one hyperplane:
$$\forall i \leq m \quad \sum_{j \leq n} x_{ij} = 1,$$
and the hyperplanes must be nontrivial:
$$\forall j \leq n \quad \sum_{k \leq d} |w_{jk}| = 1,$$
for otherwise the trivial solution with $w = 0, w^0 = 0$ would be optimal.

This is a MINLP formulation because of the presence of the nonlinear terms (absolute values and products in the objective function) and of the binary assignment variables. We shall now apply several of the elementary reformulations presented in this chapter to obtain a MILP reformulation Q of P.

Let $K = \{(i,j) \mid i \leq m, j \leq n\}$.

1. Because x is nonnegative and because we are going to solve the reformulated MILP to global optimality, we can apply an reformulation similar to MinAbs(P, K) (see Sect. 3.2.5) to obtain an opt-reformulation P_1 as follows:
$$\begin{array}{rrcl} \min \sum_{i,j} (t^+_{ij} x_{ij} + t^-_{ij} x_{ij}) & & & \\ \text{s.t.} \quad \forall i & \sum_j x_{ij} & = & 1 \\ \forall j & |w_j| \mathbf{1} & = & 1 \\ \forall i, j & t^+_{ij} - t^-_{ij} & = & w_j p_i - w^0_j, \end{array}$$
where $t^+_{ij}, t^-_{ij} \in [0, M]$ are continuous added variables bounded above by a (large and arbitrary) constant M which we add to the parameter set $\mathcal{P}$. We remark that this upper bound is enforced without loss of generality because w, w^0 can be scaled arbitrarily.

2. Apply ProdBinCont(P_1, K) (see Sect. 3.2.3) to the products $t^+_{ij} x_{ij}$ and $t^-_{ij} x_{ij}$ to obtain a

opt-reformulation P_2 as follows:

$$\begin{aligned}
\min \sum_{i,j} (y_{ij}^+ + y_{ij}^-) & \\
\text{s.t.} \quad \forall i \sum_j x_{ij} &= 1 \\
\forall j \quad |w_j|\mathbf{1} &= 1 \\
\forall i,j \quad t_{ij}^+ - t_{ij}^- &= w_j p_i - w_j^0 \\
\forall i,j \quad y_{ij}^+ &\leq \min(Mx_{ij}, t_{ij}^+) \\
\forall i,j \quad y_{ij}^+ &\geq Mx_{ij} + t_{ij}^+ - M \\
\forall i,j \quad y_{ij}^- &\leq \min(Mx_{ij}, t_{ij}^-) \\
\forall i,j \quad y_{ij}^- &\geq Mx_{ij} + t_{ij}^- - M,
\end{aligned}$$

where $y_{ij}^+, y_{ij}^- \in [0, M]$ are continuous added variables.

3. For each term $e_{jk} = |w_{jk}|$ apply AbsDiff(P_2, e_{jk}) to obtain an opt-reformulation P_3 as follows:

$$\begin{aligned}
\min \sum_{i,j} (y_{ij}^+ + y_{ij}^-) & \\
\text{s.t.} \quad \forall i \sum_j x_{ij} &= 1 \\
\forall i,j \quad t_{ij}^+ - t_{ij}^- &= w_j p_i - w_j^0 \\
\forall i,j \quad y_{ij}^+ &\leq \min(Mx_{ij}, t_{ij}^+) \\
\forall i,j \quad y_{ij}^+ &\geq Mx_{ij} + t_{ij}^+ - M \\
\forall i,j \quad y_{ij}^- &\leq \min(Mx_{ij}, t_{ij}^-) \\
\forall i,j \quad y_{ij}^- &\geq Mx_{ij} + t_{ij}^- - M \\
\forall j \sum_{k \leq d} (u_{jk}^+ + u_{jk}^-) &= 1 \\
\forall j,k \; u_{jk}^+ - u_{jk}^- &= w_{jk} \\
\forall j,k \; u_{jk}^+ u_{jk}^- &= 0,
\end{aligned}$$

where $u_{jk}^+, u_{jk}^- \in [0, M]$ are continuous variables for all j, k. Again, the upper bound does not enforce loss of generality. P_3 is an opt-reformulation of P: whereas P was not everywhere differentiable because of the absolute values, P_3 only involves differentiable terms.

4. We remark that the last constraints of P_3 are in fact complementarity constraints. We apply CCLin(P_3) to obtain the reformulated problem Q:

$$
\begin{array}{rrcl}
\min \sum_{i,j} (y^+_{ij} + y^-_{ij}) & & & \\
\text{s.t.} \quad \forall i \sum_j x_{ij} & = & 1 \\
\forall i,j \quad t^+_{ij} - t^-_{ij} & = & w_j p_i - w^0_j \\
\forall i,j \quad y^+_{ij} & \leq & \min(M x_{ij}, t^+_{ij}) \\
\forall i,j \quad y^+_{ij} & \geq & M x_{ij} + t^+_{ij} - M \\
\forall i,j \quad y^-_{ij} & \leq & \min(M x_{ij}, t^-_{ij}) \\
\forall i,j \quad y^-_{ij} & \geq & M x_{ij} + t^-_{ij} - M \\
\forall j \sum_{k \leq d} (u^+_{jk} + u^-_{jk}) & = & 1 \\
\forall j,k \; u^+_{jk} - u^-_{jk} & = & w_{jk} \\
\forall j,k \; u^+_{jk} & \leq & M z_{jk} \\
\forall j,k \; u^-_{jk} & \leq & M(1 - z_{jk}),
\end{array}
$$

where $z_{jk} \in \{0,1\}$ are binary variables for all j,k. Q is a MILP reformulation of P (see Sect. 2.4.2).

This reformulation allows us to solve P by using a MILP solver — these have desirable properties with respect to MINLP solvers, such as numerical stability and robustness, as well as scalability and an optimality guarantee. A small instance consisting of 8 points and 2 planes in $\mathbb{R}^2$, with $p = \{(1,7),(1,1),(2,2),(4,3),(4,5),(8,3),(10,1),(10,5)\}$ is solved to optimality by the ILOG CPLEX solver [47] to produce the following output:

```
Normalized hyperplanes:
1: (-0.205479) x_1 + (0.547945) x_2 + (-0.684932) = 0
2: (0.769231) x_1 + (1.15385) x_2 + (-8.84615) = 0
Assignment of points to hyperplanar clusters:
hyp_cluster 1 = { 2 3 4 8 }
hyp_cluster 2 = { 1 5 6 7 }.
```

3.4.2 Selection of software components

Large software systems consist of a complex architecture of interdependent, modular software components. These may either be built or bought off-the-shelf. The decision of whether to

build or buy software components influencese the cost, delivery time and reliability of the whole system, and should therefore be taken in an optimal way [115].

Consider a software architecture with n component slots. Let I_i be the set of off-the-shelf components and J_i the set of purpose-built components that can be plugged in the i-th component slot, and assume $I_i \cap J_i = \emptyset$. Let T be the maximum assembly time and R be the minimum reliability level. We want to select a sequence of n off-the-shelf or purpose-built components compatible with the software architecture requirements that minimize the total cost whilst satisfying delivery time and reliability constraints. This problem can be modelled as follows.

- *Parameters*:
  1. Let $N \in \mathbb{N}$;
  2. for all $i \leq n$, s_i is the expected number of invocations;
  3. for all $i \leq n, j \in I_i$, c_{ij} is the cost, d_{ij} is the delivery time, and μ_{ij} the probability of failure on demand of the j-th off-the-shelf component for slot i;
  4. for all $i \leq n, j \in J_i$, $\bar{c}_{ij}$ is the cost, t_{ij} is the estimated development time, τ_{ij} the average time required to perform a test case, p_{ij} is the probability that the instance is faulty, and b_{ij} the testability of the j-th purpose-built component for slot i.
- *Variables*:
  1. Let $x_{ij} = 1$ if component $j \in I_j \cup J_i$ is chosen for slot $i \leq n$, and 0 otherwise;
  2. Let $N_{ij} \in \mathbb{Z}$ be the (non-negative) number of tests to be performed on the purpose-built component $j \in J_i$ for $i \leq n$: we assume $N_{ij} \in \{0, \ldots, N\}$.
- *Objective function*. We minimize the total cost, i.e. the cost of the off-the-shelf components c_{ij} and the cost of the purpose-built components $\bar{c}_{ij}(t_{ij} + \tau_{ij}N_{ij})$:

$$\min \sum_{i \leq n} \left(\sum_{j \in I_i} c_{ij} x_{ij} + \sum_{jinJ_i} \bar{c}_{ij}(t_{ij} + \tau_{ij}N_{ij}) x_{ij} \right).$$

- *Constraints*:
  1. assignment constraints: each component slot in the architecture must be filled by exactly one software component

$$\forall i \leq n \quad \sum_{j \in I_i \cup J_i} x_{ij} = 1;$$

2. delivery time constraints: the delivery time for an off-the-shelf component is simply d_{ij}, whereas for purpose-built components it is $t_{ij} + \tau_{ij} N_{ij}$

$$\forall i \leq n \quad \sum_{j \in I_i} d_{ij} x_{ij} + \sum_{j \in J_i} (t_{ij} + \tau_{ij} N_{ij}) x_{ij} \leq T;$$

3. reliability constraints: the probability of failure on demand of off-the shelf components is μ_{ij}, whereas for purpose-built components it is given by

$$\vartheta_{ij} = \frac{p_{ij} b_{ij} (1 - b_{ij})^{(1-b_{ij})N_{ij}}}{(1 - p_{ij}) + p_{ij}(1 - b_{ij})^{(1-b_{ij})N_{ij}}},$$

so the probability that no failure occurs during the execution of the i-th component is

$$\varphi_i = e^{s_i \left(\sum\limits_{j \in I_i} \mu_{ij} x_{ij} + \sum\limits_{j \in J_i} \vartheta_{ij} x_{ij} \right)},$$

whence the constraint is

$$\prod_{i \leq n} \varphi_i \geq R;$$

notice we have three classes of reliability constraints involving two sets of added variables ϑ, φ.

This problem is a MINLP with no continuous variables. We shall now apply several reformulations to this problem (call it P).

