DOCTOR OF SCIENCES

A global optimization method for mixed integer nonlinear nonconvex problems related to power systems analysis

Wanufelle, Emilie

Award date:
2007

Awarding institution:
University of Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX NAMUR

FACULTE DES SCIENCES

DEPARTEMENT DE MATHEMATIQUE

A global optimization method for mixed integer nonlinear nonconvex problems related to power systems analysis

Dissertation présentée par
Emilie Wanufelle
pour l'obtention du grade
de Docteur en Sciences

Composition du Jury:

Sven LEYFFER
Christian MERCKX
Annick SARTENAER (Promoteur)
Jean-Jacques STRODIOT
Philippe TOINT (Copromoteur)

2007

Rempart de la Vierge, 13
B-5000 Namur (Belgique)

Imprimé en Belgique

ISBN : 978-2-87037-576-1
Dépôt légal: D / 2007 / 1881 / 36

Facultés Universitaires Notre-Dame de la Paix
Faculté des Sciences
rue de Bruxelles, 61, B-5000 Namur, Belgium

Une méthode d'optimisation globale pour problèmes non linéaires et non convexes avec variables mixtes (entières et continues) issus de l'analyse des réseaux électriques

par Emilie Wanufelle

Résumé: Ce travail a pour objet la conception et l'implémentation d'une méthode d'optimisation globale pour la résolution de problèmes non linéaires et non convexes, continus ou avec variables mixtes (entières et continues), issus de l'analyse des réseaux électriques. La méthode proposée relâche le problème traité en un problème d'approximation externe linéaire en se basant sur le concept d'ensembles spécialement ordonnés. Le problème obtenu est alors successivement raffiné grâce à une stratégie de branch-and-bound. La convergence vers un optimum global est ainsi assurée, pour autant que les variables discrètes ou apparaissant non linéairement dans le problème de départ soient bornées. Notre méthode, mise au point pour résoudre un type de problème bien particulier, a été conçue dans un cadre général permettant une extension aisée à la résolution d'une grande variété de problèmes. Nous développons tout d'abord la méthode théoriquement et présentons ensuite des résultats numériques dont le but est de fixer certains choix inhérents à la méthode afin de la rendre la plus optimale possible.

A global optimization method for mixed integer nonlinear nonconvex problems related to power systems analysis

by Emilie Wanufelle

Abstract: This work is concerned with the development and the implementation of a global optimization method for solving nonlinear nonconvex problems with continuous or mixed integer variables, related to power systems analysis. The proposed method relaxes the problem under study into a linear outer approximation problem by using the concept of special ordered sets. The obtained problem is then successively refined by a branch-and-bound strategy. In this way, the convergence to a global optimum is guaranteed, provided the discrete variables or those appearing nonlinearly in the original problem are bounded. Our method, conceived to solve a specific kind of problem, has been developed in a general framework in such a way that it can be easily extended to solve a large class of problems. We first derive the method theoretically and next present numerical results, fixing some choices inherent to the method to make it as optimal as possible.

Dissertation doctorale en Sciences mathématiques (Ph.D. thesis in Mathematics)
Date: 06-12-2007
Département de Mathématique
Promoteur (Advisor): Prof. A. SARTENAER
Copromoteur (Coadvisor): Prof. Ph.L. TOINT

Remerciements

En premier lieu, je tiens à exprimer ma gratitude envers ma promotrice, Annick Sartenaer, pour son soutien et la confiance qu'elle m'a toujours accordée, tout d'abord en me proposant ce doctorat et ensuite, tout au long de ces quatre années de recherches. Je la remercie, ainsi que Philippe Toint, de m'avoir encouragée à participer à des conférences internationales et à effectuer un séjour de recherches à l'étranger. Ces voyages se sont toujours révélés être des expériences enrichissantes dont je garde des souvenirs exceptionnels. J'en profite pour remercier tous les chercheurs que j'ai côtoyés durant ces voyages pour leur accueil, leur sympathie et les discussions intéressantes que nous avons pu avoir.

Je souhaite aussi adresser un immense merci à Sven Leyffer pour son implication, son enthousiasme et ses remarques avisées vis-à-vis du travail de recherches présenté ici, mais aussi pour m'avoir offert l'opportunité d'aller travailler trois semaines à l'Argonne National Laboratory. Les mots me manquent pour lui exprimer ma reconnaissance, ainsi qu'à son épouse Gwen, pour le chaleureux accueil qu'ils m'ont réservé à Chicago. Merci de tout coeur pour ces souvenirs!

Merci à Christian Merckx, Ludovic Platbrood et Karim Karoui de Tractebel pour m'avoir fourni un problème intéressant à traiter ainsi que pour les discussions constructives s'y rapportant.

Merci aussi aux membres du jury de cette thèse d'avoir accepté cette tâche et pour les remarques et suggestions pertinentes qui ont permis d'améliorer ce manuscrit.

Merci au PAI (Pôle d'Attraction Interuniversitaire), et notamment à François Maniquet et Sébastien Laurent, d'avoir financé en grande partie ces années de recherches. Merci aussi à Pierrette Noël pour son support logistique.

Merci à tous les membres du département de mathématiques pour la formation qu'ils m'ont donnée lorsque j'étais étudiante ainsi que pour leur convivialité.

Merci à Michel Goffin, mon professeur de mathématiques du secondaire pour son enseignement haut en couleurs et sans qui, je ne me serais sans doute pas orientée vers les mathématiques.

Mon merci suivant s'adresse à tous mes collègues, anciens ou actuels, avec qui j'ai partagé de bons moments qui ne se cantonnent pas aux frontières de l'université. Sans eux, ces quatre années auraient été sans nul doute fort différentes. Pour ces agréables souvenirs, et pour leur bonne humeur et leur amitié, je voudrais remercier les "copains du midi", en particulier: Charlotte Beauthier, Katia Demaseure, Florent Deleflie, Anne-Sophie Libert, Benoît Noyelles, Caroline Sainvitu, Dimitri Tomanos, Stéphane Valk, Mélissa Weber Mendonça et Sebastian Xhonneux. J'adresse un merci tout particulier à ma collègue de bureau et à ma "soeur jumelle" pour m'avoir patiemment écoutée et soutenue pendant ma période de rédaction. Merci aussi à Benoît Colson pour m'avoir épaulée lors de mon arrivée au département ainsi que pour tous les

bons moments passés quand nous partagions le même bureau.

Je remercie également mes amis de m'avoir soutenue et encouragée. Les instants de détente passés en leur compagnie m'ont permis de rester zen et de relativiser.

Je voudrais aussi remercier mes parents, ma soeur et mes grands-parents pour leur soutien sans faille durant cette thèse de doctorat, et plus largement au cours de ma vie.

Enfin, mon dernier remerciement, mais non le moindre, s'adresse à mon mari, Fabian, qui a accepté de se lancer avec moi dans cette aventure. Sans son réconfort, son aide, sa confiance, sa patience à l'égard de mes absences et de mon anxieté, et sans son amour, je n'aurais pu mener ce projet à bien.

A vous tous, merci pour tout ce que vous m'avez apporté,

Emilie

Contents

Introduction

Optimization, also called *mathematical programming*, is a mathematical discipline devoted to the search of the solution for minimization and maximization problems. Optimization problems appear in a lot of real-world applications and domains: economics, physics, industry, etc. Because better and better performances are desired in such domains, the demand for solving optimization problems is continually growing. So, the subject of this thesis has been induced by the power systems analysis discipline.

To solve these problems, designers in optimization use systematic processes which, to be efficient, are conceived to exploit as much as possible the specificities of the problem. As a consequence, the optimization problems are divided into several categories. The research work presented therein is concerned among other, with one of the classes of problems the most difficult to solve at the present time, namely the *mixed integer nonlinear and nonconvex problems*. Due to the difficulty to characterize a solution for such problems, we have chosen to develop a *global optimization* method to solve them, that is, a method which is guaranteed to find the *best solution* of the problem, contrary to the methods known as *local methods*. Global optimization is a new discipline in optimization since its emergence dates from a few decades. The reason of this is simple: global optimization is *expensive* and before the appearance of technologies like computers, it was utopian to hope finding with certainty the best solution of a general problem. Nowadays, things have evolved and the development of more and more powerful computers has enlarged the class of problems that are possible to solve, which makes global optimization a discipline of intense activity.

In this thesis, we present a new global optimization method for solving two categories of optimization problems, *nonconvex and nonlinear continuous problems* and *mixed integer nonlinear and nonconvex problems*. The method being motivated to solve a particular problem coming from power system management, it is at present, only applicable on problems having the specificities of this particular problem, although it is based on a general framework and can be easily extended to solve a more general class of problems. The method is first theoretically derived and then implemented and validated on a collection of test problems. More precisely, this thesis is organized as following: The two first chapters are *introductory*. Chapter 1 gives some basic notions in optimization and describes optimization methods to solve some specific classes of problems, including the problems that we tackle. Chapter 2 is devoted to the application which motivates the development of a new optimization method. Its physical interpretation and its modelling are detailed. The results of some existing methods to solve this problem are also reported. We mainly focus on the special ordered set approximation method. After having reviewed the notion of special ordered sets, we highlight the failure of an approximation method on the problem under study and point out that a global optimization method is more suitable.

In Chapter 3, we present the method that we have developed, namely an *outer approximation*

method based on special ordered sets. The way of building the outer approximation problem and to refine it are then thoroughly described. The key notions of our method are those of *approximation errors* and *branch-and-bound*. The proof that our method converges to a global solution of the problem is also established. Chapter 4 then deals with *theoretical* considerations and shows on one hand that, for the problem under study, the outer approximations developed in Chapter 3 are competitive with regard to the usual ones. On the other hand, this chapter allows to theoretically justify some choices taken to refine the outer approximation problem.

Chapters 5 to 8 are dedicated to more *practical* aspects of the method. The implementation is first described and several alternatives allowing us to improve the branch-and-bound strategy used to refine the outer approximation problem are then investigated. The results obtained on a collection of problems by applying these alternatives are also reported. More specifically, Chapter 5 is interested in the *tightening of the bounds* on the variables before and during the process of the method. Chapter 6 deals with the choice of the *branching variable* and Chapter 7 is concerned with the *node selection* strategy. Chapter 8 presents several comparaisons of different methods.

Finally, we conclude this thesis by recapitulating our approach and the obtained results, by presenting some perspectives for future research and by giving a summary of our contributions.

Chapter 1

Background on optimization

Optimization is a mathematical branch which aims at solving a problem in the best way. The optimization problems being formulated as *minimization* or *maximization* problems, this amounts to look for the value, known as the *optimum value*, which corresponds to the minimum or maximum of the problem. Optimization problems arise everywhere in the real-world: it can be, for instance, the maximization of the profit of an industry or the minimization of its costs, the minimization of energy losses in an electrical network, the maximization of efficiency for a production line or the minimization of travel times. Anybody is confronted with optimization problems. However, optimization mainly covers the following domains: industry, economics, physics, biology, chemistry, medicine, planning, social management and statistics.

To tackle the problems, designers in optimization develop systematic and specific methods depending on the features of the problem. Encouraged by the success of optimization methods to solve efficiently some classes of problems (linear and nonlinear continuous problems or mixed integer linear problems, for example), industry, engineers and researchers look for solving problems more and more complex, which makes of optimization a continually evolving discipline.

In this chapter, we first review some basic notions on optimization and then present some of the methods among the most popular for solving general continuous and discrete optimization problems.

1.1 Basic notions on optimization

We first explain how to formulate an optimization problem and then focus on the notions of feasibility, optimality, convexity and on the optimality conditions. We conclude this section by describing the way chosen in this thesis to classify the optimization problems.

1.1.1 Formulation of a mathematical program

In order to be solved, the considered problems are first *modelled* in the sense that they are translated in mathematical terms. This operation simplifies the problem since all its features cannot be taken into account in the modelling. Indeed, this would make the problem too complex and untractable although progresses have been done in the last decades to solve more and more difficult problems. In fact, only the relevant characteristics of the problem are used in

the modelling. Note that the modelling is a crucial phase in an optimization process since the quality of the solution for the modelled real situation is strongly based on the quality of this modelling.

An optimization problem depends on different components: its *objective function*, its *variables* and also frequently, its *constraints*. The objective function, denoted f, corresponds to the function to minimize or to maximize. It generally gives a single number which measures the quality of a solution for the problem (for example, the production costs of a good). The variables, denoted x, are the unknowns of the problem to which the best values as possible must be given in order to optimize the value of the objective function (the quantity of raw materials to produce a good, for instance). The constraints, denoted g, are functions establishing some relations that the variables must fulfill (for example, the maximum available quantity of raw materials). They usually are of two types: inequality or equality constraints (even if an equality constraint can be modelled as two inequality constraints). When an optimization problem is expressed in terms of objective function and constraints, the underlying formulation is referred to as a *mathematical program*.

If the problem is unconstrained, the optimization problem is formulated as:

$$\min_{x \in \mathbb{R}^n} f(x), \tag{1.1}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$, while if the problem is subject to some constraints, the mathematical program can be written like:

$$(P) \begin{cases} \min_x & f(x), \\ \text{s.t.} & g_i(x) = 0,\ i \in \mathcal{E}, \\ & g_i(x) \leq 0,\ i \in \mathcal{I}, \\ & x \in \Omega, \end{cases} \tag{1.2}$$

where $g_i : \mathbb{R}^n \rightarrow \mathbb{R},\ i \in \mathcal{E} \cup \mathcal{I}$. In this formulation, $\mathcal{E}$ and $\mathcal{I}$ are two disjoint index sets: $\mathcal{E}$ comprises the indices of equality constraints while $\mathcal{I}$ is composed of the ones of inequality constraints. Ω is a subset of $\mathbb{R}^n$ to which x must belong. For instance, the set Ω can contain the possible *discrete restrictions* on the variables which require that the concerned variables take only some particular values, like integer values for example. The variables which are subject to such restrictions are known as the *discrete* variables. Note that it can be shown (see Floudas [45]) that each discrete restriction can be modelled by a set of binary restrictions. A problem which has no discrete restriction is said to be *continuous*, otherwise it is referred to as *discrete*. If the problem comprises continuous and discrete variables, the problem is also known as a *mixed integer* problem.

Note finally that we limit ourselves to study minimization problems since the maximization problems can be easily transformed into minimization ones by using the fact that:

$$\max f(x) = -\min -f(x).$$

1.1.2 Feasibility and optimality

Some vocabulary related to the concepts of *feasibility* and *optimality* is now introduced. A point x which satisfies the constraints of (1.2) is said to be *feasible*. The set S of all points fulfilling these constraints defined by:

$$S = \{x \in \mathbb{R}^n : \ x \in \Omega, \ \ g_i(x) = 0 \ \forall i \in \mathcal{E}, \ \ g_i(x) \leq 0 \ \forall i \in \mathcal{I}\},$$

is referred to as the *feasible set* or the *feasible domain*. When, for a given problem, the feasible set S is empty, the problem has no solution and is said to be *infeasible*.

Solving the mathematical program (1.2) thus amounts to find among the feasible set, a point which minimizes the objective function $f(x)$. Such a particular point, denoted x^*, is known as an *optimum solution* and the value of f at this point is the *optimum value*. A *global minimum* x^* for a problem (1.2) is a point of the feasible set which produces the minimum value for the objective function over S. Mathematically, x^* satisfies:

$$f(x^*) \leq f(x) \quad \forall x \in S. \tag{1.3}$$

However, for a general problem, it is expensive to obtain the global minimum and to guarantee that a point is really the global minimum since this implies to explore the whole feasible space. Therefore, another type of minimum, called *local minimum* is considered. A local minimum is a feasible point x^* for which there exists $\mathcal{N}(x^*)$ a neighbourhood around x^* such that:

$$f(x^*) \leq f(x) \quad \forall x \in S \cap \mathcal{N}(x^*). \tag{1.4}$$

Note that in lots of real applications, local minima are often sufficient. In the case of a discrete problem, (1.4) implies that each local minimum x^* for the problem obtained by fixing the values of the discrete variables to some feasible values is also a local minimum for the discrete problem since it is always possible to find a neighbourhood around x^* in which x^* is the unique feasible point with respect to the discrete restrictions. Therefore, the definition (1.4) does not take the discrete restrictions into account. An alternative definition for (1.4) could be considered by requiring that a local minimum for a mixed integer problem produces the smallest value for the objective function among all the values which can be reached at one of its feasible direct neighbours with regard to the discrete restrictions. However, the optimality conditions (see Section 1.1.4) developed to check if a point can be, or not, a local minimum are based on local information. Therefore, comparing the value of the objective function at a point satisfying (1.4) with the best value reachable for each of its neighbours with respect to the discrete restrictions would imply to solve an optimization problem for each of these neighbours. Indeed, even if the values of the discrete variables are fixed for these neighbours, an optimization problem must be solved to determine the values of the continuous variables which minimize the objective function. Since the number of neighbours is exponential with the number of discrete restrictions, finding a "good" local minimum with respect to these restrictions is very hard. As a consequence, the search of solutions is a more challenging task in the discrete case than in the continuous one, since it needs not only to find a solution but also to prove that it is a good solution with respect to the discrete restrictions.

1.1.3 Convexity

A useful notion in optimization is that of *convexity*:

- A *set* X is said to be *convex* if any points x and y of X satisfy:

  $$\lambda x + (1 - \lambda)y \in X, \quad \forall \lambda \in [0, 1].$$

  Geometrically, a convex set is characterized by the fact that any point of a segment joining two points of this set also belongs to this set.

- A point x is referred to as a *convex combination* of k points x_i of X if it can be expressed as:

$$x = \sum_{i=1}^{k} \lambda_i x_i, \quad \sum_{i=1}^{k} \lambda_i = 1, \quad \lambda_i \geq 0, \quad i = 1, .., k.$$

  If the assumption on the positivity of the λ_i is removed, the combination is said to be *affine*. Therefore, each convex combination is also affine.

- The *convex hull* of a set X is the smallest convex set containing X.

- A *function* f defined on a convex set X is *convex* if it fulfills:

$$\lambda f(x) + (1 - \lambda) f(y) \geq f(\lambda x + (1 - \lambda) y) \quad \forall x, y \in X, \ \forall \lambda \in [0, 1].$$

  Geometrically, this means that a segment joining two points of a convex function always overestimates this function. Furthermore, as shown in Luenberger's monograph [78], a continuously differentiable function on a convex set X is convex if and only if:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x), \quad \forall x, y \in X, \tag{1.5}$$

  where $\nabla f(x)$ is the gradient of $f(x)$. As the right term of this inequality is the expression of the tangent of f at point x, this property means that a convex function always overestimates its tangents at any point of its domain. The inverse of a convex function is a *concave function* in the sense that f is concave if $-f$ is convex. Accordingly, geometrically, a segment joining two points of a concave function always underestimates this function while the tangent at a point of this function overestimates it. Finally, an affine function on a convex set X is a function f such that:

$$\lambda f(x) + (1 - \lambda) f(y) = f(\lambda x + (1 - \lambda) y) \quad \forall x, y \in X, \ \forall \lambda \in [0, 1].$$

  Therefore, each affine function is also convex.