1. Consider the term $g = \prod_{i \leq n} \varphi_i$ and apply ProdExp(P, g) to P to obtain P_1 as follows:

$$\begin{array}{rrcl}
\min \sum\limits_{i \leq n} \left(\sum\limits_{j \in I_i} c_{ij} x_{ij} + \sum\limits_{j \in J_i} \bar{c}_{ij} (t_{ij} + \tau_{ij} N_{ij}) x_{ij} \right) & & & \\
\forall i \leq n & \sum\limits_{j \in I_i \cup J_i} x_{ij} & = & 1 \\
\forall i \leq n & \sum\limits_{j \in I_i} d_{ij} x_{ij} + \sum\limits_{j \in J_i} (t_{ij} + \tau_{ij} N_{ij}) x_{ij} & \leq & T \\
& \dfrac{p_{ij} b_{ij} (1 - b_{ij})^{(1-b_{ij})N_{ij}}}{(1 - p_{ij}) + p_{ij}(1 - b_{ij})^{(1-b_{ij})N_{ij}}} & = & \vartheta_{ij} \\
& w & \geq & R \\
& \sum\limits_{i \leq n} s_i \left(\sum\limits_{j \in I_i} \mu_{ij} x_{ij} + \sum\limits_{j \in J_i} \vartheta_{ij} x_{ij} \right) & = & \log(w),
\end{array}$$

and observe that $w \geq R$ implies $\log(w) \geq \log(R)$ because the $\log$ function is monotonically increasing, so the last two constraints can be grouped into a simpler one not involving logarithms of problem variables:

$$\sum_{i \leq n} s_i \left(\sum_{j \in I_i} \mu_{ij} x_{ij} + \sum_{j \in J_i} \vartheta_{ij} x_{ij} \right) \geq \log(R).$$

2. We now make use of the fact that N_{ij} is an integer variable for all $i \leq n, j \in J_i$, and apply Int2Bin(P, N_{ij}). For $k \in \{0, \ldots, N\}$ we add assignment variables ν_{ij}^k so that $\nu_{ij}^k = 1$ if $N_{ij} = k$ and 0 otherwise. Now for all $k \in \{0, \ldots, N\}$ we compute the constants $\vartheta^k = \frac{p_{ij} b_{ij} (1-b_{ij})^{(1-b_{ij})k}}{(1-p_{ij}) + p_{ij}(1-b_{ij})^{(1-b_{ij})k}}$, which we add to the problem parameters. We remove the constraints defining ϑ_{ij} in function of N_{ij}: since the following constraints are valid:

$$\forall i \leq n, j \in J_i \quad \sum_{k \leq N} \nu_{ij}^k = 1 \tag{3.1}$$

$$\forall i \leq n, j \in J_i \quad \sum_{k \leq N} k \nu_{ij}^k = N_{ij} \tag{3.2}$$

$$\forall i \leq n, j \in J_i \sum_{k \leq N} \vartheta^k \nu_{ij}^k = \vartheta_{ij}, \tag{3.3}$$

the second constraints are used to replace N_{ij} and the third to replace ϑ_{ij}. The first constraints are added to the formulation. We obtain:

$$\begin{aligned}
\min \sum_{i \leq n} \left(\sum_{j \in I_i} c_{ij} x_{ij} + \sum_{j \in J_i} \bar{c}_{ij} (t_{ij} + \tau_{ij} \sum_{k \leq N} k \nu_{ij}^k) x_{ij} \right) & \\
\forall i \leq n \quad \sum_{j \in I_i \cup J_i} x_{ij} &= 1 \\
\forall i \leq n \quad \sum_{j \in I_i} d_{ij} x_{ij} + \sum_{j \in J_i} (t_{ij} + \tau_{ij} \sum_{k \leq N} k \nu_{ij}^k) x_{ij} &\leq T \\
\sum_{i \leq n} s_i \left(\sum_{j \in I_i} \mu_{ij} x_{ij} + \sum_{j \in J_i} x_{ij} \sum_{k \leq N} \vartheta^k \nu_{ij}^k \right) &\geq \log(R) \\
\forall i \leq n, j \in J_i \quad \sum_{k \leq N} \nu_{ij}^k &= 1.
\end{aligned}$$

3. We distribute products over sums in the formulation to obtain the binary product sets $\{x_{ij} \nu_{ij}^k \mid k \leq N\}$ for all $i \leq n, j \in J_i$: by repeatedly applying the Prod reformulation to all binary products of binary variables, we get a MILP opt-reformulation Q of P where all the variables are binary.

We remark that the MILP opt-reformulation Q derived above has a considerably higher cardinality than $|P|$. More compact reformulations are applicable in step 3 because of the presence of the assignment constraints (see Sect. 3.3.2).

Reformulation Q essentially rests on linearization variables w_{ij}^k which replace the quadratic terms $x_{ij}\nu_{ij}^k$ throughout the formulation. A semantic interpretation of step 3 is as follows. We notice that for $i \leq n, j \in J_i$, if $x_{ij} = 1$, then $x_{ij} = \sum_k \nu_{ij}^k$ (because only one value k will be selected), and if $x_{ij} = 0$, then $x_{ij} = \sum_k \nu_{ij}^k$ (because no value k will be selected). This means that

$$\forall i \leq n, j \in J_i \quad x_{ij} = \sum_{k \leq N} \nu_{ij}^k \tag{3.4}$$

is a valid problem constraint. We use it to replace x_{ij} everywhere in the formulation where it appears with $j \in I_i$, obtaining a opt-reformulation with x_{ij} for $j \in I_i$ and quadratic terms $\nu_{ij}^k \nu_{lp}^h$. Now, because of (3.1), these are zero when $(i,j) \neq (l,p)$ or $k \neq h$ and are equal to ν_{ij}^k when $(i,j) = (l,p)$ and $k = h$, so they can be linearized exactly by replacing them by either 0 or ν_{ij}^k according to their indices. What this really means is that the reformulation Q, obtained through a series of automatic reformulation steps, is a semantically different formulation defined in terms of the following decision variables:

$$\forall i \leq n, j \in I_i \quad x_{ij} = \begin{cases} 1 & \text{if } j \in I_i \text{ is assigned to } i \\ 0 & \text{otherwise.} \end{cases}$$

$$\forall i \leq n, j \in J_i, k \leq N \quad \nu_{ij}^k = \begin{cases} 1 & \text{if } j \in J_i \text{ is assigned to } i \text{ and there are } k \text{ tests to be performed} \\ 0 & \text{otherwise.} \end{cases}$$

This is an important hint to the importance of automatic reformulation in problem analysis: automatic reformulation is a syntactical operation, the result of which, when interpreted can really be given a new meaning.

Chapter 4

Relaxations

Loosely speaking, a relaxation of a problem P is an auxiliary problem of P with fewer constraints. Relaxations are useful because they often yield problems which are simpler to solve yet they provide a bound on the objective function value at the optimum.

Such bounds are mainly used in Branch-and-Bound type algorithms, which are the most common exact or ε-approximate (for a given $\varepsilon > 0$) solution algorithms for MILPs, nonconvex NLPs and MINLPs. Although the variants for solving MILPs, NLPs and MINLPs are rather different, they all conform to the same implicit enumeration search type. Lower and upper bounds are computed for the problem over the current variable domains. If the bounds are sufficiently close, a global optimum was found in the current domain: store it if it improves the incumbent (i.e. the current best optimum). Otherwise, partition the domain and recurse over each subdomain in the partition. Should a bound be worse off than the current incumbent during the search, discard the domain immediately without recursing on it. Under some regularity conditions, the recursion terminates. The Branch-and-Bound algorithm has been used on combinatorial optimization problems since the 1950s [5]. Its first application to nonconvex NLPs is [29]. More recently, Branch-and-Bound has evolved into Branch-and-Cut and Branch-and-Price for MILPs [83, 120, 47], which have been used to solve some practically difficult problems such as the Travelling Salesman Problem (TSP) [11]. Some recent MINLP-specific Branch-and-Bound approaches are [90, 9, 4, 3, 103, 111, 64].

A further use of bounds provided by mathematical programming formulations is to evaluate the performance of heuristic algorithms without an approximation guarantee [25]. Bounds are sometimes also used to guide heuristics [88].

In this chapter we define relaxations and review the most useful ones. In Sect. 4.1 we give

some basic definitions. We then list elementary relaxations in Sect. 4.2 and more advanced ones in Sect. 4.3. We discuss relaxation strengthening in Sect. 4.4.

4.1 Definitions

Consider an optimization problem $P = (\mathcal{P}, \mathcal{V}, \mathcal{E}, \mathcal{O}, \mathcal{C}, \mathcal{B}, \mathcal{T})$ and let Q be such that: $\mathcal{P}(Q) \supseteq \mathcal{P}(P)$, $\mathcal{V}(Q) = \mathcal{V}(P)$, $\mathcal{E}(Q) \supseteq \mathcal{E}(P)$ and $\mathcal{O}(Q) = \mathcal{O}(P)$.

We first define relaxations in full generality.

4.1.1 Definition
Q is a relaxation of P if $\mathcal{F}(P) \subsetneq \mathcal{F}(Q)$.

What we might call the fundamental theorem of relaxations states that relaxations provide bounds to the objective function.

4.1.2 Theorem
Let Q be a relaxation of P and let (f, d) be an objective function of P, $x \in \mathcal{G}(P)$ and $y \in \mathcal{G}(Q)$. Then $df(y) \geq df(x)$.

Proof. Since $\mathcal{F}(Q) \supseteq \mathcal{F}(P)$, for all $x \in \mathcal{G}(P)$, $x \in \mathcal{F}(Q)$, which implies the result. □

Defn. 4.1.1 is not used very often in practice because it does not say anything on how to construct Q. The following elementary relaxations are more useful.

4.1.3 Definition
Q is a:

- *constraint relaxation of P if $\mathcal{C}(Q) \subsetneq \mathcal{C}(Q)$;*
- *bound relaxation of P if $\mathcal{B}(Q) \subsetneq \mathcal{B}(Q)$;*
- *a continuous relaxation of P if $\exists v \in \mathcal{V}(P)$ $(\mathcal{T}(v) > 0)$ and $\mathcal{T}(Q) = 0$.*

4.2 Elementary relaxations

We shall consider two types of elementary relaxations: the continuous relaxation and the convex relaxation. The former is applicable to MILPs and MINLPs, and the latter to (nonconvex) NLPs and MINLPs. They are both based on the fact that whereas solving MILPs and MINLPs is

considered difficult, there are efficient algorithms for solving LPs and convex NLPs. Since the continuous relaxation was already defined in Defn. 4.1.3 and trivially consists in considering integer/discrete variables as continuous ones, in the rest of this section we focus on convex relaxations.

Formally (and somewhat obviously), Q is a *convex relaxation* of a given problem P if Q is a relaxation of P and Q is convex. Associated to all sBB in the literature there is a (nonconvex) NLP or MINLP in standard form, which is then used as a starting point for the convex relaxation.

4.2.1 Outer approximation

Outer approximation (OA) is a technique for defining a polyhedral approximation of a convex nonlinear feasible region, based on computing tangents to the convex feasible set at suitable boundary points [27, 31, 51]. An outer approximation relaxation relaxes a convex NLP to an LP, (or a MINLP to a MILP) and is really a "relaxation scheme" rather than a relaxation: since the tangents to *all* boundary points of a convex set define the convex set itself, any choice of (finite) set of boundary points of the convex can be used to define a different outer approximation. OA-based optimization algorithms identify sets of boundary points that eventually guarantee that the outer approximation will be exact near the optimum. In [51], the following convex MINLP is considered:

$$\left.\begin{array}{rrcl} \min & L_0(x) + cy & & \\ \text{s.t.} & L(x) + By & \leq & 0 \\ & x^L \leq & x & \leq x^U \\ & y & \in & \{0,1\}^q, \end{array}\right\} \tag{4.1}$$

where $L_1 : \mathbb{R}^n \to \mathbb{R}$, $L : \mathbb{R}^n \to \mathbb{R}^m$ are convex once-differentiable functions, $c \in \mathbb{R}^q$, B is an $m \times q$ matrix. For a given $y' \in \{0,1\}^q$, let $P(y')$ be (4.1) with y fixed at y'. Let $\{y^j\}$ be a sequence of binary q-vectors. Let $T = \{j \mid P(y^j)$ is feasible with solution $x^j\}$. Then the following is a MILP outer approximation for (4.1):

$$\left.\begin{array}{rrcl} \min_{x,y,\eta} & \eta & & \\ \forall j \in T & L_0(x^j) + \nabla L_0(x^j)(x - x^j) + cy & \leq & \eta \\ \forall j & L(x^j) + \nabla L(x^j)(x - x^j) + By & \leq & 0 \\ & x^L \leq & x & \leq x^U \\ & y & \in & \{0,1\}^q, \end{array}\right\}$$

where x^j is the solution to $F(y^j)$ (defined in [31]) whenever $P(y^j)$ is infeasible.

4.2.2 αBB convex relaxation

The αBB algorithm [9, 4, 3, 32] targets single-objective NLPs where the expressions in the objective and constraints are twice-differentiable. The convex relaxation of the problem P:

$$\left.\begin{array}{rrcl} \min_x & f(x) & & \\ \text{s.t.} & g(x) & \leq & 0 \\ & h(x) & = & 0 \\ & x^L \leq & x & \leq x^U \end{array}\right\} \tag{4.2}$$

is obtained as follows.

1. Apply the Eq2Ineq reformulation (see Sect. 3.1.4) to each nonlinear equality constraint in $\mathcal{C}$, obtaining an opt-reformulation P_1 of P.
2. For every nonconvex inequality constraint $c = (e_c, s_c, b_c) \in \mathcal{C}(P_1)$:
   (a) if the root node r of the expression tree e_c is a sum operator, for every subnode $s \in \delta^+(r)$ replace s with a specialized convex underestimator if s is a bilinear, trilinear, linear fractional, fractional trilinear, univariate concave term. Otherwise replace with α-underestimator;
   (b) otherwise, replace r with a specialized if s is a bilinear, trilinear, linear fractional, fractional trilinear, univariate concave term. Otherwise replace with α-underestimator.

The specialized underestimators are as follows: McCormick's envelopes for bilinear terms [80, 6], the second-level RLT bound factor linearized products [99, 98, 95] for trilinear terms, and a secant underestimator for univariate concave terms. Fractional terms are dealt with by extending the bilinear/trilinear underestimators to bilinear/trilinear products of univariate functions and then noting that $x/y = \phi_1(x)\phi_2(y)$ where ϕ_1 is the identity and $\phi_2(y) = 1/y$ [77]. Recently, the convex underestimator for trilinear terms have been replaced with the convex envelopes [81].

The general-purpose α-underestimator:

$$\alpha(x^L - x)^\top (x^U - x) \tag{4.3}$$

is a quadratic convex function that for suitable values of α is "convex enough" to overpower the generic nonconvex term. This occurs for

$$\alpha \geq \max\{0, -\frac{1}{2} \min_{x^L \leq x \leq x^U} \lambda(x)\},$$

where $\min \lambda(x)$ is the minimum eigenvalue of the Hessian of the generic nonconvex term in function of the problem variables.

The resulting αBB relaxation Q of P is a convex NLP.

4.2.3 Branch-and-Contract convex relaxation

The convex relaxation is used in the Branch-and-Contract algorithm [121], targeting nonconvex NLPs with twice-differentiable objective function and constraints. This relaxation is derived essentially in the same way as Sect. 4.2.2. The differences are:

- the problem is assumed to only have inequality constraints of the form $c = (e_c, -1, 0)$;
- each function (in the objective and constraints) consists of a sum of nonlinear terms including: bilinear, linear fractional, univariate concave, and generic convex.

The convex relaxation is then constructed by replacing each nonconvex nonlinear term in the objective and constraints by a corresponding envelope or relaxation. The convex relaxation for linear fractional term had not appeared in the literature before [121].

4.2.4 Symbolic reformulation based convex relaxation

This relaxation is used in the symbolic reformulation spatial Branch-and-Bound algorithm proposed in [102, 103]. It can be applied to all NLPs and MINLPs for which a convex underestimator and a concave overestimator are available. It consists in reformulating P to the Smith standard form (see Sect. 2.4.4) and then replacing every defining constraint with the convex and concave under/over-estimators. In his Ph.D. thesis [101], Smith had tried both NLP and LP convex relaxations, finding that LP relaxations were more reliable and faster to compute, although of course with slacker bounds. The second implementation of the sBB algorithm he proposed is described in [62, 64] and implemented in the $oo\mathcal{OPS}$ software framework [73]. Both versions of this algorithm consider under/overestimators for the following terms: bilinear, univariate concave, univariate convex (linear fractional being reformulated to bilinear). The second version also included estimators for piecewise convex/concave terms. One notable feature of this relaxation is that it can be adapted to deal with more terms. Some recent work in polyhedral envelopes, for example [106], gives conditions under which the sum of the envelopes is the envelope of the sum: this would yield a convex envelope for a sum of terms. It

would then suffice to provide for a defining constraint in the Smith standard form linearizing the corresponding sum. The Smith relaxation is optionally strengthened via LP-based optimality and feasibility based range reduction techniques. After every range reduction step, the convex relaxation is updated with the new variable ranges in an iterative fashion until no further range tightening occurs [101, 62, 64].

This relaxation is at the basis of the sBB solver [64] in the $oo\mathcal{OPS}$ software framework [73], which was used to obtain solutions of many different problem classes: pooling and blending problems [43, 72], distance geometry problems [54, 56], and a quantum chemistry problem [57, 69].

4.2.5 BARON's convex relaxation

BARON (Branch And Reduce Optimization Navigator) is a commercial Branch-and-Bound based global optimization solver (packaged within the GAMS [21] modelling environment) which is often quoted as being the *de facto* standard solver for MINLPs [111, 110]. Its convex relaxation is derived essentially in the same way as in Sect. 4.2.4. The differences are:

- better handling of fractional terms [107, 109]
- advanced range reduction techniques (optimality, feasibility and duality based, plus a learning reduction heuristic)
- optionally, an LP relaxation is derived via outer approximation.

4.3 Advanced relaxations

In this section we shall describe some more advanced relaxations, namely the Lagrangian relaxation, the semidefinite relaxation, the reformulation-linearization technique and the signomial relaxation.

4.3.1 Lagrangian relaxation

Consider a MINLP

$$\left.\begin{array}{rrcl} f^* = \min_x & f(x) & & \\ \text{s.t.} & g(x) & \leq & 0 \\ & x & \in & X \subseteq \mathbb{R}^n, \end{array}\right\} \tag{4.4}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ are continuous functions and X is an arbitrary set. The Lagrangian relaxation consists in "moving" the weighted constraints to the objective function, namely:

$$\left.\begin{array}{rrcl} L(\mu) = \inf_x & f(x) & + & \mu^\top g(x) \\ & x & \in & X \subseteq \mathbb{R}^n, \end{array}\right\}$$

for some nonnegative $\mu \in \mathbb{R}^m_+$. For all $x \in X$ with $g(x) \leq 0$, we have $\mu^\top g(x) \leq 0$, which implies $L(\mu) \leq f^*$ for all $\mu \geq 0$. In other words, $L(\mu)$ provides a lower bound to (4.4) for all $\mu \geq 0$. Thus, we can improve the tightness of the relaxation by solving the Lagrangian problem

$$\max_{\mu \geq 0} L(\mu), \tag{4.5}$$

(namely, we attempt to find the largest possible lower bound). If (4.4) is an LP problem, it is easy to show that the Lagrangian problem (4.5) is the dual LP problem. In general, solving (4.5) is not a computationally easy task [84]. However, one of the nice features of Lagrangian relaxations is that they provide a lower bound for each value of $\mu \geq 0$, so (4.5) does not need to be solved at optimality. Another useful feature is that any subset of problem constraints can be relaxed, for X can be defined arbitrarily. This is useful for problems that are almost block-separable, i.e. those problems that can be decomposed in some independent subproblems bar a few constraints involving all the problem variables (also called complicating constraints). In these cases, one considers a Lagrangian relaxation of the complicating constraints and then solves a block-separable Lagrangian problem. This approach is called Lagrangian decomposition.

The Lagrangian relaxation has some interesting theoretical properties: (a) for convex NLPs it is a global reformulation [20]; (b) for MILPs, it is at least as tight as the continuous relaxation [120]; (c) for MINLPs, under some conditions (i.e. some constraint qualification and no equality constraints) it is at least as tight as any convex relaxation obtained by relaxing each nonconvex term or each constraint one by one [46], such as all those given in Sect. 4.2. Further material on the use of Lagrangian relaxation in NLPs and MINLPs can be found in [84, 46].

Consider a problem P such that $\mathcal{O}(P) = \{(e_o, d_o)\}$ and a subset of constraints $C \subseteq \mathcal{C}(P)$. A Lagrangian relaxation of C in P (denoted by LagRel(P, C)) is a problem Q defined as follows.

- $\mathcal{V}(Q) = \mathcal{V}(P)$, $\mathcal{B}(Q) = \mathcal{B}(P)$, $\mathcal{T}(Q) = \mathcal{T}(P)$,
- $\mathcal{P}(Q) = \mathcal{P}(P) \cup \{\mu_c \mid c \in C\}$,
- $\mathcal{C}(Q) = \mathcal{C}(P) \smallsetminus C$,
- $\mathcal{O}(Q) = \{(e'_o, d'_o)\}$, where $e'_o = e_o + \sum_{c \in C} \mu_c c$.