After having defined the above concepts, the expression of a *convex problem* can be given. The problem (1.2) is convex if the objective function f and the inequality constraint g_i $(i \in \mathcal{I})$ are convex, the equality constraints g_i $(i \in \mathcal{E})$ are affine and the set Ω is convex. The convex problems have nice properties, as shown by the following result (see Nash and Sofer [86] for the detail of the proof).

Theorem 1.1 *Let x^* be a local minimum of a convex problem. Then x^* is also a global minimum.*

To conclude this section, we refine the notion of a convex problem when the problem is discrete. Indeed, in this case, the set Ω is nonconvex since it contains the discrete restrictions. A discrete problem is said to be convex if its *continous relaxation* is convex, that is, if the problem obtained by dropping the discrete restrictions is convex.

1.1.4 Optimality conditions

This section deals with the *optimality conditions* which allow us to check if a point can be, or not, a minimum for the considered problem. The optimality conditions are based on local information. They thus give a characterization for the *local* minima only and not for the global ones, except for particular problems like convex ones (see Theorem 1.1) and l_2-norm trust-region subproblems (see Theorem 7.2.1 of Conn *et al.* [26]), for instance. As a consequence, the optimality conditions, which have been developed for the continuous case, cannot suitably characterize a minimum of a general problem in the discrete case (see Section 1.1.2). Even attempts are made nowadays to design optimality conditions for general problems in the discrete case, nothing practical exists for general discrete problems.

There exist several types of optimality conditions for general continuous problems: necessary first-order, necessary second-order and sufficient second-order, and they differ if the problem is unconstrained or constrained. In this section, only the classical *necessary first-order optimality conditions* are presented (the other ones being not explicitly used in what follows). For a detailed discussion about optimality conditions, we refer the reader to the books of Conn *et al.* [26] and Nocedal and Wright [90]. We now state the first-order necessary optimality condition for an unconstrained problem.

Theorem 1.2 *(First-order necessary optimality condition for unconstrained problems) If x^* is a local minimum of problem (1.1) and if f is continuously differentiable in a open neighbourhood of x^*, then $\nabla f(x^*) = 0$.*

Before giving the expression of the first-order necessary optimality condition for a constrained problem, we introduce some further concepts and notations. The *active set* at a given feasible point x is the set $\mathcal{A}(x)$ of the indices of constraints g_i equal to zero at this point, that is:

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} \text{ such that } g_i(x) = 0\}.$$

An inequality constraint g_i which is equal to zero at x is said to be *active* at x. The second concept which has to be defined is that of *Lagrangian*. For the problem (1.2), the Lagrangian is given by:

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i g_i(x), \tag{1.6}$$

where the variables λ_i $(i \in \mathcal{E} \cup \mathcal{I})$ are known as the *Lagrangian multipliers* or the *dual variables*. With this latter terminology, the variables x are called the *primal variables*. We finally introduce the last notion needed to establish the first-order necessary optimality condition for constrained problems which is the concept of *constrained qualification conditions*. These conditions aim at guaranteeing the exclusion of pathological geometric cases. More precisely, they ensure that the feasible set is reasonably represented by the first-order Taylor approximation of the constraints around a potential local minimum x^*, that is, each constraint g_i $(i \in \mathcal{E} \cup \mathcal{I})$ is reasonably approximated at points $x^* + \epsilon$ (where ϵ is a vector with components of small values) by:

$$g_i(x^* + \epsilon) \approx g_i(x^*) + \nabla g_i(x^*)^T \epsilon.$$

There exist several constrained qualification conditions in the literature (*Mangasarian-Fromovitz constraint qualification* and in the convex case, *Slater's constraint qualification*, for instance). We restrict here our attention to the so-called *linear independency constraint qualification* (LICQ for short).

Condition 1.1 *(Linear independency constraint qualification)*
Given a point x^* and the active set $\mathcal{A}(x^*)$ at this point, the linear independency constraint qualification holds at x^* for problem (1.2) if the set of gradients of the active constraints at x^*:

$$\{\nabla g_i(x^*) : i \in \mathcal{A}(x^*)\}$$

are linearly independent.

Having explained the above concepts, we can now give the first-order necessary optimality conditions for a constrained problem.

Theorem 1.3 *(First-order necessary optimality conditions for constrained problems)*
Let x^* be a local minimum of the problem (1.2) at which the LICQ condition holds. Then there exists a Lagrangian multiplier vector λ^*, with components λ_i^* $(i \in \mathcal{E} \cup \mathcal{I})$, such that the following conditions are satisfied:

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0 \qquad \textit{(stationarity)}, \tag{1.7}$$

$$g_i(x^*) = 0 \quad \forall i \in \mathcal{E} \qquad \textit{(feasibility)}, \tag{1.8}$$

$$g_i(x^*) \leq 0 \quad \forall i \in \mathcal{I} \qquad \textit{(feasibility)}, \tag{1.9}$$

$$\lambda_i^* \geq 0 \quad \forall i \in \mathcal{I} \qquad \textit{(nonnegativity of the multipliers)}, \tag{1.10}$$

$$\lambda_i^* g_i(x^*) = 0 \quad \forall i \in \mathcal{I} \qquad \textit{(complementarity)}. \tag{1.11}$$

These conditions developed independently by Karush [64] and Kuhn and Tucker [70] are referred to as the *Karush-Kuhn-Tucker conditions* (or shortly, the *KKT* conditions). A point x^* which satisfies these conditions is called a *KKT point*.

1.1.5 Classification of optimization problems

The optimization problems can be classified according to lots of criteria. However, a major distinction is probably concerned with the presence or not of *discrete restrictions*. As explained in Section 1.1.2, the task related to the solution of such problems is double since it requires to find a minimum (local or global) but also to prove that it is also a good optimum with regard to the discrete restrictions. The methods developed to solve discrete problems are thus very different from those employed in the continuous case, even if they generally use continuous methods to solve subproblems. Therefore, in this chapter, we present separately the optimization methods to solve continuous and discrete problems. A second distinction could be the presence or not of *constraints* in the problem. Among the constrained problems, we also distinguish the methods used to solve *linear* and *nonlinear* problems. The last distinction that we consider in this thesis is related to the quality of the optimum since we are interested in *global optimization* methods which aim at finding the global optimum of a problem contrary to the *local optimization* ones which can guarantee no more than finding a local minimum, unless the problem is convex.

Note that there exist lots of other optimization problems, as for example, stochastic mathematical programs (see Birge and Louveaux [18] or Kall and Wallace [62], for instance), semidefinite programs (see Boyd *et al.* [20], Vandenberghe and Boyd [112] and also Wolkowicz *et al.* [126]), mathematical programs with complementarity constraints (see Ferris and Pang [38], Luo *et al.* [79], Outrata *et al.* [92]) and bilevel programs (see Dempe [30] for more details).

1.2 Methods for continuous optimization problems

This section is concerned with optimization methods for solving *continuous* problems, notably unconstrained problems, linear problems (LP) and nonlinear problems (NLP). For the latter, a distinction is performed between local optimization methods and global ones. Note that this overview of optimization methods for continuous problems is far from being exhaustive since it aims at mainly focusing on methods and concepts which will be used in the following.

1.2.1 Methods to solve continuous unconstrained problems

Our study of continuous optimization methods starts with the techniques developed to solve the unconstrained problem defined in (1.1). As pointed out in Section 1.1.4, the first-order optimality condition stipulates that a local minimum x^* for an unconstrained problem must satisfy:

$$\nabla f(x^*) = 0. \tag{1.12}$$

Therefore, the methods designed for solving problems (1.1) look for points where the gradient of the objective function of the problem is zero. A basic idea to reach this goal is to solve the system (1.12) of n equations with n unknowns. To this aim, an iterative method producing a sequence of iterates that we hope converging to a point fulfilling (1.12) is used. *Newton's method* is such a technique.

Newton's method
To motivate the use of this method for the system of equations (1.12), let us consider the following model of f (easier to minimize than f) around the current iterate x_k:

$$m(x_k + s) = f(x_k) + \nabla f(x_k)^T s + \frac{1}{2} s^T \nabla^2 f(x_k) s, \tag{1.13}$$

where $\nabla f(x_k)$ and $\nabla^2 f(x_k)$ are, respectively, the gradient and the Hessian of f at x_k. If m is a sufficiently accurate approximation of f around x_k, minimizing $m(x_k + s)$ should help to minimize $f(x)$. By the first-order optimality condition, the minimizer s_k of $m(x_k + s)$, if it exists, must satisfy the following system of equations which are known as *Newton equations*:

$$\nabla^2 f(x_k) s_k = -\nabla f(x_k). \tag{1.14}$$

If the matrix $\nabla^2 f(x_k)$ is nonsingular, the solution s_k of system (1.14) is unique and is called the *Newton step* or the *Newton direction*. Moreover, if, in addition, $\nabla^2 f(x_k)$ is positive definite (that is, if $s^T \nabla^2 f(x_k) s > 0,\ \forall s \in \mathbb{R}^n_0$) and the gradient of f is nonzero at x_k, the Newton step is a *descent direction* (that is, $s_k^T \nabla f(x_k) < 0$). The next iterate is then given by:

$$x_{k+1} = x_k + s_k. \tag{1.15}$$

Newton's method iterates the construction of a model given by (1.13) around the current iterate x_k, the minimization of this model by solving (1.14) and finally, the update of the current iterate by (1.15). This method is well defined as long as the matrix $\nabla^2 f(x_k)$ is nonsingular. However, it happens that this matrix is singular far from a local minimum for f or that it is not positive definite, in which case the Newton direction is not a descent one. Actually, Newton's method

has only local convergence properties. Accordingly, it is only suitable if the starting point of the iterative process is sufficiently close to the solution, which is not known a priori. In order to enforce the convergence to a point which fulfills (1.12) from any starting point, a *globalization technique* must be integrated into the method. Another drawback of Newton's method is its need of the expression of the Hessian of the objective function at the current iterate, which can be expensive to compute. *Quasi-Newton* methods get round this difficulty by using an approximation of the Hessian of the objective function, as explained thoroughly in the book of Dennis and Schnabel [31]. Nevertheless, Newton's method has an important advantage in the sense that, when it converges, it converges rapidly.

In addition to Newton and quasi-Newton methods, the *conjugate-gradient* methods introduced by Hestenes and Stiefel [58] are another popular way to solve unconstrained problems. For more details on these latter methods and more generally on methods for solving unconstrained problems, we suggest the reader to consult the books of Luenberger [78], Nocedal and Wright [90] and Ruszczynski [99].

Globalization techniques

We now consider two well-known globalization techniques, namely the *line-search* and the *trust-region* methods, whose goal is to enforce the convergence from any starting point to a point fulfilling (1.12).

Line-search techniques

Having computed a descent direction d_k from a current iterate x_k, the idea of line-search methods is to find, along this direction, a point which produces a *sufficient* decrease in the value of the objective function (in the sense that it will allow us to converge to a point satisfying (1.12)). This point then becomes the next iterate:

$$x_{k+1} = x_k + \alpha d_k, \tag{1.16}$$

where $\alpha > 0$. Ideally, to obtain the largest decrease in the value of the objective function, α should be computed as the solution of:

$$\min_{\alpha>0} f(x_k + \alpha d_k). \tag{1.17}$$

However, solving (1.17) is often expensive and useless. Accordingly, it is preferable to generate candidate values for α until one of these values produces a sufficient decrease, in which case, this value is accepted and the iterate is updated by (1.16). In practice, conditions are used to determine if a particular value for α produces, or not, a sufficient decrease in the value of the objective function. The most popular among these conditions are *Armijo*, *Wolfe* and *Goldstein conditions* (see Nocedal and Wright [90] for more details).

Trust-region methods

While line-search techniques choose a direction and then determine the step length, trust-region methods define a maximum step length and then select a direction within the region determined by this step length. Trust-region methods start by building a model of f around the current iterate x_k. Usually, a quadratic model is employed:

$$m(x_k + s) = f(x_k) + \nabla f(x_k)^T s + \frac{1}{2} s^T H_k s,$$

where H_k is the Hessian of f at x_k or some approximation of it. Instead of minimizing this model as in Newton's method, we minimize it on a region where we *trust* the model, the *trust region*. Such a region, denoted $\mathcal{B}_k$, can be defined by:

$$\mathcal{B}_k = \{x_k + s \mid \|s\| \leq \Delta_k\},$$

where $\Delta_k > 0$ is known as the *trust-region radius* and $\|.\|$ is a given norm. The *trust-region problem* can thus be expressed as:

$$\begin{cases} \min_s & m(x_k + s), \\ \text{s.t.} & \|s\| \leq \Delta_k. \end{cases} \tag{1.18}$$

After having solved this problem (possibly approximatively), a trial step s_k is obtained. Its acceptance depends on the decrease in the value of the objective function at the candidate iterate defined by $x^+ = x_k + s_k$, compared to the decrease predicted in the model. If the candidate iterate produces a sufficient decrease in the objective function with regard to the predicted decrease in the model, it is accepted. If the decrease is more than sufficient, the trust-region radius Δ_k is increased. Otherwise, if the candidate iterate x^+ generates a poor decrease in the objective function compared to the predicted decrease, or even an increase, x^+ is rejected and the trust-region radius Δ_k is reduced. More precisely, the ratio of the *actual* reduction versus the *predicted* reduction is computed as:

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m(x_k) - m(x_k + s_k)}. \tag{1.19}$$

According to the value of ρ_k with regard to some fixed parameters, the candidate step x^+ is accepted or rejected and the trust-region radius is enlarged, reduced or unchanged. For more details about trust-region methods, including the way of solving the trust-region problem (1.18) exactly or approximatively, we refer the reader to the book of Conn *et al.* [26].

1.2.2 Methods to solve continuous linear problems

After the unconstrained problems, we examine the constrained problems and, at first, the class of the easiest problems in most cases, namely the linear ones. A linear program is characterized by a linear objective function and linear constraints. Accordingly, each function involved in the problem can be expressed as $ax + b$, where a and b are some constants. When the continuous problem (1.2) is linear, it is usual to consider its *standard form* given below:

$$(LP) \begin{cases} \min_x & c^T x, \\ \text{s.t.} & Ax = b, \\ & x \geq 0, \end{cases} \tag{1.20}$$

where x and c are vectors of length n, b is a vector of length m and A is an $m \times n$ matrix. Without loss of generality, we can assume that A has full rank and $m \leq n$. Note that any linear problem can be expressed under the standard form (1.20), as illustrated in [86]. Moreover, since it is defined by linear constraints, the feasible set S of a continuous linear problem is convex and the problem (LP) is *convex*. In fact, linear problems are particular cases of convex problems.

We now introduce the notion of extreme point. An *extreme point* x of the feasible set S is

a point which cannot be expressed as a convex combination of two other points of this set, that is:

$$\nexists \lambda \in [0,1],\ y,z \in S\ (x \neq y,\ x \neq z)\ \text{ such that } x = \lambda y + (1-\lambda)z.$$

For example, in two dimensions, if the feasible set S is a triangle, its extreme points correspond to its three vertices. The notion of extreme point is crucial in linear programming since it can be shown (see [86], for instance) that the optimum value of a linear problem, if an optimum solution exists, is reached at an extreme point of the feasible set. Therefore, it is sufficient to consider the boundary of the feasible domain, which is the main idea of the *simplex method* proposed by Dantzig (see [29]). The simplex method is an iterative method which passes from an extreme point to another to which it is adjacent. All generated iterates are extreme points. At each iterate, one seeks a direction (in fact an edge of the feasible set) along which the value of the objective function decreases. If such a direction exists, it is taken towards the next extreme point and then, the operation is repeated. Otherwise, the optimum solution is found. Accordingly, the simplex method explores the feasible domain by moving on its boundary, which distinguishes it from the second popular class of methods used in linear programming, namely, the *interior point methods*. Indeed, interior point methods search the feasible set by the interior and reach the boundary only at the limit. Historically, interior point methods have been first developed by Fiacco and McCormick to solve nonlinear problems, as detailed in [39] and [40] (see also Section 1.2.3), and then, have been applied by Karmarkar in the linear programming framework [63]. For more details about interior point methods, we refer the reader to the monograph of Wright [127] and about the simplex method to the books of Luenberger [78] and Nash and Sofer [86].

1.2.3 Methods to solve continuous nonlinear problems (not necessarily convex)

We are now interested in (continuous) nonlinear programs, that is, in problems (1.2) where the objective function and constraints may be nonlinear. To find an optimum solution, the methods designed for nonlinear programs generate a succession of subproblems easier to solve. The methods differ in their way of constructing and solving these subproblems. Here, we focus on two classes of methods to solve nonlinear problems: *sequential quadratic programming methods* and *interior point methods* (the latter being already mentioned in the linear case).

This overview of nonlinear optimization methods being incomplete, we refer the reader to the books of Conn *et al.* [26], Luenberger [78], Nocedal and Wright [90] and Ruszczynski [99], and to the paper of Sartenaer [104] for a thorough analysis of this topic.

Sequential Quadratic Programming methods

Sequential Quadratic Programming methods, brievly referred to as *SQP* methods, have been introduced by Wilson [125] for solving nonlinear problems. These methods can be seen as an extension of Newton's method to the constrained case. The idea of SQP methods is to represent the nonlinear problem by a quadratic one (easier to solve) around the current iterate, and to solve this latter to produce the next iterate. This quadratic problem built around the current

iterate x_k is defined by:

$$(QP_k) \begin{cases} \min_p & \frac{1}{2}p^T \nabla^2_{xx}\mathcal{L}(x_k, \lambda_k)p + \nabla f(x_k)^T p \\ \text{s.t.} & \nabla g_i(x_k)^T p + g_i(x_k) = 0,\ i \in \mathcal{E}, \\ & \nabla g_i(x_k)^T p + g_i(x_k) \leq 0,\ i \in \mathcal{I}, \end{cases} \tag{1.21}$$

where $\nabla^2_{xx}\mathcal{L}(x_k, \lambda_k)$ is the Hessian of the Lagrangian defined in (1.6) for problem (1.2). In (1.21), the contraints are the first-order Taylor's approximations of the original contraints g_i $(i \in \mathcal{I} \cup \mathcal{E})$ around the current iterate x_k. Denoting p_k, the solution (possibly approximate) of (QP_k) and λ, the associated Lagrangian multipliers, the next iterate is then given by $x_{k+1} = x_k + p_k$ and $\lambda_{k+1} = \lambda$. To solve quadratic problems, some specific techniques have been developed (see Nocedal and Wright [90]) and implemented in some solvers like *BQPD* [41] for example.