The Lagrangian problem cannot itself be defined in the data structure of Defn. 2.1.1, for the $\max$ operator is only part of $O_{\mathcal{L}}$ as long as it has a finite number of arguments.

4.3.2 Semidefinite relaxation

As was pointed out in Sect. 2.4.10, SDPs provide very tight relaxations for quadratically constrained quadratic MINLPs (QCQP). A QCQP in general form is as follows [10]:

$$\left.\begin{array}{rrcl} \min_x & x^\top Q_0 x + a_0^\top x & & \\ \forall i \in I & x^\top Q_i x + a_i^\top x & \leq & b_i \\ \forall i \in E & x^\top Q_i x + a_i^\top x & = & b_i \\ & x^L \leq & x & \leq x^U \\ \forall j \in J & x_i \in \mathbb{Z}, & & \end{array}\right\} \tag{4.6}$$

where $I \cup E = \{1, \ldots, m\}$, $J \subseteq \{1, \ldots, n\}$, $x \in \mathbb{R}^n$, Q_i is an $n \times n$ symmetric matrix for all $i \leq m$. For general matrices Q_i and $J \neq$, the QCQP is nonconvex. Optionally, the integer variables can be reformulated exactly to binary (see Int2Bin, Sect. 3.1.9) and subsequently to continuous (see Bin2Cont, Sect. 3.1.8) via the introduction of the constraints $x_i^2 - x_i = 0$ for all $i \in J$: since these constraints are quadratic, they can be accommodated in formulation (4.6) by suitably modifying the Q_i matrices. Many important applications can be modelled as QCQPs, including graph bisection (see Sect. 2.1.1.2) and graph partitioning [66], scheduling with communication delays [25], distance geometry problems such as the KNP (see Sect. 2.1.1.3) [54] and the Molecular Distance Geometry Problem (MDGP) [56, 67], pooling and blending problems from the oil industry [43, 72] and so on.

The SDP relaxation of the QCQP is constructed as follows:

- replace all quadratic products $x_i x_j$ in (4.6) with an added linearization variable X_{ij}
- form the matrix $X = (X_{ij})$ and the variable matrix

$$\bar{X} = \begin{pmatrix} 1 & x^\top \\ x & X \end{pmatrix}$$

- for all $0 \leq i \leq m$ form the matrices

$$\bar{Q}_i = \begin{pmatrix} -b_i & a_i^\top/2 \\ a_i/2 & Q_i \end{pmatrix}$$

- the following is an SDP relaxation for QCQP:

$$\left.\begin{array}{rrcl} \min_X & \bar{Q}_0 \bullet \bar{X} & & \\ \forall i \in I & \bar{Q}_i \bullet \bar{X} & \leq & 0 \\ \forall i \in E & \bar{Q}_i \bullet \bar{X} & = & 0 \\ & x^L \leq & x & \leq x^U \\ & \bar{X} & \succeq & 0. \end{array}\right\} \tag{4.7}$$

As for the SDP formulation of Sect. 2.4.10, the SDP relaxation can be easily represented by the data structure described in Defn. 2.1.1.

4.3.3 Reformulation-Linearization Technique

The Reformulation-Linearization Technique (RLT) is a relaxation method for mathematical programming problems with quadratic terms. The RLT linearizes all quadratic terms in the problem and generates valid linear equation and inequality constraints by considering multiplications of bound factors (terms like $x_i - x_i^L$ and $x_i^U - x_i$) and constraint factors (the left hand side of a constraint such as $\sum_{j=1}^n a_j x_j - b \geq 0$ or $\sum_{j=1}^n a_j x_j - b = 0$). Since bound and constraint factors are always non-negative, so are their products: this way one can generate sets of valid problem constraints. In a sequence of papers published from the 1980s onwards (see e.g. [96, 99, 93, 98, 94, 100, 92]), RLT-based relaxations were derived for many different classes of problems, including IPs, NLPs, MINLPs in general formulation, and several real-life applications. It was shown that the RLT can be used in a lift-and-project fashion to generate the convex envelope of binary and general discrete problems [97, 2].

4.3.3.1 Basic RLT

The RLT consists of two symbolic manipulation steps: reformulation and linearization. The reformulation step is a reformulation in the sense of Defn. 2.3.10. Given a problem P, the reformulation step produces a reformulation Q' where:

- $\mathcal{P}(Q') = \mathcal{P}(P)$;

- $\mathcal{V}(Q') = \mathcal{V}(P)$;
- $\mathcal{E}(Q') \supseteq \mathcal{E}(P)$;
- $\mathcal{C}(Q') \supseteq \mathcal{C}(P)$;
- $\mathcal{O}(Q') = \mathcal{O}(P)$;
- $\mathcal{B}(Q') = \mathcal{B}(P)$;
- $\mathcal{T}(Q') = \mathcal{T}(P)$;
- $\forall x, y \in \mathcal{V}(P)$, add the following constraints to $\mathcal{C}(Q')$:

$$(x - L_x)(y - L_y) \geq 0 \tag{4.8}$$
$$(x - L_x)(U_y - y) \geq 0 \tag{4.9}$$
$$(U_x - x)(y - L_y) \geq 0 \tag{4.10}$$
$$(U_x - x)(L_y - y) \geq 0; \tag{4.11}$$

- $\forall x \in \mathcal{V}(P), c = (e_c, s_c, b_c) \in \mathcal{C}(P)$ such that e_c is an affine form, $s_c = 1$ and $b_c = 0$ (we remark that all linear inequality constraints can be easily reformulated to this form, see Sect. 3.1.2), add the following constraints to $\mathcal{C}(Q')$:

$$e_c(x - L_x) \geq 0 \tag{4.12}$$
$$e_c(U_x - x) \geq 0; \tag{4.13}$$

- $\forall x \in \mathcal{V}(P), c = (e_c, s_c, b_c) \in \mathcal{C}(P)$ such that e_c is an affine form, $s_c = 0$ and $b_c = 0$ (we remark that all linear equality constraints can be trivially reformulated to this form), add the following constraint to $\mathcal{C}(Q')$:

$$e_c x = 0. \tag{4.14}$$

Having obtained Q', we proceed to linearize all the quadratic products engendered by (4.8)-(4.14). We derive the auxiliary problem Q from Q' by reformulating Q' in Smith's standard form (see Sect. 2.4.4) and then performing a constraint relaxation with respect to all defining constraints. Smith's standard form is a reformulation of the lifting type, and the obtained constraint relaxation Q is a MILP whose optimal objective function value $\bar{f}$ is a bound to the optimal objective function value f^* of the original problem P. The bound obtained in this way is shown to dominate, or be equivalent to, several other bounds in the literature [2].

We remark in passing that (4.8)-(4.11), when linearized by replacing the bilinear term xy with an added variable w, are also known in the literature as McCormick relaxation, as they were first proposed as a convex relaxation of the nonconvex constraint $w = xy$ [80], shown to be the convex envelope [6], and widely used in spatial Branch-and-Bound (sBB) algorithms for global optimization [103, 4, 3, 111, 64]. RLT constraints of type (4.14) have been the object of further research showing their reformulating power [60, 61, 63, 72, 66] (also see Sect. 3.3.2 and Sect. 3.3.3).

4.3.3.2 RLT Hierarchy

The basic RLT method can be extended to provide a hierarchy of relaxations, by noticing that we can form valid RLT constraints by multiplying sets of bound and constraint factors of cardinality higher than 2, and then projecting the obtained constraints back to the original variable space. In [97, 2] it is shown that this fact can be used to construct the convex hull of an arbitrary MILP P. For simplicity, we only report the procedure for MILP in standard canonical form (see Sect. 2.4.2) where all discrete variables are binary, i.e. $\mathcal{T}(v) = 2$ for all $v \in \mathcal{V}(P)$. Let $|\mathcal{V}(P)| = n$. For all integer $d \leq n$, let P_d be the relaxation of P obtained as follows:

- for all linear constraint $c = (e_c, 1, 0) \in \mathcal{C}(P)$, subset $V \subseteq \mathcal{V}(P)$ and finite binary sequence B with $|V| = |B| = d$ such that B_x is the x-th term of the sequence for $x \in V$, add the valid constraint:
$$e_c \left(\prod_{\substack{x \in V \\ B_x = 0}} x \right) \left(\prod_{\substack{x \in V \\ B_x = 1}} (1 - x) \right) \geq 0; \tag{4.15}$$
we remark that (4.15) is a multivariate polynomial inequality;
- for all monomials of the form
$$a \prod_{x \in J \subseteq \mathcal{V}(P)} x$$
with $a \in \mathbb{R}$ in a constraint (4.15), replace $\prod_{x \in J} x$ with an added variable w_J (this is equivalent to relaxing a defining constraint $w_J = \prod_{x \in J}$ in the Smith's standard form restricted to (4.15).

Now consider the projection X_d of P_d in the $\mathcal{V}(P)$ variable space (see Sect. 3.1.3.3). It can be shown that
$$\text{conv}(\mathcal{F}(P)) \subseteq \mathcal{F}(X_n) \subseteq \mathcal{F}(X_{n-1}) \ldots \subseteq \mathcal{F}(X_1) \subseteq \mathcal{F}(P).$$

We recall that for a set $Y \subseteq \mathbb{R}^n$, conv(Y) is defined as the smallest convex subset of $\mathbb{R}^n$ containing Y.

A natural practical application of the RLT hierarchy is to generate relaxations for polynomial programming problems [94], where the various multivariate monomials generated by the RLT hierarchy might already be present in the problem formulation.

4.3.4 Signomial programming relaxations

A signomial programming problem is an optimization problem where every objective function is a signomial function and every constraint is of the form $c = (g, s, 0)$ where g is a signomial function of the problem variables, and $s \neq 0$ (so signomial equality constraints must be reformulated to pairs of inequality constraints as per the Eq2Ineq reformulation of Sect. 3.1.4). A *signomial* is a term of the form:

$$a \prod_{k=1}^{K} x_k^{r_k}, \tag{4.16}$$

where $a, r_k \in \mathbb{R}$ for all $k \in K$, and the r_k exponents are assumed ordered so that $r_k > 0$ for all $k \leq m$ and $r_k < 0$ for $m \leq k \leq K$. Because the exponents of the variables are real constants, this is a generalization of a multivariate monomial term. A *signomial function* is a sum of signomial terms. In [17], a set of transformations of the form $x_k = f_k(z_k)$ are proposed, where x_k is a problem variable, z_k is a variable in the reformulated problem and f_k is suitable function that can be either exponential or power. This yields an opt-reformulation where all the inequality constraints are convex, and the variables z and the associated (inverse) defining constraints $x_k = f_k(z_k)$ are added to the reformulation for all $k \in K$ (over each signomial term of each signomial constraint).

We distinguish the following cases:

- If $a > 0$, the transformation functions f_k are exponential univariate, i.e. $x_k = e^{z_k}$. This reformulates (4.16) as follows:

$$\left.\begin{array}{c} a \frac{e^{\sum_{k \leq m} r_k z_k}}{\prod_{k=m+1}^{K} x_k^{|r_k|}} \\ \forall k \leq K \quad x_k = e^{z_k}. \end{array}\right\}$$

- If $a < 0$, the transformation functions are power univariate, i.e. $x_k = z_k^{\frac{1}{R}}$ for $k \leq m$ and $x_k = z_k^{-\frac{1}{R}}$ for $k > m$, where $R = \sum_{k \leq K} |r_k|$. This is also called a *potential*

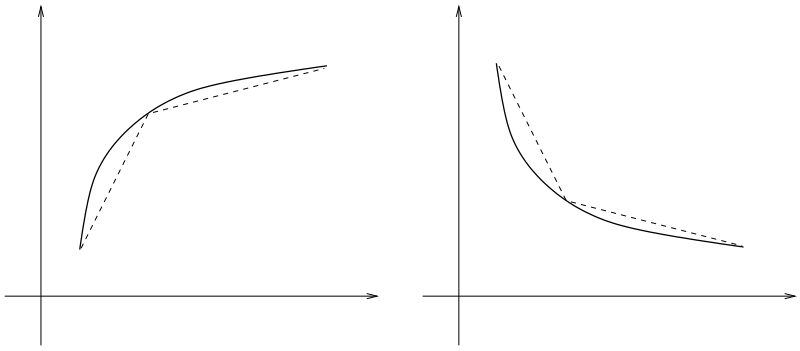

Figure 4.1: Piecewise linear underestimating approximations for concave (left) and convex (right) univariate functions.

transformation. This reformulates (4.16) as follows:

$$\left.\begin{array}{c} a\prod_{k\leq K} z_k^{\frac{|r_k|}{R}} \\ \forall k \leq m \quad x_k = z_k^{\frac{1}{R}} \\ \forall k > m \quad x_k = z_k^{-\frac{1}{R}} \\ R = \sum_{k\leq K} |r_k|. \end{array}\right\}$$

This opt-reformulation isolates all nonconvexities in the inverse defining constraints. These are transformed as follows:

$$\begin{array}{rcl} \forall k \leq K \; x_k = e^{z_k} & \rightarrow & \forall k \leq K \; z_k = \log x_k \\ \forall k \leq m \quad z_k = x_k^R & & \\ \forall k > m \quad z_k = x_k^{-R}, & & \end{array}$$

and then relaxed using a piecewise linear approximation as per Fig. 4.1. This requires the introduction of binary variables (one per turning point).

The signomial relaxation is a convex MINLP; it can be further relaxed to a MILP by outer approximation of the convex terms, or to a convex NLP by continuous relaxation of the discrete variables.

4.4 Valid cuts

Once a relaxation has been derived, it should be strengthened (i.e. it should be modified so that the deriving bound becomes tighter). This is usually done by tightening the relaxation, i.e. by

adding inequalities. These inequalities have the property that they are redundant with respect to the original (or reformulated) problem but they are not redundant with respect to the relaxation. Thus, they tighten the relaxation but do not change the original problem. In this section we discuss such inequalities for MILPs, NLPs and MINLPs.

4.4.1 Definition
Given an optimization problem P and a relaxation Q, a valid inequality is a constraint $c = (e_c, s_c, b_c)$ such that the problem Q' obtained by Q from adding c to $\mathcal{C}(Q)$ has $\mathcal{F}(P) \subseteq \mathcal{F}(Q')$.

Naturally, because Q can be seen as a constraint relaxation of Q', we also have $\mathcal{F}(Q') \subseteq \mathcal{F}(Q)$. Linear valid inequalities are very important as adding a linear inequality to an optimization problem usually does not significantly alter the solution time.

For any problem P and any $c \in \mathcal{C}(P)$, let $\mathcal{F}_c$ be the set of points in $\mathbb{R}^n$ that satisfy c. Let Q be a relaxation of P.

4.4.2 Definition
A linear valid inequality c is a valid cut if there exists $y \in \mathcal{Q}$ such that $y \notin \mathcal{F}_c$.

Valid cuts are linear valid inequalities that "cut away" a part of the feasible region of the relaxation. They are used in two types of algorithms: cutting plane algorithms and Branch-and-Bound algorithms. The typical iteration of a cutting plane algorithm solves a problem relaxation Q (say with solution x'), derives a valid cut that cuts away x'; the cut is then added to the relaxation and the iteration is repeated. Convergence is attained when $x' \in \mathcal{F}(P)$. Cutting plane algorithms were proposed for MILPs [37] but then deemed to be too slow for practical purposes, and replaced by Branch-and-Bound. Cutting plane algorithms were also proposed for convex [50] and bilinear [53] NLPs, and pseudoconvex MINLPs [119, 118].

4.4.1 Valid cuts for MILPs

This is possibly the area of integer programming where the highest number of papers is published annually. It would be outside the scope of this thesis to relate on all valid cuts for MILPs, so we limit this section to a brief summary. The most effective cutting techniques usually rely on problem structure. See [83], Ch. II.2 for a good technical discussion on the most standard techniques, and [78, 79, 49] for recent interesting group-theoretical approaches which are applicable to large subclasses of IPs. Valid inequalities are generated by all relaxation hierarchies (like e.g. Chvátal-Gomory [120] or Sherali-Adams' [98]). The best known general-purpose valid cuts are the Gomory cuts [37], for they are simple to define and can be written in a form

suitable for straightforward insertion in a simplex tableau; many strengthenings of Gomory cuts have been proposed (see e.g. [58]). Lift-and-project techniques are used to generate new cuts from existing inequalities [14]. Families of valid cuts for general Binary Integer Programming (BIP) problems have been derived, for example, in [15, 74], based on geometrical properties of the definition hypercube $\{0, 1\}^n$. In [15], inequalities defining the various faces of the unit hypercube are derived. The cuts proposed in [74] are defined by finding a suitable hyperplane separating a unit hypercube vertex $\bar{x}$ from its adjacent vertices. Intersection cuts [13] are defined as the hyperplane passing through the intersection points between the smallest hypersphere containing the unit hypercube and n half-lines of a cone rooted at the current relaxed solution of Q. Spherical cuts are similar to intersection cuts, but the considered sphere is centered at the current relaxed solution, with radius equal to the distance to the nearest integral point [65]. In [19], Fenchel duality arguments are used to find the maximum distance between the solution of Q and the convex hull of the $\mathcal{F}(P)$; this gives rise to provably deep cuts called *Fenchel cuts*. See [23] for a survey touching on the most important general-purpose MILP cuts, including Gomory cuts, Lift-and-project techniques, Mixed Integer Rounding (MIR) cuts, Intersection cuts and Reduce-and-split cuts.

4.4.2 Valid cuts for NLPs

Valid cuts for NLPs with a single objective function f subject to linear constraints are described in [45] (Ch. III) when an incumbent x^* with $f(x^*) = \gamma$ is known, in order to cut away feasible points x' with $f(x') > \gamma$. Such cuts are called γ-*valid cuts*. Given a nondegenerate vertex x' of the feasible polyhedron for which $f(x') > \gamma$, we consider the n polyhedron edges emanating from x'. For each $i \leq n$ we consider a point x^i on the i-th edge from x' such that $f(x^i) \geq \gamma$. The hyperplane passing through the intersection of the x^i is a γ-valid cut (see Fig. 4.2). More precisely, let Q be the matrix whose i-th column is $x^i - x'$ and e the unit n-vector. Then by [45] Thm. III.1 $eQ^{-1}(x - x') \geq 1$ defines a γ-valid cut. Under some conditions, we can find x^i such that $f(x) = x^i$ and define the strongest possible γ-valid cut, also called *concavity cut*.

The idea for defining γ-valid cuts was first proposed in [113]; this was applied to 0-1 linear programs by means of a simple reformulation in [89]. It is likely that this work influenced the inception of intersection cuts [13] (see Sect. 4.4.1), which was then used as the basis for current work on Reduce-and-Split cuts [8].

Some valid cuts for pseudoconvex optimization problems are proposed in [119]. An optimization problem is pseudoconvex if the objective function is a linear form and the constraints

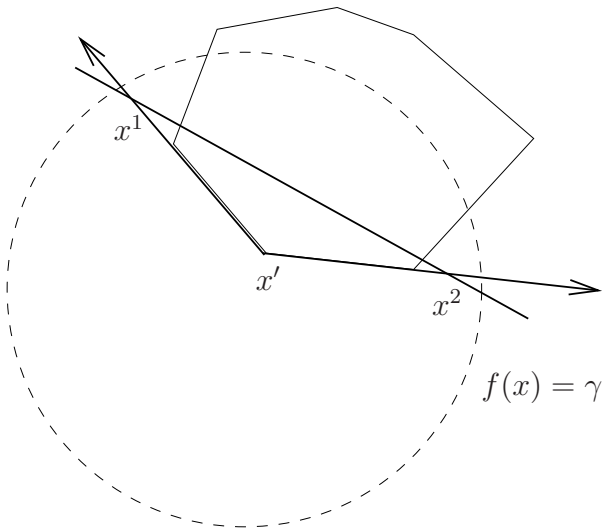

Figure 4.2: A γ-valid cut.

are in the form $c = (g, -1, 0)$ where $g(x)$ is a pseudoconvex function of the problem variable vector x. A function $g : S \subseteq \mathbb{R}^n \to \mathbb{R}$ is *pseudoconvex* if for all $x, y \in S$, $g(x) < g(y)$ implies $\nabla g(y)(x - y) < 0$. So it follows that for each $x, y \in S$ with $g(y) > 0$, there is a constant $\alpha \geq 1$ such that

$$g(y) + \alpha(\nabla g(y))(x - y) \leq g(x) \tag{4.17}$$

is a (linear) outer approximation to the feasible region of the problem. If g is convex, $\alpha = 1$ suffices.