Globalization techniques

Like for the unconstrained case, some *globalization techniques* must be added to the method in order to enforce its convergence even if the starting point is far from the solution. Line-search and trust-region approaches are again employed. We first consider the line-search technique. In the unconstrained case, the *line search* aims to obtain sufficient decreases along the descent directions, since the unique goal of the optimization is to minimize the value of the objective function. Here, however, we must also attempt to reach the feasibility. To balance these two goals often conflicting, several techniques are proposed. The use of a *merit function* ϕ which allows us to quantify the satisfaction of these two goals, is one of them. There exist numerous merit functions. Restricting our attention to the nonlinear problems with equality constraints only, we give the expression of one of the most known merit functions, that is, the l_1 *merit function*:

$$\phi_1(x, \mu) = f(x) + \frac{1}{\mu}\|g(x)\|_1,$$

where $\mu > 0$ is the *penalty parameter*, $g(x)$ is the vector of (equality) constraints and $\|.\|_1$ is the l_1-norm. In order to converge, the direction p_k, solution of the quadratic problem (QP_k), must be a descent direction for the selected merit function ϕ. To this aim, μ must be sufficiently small. A value for $\frac{1}{\mu}$ adapted to the l_1 merit function is given at each iteration by:

$$\frac{1}{\mu} = \delta + \|\lambda_{k+1}\|_\infty,$$

where $\|.\|_\infty$ is the l_∞-norm and $\delta > 0$ is a constant. Moreover, ϕ has to fulfill a line-search condition like the ones cited in Section 1.2.1. This implies to impose some conditions on the merit function but also on the quadratic subproblem (QP_k), and in particular, to add suitable modifications to $\nabla^2_{xx}\mathcal{L}(x_k, \lambda_k)$, the Hessian of the Lagrangian. SQP methods differ in their choice of the merit function and also in the modifications introduced in the latter matrix.

Let us now switch to the *trust-region approach* for SQP methods which is due to Beale [15] and Sargent [103]. In this case, an additional constraint preventing the new iterate from leaving the trust region is introduced in the subproblem (QP_k), that is,

$$\|p\| \leq \Delta_k,$$

where $\|.\|$ is a given norm and $\Delta_k > 0$ is the current trust-region radius. In order to measure the progress in the convergence produced by the candidate iterate, a merit function is again employed. As a consequence, the numerator of the ratio ρ_k defined in (1.19) must be modified in order to compute the decrease in ϕ instead of in f. However, the objective function of (QP_k) is not the model of the merit function itself and thus cannot predict appropriately the decrease in ϕ. To remedy this, several possibilities can be used: one may modify the subproblem (QP_k) by adding to it a penalty term in order that it represents more suitably the merit function ϕ, or one may add to the denominator of the ratio ρ_k a term reflecting the predicted reduction in the constraint violation.

Filter approach

As the efficiency of methods using a merit function is strongly related to the initialization and update phases of the penalty parameter μ, a different approach based on the notion of *filter* has been proposed by Fletcher and Leyffer [43]. The principle of filter methods is that a step is accepted if it produces a sufficient decrease either in the value of the objective function *or* in the constraint violation. In practice, a couple $(\theta(x), f(x))$ measuring the constraint violation and the value of the objective function at a point x is associated to each considered point x. Lots of ways exist to measure the constraint violation, as for example, the *maximum violation*:

$$\theta(x) = \max\{\max_{i\in\mathcal{E}} |g_i(x)|, \max_{i\in\mathcal{I}}(g_i(x), 0)\}.$$

The filter is based on the concept of *dominance*. A point x dominates a point y if it satisfies:

$$\theta(x) \leq \theta(y) \text{ and } f(x) \leq f(y).$$

The filter is composed of pairs (θ_i, f_i) associated to points x_i which are not dominated by other ones. The filter, denoted $\mathcal{F}$, is thus defined by:

$$\mathcal{F} = \{(\theta_i, f_i) \mid \theta_i < \theta_j \text{ or } f_i < f_j \text{ for } i \neq j\}.$$

A point x_k is accepted as a new iterate if the pair (θ_k, f_k) can enter in the filter. More details about filter methods can be found in Conn *et al.* [26], Fletcher *et al.* [42], Fletcher and Leyffer [43] or in the PhD thesis of Sainvitu [102]. Concerning the SQP methods, we refer also the reader to the book of Nocedal and Wright [90]. To conclude this section, we mention that some competitive software packages are based on SQP methods, like *FilterSQP* [44] which is based on an SQP method using a trust region and a filter, and *SNOPT* [50] which implements an SQP method using a line-search strategy with a merit function of augmented Lagrangian type.

Interior point methods

In order to motivate the use of interior point methods called primal-dual, we first focus on the barrier methods.

Barrier methods

Instead of replacing the nonlinear problem by a succession of quadratic subproblems like in SQP methods, the *barrier methods* reformulate the constrained nonlinear program as an *unconstrained* one by introducing the constraints in the objective function. Assume that the nonlinear problem involves inequality constraints only, that is, we consider:

$$(P_i) \begin{cases} \min & f(x), \\ \text{s.t.} & g_i(x) \geq 0,\ i \in \mathcal{I}. \end{cases} \tag{1.22}$$

The functions used in barrier methods to combine the objective function and the constraints in a specific way are known as the *barrier functions*. The most common barrier function is probably the *logarithmic barrier function* which is given by:

$$B(x, \mu) = f(x) - \mu \sum_{i \in \mathcal{I}} \log(g_i(x)),$$

where $\mu > 0$ is the barrier parameter. Therefore, barrier methods aim at solving:

$$\min_x B(x, \mu). \tag{1.23}$$

These methods have the property that the generated iterates remain strictly feasible. By minimizing $B(x, \mu)$ with smaller and smaller values for μ, it can be shown (see Forsgren *et al.* [47]) that, under some assumptions, the sequence of minima for $B(x, \mu)$, denoted $x(\mu)$, converges to a local minimum of (P_i). To produce the iterates, a variant of Newton's method is applied. However, with this approach, the full Newton step s_k computed at $x(\mu)$ produces an infeasible point for (P_i). As a consequence, the full Newton step must be reduced to generate a new iterate belonging to the feasible set. Shorten the Newton step prevents the barrier methods from having the fast convergence of Newton's methods.

Primal-dual interior point methods

To get round the drawback of the barrier methods mentioned above, *primal-dual interior point methods* are considered. The basic idea of these methods is to use the *dual variables* λ and to treat them independently from the primal variables, contrary to the barrier methods which only employ the primal variables. Exploiting the KKT conditions associated to problem (P_i) allows us to reformulate the KKT conditions related to the minimization problem (1.23) like:

$$KKT(B(x, \mu)) \begin{cases} \nabla f(x) - \nabla g(x)^T \lambda = 0, \\ G(x)\lambda - \mu e = 0, \\ g(x) \geq 0, \\ \lambda \geq 0, \end{cases} \tag{1.24}$$

where $G(x)$ is equal to diag$(g_1(x), g_2(x), ..., g_m(x))$ $(m = \#\mathcal{I})$, the diagonal matrix composed of elements $g_i(x)$ and e is the vector of length m where all components are equal to 1. Note that, for the sake of clarity, in the notation, we often use x for $x(\mu)$ and λ for $\lambda(\mu)$. The system (1.24) is known as the *primal-dual equations*. The solutions $(x(\mu), \lambda(\mu))$ of these equations define a *primal-dual central path* $\mathcal{C}$, that is:

$$\mathcal{C} = (x(\mu), \lambda(\mu)),$$

which tends to a KKT point when μ tends to zero. The system (1.24) is again solved by Newton's method. Therefore, the *primal-dual Newton direction* (p_x, p_λ) is such that:

$$\begin{pmatrix} \nabla^2_{xx}\mathcal{L}(x, \lambda) & -\nabla g(x)^T \\ \Lambda \nabla g(x) & G(x) \end{pmatrix} \begin{pmatrix} p_x \\ p_\lambda \end{pmatrix} = - \begin{pmatrix} \nabla_x \mathcal{L}(x, \lambda) \\ G(x)\lambda - \mu e \end{pmatrix},$$

where $\Lambda = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_m)$ $(m = \#\mathcal{I})$. With this update of iterates, the drawback of barrier methods is avoided since the full Newton step computed at a point (x, λ) of the primal-dual central path generates a feasible iterate. Therefore, the full Newton step has not to be

reduced contrary to the barrier methods, and the convergence can be reached at an appropriate rate, which makes primal-dual interior point methods successful to solve nonlinear problems. Note that some globalization techniques (line-search or trust-region) and a merit function are also introduced in the method to promote the convergence. Further details about interior point methods can be obtained in the books of Vanderbei [114] and Wright [127]. The paper of Forsgren *et al.* [47] also offers an extensive survey of this topic. Note finally that interior point methods have been implemented in numerous solvers like *LOQO* [113], *KNITRO* [116] and *IPOPT* [128].

1.2.4 Global methods to solve continuous nonlinear nonconvex problems

The nonlinear optimization methods presented above cannot yield more than local solutions, unless the nonlinear program has specific properties (for example, if the problem is convex (see Theorem 1.1)). In this section, we are interested in general nonlinear problems for which optimality conditions for a global optimum are not available. For some of such problems, notably highly nonconvex, the nonlinear optimization methods described above can declare the problem (locally) infeasible while the feasible set is nonempty. For numerous applications, it is however desirable to be guaranteed to detect a minimum, if such a point exists, and even more, the global one.

The aim of a global optimization method is twofold since it amounts to *detect* a minimum, like for local optimization methods detailed above, but also to *prove* that this minimum is the global one. Since the optimality conditions used in the search of a minimum are based on local information, these cannot be exploited to prove that the minimum of a general nonlinear problem is global. Therefore, the *whole feasible set* must be explored (in a clever way) to check if a minimum is global or not, which implies that global optimization methods are much more *expensive* than the local ones. Global optimization is a recent discipline in optimization because it is born a few decades ago while local optimization methods are developed for centuries. The progresses in the computation time due to the development of more and more powerful computers have made possible the solution of problems which were expected unsolvable before.

In this thesis, we focus on global optimization methods that *ensure* to detect a global minimum. As a consequence, we are not interested in *heuristics*, that is, in methods which aim at finding the global minimum but which have no guarantee to converge to it. However, in practice, such methods can also lead to good results. The use of heuristics is motivated by their cheap cost. Simulated annealing [77] and genetic algorithms [52] are common heuristics for global optimization.

Let us come back to methods having the guarantee to find the global minimum. Before explaining the general idea of these methods, we give the expression of the nonlinear problem (P_G) on which we focus:

$$(P_G)\begin{cases} \min_x & f(x), \\ \text{s.t.} & g_i(x) \leq 0,\ i \in \mathcal{I}, \\ & x \in \Omega. \end{cases} \tag{1.25}$$

Note that the problem (P) defined in (1.2) can be reformulated like problem (P_G) by replacing each equality constraint by two inequality constraints.

Before going further, we introduce the notion of *relaxation*. The relaxation of a problem

(P_G) is a problem, denoted (R), which underestimates the objective function and the constraints of (P_G). As a consequence, the optimum value of (R) is a lower bound for that of (P_G). The relaxation (R) can thus be expressed as:

$$(R) \begin{cases} \min_x & \bar{f}(x), \\ \text{s.t.} & \bar{g}_i(x) \leq 0,\ i \in \mathcal{I}, \\ & x \in \bar{\Omega} \end{cases} \tag{1.26}$$

where $\bar{f} : \mathbb{R}^n \to \mathbb{R}$ and $\bar{g}_i : \mathbb{R}^n \to \mathbb{R}$, $\Omega \subset \bar{\Omega}$ and $\forall x \in \Omega$, $\bar{g}_i(x) \leq g_i(x)\ \forall i \in \mathcal{I}$, and $\bar{f}(x) \leq f(x)$. Two common relaxations are the *convex relaxation* and the *continuous relaxation*. A relaxation is said to be convex if $\bar{\Omega}$, $\bar{f}$ and $\bar{g}_i$ $(i \in \mathcal{I})$ are convex. In case of a discrete problem, the continuous relaxation corresponds to the problem obtained by dropping the discrete restrictions.

Global optimization methods developed for solving nonconvex problems are in most cases based on the same scheme: they build a convex relaxation, solve it (hence finding its global solution) and refine it by partitioning the feasible domain. New convex relaxation subproblems are constructed on the generated subdomains. These subproblems are themselves solved and refined until the subdomains under study are guaranteed not to contain the global optimum. During the process of the method, candidates for the global optimum value, also referred to as upper bounds on this optimum, can be obtained by solving the problem (P_G) by means of local methods like the ones presented in Section 1.2.3 or when the solution of the convex relaxation subproblems is feasible for the problem (P_G). The global optimum is only found with certainty once the whole feasible domain has been explored. It then corresponds to the best upper bound found during the process of the method if such a bound has been found, otherwise, the problem (P_G) is infeasible.

Convex relaxations are generally used in global optimization methods since a minimum found for these problems is always global by Theorem 1.1. As a consequence, by definition of a relaxation, the optimum value which has been found for the convex relaxation consists of a valid lower bound on the optimum value of the problem (P_G). Since the lower bound is an information used to determine if a subdomain can contain, or not, a global optimum and thus, if the subdomain must, or not, still be considered, the use of *valid* lower bounds is crucial for the convergence to the global optimum. Global optimization methods differ in their way of building the convex relaxation and of refining it. Usually, the refinement of the relaxation is obtained thanks to *branch-and-bound* (see below).

We next explain the main ideas of two of the most popular solvers for global optimization, namely *BARON* and *αBB*, and that of another method proposed by Polisetty and Gatzke, which is close to the one developed in this thesis. But before going further, we detail the principle of the branch-and-bound strategy.

Branch-and-Bound

Branch-and-bound has been first proposed by Land and Doig [71] to solve mixed integer linear problems. It has then been extended to the mixed integer nonlinear case by Dakin [28] and improved by Gupta and Ravindran [56]. Branch-and-bound has been also adapted to solve more general problems, like continuous global optimization ones (see Falk and Soland [37] and Horst and Tuy [61]). In all cases, the same general scheme is used to solve with branch-and-bound a nonlinear problem also referred to as the *original problem*: a relaxation of this problem is constructed, solved and refined by *partitioning* the feasible domain. Subproblems are built on

the subdomains generated by this partitioning and are successively refined until the subdomain under study is guaranteed not to contain a global optimum of the nonlinear problem or a better optimum than the one found so far, if such an optimum has been found.

More precisely, a first relaxation (R) of the original problem is built and globally solved. Usually, the relaxation problem is convex, which allows us to guarantee the convergence to its global optimum value. This optimum value thus corresponds to a valid *lower bound* L on the optimum value of the original problem, by definition of a relaxation. To obtain an *upper bound* U, the original problem can be solved by using a local optimization method, like the ones described in Section 1.2.3. However, because of the cost of solving a nonlinear problem, an upper bound is not always computed. Note that, if the relaxation problem is infeasible, the original problem is infeasible too by definition of a relaxation and the algorithm stops. Otherwise, if the solution of the relaxation problem is feasible for the original problem, it corresponds to the global optimum of this latter problem. But these two cases are trivial. When they do not happen, the relaxation problem is refined by partitioning, that is, the feasible domain is divided in several subdomains (generally two). To this aim, in case of a partitioning in two subdomains, a variable, known as the *branching variable*, is selected and by bounding this variable, the domain is partitioned in two subdomains. For example, assume that the variable x_j defined on $[l_j, u_j]$ is selected. The domain is then divided in two subdomains, the first one containing all the points of the current domain satisfying:

$$x_j \leq m_j,$$

and the second one, the points fulfilling:

$$m_j \leq x_j,$$

where m_j is a point belonging to $]l_j, u_j[$. This operation is called *branching*. The relaxation is then refined on each of the subdomains. Indeed, the smaller the feasible domain, the tighter the relaxation.

To each generated subproblem is associated a node which is ordered in a *branch-and-bound tree* (see Figure 3.15 for an example of such a tree). According to the place of the associated node in the branch-and-bound tree, a generated subproblem is referred to as a *left subproblem* or a *right subproblem*. These nodes are also put in a *stack* containing all nodes to explore. Then, a node is chosen and removed from the stack. The related relaxation subproblem is solved, which gives a new lower bound L on the optimum value of the original problem *valid on the subdomain under study* (by assuming that the relaxation subproblem is feasible). If the obtained optimum solution is feasible for the original problem and if it produces a smaller optimum value than the current one, then the upper bound U is updated with this optimum value. The process is repeated: partitioning the domain under study, generating subproblems, choosing a node in the stack and solving it. The branch-and-bound tree (which is thus dynamically created) is extended in this way until one of the following conditions is satisfied at a considered node:

1. the relaxation problem corresponding to the treated node is infeasible,
2. the lower bound L generated at the node under study is larger than the upper bound U,
3. the difference between the upper and lower bounds, L and U, is smaller than a fixed threshold ϵ.

If one of the two first conditions holds, the node can be *cut* (or equivalently, *fathomed* or *discarded*), since the part of the domain considered at this node cannot contain a global minimum. Indeed, if the first condition is fulfilled, the original problem is also infeasible on this part of the domain by definition of a relaxation. If the second condition is satisfied, no better solution than the current one can be reached on this part of the domain since the better optimum value which can be possibly achieved, that is, L, is larger than the current optimum value U. By using this argument, when the upper bound U is updated during the branch-and-bound process, all nodes in the stack associated to a lower bound larger than the new current upper bound can be removed from this stack. Note that the lower bound for a node in the stack corresponds to the optimum value of the relaxation problem solved at its parent. Furthermore, if the third condition holds, the node can also be fathomed since, by a similar reasoning as for the second condition, a better optimum value than the current one within a thresold ϵ cannot be produced on the part of the domain under study. Note finally that the part of the domain containing a global optimum is also discarded (if this optimum has been found) by the third condition since the relaxation can be refined sufficiently in order that the lower and upper bounds on this part of the domain are within the threshold ϵ. A node is thus fathomed if it is associated to a domain on which the original problem is infeasible or if this domain cannot contain a better solution than the current one (even if the current solution belongs to this domain).

The branch-and-bound process is complete when all the subproblems have been treated, that is, when the stack is empty. Horst and Tuy [61] have shown that under mild assumptions, branch-and-bound converges to the global optimum for continuous optimization problems. More formally, Algorithm 1.1 given below is applied. When this algorithm stops, the optimum solution and value of the original problem are given by x^* and U respectively, unless U is equal to $+\infty$, in which case the original problem is infeasible.