In [84], Ch. 7 presents a non-exhaustive list of NLP cuts, applicable to a MINLP standard form ([84] Eq. (7.1): minimization of a linear objective subject to linear inequality constraints and nonlinear inequality constraints): linearization cuts (outer approximation, see Sect. 4.2.1), knapsack cuts (used for improving loose convex relaxations of given constraints), interval-gradient cuts (a linearization carried out on an interval where the gradient of a given constraint is defined), Lagrangian cuts (derived by solving Lagrangian subproblems), level cuts (defined for a given objective function upper bound), deeper cuts (used to tighten loose Lagrangian relaxation; they involve the solution of separation problems involving several variable blocks).

Another NLP cut based on the Lagrangian relaxation is proposed in [111]: consider a MINLP in the canonical form $\min_{g(x) \leq 0} f(x)$ and let $L(\cdot, \mu) = f(x) + \mu^\top g(x)$ be its Lagrangian relaxation. Let $\underline{f}$ be a lower bound obtained by solving L and $\bar{f}$ be an upper bound computed by evaluating f at a feasible point x'. From $\underline{f} \leq f(x) + \mu^\top g(x) \leq \bar{f} + \mu^\top g(x)$ one derives the valid cut $g_i(x) \geq -\frac{1}{\mu_i}(\bar{f} - \underline{f})$ for all $i \leq m$ (where $g : \mathbb{R}^n \to \mathbb{R}^m$).

4.4.3 Valid cuts for MINLPs

Naturally, both MILP and NLP cuts may apply to MINLPs. Some more specific MINLP cuts can be derived by reformulating integer variables to binary (see Sect. 3.1.9) and successively to continuous (see Sect. 3.1.8). The added quadratic constraints may then be relaxed in a Lagrangian (see Sect. 4.3.1) or SDP fashion (see Sect. 4.3.2) [87]: any of the NLP cuts described in Sect. 4.4.2 applied to such a reformulation is essentially a specific MINLP valid cut.

Chapter 5

Conclusion

This thesis is a study of mathematical programming reformulation and relaxation techniques. The introductory chapter presents some motivations towards such a study, the principle of which being that Mixed Integer Nonlinear Programming solvers need to be endowed with automatic reformulation capabilities before they can be as reliable, functional and efficient as their industrial-strength Mixed Integer Linear Programming solvers have been. The second chapter presents a general framework for representing and manipulating mathematical programming formulations, as well as some definitions of the concept of reformulation together with some theoretical results; the chapter is concluded by listing some of the most common standard forms in mathematical programming. In the third chapter we present a partial systematic study of existing reformulations. Each reformulation is presented both in symbolic algorithmic terms (i.e. a prototype for carrying out the reformulation automatically in terms of the provided data structures is always supplied) and in the more usual mathematical terms. This should be seen as the starting point for an exhaustive such study: eventually, all useful reformulations might find their place in an automatic reformulation preprocessing software for Mixed Integer Nonlinear Programming. In chapter four, we attempt a similar work with respect to relaxations, although this seems to be an even larger task, for there are in general more ways to relax a mathematical programming problem rather than to reformulate it; again, it should be seen as the paving stones for an exhaustive work that is still to come.

Thus, the future work stemming from this thesis will focus on completing the existing reviews in reformulation and relaxation techniques, and on producing new and useful reformulation methods for mathematical programming. Hopefully, we shall one day also produce the accompanying software.

Bibliography

[1] W.P. Adams, R.J. Forrester, and F.W. Glover. Comparisons and enhancement strategies for linearizing mixed 0-1 quadratic programs. *Discrete Optimization*, 1:99–120, 2004.

[2] W.P. Adams and H.D. Sherali. A hierarchy of relaxations leading to the convex hull representation for general discrete optimization problems. *Annals of Operations Research*, 140:21–47, 2005.

[3] C. S. Adjiman, I. P. Androulakis, and C. A. Floudas. A global optimization method, αBB, for general twice-differentiable constrained NLPs: II. Implementation and computational results. *Computers & Chemical Engineering*, 22(9):1159–1179, 1998.

[4] C.S. Adjiman, S. Dallwig, C.A. Floudas, and A. Neumaier. A global optimization method, αBB, for general twice-differentiable constrained NLPs: I. Theoretical advances. *Computers & Chemical Engineering*, 22(9):1137–1158, 1998.

[5] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, MA, 1983.

[6] F.A. Al-Khayyal and J.E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8(2):273–286, 1983.

[7] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.

[8] K. Andersen, G. Cornuéjols, and Y. Li. Reduce-and-split cuts: Improving the performance of mixed-integer gomory cuts. *Management Science*, 51(11):1720–1732, 2005.

[9] I. P. Androulakis, C. D. Maranas, and C. A. Floudas. αBB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, December 1995.

[10] K. Anstreicher. Sdp versus rlt for nonconvex qcqps. In C.A. Floudas and P. Pardalos, editors, *Proceedings of Advances in Global Optimization: Methods and Applications*, Mykonos, Greece, June 2007.

[11] D.L. Applegate, R. Bixby, V. Chvátal, and W.J. Cook. *The Travelling Salesman Problem: a Computational Study*. Princeton University Press, Princeton, 2007.

[12] C. Audet, P. Hansen, B. Jaumard, and G. Savard. Links between linear bilevel and mixed 0-1 programming problems. *Journal of Optimization Theory and Applications*, 93(2):273–300, 1997.

[13] E. Balas. Intersection cuts — a new type of cutting planes for integer programming. *Operations Research*, 19(1):19–39, 1971.

[14] E. Balas. Projection, lifting and extended formulation in integer and combinatorial optimization. *Annals of Operations Research*, 140:125–161, 2005.

[15] E. Balas and R. Jeroslow. Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23(1):61–69, 1972.

[16] A. Billionnet, S. Elloumi, and M.-C. Plateau. Improving standard solvers convex reformulation for constrained quadratic 0-1 programs: the QCR method. *Discrete Applied Mathematics*, to appear.

[17] K.-M. Björk, P.O. Lindberg, and T. Westerlund. Some convexifications in global optimization of problems containing signomial terms. *Computers & Chemical Engineering*, 27:669–679, 2003.

[18] J. Bjorkqvist and T. Westerlund. Automated reformulation of disjunctive constraints in MINLP optimization. *Computers & Chemical Engineering*, 23:S11–S14, June 1999.

[19] E.A. Boyd. Fenchel cutting planes for integer programs. *Operations Research*, 42(1):53–64, 1994.

[20] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, 2004.

[21] A. Brook, D. Kendrick, and A. Meeraus. Gams, a user's guide. *ACM SIGNUM Newsletter*, 23(3-4):10–11, 1988.

[22] G. Caporossi, D. Alamargot, and D. Chesnet. Using the computer to study the dyamics of the handwriting processes. In *DS 2004 Proceedings*, volume 3245 of *LNAI*, pages 242–254. Springer-Verlag, 2004.

[23] G. Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming B*, 112(1):3–44, 2008.

[24] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[25] T. Davidović, L. Liberti, N. Maculan, and N. Mladenović. Towards the optimal solution of the multiprocessor scheduling problem with communication delays. In *MISTA Proceedings*, 2007.

[26] K. Dhyani. Personal communication. 2007.

[27] M. Duran and I. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.

[28] J.E. Falk and J. Liu. On bilevel programming, part I: General nonlinear cases. *Mathematical Programming*, 70:47–72, 1995.

[29] J.E. Falk and R.M. Soland. An algorithm for separable nonconvex programming problems. *Management Science*, 15:550–569, 1969.

[30] A. Fischer. New constrained optimization reformulation of complementarity problems. *Journal of Optimization Theory and Applications*, 99(2):481–507, November 1998.

[31] R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994.

[32] C.A. Floudas. *Deterministic Global Optimization*. Kluwer Academic Publishers, Dordrecht, 2000.

[33] R. Fortet. Applications de l'algèbre de boole en recherche opérationelle. *Revue Française de Recherche Opérationelle*, 4:17–26, 1960.

[34] R. Fourer and D. Gay. *The AMPL Book*. Duxbury Press, Pacific Grove, 2002.

[35] L. Di Giacomo. *Mathematical programming methods in dynamical nonlinear stochastic Supply Chain management*. PhD thesis, DSPSA, Università di Roma "La Sapienza", 2007.

[36] P.E. Gill. *User's Guide for SNOPT 5.3*. Systems Optimization Laboratory, Department of EESOR, Stanford University, California, February 1999.

[37] R.E. Gomory. Essentials of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):256, 1958.

[38] M. Grant, S. Boyd, and Y. Ye. Disciplined convex programming. In Liberti and Maculan [70], pages 155–210.

[39] S. Gueye and P. Michelon. "Miniaturized" linearizations for quadratic 0/1 problems. *Annals of Operations Research*, 140:235–261, 2005.

[40] P.L. Hammer and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas*. Springer, Berlin, 1968.

[41] P. Hansen. Method of non-linear 0-1 programming. *Annals of Discrete Mathematics*, 5:53–70, 1979.

[42] P. Hansen and C. Meyer. Improved compact linearizations for the unconstrained quadratic 0-1 minimization problem. *Discrete Applied Mathematics*, to appear.

[43] C.A. Haverly. Studies of the behaviour of recursion for the pooling problem. *ACM SIGMAP Bulletin*, 25:19–28, 1978.

[44] R. Horst. On the convexification of nonlinear programming problems: an applications-oriented approach. *European Journal of Operations Research*, 15:382–392, 1984.

[45] R. Horst and Hoang Tuy. *Global Optimization: Deterministic Approaches*. Springer-Verlag, Berlin, third edition, 1996.

[46] R. Horst and N. Van Thoai. Duality bound methods in global optimization. In C. Audet, P. Hansen, and G. Savard, editors, *Essays and Surveys in Global Optimization*, pages 79–105. Springer, Berlin, 2005.

[47] ILOG. *ILOG CPLEX 8.0 User's Manual*. ILOG S.A., Gentilly, France, 2002.

[48] J. Judice and G. Mitra. Reformulation of mathematical programming problems as linear complementarity problems and investigation of their solution methods. *Journal of Optimization Theory and Applications*, 57(1):123–149, 1988.

[49] V. Kaibel and M. Pfetsch. Packing and partitioning orbitopes. *Mathematical Programming*, accepted for publication.

[50] J. Kelley. The cutting plane method for solving convex programs. *Journal of SIAM*, VIII(6):703–712, 1960.

[51] P. Kesavan, R.I. Allgor, E.P. Gatzke, and P.I. Barton. Outer-approximation algorithms for nonconvex mixed-integer nonlinear programs. *Mathematical Programming*, 100(3):517–535, 2004.

[52] M. Kojima, N. Megiddo, and Y. Ye. An interior point potential reduction algorithm for the linear complementarity problem. *Mathematical Programming*, 54:267–279, 1992.

[53] H. Konno. A cutting plane algorithm for solving bilinear programs. *Mathematical Programming*, 11:14–27, 1976.

[54] S. Kucherenko, P. Belotti, L. Liberti, and N. Maculan. New formulations for the kissing number problem. *Discrete Applied Mathematics*, 155(14):1837–1841, 2007.

[55] K. Kunen. *Set Theory. An Introduction to Independence Proofs*. North Holland, Amsterdam, 1980.

[56] C. Lavor, L. Liberti, and N. Maculan. Computational experience with the molecular distance geometry problem. In J. Pintér, editor, *Global Optimization: Scientific and Engineering Case Studies*, pages 213–225. Springer, Berlin, 2006.

[57] C. Lavor, L. Liberti, N. Maculan, and M.A. Chaer Nascimento. Solving hartree-fock systems with global optimization metohds. *Europhysics Letters*, 5(77):50006p1–50006p5, 2007.

[58] A. Letchford and A. Lodi. Strengthening chvátal-gomory cuts and gomory fractional cuts. *Operations Research Letters*, 30:74–82, 2002.

[59] L. Liberti. Comparison of convex relaxations for monomials of odd degree. In I. Tseveendorj, P.M. Pardalos, and R. Enkhbat, editors, *Optimization and Optimal Control*. World Scientific, 2003.

[60] L. Liberti. Reduction constraints for the global optimization of NLPs. *International Transactions in Operations Research*, 11(1):34–41, 2004.

[61] L. Liberti. Reformulation and convex relaxation techniques for global optimization. *4OR*, 2:255–258, 2004.

[62] L. Liberti. *Reformulation and Convex Relaxation Techniques for Global Optimization*. PhD thesis, Imperial College London, UK, March 2004.

[63] L. Liberti. Linearity embedded in nonconvex programs. *Journal of Global Optimization*, 33(2):157–196, 2005.

[64] L. Liberti. Writing global optimization software. In Liberti and Maculan [70], pages 211–262.

[65] L. Liberti. Spherical cuts for integer programming problems. *International Transactions in Operations Research*, accepted for publication.

[66] L. Liberti. Compact linearization of binary quadratic problems. *4OR*, 5(3):231-245, 2007.

[67] L. Liberti, C. Lavor, and N. Maculan. Double VNS for the molecular distance geometry problem. In *Proc. of Mini Euro Conference on Variable Neighbourhood Search, Tenerife, Spain*, 2005.

[68] L. Liberti, C. Lavor, N. Maculan and F. Marinelli. Double Variable Neighbourhood Search with smoothing for the Molecular Distance Geometry Problem. *Journal of Global Optimization*, accepted for publication.

[69] L. Liberti, C. Lavor, M.A. Chaer Nascimento, and N. Maculan. Reformulation in mathematical programming: an application to quantum chemistry. *Discrete Applied Mathematics*, accepted for publication.

[70] L. Liberti and N. Maculan, editors. *Global Optimization: from Theory to Implementation*. Springer, Berlin, 2006.

[71] L. Liberti and C.C. Pantelides. Convex envelopes of monomials of odd degree. *Journal of Global Optimization*, 25:157–168, 2003.

[72] L. Liberti and C.C. Pantelides. An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms. *Journal of Global Optimization*, 36:161–189, 2006.

[73] L. Liberti, P. Tsiakis, B. Keeping, and C.C. Pantelides. $oo\mathcal{OPS}$. Centre for Process Systems Engineering, Chemical Engineering Department, Imperial College, London, UK, 2001.

[74] N. Maculan, E.M. Macambira, and C.C. de Souza. Geometrical cuts for 0-1 integer programming. Technical Report IC-02-006, Instituto de Computação, Universidade Estadual de Campinas, July 2002.

[75] O.L. Mangasarian. Linear complementarity problems solvable by a single linear program. *Mathematical Programming*, 10:263–270, 1976.

[76] O.L. Mangasarian. The linear complementarity problem as a separable bilinear program. *Journal of Global Optimization*, 6:153–161, 1995.

[77] C. D. Maranas and C. A. Floudas. Finding all solutions to nonlinearly constrained systems of equations. *Journal of Global Optimization*, 7(2):143–182, 1995.

[78] F. Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94:71–90, 2002.

[79] F. Margot. Exploiting orbits in symmetric ILP. *Mathematical Programming B*, 98:3–21, 2003.

[80] G.P. McCormick. Computability of global solutions to factorable nonconvex programs: Part i — convex underestimating problems. *Mathematical Programming*, 10:146–175, 1976.

[81] C.A. Meyer and C.A. Floudas. Convex hull of trilinear monomials with mixed sign domains. *Journal of Global Optimization*, 29:125–155, 2004.

[82] N. Mladenović, F. Plastria, and D. Urošević. Reformulation descent applied to circle packing problems. *Computers and Operations Research*, 32(9):2419–2434, 2005.

[83] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, New York, 1988.

[84] I. Nowak. *Relaxation and Decomposition Methods for Mixed Integer Nonlinear Programming*. Birkhäuser, Basel, 2005.

[85] C.C. Pantelides, L. Liberti, P. Tsiakis, and T. Crombie. Mixed integer linear/nonlinear programming interface specification. *Global Cape-Open Deliverable WP2.3-04*, February 2002.

[86] P.M. Pardalos and H.E. Romeijn, editors. *Handbook of Global Optimization*, volume 2. Kluwer Academic Publishers, Dordrecht, 2002.

[87] M.-C. Plateau. *Reformulations quadratiques convexes pour la programmation quadratique en variables 0-1*. PhD thesis, Conservatoire National d'Arts et Métiers, 2006.

[88] J. Puchinger and G.R. Raidl. Relaxation guided variable neighbourhood search. In *Proc. of Mini Euro Conference on Variable Neighbourhood Search, Tenerife, Spain*, 2005.

[89] M. Raghavachari. On connections between zero-one integer programming and concave programming under linear constraints. *Operations Research*, 17(4):680–684, 1969.

[90] H.S. Ryoo and N.V. Sahinidis. Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Computers & Chemical Engineering*, 19(5):551–566, 1995.

[91] H. Sherali. Personal communication. June 2007.

[92] H. Sherali and L. Liberti. Reformulation-linearization methods for global optimization. In C. Floudas and P. Pardalos, editors, *Encyclopoedia of Optimization*. Springer, New York, to appear.

[93] H. Sherali and C.H. Tuncbilek. New reformulation linearization/convexification relaxations for univariate and multivariate polynomial programming problems. *Operations Research Letters*, 21:1–9, 1997.

[94] H.D. Sherali. Global optimization of nonconvex polynomial programming problems having rational exponents. *Journal of Global Optimization*, 12:267–283, 1998.

[95] H.D. Sherali. Tight relaxations for nonconvex optimization problems using the reformulation-linearization/convexification technique (rlt). In Pardalos and Romeijn [86], pages 1–63.

[96] H.D. Sherali and W.P. Adams. A tight linearization and an algorithm for 0-1 quadratic programming problems. *Management Science*, 32(10):1274–1290, 1986.

[97] H.D. Sherali and W.P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal of Discrete Mathematics*, 3:411–430, 1990.

[98] H.D. Sherali and W.P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Kluwer Academic Publishers, Dodrecht, 1999.

[99] H.D. Sherali and A. Alameddine. A new reformulation-linearization technique for bilinear programming problems. *Journal of Global Optimization*, 2:379–410, 1992.

[100] H.D. Sherali and H. Wang. Global optimization of nonconvex factorable programming problems. *Mathematical Programming*, 89:459–478, 2001.

[101] E.M.B. Smith. *On the Optimal Design of Continuous Processes*. PhD thesis, Imperial College of Science, Technology and Medicine, University of London, October 1996.

[102] E.M.B. Smith and C.C. Pantelides. Global optimisation of nonconvex MINLPs. *Computers & Chemical Engineering*, 21:S791–S796, 1997.

[103] E.M.B. Smith and C.C. Pantelides. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. *Computers & Chemical Engineering*, 23:457–478, 1999.

[104] A.S. Strekalovsky. On global optimality conditions for d.c. programming problems. *Technical Paper, Irkutsk State University*, 1997.

[105] A.S. Strekalovsky. Extremal problems with d.c. constraints. *Computational Mathematics and Mathematical Physics*, 41(12):1742–1751, 2001.

[106] F. Tardella. Existence and sum decomposition of vertex polyhedral convex envelopes. Technical report, Facoltà di Economia e Commercio, Università di Roma “La Sapienza”, 2007.

[107] M. Tawarmalani, S. Ahmed, and N.V. Sahinidis. Global optimization of 0-1 hyperbolic programs. *Journal of Global Optimization*, 24:385–416, 2002.

[108] M. Tawarmalani and N. Sahinidis. Convex extensions and envelopes of semi-continuous functions. *Mathematical Programming*, 93(2):247–263, 2002.

[109] M. Tawarmalani and N.V. Sahinidis. Semidefinite relaxations of fractional programming via novel techniques for constructing convex envelopes of nonlinear functions. *Journal of Global Optimization*, 20(2):137–158, 2001.

[110] M. Tawarmalani and N.V. Sahinidis. Exact algorithms for global optimization of mixed-integer nonlinear programs. In Pardalos and Romeijn [86], pages 65–86.

[111] M. Tawarmalani and N.V. Sahinidis. Global optimization of mixed integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99:563–591, 2004.

[112] M.J. Todd. Semidefinite optimization. *Acta Numerica*, 10:515–560, 2001.

[113] H. Tuy. Concave programming under linear constraints. *Soviet Mathematics*, pages 1437–1440, 1964.

[114] H. Tuy. D.c. optimization: Theory, methods and algorithms. In R. Horst and P.M. Pardalos, editors, *Handbook of Global Optimization*, volume 1, pages 149–216. Kluwer Academic Publishers, Dordrecht, 1995.

[115] P. Potena V. Cortellessa, F. Marinelli. Automated selection of software components based on cost/reliability tradeoff. In V. Gruhn and F. Oquendo, editors, *EWSA 2006*, volume 4344 of *LNCS*, pages 66–81. Springer-Verlag, 2006.

[116] T.J. van Roy and L.A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35(1):45–57, 1987.

[117] X. Wang and T.S. Change. A multivariate global optimization using linear bounding functions. *Journal of Global Optimization*, 12:383–404, 1998.

[118] T. Westerlund. Some transformation techniques in global optimization. In Liberti and Maculan [70], pages 45–74.

[119] T. Westerlund, H. Skrifvars, I. Harjunkoski, and R. Pörn. An extended cutting plane method for a class of non-convex MINLP problems. *Computers & Chemical Engineering*, 22(3):357–365, 1998.