In order for branch-and-bound to be efficient, the choices of the *node to treat* and the *branching variable* must be handled carefully. So, the choice of the node allows us to develop and explore the tree in some order. Ideally, the tree should be explored in such a way that the optimum value is rapidly found. If the optimum value, or a good upper bound on it, is quickly found, this value can be used to fathom nodes and then to converge faster. The choice of the branching variable is also important because the way in which the branch-and-bound tree is built strongly depends on it. Indeed, the subdomains on which the subproblems are constructed are determined by the branching variables. Branching on variables having a lot of influence on the problem, especially at the top of the tree, allows us to reduce the size of this tree and thus, the number of problems to solve, which improves the speed of convergence of the branch-and-bound process. For instance, a branching allowing us to increase the value of the objective function for the generated subproblems may be preferred, since in this way, it can be expected that the lower bound on the optimum value will be rapidly larger than the current upper bound U and the node will be discarded. To select the branching variables, numerous *branching rules* have been proposed (see Chapter 6 for more details). A third way to get better a branch-and-bound process, consists in trying to *tighten the bounds* on the variables. This is motivated by the fact, that, the smaller the domains of the relaxation problems, the tighter these problems and then, quicker nodes can be fathomed.

Numerous variants of the branch-and-bound strategy exist, depending on the use or not of techniques to tighten the bounds on the variables, on the branching rule and also on the node selection strategy. These notions will be discussed further in Chapters 5, 6 and 7, respectively. More details can also be found in Adjiman *et al.* [8] and in Linderoth and Savelsbergh [75] (the

latter reference being devoted to the mixed integer case). The branch-and-bound can also be improved by adding valid *cuts* to refine the relaxation. In this case, the branch-and-bound is referred to as *branch-and-cut*, which has been introduced by Padberg and Rinaldi [93].

Algorithm 1.1: Branch-and-bound

Let ϵ be a fixed accuracy.

Init: Set $U = +\infty$ as upper bound and $k = 0$.
Build a relaxation (R_0) for the original problem and put it in the stack.

while (the stack is not empty) **do**

1. Choose a node in the stack and solve the associated relaxation problem (R_k).
   If (this problem is infeasible) **then**
   Fathom the node and **go to 3**
   else
   Set L_k and x_k the optimum value and solution of (R_k), respectively.
   If ($L_k \geq U - \epsilon$) **then** Fathom the node and **go to 3.**
   If (x_k is feasible for the original problem) **then**
   If ($f(x_k) < U$) **then**
   Update: $U = f(x_k)$ and $x^* = x^k$. Remove from the stack all the nodes having a lower bound larger or equal to $U - \epsilon$.
   If ($f(x_k) - L_k \leq \epsilon$) **then** Fathom the node and **go to 3.**
2. Choose a branching variable, branch on it and add the generated subproblems to the stack.
3. Set $k = k + 1$.

end while

Branch-and-reduce

Ryoo and Sahinidis have extended in [100] the concept of branch-and-bound to solve problems for which a valid convex relaxation problem can be built, to that of *branch-and-reduce*. This approach is motivated by the fact that to achieve a fast convergence, the range of the variables must be reduced as much as possible. The branch-and-reduce strategy has been implemented in *BARON*, the *Branch-And-Reduce Optimization Navigator*, developed by Sahinidis and Tawarmalani [109]. This solver is conceived to handle problems for which it can build a convex relaxation. This includes a large class of problems. However, *BARON* cannot treat problems involving trigonometric functions, for example. In order for *BARON* to be applicable, each general function f or g_i is decomposed in a sum of functions and each of these latter functions is underestimated by a convex function (if *BARON* can treat the considered function). Then, the resulting relaxation problem is refined through a branch-and-reduce process, as explained below.

To improve the speed of convergence of branch-and-bound, *BARON* uses *optimality-based*

range reduction mechanisms which allow it to tighten the bounds on the variables and constraints. These mechanisms are based on *dual* variables and more particularly, on the *sensitivity* information that they provide. Assume, for example, that at the solution of the relaxation problem, the constraint $x_j - u_j \leq 0$ is active and that the value of the objective function of this relaxation problem is given by L. The Lagrangian multiplier λ associated to this constraint gives the *rate of change* in the value of the objective function with respect to a modification in u_j. This information allows us to tighten the lower bound on x_j if an upper bound U on the value of the objective function is known. Indeed, as the rate of change in the value of the objective function by moving away from u_j is known, it is possible to predict the point $x_j = \kappa$ from which the value of the objective function is larger or equal to the current upper bound U. Thus, the point κ must satisfy the following equality:

$$L + \lambda(u_j - \kappa) = U,$$

or equivalently,

$$\kappa = u_j - \frac{(U - L)}{\lambda}.$$

For $x_j < \kappa$, the value of the objective function is guaranteed to overestimate the current upper bound U. Therefore, the current feasible interval for x_j can be reduced to $[\kappa, u_j]$. Similar reasonings can be held to strengthen the upper bound on a variable active at its lower bound, and for the bounds on active general constraints. When the constraints are not active, some tricks can be employed in order that the optimality-based range reduction mechanisms are even though applicable. For more details about optimality-based range reduction, we refer the reader to the paper of Ryoo and Sahinidis [100] and for, in addition, further information about *BARON*, to the book of Tawarmalani and Sahinidis [109] and to the user's guide [101].

A solver based on branch-and-bound: α*BB*

The drawback of *BARON* lies on the fact that it cannot be applied on any problem (1.25) because it cannot build a convex underestimator for all functions. The solver α*BB* circumvents this difficulty by using a generic formula for underestimating any nonconvex twice-differentiable function. To build the convex relaxation problems, each function f and g_i is decomposed in a sum of nonlinear functions. These latter are then classified in three categories: the convex functions (which are underestimated by themselves), the nonconvex functions with a special structure (for which the expression of a convex underestimator is known: bilinear functions, trilinear functions, concave univariate functions, etc.), and finally the other ones: the general nonlinear functions. For a general nonlinear function f, the following convex underestimator is used by α*BB*:

$$u(x) = f(x) + \sum_{i=1}^{n} \alpha_i (l_i - x_i)(u_i - x_i). \tag{1.27}$$

In order for $u(x)$ to be convex, its Hessian must be positive definite for all $x \in \bar{\Omega}$. To this aim, the parameters α_i are positive scalars chosen in such a way that the matrix:

$$H + \text{diag}(\alpha_i)$$

has positive eigenvalues (with H, the Hessian of the function $f(x)$, and diag(α_i), the diagonal matrix composed of elements α_i). Adding the term $\sum_{i=1}^{n} \alpha_i (l_i - x_i)(u_i - x_i)$ to the function $f(x)$ allows us to obtain an underestimating function $u(x)$ having a Hessian with positive

eigenvalues, which characterizes a convex function. Using $\alpha_i = \alpha$, for all i, the condition on the eigenvalues of (1.27) amounts to require that α satisfies:

$$\alpha \geq \max\left\{0,\ -\frac{1}{2}\min_{k,\ l\leq x\leq u}\lambda_k(x)\right\},$$

where the $\lambda_k(x)$'s are the eigenvalues of the Hessian of $f(x)$. The transformation (1.27) and the way to compute the α_i's are discussed thoroughly in the paper of Adjiman *et al.* [9].

The convex underestimator (1.27) is applicable to each twice-differentiable function. However, it is expensive to compute and it does not exploit all the specificities of the function under study and thus, corresponds to a less tight convex underestimator than the one defined typically for this function. Therefore, convex underestimators have been specifically developed for functions like trigonometric ones (see Caratzoulas and Floudas [24]) for instance, in order to reduce the use of formula (1.27). In α*BB*, the convex relaxation problems involving the convex underestimators mentioned above are refined by branch-and-bound. For further details about the theoretical aspects of this method, we refer the reader to [9] and for a more practical point of view (implementation, branch-and-bound choices, etc.), to [8].

Piecewise linear relaxation

The method presented by Polisetty and Gatzke in [96] is now examined because it is, in a sense, quite close to ours since it uses a *piecewise linear relaxation*. To build this relaxation, the feasible domain is divided in pieces. Each nonlinear function (obtained by decomposing each function of the problem (1.25) in a sum of nonlinear functions) is then underestimated separately on each piece of the domain by a convex underestimator (by assuming that such an underestimator is known, otherwise, the method is not applicable). These convex underestimators are themselves underestimated on each piece by linear functions corresponding to their tangent at some points (see (1.5)), unless these convex underestimators are already linear. Computing the underestimators on each piece separately allows us to obtain a tighter underestimator than if it had been computed on the whole piece, as illustrated on Figure 1.1. It can be observed on this figure that the resulting piecewise underestimator is no longer a convex underestimator on the whole domain. However, the formulation of the piecewise relaxation is linear thanks to the introduction of *binary variables* and of the so-called *big-M constraints* employed for determining the linear underestimator adapted to the considered piece. Indeed, assume that for an univariate function, the feasible interval $[s_0, s_p]$ is divided in p pieces: $[s_0, s_1], [s_1, s_2], [s_2, s_3], ..., [s_{p-1}, s_p]$. Let M be a large value, δ a positive small one and b_i, $1 \leq i \leq p$, binary variables. The big-M formulation is then given by:

$$\begin{array}{lcl} -s_1 + x & \leq & M(1-b_1), \\ s_1 - x + \delta & \leq & M(1-b_2), \\ -s_2 + x & \leq & M(1-b_2), \\ s_2 - x + \delta & \leq & M(1-b_3), \\ -s_3 + x & \leq & M(1-b_3), \\ & \cdots & \\ s_{p-1} - x + \delta & \leq & M(1-b_p), \end{array} \tag{1.28}$$

together with:

$$\sum_{i=1}^{p} b_i = 1. \tag{1.29}$$

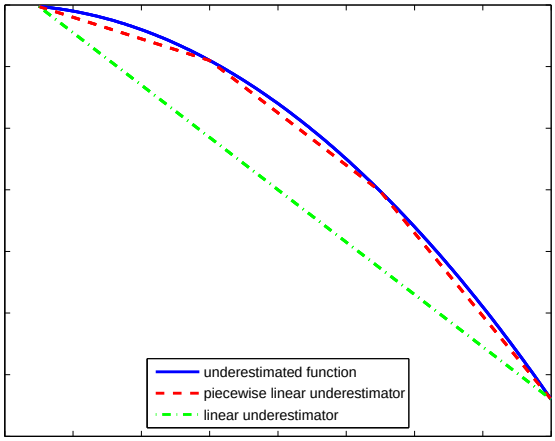


Figure 1.1: Piecewise linear underestimator versus linear underestimator.

This system implies that if x belongs to a piece $[s_{k-1}, s_k]$, the associated binary variable b_k is necessarily equal to one while the other b_i are equal to zero. For example, if $x \in [s_0, s_1]$, b_1 is forced to be equal to one and the first constraint amounts to $x \leq s_1$. The second constraint is given by $s_1 + \delta - M \leq x$, and the third one by $x \leq s_2 + M$. These constraints are obviously satisfied for $x \in [s_0, s_1]$ since M is a large value. Note that the parameter δ is used to define a unique piece to which x belongs in order to avoid problems at the boundaries of the pieces.

Denoting now $u_i(x)$ the linear underestimator of $f(x)$ computed on the i^{th} piece, the linear relaxation on the i^{th} piece employed in the piecewise linear relaxation problem is given by:

$$u_i(x) - M(1 - b_i). \tag{1.30}$$

Therefore, if x belongs to the i^{th} piece, b_i is equal to one and the underestimator is given by $u_i(x)$. Otherwise, b_i is equal to zero and the underestimator on the i^{th} piece corresponds to $u_i(x) - M$, which is a relaxation of the constraint for M sufficiently large. However, this relaxation has no influence because x does not belong to this piece. Replacing the nonlinear function by a new variable w, a constraint requiring that the function $u_i(x)$ is underestimating on each piece is imposed, that is:

$$u_i(x) - M(1 - b_i) \leq w \quad \forall i = 1, .., p. \tag{1.31}$$

As each constraint of (1.28), (1.29) and (1.31) is linear, the resulting piecewise problem is also linear but *mixed integer* because of the variables b_i. The use of tighter linear underestimators on each piece (but not linear on the whole domain) has been made possible by the introduction in the relaxation problem of binary variables. To refine the relaxation, branch-and-bound is again used. As a consequence, at each node of the branch-and-bound tree, a *mixed integer linear problem* must be solved. Methods conceived for solving such problems are cited at the begin of Section 1.3.

The method developed in this thesis, and more particularly in Chapter 3, also builds a piecewise linear relaxation, in a sense. However, the way of constructing this relaxation is different

since it is not based on a big-M formulation but on *special ordered sets* (see Section 2.2.3 for more details).

Summary of this section
The ideas of some global optimization methods have been brievly given. Obviously, there exist numerous other global optimization methods like interval methods (Hansen [57]), for instance. A survey of the subject has been realized by Neumaier in [89]. Several books like those of Horst and Pardalos [60] and Horst and Tuy [61] are references in global optimization. Finally, we mention the website [2] maintained by Neumaier and entirely dedicated to global optimization, and also the *COCONUT* project [107] which aims at combining the techniques developed in different optimization domains in order to produce efficient algorithms for global optimization.

1.3 Methods for discrete optimization problems

Concerning the discrete optimization problems, we directly focus on mixed integer nonlinear problems (MINLP) without detailing the methods used to solve mixed integer linear problems (MILP). Indeed, solving mixed integer linear problems is out of the scope of this thesis, even for the solution of MILP subproblems contrary to some methods. We thus limit ourselves to cite the most common optimization methods for MILP, which are *branch-and-bound, branch-and-cut, branch-and-price* and *cutting-plane* methods. Note that except the branch-and-price, the nonlinear version of these methods will be explained in the next subsection. The book of Nemhauser and Wolsey [87] treats in depth the subject of mixed integer linear optimization.

1.3.1 Methods to solve mixed integer nonlinear convex problems

This section presents some methods for solving mixed integer nonlinear and *convex* problems. For such problems, if *the set of possible values for the discrete variables is finite*, the methods described below are guaranteed to converge to the global optimum under mild assumptions, contrary to the nonconvex problems (see below). However, in some cases, the methods developed for mixed integer convex problems can be efficient to solve mixed integer nonlinear and nonconvex problems. Therefore, they can also be used in this context.

For the sake of clarity, we express the problem considered in this section by:

$$(P_I)\begin{cases} \min_{x,y} & f(x,y), \\ \text{s.t.} & g_i(x,y) \leq 0,\ i \in \mathcal{I}, \\ & x \in X,\ y \in Y, \end{cases} \tag{1.32}$$

where the variables x and y denote the continuous and discrete variables, respectively, the functions f and g_i $(i \in \mathcal{I})$ are convex, the set X is convex and the continuous relaxation of Y is also convex. Note that the continuous relaxation of (P_I) is obtained by replacing Y by its convex hull.

Branch-and-bound
Branch-and-bound for solving MINLP problems can be seen as a variant of the branch-and-bound detailed in Section 1.2.4 to refine the relaxation problems, although it has been developed first. Here, the relaxation problem used is the *continuous relaxation* of the convex MINLP

problem which is thus a convex nonlinear continuous problem. The function f and g_i remain unchanged. Accordingly, branch-and-bound is only used to satisfy the integer restrictions. Assume, for instance, that an integer variable y is equal to a noninteger value y^* at the current solution of a relaxation problem. To discard this infeasible solution, branching is used by imposing to the left subproblem:

$$y \leq \lfloor y^* \rfloor,$$

and to the right one:

$$\lceil y^* \rceil \leq y,$$

where $\lfloor y^* \rfloor$ and $\lceil y^* \rceil$ correspond to the closest integers to y^*, smaller or equal to it and larger or equal to it, respectively. Therefore, as one goes down in the tree, the bounds on the discrete variables are tighter and tighter and the discrete restrictions are gradually enforced.

At each node of the branch-and-bound tree, a nonlinear problem is solved. As this nonlinear problem is convex, the solution found for this problem is a global optimum and thus, the branch-and-bound is guaranteed to converge to a global optimum of the convex MINLP, if the set of possible values for the discrete variables is finite. Note that if the assumption on the convexity of the problem is removed, the lower bounds on the optimum value of the original problem corresponding to the solutions of the continuous nonconvex NLPs are not necessarily valid, unless global optimization methods are employed to solve these nonlinear problems at each node. As a consequence, the convergence to a global optimum cannot be ensured in this case.

MINLP_BB [74] and *SBB* [105] are two solvers based on a branch-and-bound framework designed to solve convex MINLP problems. The open-source code *Bonmin* (Basic Open-source Nonlinear Mixed INteger programming) [19] also offers the possibility to solve convex MINLP problems by a branch-and-bound process. These solvers use local optimization methods to solve the nonlinear problems and thus, guarantee the convergence to a global optimum only if the problem under study is convex.

Generalized Benders Decomposition and Outer Approximations

We now consider two other kinds of methods for convex MINLPs, namely the generalized Benders decomposition and the outer approximation methods. They are presented together since they are based on the same scheme. Indeed, they successively solve two problems: the so-called *primal problem* and *master problem*. On one hand, the primal problem corresponds to the nonlinear problem (1.32) in which the discrete variables have been fixed to some feasible values. The primal problem is thus a convex nonlinear continuous problem. By fixing the value of the discrete variables at y_k, this problem is given by:

$$(NLP_k) \begin{cases} \min_x & f(x, y_k), \\ \text{s.t.} & g_i(x, y_k) \leq 0,\ i \in \mathcal{I}, \\ & x \in X. \end{cases} \tag{1.33}$$

If this problem is feasible, its optimum value provides a valid *upper bound* on the optimum value of problem (1.32). On the other hand, the optimum value of the master problem, which is a mixed integer linear relaxation of problem (1.32), gives a valid *lower bound* on the optimum value of problem (1.32). The generalized Benders decomposition and outer approximation methods iterate the solution of the primal and master problems until the difference between the lower and upper bound is within a tolerance ϵ, in which case the algorithm stops since it

has detected a global optimum for problem (1.32). Note that both methods do not partition the feasible domain and thus, do not employ a branch-and-bound process. The generalized Benders decomposition and outer approximation methods differ in their way of building the master problem, which is given below for both methods.

Generalized Benders decomposition
This method due to Geoffrion [49] is an extension of the work of Benders [17]. Here, the master problem is based on the dual information provided at the solution of the nonlinear problem (NLP_k), which is reformulated as:

$$(NLP'_k)\begin{cases} \min_{x,y} & f(x,y), \\ \text{s.t.} & g_i(x,y) \leq 0,\ i \in \mathcal{I}, \\ & y = y_k, \\ & x \in X,\ y \in \text{conv}(Y), \end{cases} \tag{1.34}$$

where conv(Y) denotes the convex hull of Y. If (NLP'_k) is feasible, let (x_k, y_k) be its primal solution and μ_k the optimum Lagrangian multiplier associated to the equality constraint $y = y_k$. If the problem (NLP'_k) is infeasible, the NLP solver returns a point (x_k, y_k) which is the solution of the following feasibility problem:

$$(F'_k)\begin{cases} \min_{x,y} & \sum_{i \in C^+} w_i^k g_i^+(x,y), \\ \text{s.t.} & g_i(x,y) \leq 0,\ i \in C^-, \\ & y = y_k, \\ & x \in X,\ y \in \text{conv}(Y), \end{cases} \tag{1.35}$$

where C^- is the set of the indices of feasible constraint at the current point and C^+ its complementary, the function $g_i^+(x,y)$ is defined by $g_i^+(x,y) = \max(0, g_i(x,y))$ and $w_i^k \geq 0$ are weighting parameters not all zero. Let ν_k, the optimum Lagrangian multiplier associated to the equality constraint $y = y_k$ at the solution of (F'_k).