[120] L.A. Wolsey. *Integer Programming*. Wiley, New York, 1998.

[121] J. M. Zamora and I. E. Grossmann. A branch and contract algorithm for problems with concave univariate, bilinear and linear fractional terms. *Journal of Global Optimization*, 14:217:249, 1999.

Index

α-underestimator, 61
αBB algorithm, 61

AbsDiff, 40
absolute value
    minimization, 46
affine, 67
algorithm, 8
    efficient, 60
    generic, 8
    heuristic, 58
    NLP, 10
    symbolic, 9
alphabet, 15
AMPL, 11
AND, 45
approximation, 58
    outer, 60
    polyhedral, 60
architecture, 53
assignment, 50
    constraint, 42
    variable, 42
    variables, 51

BARON, 63
BB, 34
bijection, 25
bilevel problem, 34
bilevel programming, 34
bilinear, 26
Bin2Cont, 41
binary
    product, 44
    representatation, 41
BinaryRep, 42
BIP, 72
bisection, 9
blending problem, 65
block-separable, 64
BLP, 34
bound, 58
    lower, 64
bound factor, 61, 66
boundary point, 60
bounds, 15
branch-and-bound, 11, 30, 34, 58, 63, 71
    pruning, 26
    spatial, 62, 68
branch-and-contract, 62

cardinality, 8
CAS, 11
CCLin, 46
clustering, 16, 50
complementarity
    constraints, 45
complexity theory, 24
complicating constraints, 64
ConSense, 37
constants, 41

constraint, 2, 14, 15, 58
    assignment, 51, 54
    binary, 41
    complicating, 64
    defining, 62, 67, 68
    equality, 38, 39
    inequality, 38, 39
    nonconvex, 41
    nonlinear, 41
    ordering, 10, 41
    quadratic, 35, 41
    qualification, 64
    relaxation, 67, 71
    reliability, 55
    sense, 37
    soft, 10
    valid, 66
constraint factor, 66
constraint relaxation, 43
constraint tolerance, 43
convex
    envelope, 26
    function, 26
    hull, 26
convex envelope, 61, 66, 68
coordinates
    Euclidean, 10
    spherical, 10
cost, 54
CPLEX, 11, 53
cut
    deeper, 73
    Gomory, 71
    intersection, 72
    interval-gradient, 73
    knapsack, 73
    Lagrangian, 73
    level, 73
    linearization, 73
    NLP, 73
    spherical, 72
    valid, 72
cuts, 26
    MIR, 72
    Reduce-and-split, 72
cutset, 23
cutting plane, 71
    convergence, 71

d.c.
    function, 32
    problem, 32
    set, 32
DAG, 15, 20
data structure, 65
decidable, 8
decision problem, 23
decomposition, 9
dense, 33
differentiable, 52
distance
    minimum, 50
distance geometry, 65
distribute
    products over sums, 56
domain
    partition, 58
duality
    Fenchel, 72
duality gap, 35

efficiency, 8
eigenvalue

minimum, 62
enumeration
complete, 8
implicit, 58
envelope
convex, 62
polyhedral, 62
equation
univariate, 41
evaluation, 20
exponent, 22, 42
exponential, 69
expression, 14, 15
nonlinear, 41
standard form, 21
tree, 15, 20
expression tree, 14

factor
bound, 66
constraint, 66
factorable, 32
Feas2Opt, 43
feasible, 23
bounds, 23
constraints, 23
point, 23, 33
region, 71
type, 22
feasible region
convex, 60
follower, 34
form
factorable, 32
separable, 31
standard, 29, 60
formulation, 9, 20, 29
mathematical programming, 14
fractional, 26
function
continuous, 64
exponential, 69
power, 69
pseudoconvex, 73
signomial, 69
transformation, 69
univariate, 61

Gödel, 8
GAMS, 63
global
optimality condition, 33
optimum, 26
GO, 41
Gomory cuts, 71
graph
bisection, 16
directed acyclic, 15
partitioning, 16
graph bisection, 65
graph partitioning, 65
group
cuts, 71

half-line, 72
HCP, 50
Hessian, 62
heuristic, 58
hypercube
unit, 72
hyperplane, 50, 72
Hyperplane Clustering Problem, 50
hypersphere, 72

identity, 39, 43
ILOG, 53
incumbent, 58
inequality
    linear, 71
    polynomial, 68
infeasible, 23
input, 8
Int2Bin, 42
interior point method, 35
intersection cuts, 72
IP, 71
IPM, 35
iteration, 71

kissing number, 9, 18
Kissing Number Problem, 9, 65
KNP, 9, 19, 65

Lagrangian decomposition, 64
Lagrangian problem, 64
Lagrangian relaxation, 64
language, 24
LCP, 33
leader, 34
leaf, 20
leaf node, 42
Lift, 37
lift-and-project, 66, 72
lifting, 9, 23, 37, 39, 67
linear
    equation, 8
    form, 8, 22
    mixed integer, 30
    programming, 30
linear complementarity, 33
linear fractional, 47
linear programming, 8
linearization, 44
LinFract, 47
logarithm, 40, 56
LP, 8, 30, 39, 60, 64
    standard form, 30

mathematical programming, 24, 42
matrix, 14, 20
    symmetric, 65
McCormick envelopes, 61
MDGP, 65
MILP, 11, 30, 34, 50, 51, 53, 56, 58–60, 70
    solver, 53
    standard form, 30
MinAbs, 46
minimality
    local, 43
MINLP, 31, 50, 51, 55, 58–60, 63–65
    convex, 70
    quadratic, 65
    solver, 53
    standard form, 73
MinMax, 44
MIR cuts, 72
molecular distance geometry, 65
monomial, 68
    multivariate, 69
    odd degree, 26
monotonically increasing, 56
multiplier, 21

NLP, 10, 30, 41, 59, 60, 64
    convex, 35
    nonconvex, 58
    standard form, 31
nonconvex, 9, 35, 59, 65

nonlinear
    mixed integer, 31
    programming, 30
NP-completeness, 24
NP-hard, 34

OA, 60
ObjDir, 37
objective, 2
    linear, 72
    quadratic, 35
objective function, 14, 15, 23
operator, 15, 20, 21
    argument, 20
    max, 65
    power, 22
    product, 22
    sum, 21
opt-reformulation, 26, 43, 69
optimality, 64
    condition, 33
    global, 23
    guarantee, 53
    local, 24, 25
optimization
    direction, 15, 36
    global, 41
optimization problem, 23
optimum
    global, 23, 43, 51
    local, 23, 43
outer approximation, 60, 63
output, 8
overestimator
    concave, 62

parameter, 2, 14, 15
partition
    balanced, 23
piecewise
    linear, 44
point, 22
polynomial, 22
polynomial time, 24
pooling problem, 65
potential transformation, 70
power, 22, 69
problem
    auxiliary, 2, 9, 29, 58, 67
    bilevel, 34
    binary, 41
    blending, 63, 65
    decision, 2, 8, 9, 23, 42
    distance geometry, 63
    feasibility, 10, 43
    follower, 34
    graph partitioning, 65
    kissing number, 18
    Lagrangian, 64, 65
    leader, 34
    molecular distance geometry, 65
    multilevel, 15, 20
    optimization, 2, 8, 23, 42
    polynomial, 69
    pooling, 63, 65
    pseudoconvex, 72
    quadratic, 41
    quantum chemistry, 63
    reformulated, 69
    scheduling, 65
    SDP, 35
    semidefinite, 15, 35
    signomial, 69

structure, 29, 71
Prod, 45
ProdBinCont, 45
ProdExp, 40
product, 22
binary, 56
binary and continuous, 45
binary variables, 44
exponential, 40
quadratic, 67
programming
mathematical, 9
mixed-integer linear, 11
nonlinear, 10
Proj, 38
projection, 9, 37, 38, 43, 68
pseudoconvex, 71, 72
function, 73
problem, 72

QCQP, 35, 65
quadratic, 35
quadratically constrained, 35
MINLP, 65

range reduction, 63
recursion, 58
Reduce-and-split cuts, 72
redundant inequality, 71
reformulation, 12, 23, 24
automatic, 57
cardinality, 57
continuous, 27
definition, 23
global, 25, 64
local, 25
product, 56
step, 66
symbolic, 62
relaxation, 9, 58, 59
bound, 59
constraint, 59, 67
continuous, 59, 64
convex, 59, 63, 64
elementary, 59
hierarchy, 68, 71
Lagrangian, 63, 64
linear, 11
LP, 63
McCormick, 68
RLT, 63
scheme, 60
semidefinite, 63
signomial, 63
tightness, 64
reliability, 54
Restrict, 38
restriction, 37
reverse convex, 10
RLT, 66, 68
hierarchy, 69
second-level, 61
robustness
numerical, 53

sBB, 60, 62, 68
scalability, 53
scheduling
communication delays, 65
SDP, 35, 65
dual, 35
primal, 35
semantic interpretation, 57

separable, 31
sequential quadratic prorgramming, 30
signomial, 69
signomial function, 69
simplex
    tableau, 72
simplex method, 30, 39
Slack, 39
SNOPT, 10
software
    architecture, 54
    component, 53
    system, 53
solution
    basic, 72
    feasible, 41
    optimal feasible, 29
    starting, 10
    symmetry, 26
    time, 71
solver, 53
    NLP, 10
sphere, 9
spherical cuts, 72
SQP, 30
stability
    numerical, 53
standard form, 29
    expression, 21
    LP, 30
    MILP, 30
    MINLP, 73
    NLP, 31
    Smith, 62, 63, 67, 68
subdomain, 58
sum, 21
symmetry, 10
    breaking, 41
    group, 26
system
    software, 53

tangent, 60
term
    bilinear, 61, 62, 68
    fractional, 61, 63
    linear fractional, 47
    monomial, 69
    nonconvex, 61, 64
    nonlinear, 40
    piecewise convex/concave, 62
    quadratic, 57, 66
    signomial, 69
    trilinear, 61
    univariate, 62
    univariate concave, 61
transformation
    potential, 70
    symbolic, 2
tree
    BB, 34
trilinear, 26
twice-differentiable, 61

underestimator
    convex, 62
    secant, 61
uniform convergence, 33

valid cut, 71
    MILP, 71
valid inequality, 71
    linear, 71

variable, 2, 14, 15
    added, 68
    assignment, 41, 56
    binary, 15, 40, 41
    bounds, 15
    continuous, 15, 30, 60
    discrete, 60
    domain, 58
    general integer, 41
    integer, 15, 56, 60
    linearization, 57
    transformation, 23

Zermelo-Fränkel, 8
ZFC, 8