The master problem employed in the generalized Benders decomposition is given by:

$$(MILP'_k)\begin{cases} \min_{\eta,y} & \eta, \\ \text{s.t.} & \eta \geq f(x_j, y_j) + \mu_j^T (y_j - y) \quad \forall j \in F^+, \\ & 0 \geq \sum_{i \in C^+} (w_i^j g_i^+(x_j, y_j)) + \nu_j^T (y_j - y) \quad \forall j \in F^-, \\ & y \in Y, \end{cases} \tag{1.36}$$

where F^+ is the set of indices k for which the problem (NLP'_k) was feasible and F^- is its complementary.

Outer approximations
We now give the expression of the master problem used in outer approximation methods. The outer approximation methods have been introduced by Duran and Grossman [35] to solve mixed integer nonlinear convex problems whose objective and constraint functions are linear in the integer variables. They have next been generalized by Fletcher and Leyffer [43] to handle nonlinearities in the integer variables. In outer approximation methods, the master problem is based on tangential linearizations around (x_k, y_k) where (x_k, y_k) is the solution of (NLP_k) if

this problem is feasible or the solution of the following feasibility problem:

$$(F_k)\begin{cases} \min_x & \sum_{i\in C^+} w_i^k g_i^+(x,y_k), \\ \text{s.t.} & g_i(x,y_k)\le 0,\ i\in C^-, \\ & x\in X, \end{cases} \tag{1.37}$$

otherwise. Note that this latter problem is equivalent to (1.35). The master problem can be expressed as:

$$(MILP_k)\begin{cases} \min_{x,y,\eta} & \eta, \\ \text{s.t.} & \eta \ge f(x_j,y_j)+\nabla f(x_j,y_j)^T\begin{pmatrix} x-x_j \\ y-y_j \end{pmatrix} & \forall j\in F^+, \\ & 0\ge g(x_j,y_j)+\nabla g(x_j,y_j)^T\begin{pmatrix} x-x_j \\ y-y_j \end{pmatrix} & \forall j\in F^+, \\ & 0\ge g(x_j,y_j)+\nabla g(x_j,y_j)^T\begin{pmatrix} x-x_j \\ y-y_j \end{pmatrix} & \forall j\in F^-, \\ & x\in X,\ y\in Y. \end{cases} \tag{1.38}$$

Comparison of both methods
By comparing the master problems of both methods, it appears that it contains less variables and less constraints for the generalized Benders decomposition than for the outer approximation methods. However, in practice, the generalized Benders decomposition provides a less tight relaxation problem than the one generated by the outer approximation method. Variants of this latter method have been proposed in order to also handle equality constraints (Kocis and Grossmann [68]) and to exploit second order information by introducing quadratic terms in the problem (Fletcher and Leyffer [43]). Concerning the implementation of outer approximation methods, we can again cite *Bonmin* [19] and the solver *DICOPT* [69] which uses a third variant of the outer approximation method developed by Viswanathan and Grossmann [115], that is, an outer approximation extended by a penalty function and able to treat equality constraints.

LP/NLP Branch-and-Bound
The generalized Benders decomposition and the outer approximation methods need to solve a MILP problem at each iteration, which is costly. To avoid this drawback, Quesada and Grossmann have proposed in [98] an alternative, which has been generalized by Leyffer in [73] to handle nonlinearities in the integer variables. The idea of the proposed methods is to integrate the outer approximation method in a branch-and-bound framework. More precisely, instead of solving a MILP at each iteration, only one MILP is solved by using branch-and-bound. The MILP which is solved corresponds to the first master problem built by the outer approximation method. By using branch-and-bound, the problem at the root node of the tree corresponds to the continuous relaxation of this MILP. Branching is then employed to try to satisfy the discrete restrictions. But contrary to the classical branch-and-bound for MILP, once a solution satisfying the integer restrictions is found in the branching tree, the branch-and-bound process is stopped and the primal problem (1.33) is solved for this integer solution, exactly as in outer approximation. New *cuts* (the tangential linearizations at the current point) can then be generated to refine the relaxation problem. These cuts are next added to each subproblem associated to a node of the stack of branch-and-bound in order to improve the quality of all relaxation problems remaining to treat. This is why this method is also referred to as a *branch-and-cut method*.

Bonami *et al.* have recently proposed an hybrid approach [19] which solves a larger number of NLPs during the branch-and-bound process (for example, every l nodes). The LP/NLP branch-and-bound method and the latter hybrid approach have been implemented in *Bonmin*. At the same time, Abhishek *et al.* [5] have developed *FilMINT*, a solver also based on the LP/NLP branch-and-bound algorithm.

Extended cutting plane

The last method designed to solve convex MINLP on which we focus is the *extended cutting plane* method. It is due to Westerlund and Petterson [119] which have extended to MINLP the cutting plane method proposed by Kelley in [65] to solve NLP. In order to apply the extended cutting plane method, the problem (1.32) is reformulated as:

$$(P_\eta)\begin{cases} \min & \eta, \\ \text{s.t.} & g_i(x,y) \leq 0,\ \forall i \in \mathcal{I} \cup \{i_f\}, \\ & x \in X,\ y \in Y, \end{cases} \tag{1.39}$$

where $g_{i_f} = f(x,y) - \eta$. The general idea of the extended cutting plane method is to solve a sequence of tighter and tighter MILPs obtained by adding to the current MILP the linearization of one constraint of the problem P_η at each iteration. More precisely, the first MILP problem is composed of the linear constraints and of the linearization of the *most violated constraint* of (P_η) at a given feasible point (x_0, y_0). Therefore, defining j_0 by $j_0 = \text{argmax}_i g_i(x_0, y_0)$, the following linear constraint is added to the problem:

$$g_{j_0}(x_0,y_0) + \nabla g_{j_0}(x_0,y_0)^T \begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix} \leq 0.$$

As a consequence, the first MILP problem can be expressed as:

$$(MILP_0^{''})\begin{cases} \min & \eta, \\ \text{s.t.} & g_i(x,y) \leq 0,\ \forall i \in \mathcal{I}_\mathcal{L}, \\ & g_{j_0}(x_0,y_0) + \nabla g_{j_0}(x_0,y_0)^T \begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix} \leq 0, \\ & x \in X,\ y \in Y, \end{cases} \tag{1.40}$$

where $\mathcal{I}_\mathcal{L} \subset \mathcal{I} \cup \{i_f\}$ is the set of indices of linear constraints. The most violated constraint is then looked for at the solution (x_1, y_1) of $(MILP_0^{''})$ and the process is repeated by adding its linearization to $(MILP_0^{''})$. The extended cutting plane solves a sequence of $(MILP_k^{''})$ until the maximum violation of the constraints at the current point is smaller than a fixed tolerance. The optimum values of these problems generate a sequence of non-decreasing lower bounds on the optimum value of the original problem. However, contrary to the last methods described above, no upper bound is generated by using an NLP solver, which can prevent the method from having a fast convergence. Indeed, since the extended cutting plane method does not employ an NLP solver which is usually based on Newton-type methods, it cannot benefit from the fast convergence of these methods. As a consequence, the extended cutting plane method is a first-order method only.

Let us consider the case of nonconvex problems. To handle such problems, the extended cutting plane algorithm has been modified to take into account the fact that the linearizations of

the constraints may discard feasible parts of the domain. For example, if it appears that a linearization is larger at the current point x_k than the constraint that it attempted to underestimate at a previous iteration, this linearization is removed from the problem $(MILP_k^{''})$. For more details about this method, we refer the reader to the papers of Westerlund and Petersson [119] and Westerlund *et al.* [121] which present the method for convex and nonconvex MINLPs respectively, and to [97] and [120] for extensions of the method. Note that this extended cutting plane method has been implemented in the software *Alpha-ECP* [118].

The main ideas of some of the most popular deterministic methods to solve mixed integer nonlinear programs have been presented. The described methods however do not cover all the existing methods. We refer the reader to the paper of Grossmann [55] for a more complete survey on the subject.

1.3.2 Global methods to solve mixed integer nonlinear nonconvex problems

Since the methods described above cannot guarantee the convergence to a global solution (or even to a feasible solution) for mixed integer nonlinear and *nonconvex* programs, global optimization methods have been proposed in the literature to remedy this. Some of them are now reviewed.

Branch-and-reduce
The branch-and-reduce method implemented in the solver *BARON* and presented in Section 1.2.4 has been conceived to treat discrete variables. Indeed, the method is included in a branch-and-bound framework. When the problem involves discrete variables, the considered relaxation is a convex and continuous relaxation of the MINLP. In this case, branching is performed to refine the convex relaxations as well as to satisfy the discrete restrictions. Again, the convergence to the global optimum is ensured under mild assumptions.

Methods based on branch-and-bound: Extensions of α*BB*
The method implemented in α*BB* has been extended to be applicable in the discrete case. Two methods have been proposed. The first one, *SMIN-*α*BB*, is applicable on problems where the discrete variables are only involved in linear and bilinear functions. It is based on the convex underestimation of the continuous functions and amounts to a branch-and-bound process where a convex MINLP is solved at each node. The second strategy, *GMIN-*α*BB*, can treat a larger class of problems, that is, problems involving twice continuously differentiable functions. It is based on the convex relaxation of the whole problem and also uses a branch-and-bound process but here, a convex NLP is solved at each node. More details about these two algorithms can be found in Adjiman [7].

Interval analysis methods
These methods proposed by Vaidyanathan and El-Halwagi [111] can be seen as a branch-and-bound approach since they reduce the domain by partitioning and bound the objective function of the problem on each subdomain. However, the bounds are not obtained by solving optimization problems, which distinguish the *interval analysis methods* from branch-and-bound ones.

Here, valid bounds are computed by *interval arithmetic* techniques (see Moore [85] and Neumaier [88]). However, these bounds can be poor and specific techniques must be employed to accelerate the convergence.

Other global optimization methods
In addition to these methods, Smith and Pantelides have proposed in [108] a spatial branch-and-bound, Pörn and Westerlund have developed an extended cutting plane method [97] for nonconvex MINLPs which ensures the convergence to the global optimum for a larger class of problems than for the convex ones and Kesavan *et al.* [66] have extended the outer approximation methods in order to find the global optimum of separable and nonconvex MINLPs. An overview of the deterministic methods developed to solve nonconvex (and also convex) MINLPs and the assumption under which they are applicable can be found in Adjiman [10]. We also refer the reader to the paper [46] of Floudas *et al.* which presents the recent advances in the global optimization domain. Geometric, algebraic and combinatorial approaches for MINLP are also given in Weismantel [117].

An alternative to the deterministic methods described above are the *approximation methods*. These latter build an approximation problem from the original one and solve this approximation problem which is easier to solve than the original one and for which the guarantee to obtain the global minimum is ensured. If the approximation problem is sufficiently accurate, its optimum solution is close to the one of the original problem, which can be easily reached from the optimum solution of the approximation problem. In this thesis, we focus on an approximation method (see Martin *et al.* [80]), which approximates a MINLP by a MILP involving *special ordered sets*. More details on this method will be given in Section 2.2.3. Finally, we mention the existence of heuristics to solve MINLP like simulated annealing (see [25] and [67]), tabu search [51], evolutionary algorithms [13] and also, heuristics based on branch-and-cut as used by the software LaGO [3].

1.4 Conclusion

This chapter aimed at fixing some basic notions and definitions useful in this thesis and also to give a brief overview of the existing optimization methods to solve different classes of problems. The list of the described methods being non-exhaustive and the methods being not presented in details, we invite the reader interested in a particular topic to consult the proposed references in the related sections.

We also recommend NEOS, an online server for optimization developed by the Optimization Technology Center (Argonne National Laboratory and Northwestern University). NEOS gives access to a lot of solvers presented in this chapter. It offers to the user the possibility of submitting a problem online and of solving it also online by the solver that he has chosen. More details about NEOS can be found in [27], [33] and [54] and on its internet address:

http://www-neos.mcs.anl.gov/

Note that a guide for optimization is also available on this website.

Chapter 2

Solution of a mixed integer nonlinear nonconvex problem related to power systems analysis

In an era where the world population continually increases while the natural resources decrease, the energy management has become an important topic. In order to tackle it in an optimal way, the underlying problems of the energy management are often modelled as optimization problems. This is the case of the problem treated in this thesis which can be expressed as a *mixed integer nonlinear and nonconvex optimization problem*. The considered problem is an optimal power flow problem which arises in the management of transmission systems. Especially, we focus on a problem known as *Tertiary Voltage Control* problem, or more briefly *TVC* problem but the method developed in this thesis is also applicable on general optimal power flow problems. The *TVC* problem has been provided to us by *Tractebel Engineering*, an important engineering consultancy company for energy and infrastructure.

This chapter aims at introducing the problem under study and at investigating the question of the best way to solve it. After explaining the physical meaning of the problem and detailing its variables and constraints, we first explain the heuristics used by Tractebel to solve the problem. We then describe the results obtained by using *MINLP_BB*, a solver dedicated to the solution of mixed integer nonlinear convex problems. As the observed results lead to think that a method preferring the solution of linear problems instead of nonlinear ones is better, we consider such a method. More precisely, we detail an approximation method for the solution of mixed integer nonlinear problems. However, this method can fail to converge to an optimum. To remedy this, a global optimization method is finally considered.

2.1 Presentation of the treated problem

The *TVC* problem takes place in the framework of the *Optimal Power Flow*, shortly referred to as OPF, (see Bacher [14], Momoh *et al.* [84]). The goal of the latter is to optimize some objective function (power loss minimization or transfer capability maximization, for example) by adjusting system control settings while satisfying operational and physical constraints (see Section 2.1.2). Therefore, the OPF is an optimization problem. The OPF applies on steady-state operating conditions. Actually, the aim of the OPF is to *predict* the effects of some planned

expansion or to determine the best operating conditions of an existing system.

In order to analyze the *TVC* problem, we first consider the variables and constraints involved in the OPF. To this aim, some basic electricity notions linked to the problem are introduced. We then focus on the specificities of the OPF and finally, give the complete formulation of the *TVC* problem.

2.1.1 Variables of the problem

Electrical networks mainly depend on three components:

- the *voltage* which corresponds to the electric potential difference between two points of an electrical circuit and which is related to the amount of energy needed to move an electric charge from one of these two points to the other;
- the *current* which represents the movement (or the flow) of electricity;
- the *power* which expresses the quantity of energy transferred from a system to another.

The OPF is based on the *alternating current* which means that contrary to the directed current, the magnitude and the direction of the current periodically vary. In this context, the three previous notions (voltage, current and power) are complex. A complex number x can be written as:

$$x = a + jb \quad \text{as well as} \quad x = c(\cos(d) + j\sin(d)), \tag{2.1}$$

where a, b, c and d are reals such that a and b represent the real and imaginary parts of x respectively, c and d are known as the modulus and the argument of x respectively, while j corresponds to the imaginary unit (usually denoted i anywhere else than in the electrical engineering domain). The polar coordinates (c, d) involve trigonometric functions contrary to the Cartesian coordinates (a, b). However, Tractebel prefers employing the formulation with polar coordinates because of its experiment. Indeed, the use of Cartesian coordinates introduces in the OPF problems functions like square roots of sum of squares which do not appear by using polar coordinates.

Moreover, as indicated by its name, the OPF focuses more on the notion of power than on voltage or current. Note that the real and imaginary parts of the power have both a physical interpretation. Indeed, the power, denoted S, and referred to as *apparent power*, is divided in two parts: the *real power* P and the *reactive power* Q which correspond respectively to the real and imaginary parts of the quantity S ($S = P + iQ$). On one hand, the real power, also called active or true power, is the equivalent of the power for the directed current. This is the amount of power used or dissipated in a electrical circuit and which is employed for the consumption of the end-user. On the other hand, the reactive power corresponds to the quantity of power which returns to the energy source at each cycle. It is produced to maintain the system and to ensure a steady voltage.

Let us now consider the representation of an electrical network by a *directed graph*, that is, by a set of *nodes* linked together by *directed branches*. To each component of the graph, node or branch, is associated a set of variables and parameters, as detailed below.

Nodes
We consider two kinds of nodes:

- the *generator nodes* which are associated to generators (devices producing electrical energy),
- the *simple nodes*, the other ones.

In the following, the sets of generator nodes and of simple nodes will be respectively denoted $\mathcal{N}_\mathcal{G}$ and $\mathcal{N}_\mathcal{S}$, while $\mathcal{N}$ will be used for the set of all nodes ($\mathcal{N} = \mathcal{N}_\mathcal{G} \cup \mathcal{N}_\mathcal{S}$). With each node, we associate a voltage. By (2.1), this variable is divided in two parts. For a node denoted i, referred to as node i, they are given by:

- ν_i, the modulus of the voltage at node i,
- θ_i, the argument of the voltage at node i.

Furthermore, the following parameters are also employed for each node:

- $\nu_{\min_i}$ and $\nu_{\max_i}$ that bound below and above the modulus of the voltage at node i,
- Q_{0_i}, a parameter associated to the reactive power compensation at node i, that is, the reactive power which is released by some components of the network such as capacitors (see Section 2.1.2),
- P_{c_i}, the real power consumed at node i,
- Q_{c_i}, the reactive power consumed at node i.

The variables and parameters depending on a node differ according to the kind of node (generator or simple). As the generator nodes are associated to an energy source, they are characterized by two more variables:

- P_i, the generated real power at node i,
- Q_i, the generated reactive power at node i.

As the produced power cannot be infinite, parameters are used to bound these variables below and above:

- $P_{\min_i}$ and $P_{\max_i}$,
- $Q_{\min_i}$ and $Q_{\max_i}$.

Branches

Like for nodes, we consider two types of branches, depending, this time, if a *transformer* is present or not on these branches. A transformer is a device allowing us to modify the amplitude of a voltage or of an alternating current. The representation of a transformer is given in Figure 2.1. On this figure, it can be observed that a transformer is composed of two windings. One of these windings is linked to the voltage source. It is referred to as primary, while the other winding is called secondary since it is associated to a voltage induced by the primary. We thus have:

- *branches with transformer*,
- *simple branches* (the other ones).

In the following, the sets of branches with transformers and of simple branches will be denoted $\mathcal{B}_\mathcal{T}$ and $\mathcal{B}_\mathcal{S}$, respectively. Among these branches, only the ones with transformer are associated to a variable which is:

- r_j, the ratio of voltage at branch j.

The *ratio of voltage* is the ratio between the induced voltages for the primary and secondary. In case of an ideal transformer with no energy loss, this ratio is equal to the number of turns of

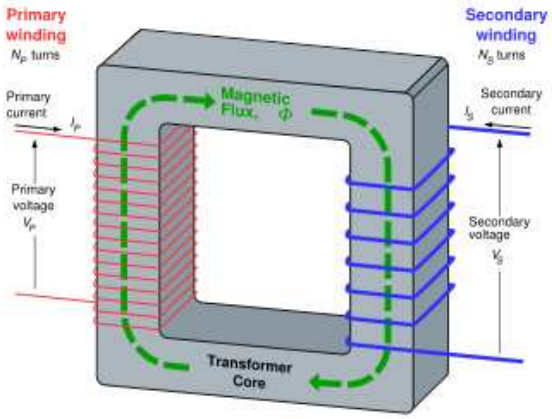


Figure 2.1: Representation of a transformer [1].

wire for the primary divided by the number of turns of wire for the secondary ($\frac{N_p}{N_s}$ in Figure 2.1). Therefore, the ratio of voltage is not a continuous number and can take only a *discrete set* of values comprised between two parameters:

- $r_{\min_j}$ and $r_{\max_j}$.

After the variables depending on branches, the parameters are now examined. With each branch, simple or with transformer, we associate a parameter called *admittance* which expresses the ease of an alternating current to flow through an electrical circuit. The admittance is composed of two parts: the *conductance* and the *susceptance*. The conductance measures the ability of a material to conduct electricity while the susceptance represents the ease of a system to free stored energy. As a direct consequence, the admittance depends on the materials composing the electrical system. While the conductance and susceptance correspond respectively to the real and imaginary parts of the admittance, the modulus and argument of (2.1) are used in the modelling to express the admittance of a branch j:

- y_j, the modulus of the admittance of branch j,
- ζ_j, the argument of the admittance of branch j.

Moreover, any branch is linked to the earth by means of a component called *shunt* which is also associated to an admittance. Depending if the shunt is connected to a branch with transformer or to a simple branch, a different definition is used for the admittance: modulus and argument or real and imaginary parts. With a shunt linked to a branch j with transformer, we associate:

- y_{0_j}, the modulus of the admittance of the shunt,
- ζ_{0_j}, the argument of the admittance of the shunt,

while a shunt connected to a simple branch j is characterized by:

- g, the conductance of the shunt,
- h, the susceptance of the shunt.

2.1.2 Constraints of the problem

After having listed the variables and parameters involved in the optimization problem provided by Tractebel Engineering, we now consider the constraints imposed on this problem.

Bound constraints
As previously underlined, the majority of the variables (ν_i, P_i, Q_i and r_j) are subject to bound constraints.

Power Flow Equations
In the OPF formulation, the electrical network is also subject to *power flow equations* which concern the conservation of power inside an electrical circuit. They can be deduced from Tellegen's theorem [94] and require that:

The total power in an electrical circuit is equal to zero, or equivalently, the total power generated in a network is equal to the total power dissipated in this network.

It can be shown (see [94]) that this law is implied by the well known Kirchoff's laws and thus expresses a physical reality. As the power is a complex value, the law on power conservation can be decomposed according to the real and reactive powers. Mathematically, in the model provided by Tractebel Engineering, these constraints are expressed as:

$$P_i - P_{c_i} - \sum_{j \in S_i^e} P_j - \sum_{j \in S_i^o} P_j - \sum_{j \in T_i^e} P_j - \sum_{j \in T_i^o} P_j = 0, \quad \forall i \in \mathcal{N}, \tag{2.2}$$

$$Q_i + \nu_i^2 Q_{0_i} - Q_{c_i} - \sum_{j \in S_i^e} Q_j - \sum_{j \in S_i^o} Q_j - \sum_{j \in T_i^e} Q_j - \sum_{j \in T_i^o} Q_j = 0, \quad \forall i \in \mathcal{N}, \tag{2.3}$$

where
- S_i^e is the set of simple branches coming from node i,
- S_i^o is the set of simple branches going to node i,
- T_i^e is the set of branches with transformer coming from node i,
- T_i^o is the set of branches with transformer going to node i,
- P_j is the real power on branch j,
- Q_j is the reactive power on branch j.

The real and reactive powers on a branch can be written in terms of the variables and parameters described above as shown in Platbrood [95]. For a branch denoted j linking node i to node k, the expressions of the real and reactive powers are given by:

$$\begin{array}{ll} P_j = \nu_i^2 \left(y_j \cos(\zeta_j) + g_j\right) - \nu_i \nu_k y_j \cos(\zeta_j + \theta_i - \theta_k) & \text{if } j \in S_i^e, \\ Q_j = \nu_i^2 \left(y_j \sin(\zeta_j) - h_j\right) - \nu_i \nu_k y_j \sin(\zeta_j + \theta_i - \theta_k) & \text{if } j \in S_i^e, \end{array}$$

$$\begin{array}{ll} P_j = \nu_i^2 r_j^2 y_j \cos(\zeta_j) - \nu_i \nu_k r_j y_j \cos(\zeta_j + \theta_i - \theta_k) & \text{if } j \in T_i^e, \\ Q_j = \nu_i^2 r_j^2 y_j \sin(\zeta_j) - \nu_i \nu_k r_j y_j \sin(\zeta_j + \theta_i - \theta_k) & \text{if } j \in T_i^e. \end{array} \tag{2.4}$$

If the branch j links node k to node i, the real and reactive power are written as:

$$\begin{array}{ll} P_j = \nu_k^2 \left(y_j \cos(\zeta_j) + g_j\right) - \nu_i \nu_k y_j \cos(\zeta_j + \theta_k - \theta_i) & \text{if } j \in S_i^o, \\ Q_j = \nu_k^2 \left(y_j \sin(\zeta_j) - h_j\right) - \nu_i \nu_k y_j \sin(\zeta_j + \theta_k - \theta_i) & \text{if } j \in S_i^o, \end{array}$$

$$\begin{array}{ll} P_j = \nu_k^2 \left(y_j \cos(\zeta_j) + y_{0_j} \cos(\zeta_{0_j})\right) - \nu_i \nu_k r_j y_j \cos(\zeta_j + \theta_k - \theta_i) & \text{if } j \in T_i^o, \\ Q_j = \nu_k^2 \left(y_j \sin(\zeta_j) + y_{0_j} \sin(\zeta_{0_j})\right) - \nu_i \nu_k r_j y_j \sin(\zeta_j + \theta_k - \theta_i) & \text{if } j \in T_i^o. \end{array} \tag{2.5}$$

Note that the equations associated to a simple branch are symmetric for a branch linking node i to node k and one linking node k to node i. This is not the case for branches with transformer due to the presence of a transformer on these branches.

By replacing the expressions of the powers on a branch given by (2.4) and (2.5) in (2.2) and (2.3), it can be seen that the power flow equations are equality constraints implying trigonometric functions, which make them nonconvex. Furthermore, as these constraints express physical realities, they must be absolutely satisfied.

Discrete restrictions

In addition to general constraints, the problem is also subject to some restrictions on the variables in the sense that some variables are allowed to take only a discrete set of values. In the considered problem, there are two kinds of such variables: the *ratio of voltage* previously mentioned and binary variables associated to the *capacitor banks*, as detailed below.

1. *Ratio of voltage*
   As explained in Section 2.1.1, the ratio of voltage is a ratio between two integer numbers N_p and N_s. In practice, N_p can take q values while the value of N_s is fixed. Therefore, the ratio of voltage is a variable which can take q *fixed values*, not necessarily integer. In the modelling used in this thesis, we have assumed, in order to simplify, that the possible values for the ratio of voltage on a branch j are *equally spaced* between its lower and upper bounds, $r_{\min_j}$ and $r_{\max_j}$. Moreover, we have also supposed that there exist eleven possible values for the ratio of voltage. Therefore, we have introduced an integer variable z_j belonging to $[0, 10]$ which satisfies:

$$r_j = r_{\min_j} + z_j \frac{r_{\max_j} - r_{\min_j}}{10}, \quad j \in \mathcal{B}_\mathcal{T}. \tag{2.6}$$

   In the suggested modelling, the ratio of voltage r_j is treated as a continuous variable while the variable z_j is considered as an integer one. The satisfaction of the discrete restriction on r_j is obtained by means of the constraint (2.6) and the integer restriction on z_j.

   Note that the simplifying assumption on the equal spacing of the possible values for the ratio of voltage can be relaxed by using another formulation requiring that:

$$r_j = r_{j_k} z_{j_k}, \quad 1 \leq k \leq q,\ j \in \mathcal{B}_\mathcal{T}, \tag{2.7}$$

$$\sum_{k=1}^{q} z_{j_k} = 1, \tag{2.8}$$

$$z_{j_k} \text{ binary}, \quad 1 \leq k \leq q. \tag{2.9}$$

   In this formulation, the q possible values for r_j are given by $\{r_{j_k}\}_{k=1}^{q}$. The constraint (2.8) compels only one z_{j_k} to be equal to one and the others to be equal to zero. As a binary variable z_{j_k} is associated to each of the possible values r_{j_k}, r_j is equal to one of the values r_{j_k} by (2.7). A set of variables where only one variable is allowed to be nonzero is called *special ordered set of type 1*. Observe that the formulation (2.6) involves two variables (r_j and z_j) while $q + 1$ variables (r_j and $\{z_{j_k}\}_{k=1}^{q}$) are needed for constraints (2.7), (2.8) and (2.9).

2. *Binary variable associated to a capacitor bank*
   A *capacitor bank* is an electrical device able to store reactive power and to free the accumulated power. In the proposed model, the constraints (2.3) assume, by means of the term $\nu_i^2 Q_{0_i}$, that the stored reactive power is always released. However, in practice, it is not always the case since it arises that the reactive power remains accumulated in the capacitor. Therefore, to improve the model, *binary* variables a_i have been added to take this characteristic into account. With these variables, the equalities (2.3) are replaced by:

$$Q_i + \boldsymbol{a_i}\nu_i^2 Q_{0_i} - Q_{c_i} - \sum_{j \in S_i^e} Q_j - \sum_{j \in S_i^o} Q_j - \sum_{j \in T_i^e} Q_j - \sum_{j \in T_i^o} Q_j = 0, \quad \forall i \in \mathcal{N}. \quad (2.10)$$

   When a_i is equal to zero, the reactive power remains in the capacitor at node i while if it is equal to one, the entirety of this power is released. Note that values for a_i different from zero and one are not consistent with the reality because when the capacitor frees power, the entirety of the stored power is released. As a consequence, the variables a_i must be binary.

2.1.3 TVC problem

The general specificities of an OPF problem being given, we now focus on the *TVC* problem. In alternating current networks, the reactive power transmission produces voltage drops and losses. In order to have a situation as regular as possible, a *tertiary voltage control* is applied. Its goal amounts to try that the produced reactive power at each generator node i remains more or less constant around a fixed value denoted obj_i, in such a way that the generated reactive power Q_i does not reach its bounds too often. As a consequence, the objective function of the *TVC* problem is given by:

$$\min \sum_{i \in \mathcal{N}_\mathcal{G}} w_i \, (Q_i - obj_i)^2, \quad (2.11)$$

where w_i are weighting parameters. Indeed, as the range of the generated reactive power can be different for every node, the weighting parameters are used to scale these quantities. In fact, (2.11) can be seen as a weighted least squares function.

The *TVC* problem also involves an additional constraint employed to control the voltage on the branches connected to other networks (of other countries for example):

$$\sum_{j \in \mathcal{S}} Q_j = obj_0, \quad (2.12)$$

where $\mathcal{S}$ is the set of simple branches connected to other networks and obj_0 is a fixed parameter.

Note that there also exist primary and secondary voltage controls. The primary voltage control is used to modify the voltage directly at the devices of the network (generators and transformers, for example) when a voltage variation is detected by the voltage regulators of these devices. The goal of the secondary voltage control is to coordinate the action of the voltage and reactive power control devices of the network on a local region in order that the voltage remains at a fixed level. Finally, the tertiary voltage control needs the solution of an optimization problem involving measurements taken on the network, in order to adjust the settings of devices (generators, inductances, capacitors, etc.).

By grouping together (2.6), (2.11) as well as (2.2), (2.10) and (2.12) in which the real and reactive powers on branches have been replaced by their expressions given in (2.4) and (2.5), and by bounding the variables in the way explained above, we finally obtain the complete *TVC* problem that we aim at solving (see below). In this formulation, the variables are highlighted in bold and the notation j_{ik} has been used to stress the fact that branch j_{ik} links node i to node k.

$$
(TVC)\left\{
\begin{array}{ll}
\min & \sum_{i\in\mathcal{N}_\mathcal{G}} w_i\,(\boldsymbol{Q_i}-obj_i)^2,\\[2ex]
\text{s.t.} & \boldsymbol{P_i}-P_{c_i}-\sum_{j_{ik}\in S_i^e}\left(\boldsymbol{\nu_i^2}\,(y_{j_{ik}}\cos(\zeta_{j_{ik}})+g_{j_{ik}})-\boldsymbol{\nu_i\nu_k}y_{j_{ik}}\cos(\zeta_{j_{ik}}+\boldsymbol{\theta_i}-\boldsymbol{\theta_k})\right)\\
& \quad-\sum_{j_{ki}\in S_i^o}\left(\boldsymbol{\nu_k^2}\,(y_{j_{ki}}\cos(\zeta_{j_{ki}})+g_{j_{ki}})-\boldsymbol{\nu_i\nu_k}y_{j_{ki}}\cos(\zeta_{j_{ki}}+\boldsymbol{\theta_k}-\boldsymbol{\theta_i})\right)\\
& \quad-\sum_{j_{ik}\in T_i^e}\left(\boldsymbol{\nu_i^2 r_{j_{ik}}^2}y_{j_{ik}}\cos(\zeta_{j_{ik}})-\boldsymbol{\nu_i\nu_k r_{j_{ik}}}y_{j_{ik}}\cos(\zeta_{j_{ik}}+\boldsymbol{\theta_i}-\boldsymbol{\theta_k})\right)\\
& \quad-\sum_{j_{ki}\in T_i^o}\left(\boldsymbol{\nu_k^2}\,(y_{j_{ki}}\cos(\zeta_{j_{ki}})+y_{0_{j_{ki}}}\cos(\zeta_{0_{j_{ki}}}))\right.\\
& \qquad\left.-\boldsymbol{\nu_i\nu_k r_{j_{ki}}}y_{j_{ki}}\cos(\zeta_{j_{ki}}+\boldsymbol{\theta_k}-\boldsymbol{\theta_i})\right)\\
& \quad=0,\quad \forall i\in\mathcal{N},\\[2ex]
& \boldsymbol{Q_i}+\boldsymbol{a_i\nu_i^2}Q_{0_i}-Q_{c_i}\\
& \quad-\sum_{j_{ik}\in S_i^e}\left(\boldsymbol{\nu_i^2}\,(y_{j_{ik}}\sin(\zeta_{j_{ik}})-h_{j_{ik}})-\boldsymbol{\nu_i\nu_k}y_{j_{ik}}\sin(\zeta_{j_{ik}}+\boldsymbol{\theta_i}-\boldsymbol{\theta_k})\right)\\
& \quad-\sum_{j_{ki}\in S_i^o}\left(\boldsymbol{\nu_k^2}\,(y_{j_{ki}}\sin(\zeta_{j_{ki}})-h_{j_{ki}})-\boldsymbol{\nu_i\nu_k}y_{j_{ki}}\sin(\zeta_{j_{ki}}+\boldsymbol{\theta_k}-\boldsymbol{\theta_i})\right)\\
& \quad-\sum_{j_{ik}\in T_i^e}\left(\boldsymbol{\nu_i^2 r_{j_{ik}}^2}y_{j_{ik}}\sin(\zeta_{j_{ik}})-\boldsymbol{\nu_i\nu_k r_{j_{ik}}}y_{j_{ik}}\sin(\zeta_{j_{ik}}+\boldsymbol{\theta_i}-\boldsymbol{\theta_k})\right)\\
& \quad-\sum_{j_{ki}\in T_i^o}\left(\boldsymbol{\nu_k^2}\,(y_{j_{ki}}\sin(\zeta_{j_{ki}})+y_{0_{j_{ki}}}\sin(\zeta_{0_{j_{ki}}}))\right.\\
& \qquad\left.-\boldsymbol{\nu_i\nu_k r_{j_{ki}}}y_{j_{ki}}\sin(\zeta_{j_{ki}}+\boldsymbol{\theta_k}-\boldsymbol{\theta_i})\right)\\
& \quad=0,\quad \forall i\in\mathcal{N},\\[2ex]
& \sum_{j_{ik}\in\mathcal{S}}\left(\boldsymbol{\nu_i^2}\,(y_{j_{ik}}\sin(\zeta_{j_{ik}})-h_{j_{ik}})-\boldsymbol{\nu_i\nu_k}y_{j_{ik}}\sin(\zeta_{j_{ik}}+\boldsymbol{\theta_i}-\boldsymbol{\theta_k})\right)=obj_0,\\[2ex]
& \boldsymbol{r_j}=r_{\min_j}+\boldsymbol{z_j}\frac{r_{\max_j}-r_{\min_j}}{10},\quad j\in\mathcal{B}_\mathcal{T},\\[2ex]
& P_{\min_i}\le\boldsymbol{P_i}\le P_{\max_i},\quad i\in\mathcal{N},\\
& Q_{\min_i}\le\boldsymbol{Q_i}\le Q_{\max_i},\quad i\in\mathcal{N},\\
& \nu_{\min_i}\le\boldsymbol{\nu_i}\le\nu_{\max_i},\quad i\in\mathcal{N},\\[2ex]
& 0\le\boldsymbol{z_j}\le 10,\ \boldsymbol{z_j}\text{ integer},\quad j\in\mathcal{B}_\mathcal{T},\\
& \boldsymbol{a_i}\text{ binary},\quad i\in\mathcal{N}.
\end{array}
\right.
$$

The *TVC* problem is thus a *mixed integer nonlinear and nonconvex problem*. For more details about electricity concepts, we refer the reader to Wildi and Sybille [122] and for a more extensive description of the modelling of the *TVC* problem, to the work of Platbrood [95] which focuses on the continuous relaxation of this problem.

2.2 Solution of the TVC problem

This section investigates some methods to solve the *TVC* problem.

2.2.1 Heuristics employed by Tractebel

At present, Tractebel Engineering does not employ a solver typically conceived to treat mixed integer and nonlinear problems to solve the *TVC* problem but uses a *heuristics*. This latter consists in solving the *continuous relaxation* of the problem (see Section 1.2.4) and next, for the discrete variables which are close within some accuracy to a feasible discrete value at the solution, in *fixing the value* of these variables to this feasible discrete value. Then, the process is repeated: the continuous relaxation of the problem in which the value of some discrete variables has been fixed is solved and the discrete variables which have a value sufficiently close to a feasible discrete value at the solution are fixed to this value. To compel the convergence, the accuracy needed to fix the value of the discrete variables is relaxed as soon as continuous relaxations of the problem are solved. However, this technique remains a heuristics and thus, there is no guarantee to converge. If a solution is found, there is neither any insurance that it corresponds to a "good" optimum. Therefore, a robuster method is desirable.

2.2.2 A mixed integer nonlinear convex solver

We have then applied *MINLP_BB* (see Section 1.3), a solver especially built to solve mixed integer and nonlinear problems and available on NEOS (see Chapter 1) to the *TVC* problem. This solver is able to detect a global optimum of a mixed integer and nonlinear optimization problem if this problem is convex. If it is not the case, there is no guarantee to find a global optimum. However, in practice, *MINLP_BB* can also give good results on mixed integer nonlinear and nonconvex problems. This solver has thus been tested on the *TVC* problem with a medium set of data (34 nodes and 48 branches) provided by Tractebel Engineering. The results obtained were encouraging since the detected optimum seemed to be the global one. Indeed, no better solution has been found with other techniques and by using other starting points. We then have experimented *MINLP_BB* on a larger set of data corresponding to the Belgian network (about 1500 nodes and 2000 branches), but here the solver failed to converge. After several days, no solution was found. The size of the problem and also the *cost* of the solution of one nonlinear problem (*MINLP_BB* solves a lot of such problems) can be an explanation.

2.2.3 A linear approximation method

In order to avoid the drawback cited above, an idea is to replace the solution of the nonlinear problems by that of *linear problems* which are less expensive to solve. Moreover, the *TVC* problem comprises, among others, trigonometric functions, which make it difficult to solve, due to the highly *nonconvex* behaviour of these functions. Therefore, the search for an optimum, not necessarily the global one, can be jeopardized if the starting point for the nonlinear solver is not judiciously chosen. To remedy this, a popular idea (see Martin *et al.* [80] and Tomlin [110], for example) which approximates the nonlinear problem by a particular linear one and solve the latter, is first investigated. This idea is based on the fact that the optimum obtained for a linear problem is guaranteed to be the global one. Accordingly, if the quality of the linear approximation problem is good enough, a solution for the nonlinear problem, possibly the global one, can be found from the solution of the linear problem. To linearly approximate the nonlinear functions, the *special ordered set approximations* introduced by Beale and Tomlin [16] are used. With these approximations, a nonlinear function is approximated by means of a *piecewise*

linear function which is defined by particular constraints having common features with *discrete restrictions*, as we will see below. This method has been used by Martin *et al.* in [80] to solve a mixed integer nonlinear and nonconvex problem arising in the gas network management and has brought good results. This section begins by detailing the expression of a special ordered set approximation of a given function in one or higher dimensions and then explains the way of building a linear approximation problem from these special ordered set approximations. Finally, the limit of this approximation method is highlighted which motivates the development of a new method.

Special ordered set approximation

The special ordered set approximation has been introduced in 1970 by Beale and Tomlin [16]. As the methods for solving linear problems were a lot more advanced than for solving nonlinear problems, their goal was to replace the nonlinear functions appearing in a problem by linear approximations of these functions. However, the determination of a good linear approximation for a nonlinear function is not obvious, especially when the nonlinear function is highly nonconvex. For instance, the approximation of $\sin(x)$ on the interval $[0, 2\pi]$ by one linear function cannot produce a good approximation on the entirety of the interval $[0, 2\pi]$. Accordingly, Beale and Tomlin have suggested to use not only one linear function to approximate a nonlinear one, but a piecewise linear function, called *special ordered set approximation*, or more briefly, SOS approximation. For example, if four pieces of size equal to $\frac{\pi}{2}$ are used to approximate $\sin(x)$ on the interval $[0, 2\pi]$, the obtained SOS approximation is the one presented in Figure 2.2. This approximation allows us to catch the nonconvex behaviour of the trigonometric function. At each extremity of a piece, the approximated function and its SOS approximation coincide. Therefore, larger the number of pieces used, better the approximation. The mathematical formulation of an SOS approximation is now given by distinguishing the cases where the nonlinear function has one or more arguments. In the last case, the general formulation is illustrated by the SOS approximation of a function of two variables.

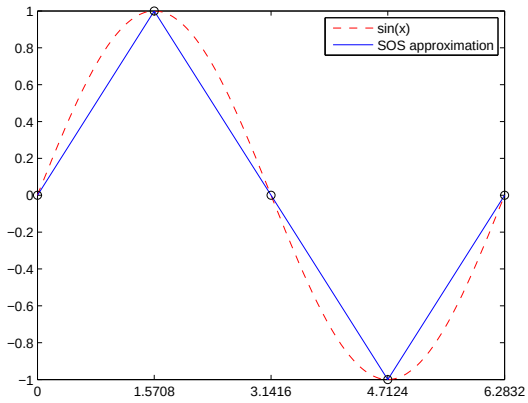


Figure 2.2: SOS approximation of $\sin(x)$ on $[0, 2\pi]$ by using four pieces of same length.

One-dimensional case

To approximate a nonlinear function f defined on an interval $[l_x, u_x]$ by its SOS approximation,

we determine a fixed number p of *breakpoints* where the nonlinear function is evaluated. These breakpoints denoted x_i, $1 \leq i \leq p$, must belong to $[l_x, u_x]$ and are ordered in such a way that $x_i < x_{i+1}$. Note that x_1 is always chosen to be equal to l_x and x_p to u_x in order to cover the entirety of the approximation interval. For example, in Figure 2.2, the breakpoints correspond to multiples of $\frac{\pi}{2}$ and are represented by circles. On each piece $[x_i, x_{i+1}]$, $1 \leq i \leq p-1$, the SOS approximation of f is defined by the linear function joining $(x_i, f(x_i))$ to $(x_{i+1}, f(x_{i+1}))$, as shown in the figure. The analytical expression of such an approximation is based on the fact that any x belonging to the interval $[l_x, u_x]$ can be expressed as a convex combination of the breakpoints x_i. This can be mathematically translated by introducing a set of new continuous variables $\lambda = \{\lambda_i\}_{i=1,p}$ and by requiring that:

$$\begin{array}{rl} x = & \displaystyle\sum_{i=1}^{p} \lambda_i x_i, \\ \text{with} & \displaystyle\sum_{i=1}^{p} \lambda_i = 1, \\ & 0 \leq \lambda_i, \quad 1 \leq i \leq p. \end{array} \tag{2.13}$$

As the breakpoints x_i are fixed, the system (2.13) is linear. Note that the variables x and λ are dependent since x is defined in function of λ. To each value of x corresponds a different convex combination of λ_i. For the sake of clarity, this dependence is omitted in the notation. Thus, we simply write x and λ.

The value at $x = \sum_{i=1}^{p} \lambda_i x_i$ of the SOS approximation of the function f, denoted $\tilde{f}$, can be computed as the same convex combination taken on the images $f(x_i)$ of the breakpoints, denoted f_i by easiness, that is,

$$\tilde{f}(x) = \tilde{f}\left(\sum_{i=1}^{p} \lambda_i x_i\right) = \sum_{i=1}^{p} \lambda_i f_i. \tag{2.14}$$

However, to define the SOS approximation of f in only one way, a condition on the variable λ must still be added to conditions (2.13) and (2.14), as explained below. For example, suppose that we want to approximate $\sin(x)$ at $x = 3$ with an SOS approximation based on five breakpoints $(x_i)_{i=1,5} = (0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi)$. Hence, by (2.13), λ must satisfy:

$$\begin{cases} 3 = 0\lambda_1 + \frac{\pi}{2}\lambda_2 + \pi\lambda_3 + \frac{3\pi}{2}\lambda_4 + 2\pi\lambda_5, \\ \displaystyle\sum_{i=1}^{5} \lambda_i = 1, \\ 0 \leq \lambda_i, \quad 1 \leq i \leq 5. \end{cases} \tag{2.15}$$

By taking $\lambda_1 = \lambda_4 = \lambda_5 = 0$, $\lambda_2 = 2 - \frac{6}{\pi}$ and $\lambda_3 = \frac{6}{\pi} - 1$ which fulfill (2.15) and by applying (2.14), the SOS approximation of $\sin(3) = 0.141120$ is given by:

$$\begin{array}{rcl} \widetilde{\sin}(3) & = & \lambda_2 \ \sin(x_2) + \ \lambda_3 \ \sin(x_3) \\ & = & \lambda_2 \ \sin(\frac{\pi}{2}) + \ \lambda_3 \ \sin(\pi) \\ & = & (2 - \frac{6}{\pi}) \, 1 - \ (\frac{6}{\pi} - 1) \, 0 \\ & = & 0.090140. \end{array}$$

Note that the choice $\lambda_2 = 2 - \frac{6}{\pi}$, $\lambda_5 = \frac{6}{\pi} - 1$ and $\lambda_1 = \lambda_3 = \lambda_4 = 0$ also satisfies (2.15) but does not give the same approximation as the one obtained with the previous value of λ. The two different approximations are represented in Figure 2.3, where it can be seen that the approximation of $\sin(x)$ on $[\frac{\pi}{2}, \pi]$ based on x_2 and x_3 is clearly better than the approximation based on x_2 and x_5.

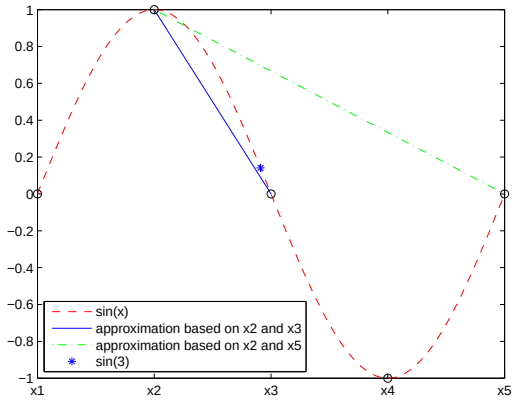


Figure 2.3: Approximations of $\sin(x)$ at $x = 3$ based on x_2 and x_3 or on x_2 and x_5.

This example highlights that the linear piecewise approximation of a function is not unique if it is only determined by constraints (2.13) and (2.14). Actually, there are numerous approximations which satisfy these constraints because the value $(x, \tilde{f}(x))$ can be obtained by any convex combination of the values (x_i, f_i). For a same value of x, there thus exist several possible values for the approximation $\tilde{f}(x)$. For the function $\sin(x)$ approximated on the interval $[0, 2\pi]$, the domain of possible values $(x, \widetilde{\sin}(x))$ satisfying constraints (2.13) and (2.14) and denoted *DPV* is represented in Figure 2.4 by the parallelogram. Indeed, it can be shown that each value $(x, \tilde{f}(x))$ inside of this parallelogram fulfills these constraints.

To obtain a unique approximating function, the one represented in Figure 2.2 for $\sin(x)$, an additional condition, known as *SOS type 2 condition*, must be imposed.

Definition 2.1 *The SOS type 2 condition, shortly referred to as SOS condition, requires that at most 2 λ_i are nonzero and that these λ_i are associated to consecutive breakpoints.*

Definition 2.2 *A set λ of variables $\{\lambda_i\}_{i=1,p}$ which satisfies the SOS condition is called SOS of type 2, shortly referred to as SOS.*

Definition 2.3 *A variable λ_i belonging to an SOS is known as an SOS variable.*

The complete definition of the SOS approximation of f can now be given.

Definition 2.4 *The SOS approximation of a function f defined on an interval $[l_x, u_x]$ is given*

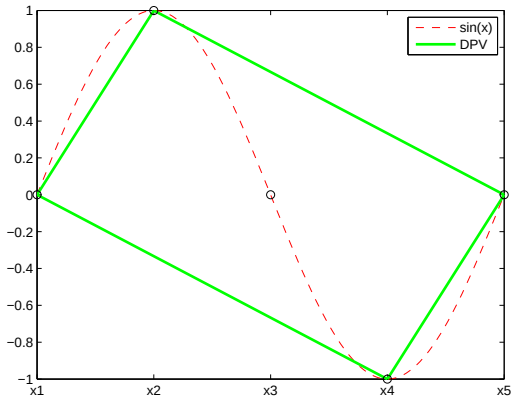


Figure 2.4: Domain of possible values $(x, \widetilde{\sin}(x))$ satisfying constraints (2.13) and (2.14) for the approximation of $\sin(x)$ on $[0, 2\pi]$.

by:

$$\tilde{f}(x) = \sum_{i=1}^{p} \lambda_i f_i, \tag{2.16}$$

$$\textit{with} \quad x = \sum_{i=1}^{p} \lambda_i x_i, \tag{2.17}$$

$$\sum_{i=1}^{p} \lambda_i = 1, \quad 0 \leq \lambda_i, \quad 1 \leq i \leq p, \tag{2.18}$$

$$\{\lambda_i\}_{i=1,p} \textit{ satisfies the SOS type 2 condition.} \tag{2.19}$$

Note that the variables λ_i which are allowed to be nonzero to satisfy the SOS condition, may be different depending on the point x where the function f is approximated. For the example illustrated in Figure 2.2, if x belongs to $[(i-1)\frac{\pi}{2}, i\frac{\pi}{2}]$, $1 \leq i \leq 4$, only λ_i and λ_{i+1} can be nonzero.

In the formulation of an SOS approximation, (2.16), (2.17) and (2.18) are linear constraints of continuous variables. It is not the case for (2.19) which can be mathematically transformed into linear constraints, but involving discrete variables. Indeed, by introducing, for each piece $[x_i, x_{i+1}]$, $1 \leq i \leq p-1$, a new binary variable y_i which is equal to one if the sum of the λ_i associated to the breakpoints defining the piece is equal to one and zero otherwise (unless only one λ_i is equal to one, in which case one of the two possibly nonzero y_i must be set to zero), the SOS type 2 condition can be mathematically expressed by system (2.20):

$$\begin{cases} y_1 \geq \lambda_1, & \\ y_i + y_{i+1} \geq \lambda_{i+1}, & 1 \leq i \leq p-2, \\ y_{p-1} \geq \lambda_p, & \\ \sum_{i=1}^{p-1} y_i = 1, & \\ y_i \text{ binary}, & 1 \leq i \leq p-1. \end{cases} \tag{2.20}$$

This formulation is known as the *lambda method* (see also Williams [124]). In (2.20), the binary condition on the variables y_i together with the constraint on the sum of these variables ensure that only one y_i can be nonzero. As a consequence, the three first constraints of (2.20) ensure that the variables λ_i and λ_{i+1} associated to the i^{th} piece are always smaller or equal to the variable y_i. By assembling this information and since the sum of the λ_i must be equal to one by (2.18), it can be derived that only two λ_i associated to two consecutive breakpoints can be nonzero.

With this formulation, after having reformulated the nonlinear problem by replacing each nonlinear function by its SOS approximation given in one dimension by (2.16), (2.17), (2.18) and conditions (2.20), we obtain a *mixed integer linear problem* which can be solved by applying the classical methods cited in Section 1.3. However, the use of the binary variables y_i increases the size of the problem. Therefore, in practice, we do not introduce them explicitly in the problem but we exploit the fact that the SOS type 2 condition can be expressed as linear constraints involving binary variables. Accordingly, to satisfy this condition, a technique often used to fulfill discrete restrictions is employed: the branch-and-bound (see Section 1.2.4). So, if the approximation of f does not satisfy the SOS type 2 condition, we branch on the variables λ_i. The popular rule for branching on such variables is to choose an index k $(k \neq 1, p)$ according to specific rules developed to this aim (see Möller [83] or Williams [123] for instance) and divide the current problem into two subproblems. The left subproblem consists of the current problem subject to the equality:

$$\sum_{j=1}^{k} \lambda_j = 1 \quad \text{(or equivalently} \quad \lambda_j = 0, \;\; k+1 \leq j \leq p), \tag{2.21}$$

while the right subproblem is subject to:

$$\sum_{j=k}^{p} \lambda_j = 1 \quad \text{(or equivalently} \quad \lambda_j = 0, \;\; 1 \leq j \leq k-1). \tag{2.22}$$

For example, if λ is equal to $(0.5, 0, 0, 0, 0.5)$, the SOS type 2 condition is not fulfilled. Assume that the branching is performed on λ_3. Accordingly, λ_4 and λ_5 must be equal to zero for the left subproblem while λ_1 and λ_2 must be equal to zero for the right one. Therefore, the current value of λ is no longer feasible for the two subproblems.

Note that this branching technique can appear as a branching rule of a branch-and-cut process more than of a branch-and-bound one because we add equalities instead of bounding some variables. But the equalities used together with (2.18) amount to impose, and thus to bound, that all variables λ_i which are not implied in the equation (2.21) for the left subproblem or in (2.22) for the right one are equal to zero. Therefore, we pursue to refer to this process as branch-and-bound.

As a preliminary summary, the SOS approximations allow us to approximate a nonlinear function by means of a piecewise linear function, which makes possible to represent a nonconvex behaviour. But these approximations have a price because they introduce in the formulation a kind of discrete restrictions which must be handled by branch-and-bound.

Case of higher dimensions

Above, the one-dimensional case has been examined but the SOS approximations can also be employed in higher dimensions. To this aim, three definitions are introduced:

Definition 2.5 *A function f is nonseparable if it cannot be decomposed as a sum of functions of a single variable.*

Definition 2.6 *A set of points is affinely independent if and only if each of these points cannot be expressed as an affine combination of the other ones (see Section 1.1.3).*

Definition 2.7 *A simplex in an n-dimensional space is the convex hull of a set of $n+1$ affinely independent points.*

Therefore, a simplex in an n-dimensional space is an n-dimensional analogue of a triangle. For example, in one dimension, the simplex is a line segment, in two dimensions, a triangle and in three dimensions, a tetrahedron.

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a *nonseparable* function defined on an n-dimensional space. Note that the case of a separable function can be discarded, because if the function is separable, it can be decomposed into a sum of functions of a single variable that can be approximated by using SOS of type 2. To define the SOS approximation of f, p_q breakpoints are chosen in each x_q-dimension, $1 \leq q \leq n$, (p_q can be different for each dimension) and an n-dimensional grid is built with them. The total number of breakpoints in this grid, denoted I_M, is thus equal to:

$$I_M = \prod_{q=1}^{n} p_q. \tag{2.23}$$

Let M be the set of the breakpoints, referred to as $\bar{x}_k, k \in I_M$, where the function f is evaluated, which gives the value f_k.

The domain $\otimes_{q=1}^{n}[l_{x_q}, u_{x_q}]$ where f is approximated can be decomposed in a set of s simplices denoted $T = \{T_1, T_2, ..., T_s\}$ (see Tomlin [110] and Ziegler [129], for instance) and which are defined by at most $n+1$ breakpoints belonging to M. The employed decomposition uses all the breakpoints in such a way that no simplex of the decomposition is included in another one and that the set of simplices covers the whole approximation domain. The SOS approximation of a function f defined on an n-dimensional space can now be given.

Definition 2.8 *The SOS approximation, $\tilde{f}$, of a function f defined on an n-dimensional space is such that:*

$$\begin{gathered} \tilde{f}(x) = \textstyle\sum_{k \in I_M} \lambda_k f_k \\ x = \textstyle\sum_{k \in I_M} \lambda_k \bar{x}_k, \\ \textstyle\sum_{k \in I_M} \lambda_k = 1, \quad 0 \leq \lambda_k, \quad k \in I_M, \end{gathered} \tag{2.24}$$

and the SOS condition is fulfilled. In the present case, it requires that:

$$\begin{gathered} \textit{At most } n+1 \ \lambda_k \textit{ are nonzero and these } \lambda_k \textit{ must be associated to breakpoints} \\ \textit{adjacent on the } n\textit{-dimensional grid refined in simplices.} \end{gathered} \tag{2.25}$$

We refer to condition (2.25) as *SOS type n+1 condition*. A way to interpret the SOS type n+1 condition is to require that all breakpoints associated to a nonzero λ_k belong to the same simplex T_j since $n+1$ is the maximum number of points which define a simplex in an n-dimensional space and since the domain has been partitioned in simplices defined by all the breakpoints.

Again, this condition can be mathematically translated by introducing additional binary variables (see Lee and Wilson [72]) while it can also be satisfied by using specific branching rules on the variables λ_k (see Martin *et al.* [80], Tomlin [110]). Note that in the particular case where n is equal to one, conditions (2.24) and (2.25) amount to conditions (2.16) to (2.19).

Illustration for the bidimensional case
The concepts defined above are now illustrated on a bidimensional space. Let (x, y) be a couple of $\mathbb{R}^2$. In order to replace $f(x, y)$ by its SOS approximation, p_x breakpoints are chosen for the x-axis and p_y breakpoints for the y-axis, which allows us to build a grid composed of $p_x p_y$ breakpoints, $\bar{x}_k = (x_i, y_j)$, where we evaluate the function f to obtain the values $f_{i,j}$, $1 \leq i \leq p_x$ and $1 \leq j \leq p_y$. To partition the domain in simplices, that is, in triangles in case of a bidimensional space, each rectangle of the grid defined by $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$, $1 \leq i \leq p_x - 1,\ 1 \leq j \leq p_y - 1$, must be divided in two triangles. An illustration of such a decomposition of the domain is given in Figure 2.5 in case of four breakpoints in each dimension. On this figure, the sixteen breakpoints are represented by dots.

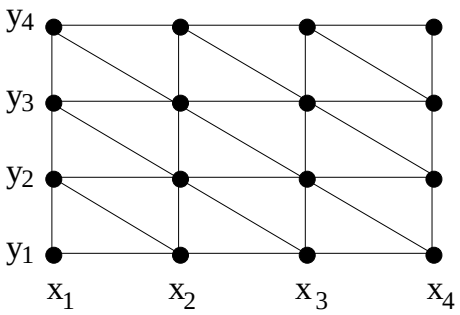


Figure 2.5: Grid generated by four breakpoints for the x and y axes.

By decomposing conditions (2.24) and (2.25) according to both dimensions and by denoting the SOS variables $\lambda_{i,j}$, $1 \leq i \leq p_x$ and $1 \leq j \leq p_y$, the SOS approximation $\tilde{f}$ of a function f is given by:

$$\tilde{f}(x, y) = \sum_{i=1}^{p_x} \sum_{j=1}^{p_y} \lambda_{i,j} f_{i,j}, \tag{2.26}$$

$$x = \sum_{i=1}^{p_x} \sum_{j=1}^{p_y} \lambda_{i,j} x_i, \tag{2.27}$$

$$y = \sum_{i=1}^{p_x} \sum_{j=1}^{p_y} \lambda_{i,j} y_j, \tag{2.28}$$

$$\sum_{i=1}^{p_x} \sum_{j=1}^{p_y} \lambda_{i,j} = 1, \quad 0 \leq \lambda_{i,j}, \quad 1 \leq i \leq p_x,\ 1 \leq j \leq p_y, \tag{2.29}$$

$$\text{at most } 3\ \lambda_{i,j} \text{ are nonzero and these } \lambda_{i,j} \text{ must be associated to breakpoints adjacent on the grid divided in triangles.} \tag{2.30}$$

The condition (2.30) corresponds to the SOS type 3 condition. For example, a solution where the nonzero $\lambda_{i,j}$ are associated to the breakpoints $(x_1, y_1),\ (x_1, y_2)$ and (x_2, y_1), satisfies the SOS type 3 condition while a solution based on the breakpoints $(x_1, y_1),\ (x_1, y_2)$ and (x_4, y_1) does not.

The SOS approximation of the function xy on the interval $[-2, 2] \times [-2, 2]$ based on four breakpoints for the x and y axes is represented in Figure 2.6, where the areas with the same color correspond to parts of the domain with similar values for the function xy. Actually, on each triangle, the SOS approximation of xy is linear and is given by the plan joining the three points $(x_{i_k}, y_{j_k}, x_{i_k} y_{j_k})$, $1 \leq k \leq 3$, where (x_{i_k}, y_{j_k}) are the three breakpoints defining the triangle.

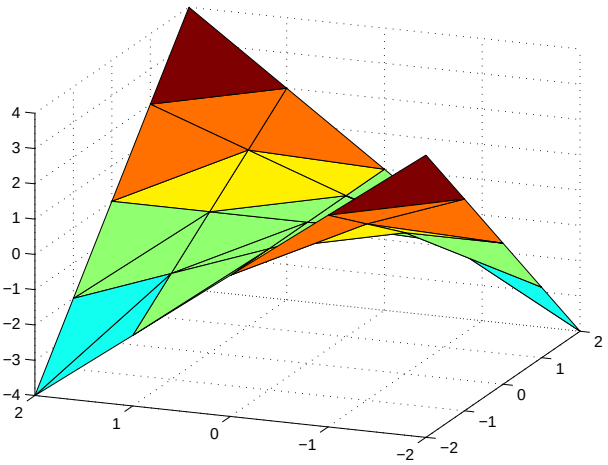


Figure 2.6: SOS approximation of xy on $[-2, 2] \times [-2, 2]$.

Figure 2.6 has been obtained by assuming that the SOS type 3 condition was fulfilled. Without explicitly imposing this condition, the variables $\lambda_{i,j}$ seldom satisfy this condition. As mentioned before, branch-and-bound can be used to fulfill the SOS condition. In case of two dimensions, the branching technique developed for SOS variables is a little bit more difficult than in one dimension because different ways of branching must be employed to satisfy the SOS type 3 condition. Firstly, the branching can be *vertical* (on x) or *horizontal* (on y). Note that the branching is actually realized on the variables $\lambda_{i,j}$ but by branching vertically (respectively horizontally), the same results as by branching on x (respectively on y) are obtained, since the variables $\lambda_{i,j}$ are associated to breakpoints. Suppose that we want to branch on $\lambda_{k,.}$, $2 \leq k \leq p_x - 1$, to divide the interval $[x_1, x_{p_x}]$ in two parts. We add to the left subproblem the

equality:

$$\sum_{i=1}^{k}\sum_{j=1}^{p_y}\lambda_{i,j}=1, \quad (\text{or equivalently } \lambda_{i,j}=0,\ \ k+1\le i\le p_x,\ 1\le j\le p_y), \tag{2.31}$$

and to the right subproblem the equality:

$$\sum_{i=k}^{p_x}\sum_{j=1}^{p_y}\lambda_{i,j}=1, \quad (\text{or equivalently } \lambda_{i,j}=0,\ \ 1\le i\le k-1,\ 1\le j\le p_y). \tag{2.32}$$

This branching is illustrated in Figure 2.7 where the breakpoints associated to possibly nonzero $\lambda_{i,j}$ are represented by black dots while the breakpoints associated to $\lambda_{i,j}$ necessarily equal to zero are represented by white dots.

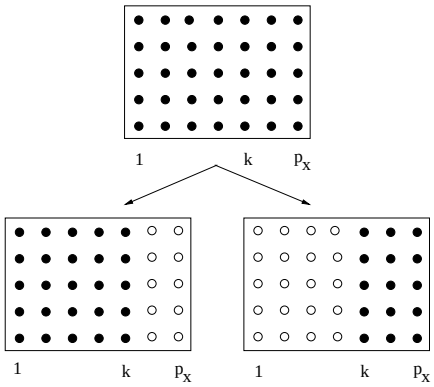


Figure 2.7: Vertical branching on $\lambda_{k,.}$.

If we branch horizontally on $\lambda_{.,k}$, $2\le k\le p_y-1$, to divide the interval $[y_1,y_{p_y}]$ in two parts, the following equality is added to the left subproblem:

$$\sum_{i=1}^{p_x}\sum_{j=1}^{k}\lambda_{i,j}=1, \quad (\text{or equivalently } \lambda_{i,j}=0,\ \ 1\le i\le p_x,\ k+1\le j\le p_y), \tag{2.33}$$

while we impose to the right subproblem:

$$\sum_{i=1}^{p_x}\sum_{j=k}^{p_y}\lambda_{i,j}=1, \quad (\text{or equivalently } \lambda_{i,j}=0,\ \ 1\le i\le p_x,\ 1\le j\le k-1). \tag{2.34}$$

Observe that by branching only vertically and horizontally, the SOS type 3 condition is not necessarily satisfied. Indeed, the best results one can expect with such branching techniques is to isolate a *rectangle* $[x_i,x_{i+1}]\times[y_j,y_{j+1}]$ of the grid instead of a triangle. In this case, we can have nonzero values for four $\lambda_{i,j}$. If this situation arises, branching must be used to divide this rectangle in two triangles in order to satisfy the SOS type 3 condition. So, one couple of indices, say (i,j), is chosen, and we impose for the left subproblem:

$$\lambda_{i,j}=0, \tag{2.35}$$

and for the right one:

$$\lambda_{i,j} + \lambda_{i,j+1} + \lambda_{i+1,j} = 1. \tag{2.36}$$

We refer to this technique as *diagonal* branching since it divides a rectangle into two triangles. This additional way to branch allows us to ensure the satisfaction of the SOS type 3 condition. In more than two dimensions, similar branching rules can be used. Indeed, similar rules as for vertical and horizontal branching can be employed to branch on each dimension. This allows us to isolate an *hyper-rectangle*. To have the satisfaction of the SOS condition, an additional condition must be again imposed, which can be expressed for the left subproblem as:

$$\lambda_q = 0, \tag{2.37}$$

and for the right one as:

$$\begin{array}{c}\text{the sum of all } \lambda_k \text{ associated to breakpoints adjacent in the grid refined in} \\ \text{simplices to the breakpoint to which is associated } \lambda_q \text{ is equal to } 1,\end{array} \tag{2.38}$$

where λ_q is nonnegative and belongs to an hyper-rectangle for which the SOS condition is not satisfied. For more details, we refer the reader to Martin *et al.* [80] or Tomlin [110].

An SOS approximation method to approximatively solve nonlinear problems

Above, we have explained how to replace a nonlinear function by its SOS approximation but we have not yet detailed the way to exploit these approximations to approximatively solve a nonlinear problem. First of all, we need to introduce some vocabulary. In the following, the system (2.24) will be referred to as *a linear approximation when it is not subject to the SOS condition and to an SOS approximation when it is*.

Suppose now that we want to solve the problem (P) expressed as:

$$(P) \begin{cases} \min & f(x), \\ \text{s.t.} & g^i(x) = 0, \ \ 1 \leq i \leq m, \\ & l_x \leq x \leq u_x, \\ & x \in \mathbb{R}^n, \end{cases}$$

where the functions f and g^i, $1 \leq i \leq m$, may be nonconvex and l_x and u_x belong to $\mathbb{R}^n$. By decomposing each function according to its linear, indexed by lin, and nonlinear parts, and by transforming, if possible, the nonlinear part into a sum of nonlinear functions, the problem (P) can be rewritten as:

$$(P) \begin{cases} \min & f_{lin}(x) + \displaystyle\sum_{j_0=1}^{t_0} f_{j_0}(x), \\ \text{s.t.} & g^i_{lin}(x) + \displaystyle\sum_{j_i=1}^{t_i} g^i_{j_i}(x) = 0, \ \ 1 \leq i \leq m, \\ & l_x \leq x \leq u_x, \\ & x \in \mathbb{R}^n. \end{cases}$$

In this formulation, only the functions f_{j_0}, $1 \leq j_0 \leq t_0$, and $g^i_{j_i}$, $1 \leq i \leq m$, $1 \leq j_i \leq t_i$, are nonlinear. If the objective function f or a constraint g^i is purely linear, the associated nonlinear

part is obviously set to zero. In order to approximate the nonlinear problem (P) by a linear one, each nonlinear function f_{j_0} and $g^i_{j_i}$ is replaced by its linear approximation given by (2.24). Note that this technique implies that all nonlinear variables of problem (P) are *bounded* since the employed linear approximations are based on intervals. The linear approximation problem obtained, denoted $(\tilde{P})$, is given by:

$$(\tilde{P}) \begin{cases} \min & f_{lin}(x) + \sum_{j_0=1}^{t_0} w^0_{j_0}, \\ \text{s.t.} & g^i_{lin}(x) + \sum_{j_i=1}^{t_i} w^i_{j_i} = 0, \quad 1 \le i \le m, \\ & w^i_{j_i} = \sum_{k \in I_{M^i_{j_i}}} (\lambda^i_{j_i})_k (g^i_{j_i})_k, \quad 0 \le i \le m,\ 1 \le j_i \le t_i, \\ & x^i_{j_i} = \sum_{k \in I_{M^i_{j_i}}} (\lambda^i_{j_i})_k (x^i_{j_i})_k, \quad 0 \le i \le m,\ 1 \le j_i \le t_i, \\ & \sum_{k \in I_{M^i_{j_i}}} (\lambda^i_{j_i})_k = 1, \quad 0 \le (\lambda^i_{j_i})_k, \quad k \in I_{M^i_{j_i}},\ 0 \le i \le m,\ 1 \le j_i \le t_i, \\ & x^i_{j_i} = x|_{g^i_{j_i}}, \quad 0 \le i \le m,\ 1 \le j_i \le t_i, \\ & l_x \le x \le u_x, \\ & x \in \mathbb{R}^n, \end{cases}$$

where:

- the variable $w^i_{j_i}$, $0 \le i \le m$, $1 \le j_i \le t_i$, is used to approximate $g^i_{j_i}(x)$. For the sake of clarity, the nonlinear functions f_{j_0} appearing in the objective function have been assimilated to $g^0_{j_0}$;
- $x^i_{j_i} = x|_{g^i_{j_i}}$ means that $x^i_{j_i}$ is the restriction of vector x containing the components actually appearing as arguments of $g^i_{j_i}$. For example, if x is defined on $\mathbb{R}^3$ and if $g^1_{1_1} = x_1 x_3$ then $x^1_{1_1} = (x_1, x_3)$;
- the set $M^i_{j_i}$ comprises all breakpoints $(x^i_{j_i})_k$ used to approximate $g^i_{j_i}$ and $I_{M^i_{j_i}} = \{1, .., \#M^i_{j_i}\}$;
- the value $(g^i_{j_i})_k$ is the evaluation of the function $g^i_{j_i}(x)$ at the k^{th} breakpoint of $M^i_{j_i}$.

Therefore, the linear problem $(\tilde{P})$ approximates the nonlinear problem (P) by using additional constraints and variables.

Solution of the linear approximation problem

The nonlinear problem (P) has thus been replaced by the linear approximation problem $(\tilde{P})$ obtained by replacing each nonlinear function of (P) by its linear approximation given by (2.24). However, as seen previously, this problem cannot be considered as a good approximation problem since no SOS condition is imposed in $(\tilde{P})$. Therefore, we cannot limit ourselves to the solution of this linear approximation problem and we must also enforce the SOS conditions. As previously mentioned, this can be achieved by explicitly introducing binary variables in the problem (see system (2.20)) or by using specific branching rules to directly branch on the

variables λ_k. Note that branching on variables λ_k has become popular since it is implemented in commercial softwares like Cplex [4] for SOS of type 2. This approach has been preferred. Therefore, to find the solution of the linear approximation problem $(\tilde{P})$ that satisfies the SOS conditions, this linear approximation problem is first solved. If, at the current solution, all sets λ do not satisfy the SOS condition associated to them, a set λ which violates this condition is chosen, and we branch on it in the way explained above. Except for this particular branching technique on λ_k, the classical algorithm of branch-and-bound (Algorithm 1.1) is applied. When the whole tree has been explored, the solution of the linear SOS approximation problem is found. To improve the speed of convergence of the method, cuts can be added during the branch-and-bound process and a convenient polynomial separation algorithm has been developed (see Martin *et al.* [80], Möller [83]).

However, the SOS approximation method is only an approximation method and it thus can fail to converge to an optimum. As it will be explained in Section 3.1.1, it is the case for the *TVC* problem. Therefore, some safeguards should be used to compel the method to converge to an optimum. As it is difficult to determine the boundary between a method which converges to a "good" optimum and one which does not converge, we have investigated the methods that *always converge to an optimum* and even more, to the global one.

2.2.4 Global optimization methods

In an attempt to check if existing global optimization methods are available to efficiently solve the *TVC* problem, we have tested *BARON* (see Section 1.2.4), a global optimization solver available on NEOS for solving mixed integer nonlinear and nonconvex problems. Note first that *BARON* cannot treat trigonometric functions directly. In order to apply even though this solver on the *TVC* problem, the trigonometric functions have been approximated by Taylor polynomials of degree 7 (see Deuflhard and Hohmann [32]). This has made the model significantly larger and inaccurate. As a consequence, *BARON* could not give good results on the considered problem(1). In this case, it is really the *trigonometric functions* which are problematic. These functions are not much treated in global optimization methods. At our knowledge, only Caratzoulas and Floudas [24] have developed specific techniques to treat these functions in a global optimization method. As their approach consists in replacing nonconvex trigonometric functions by convex ones and thus, in solving nonlinear subproblems while we would like to solve linear ones, a new method based on linear subproblems and involving adapted techniques to treat trigonometric functions is developed in this thesis.

2.3 Conclusion

The class of optimal power flow problems, and notably the *TVC* problem, has been presented and its specificities highlighted, which allows us to class it among the *mixed integer nonlinear and nonconvex problems*. Because of, on one hand, the size of the problem and the cost of the solution of a nonlinear problem and on the other hand, the successful experiment of Martin *et al.* to solve gas networks problems in this way, a method solving linear problems instead of nonlinear ones is preferred. In this chapter, we have described a technique to approximate, in one or higher dimensions nonlinear functions, possibly highly nonconvex, by means

(1)This result has been obtained thanks to Bussieck [22].

of piecewise linear functions called SOS approximations. The considered formulation allows us to approximate a nonlinear problem by a linear one subject to constraints which can be seen as discrete restrictions. However, this particular linear approximation method failed to converge on the *TVC* problem. To guarantee the detection of an optimum, and even the global one, *global optimization* methods have been also examined. However, the results obtained with the different tested methods lead to think that the development of a new method is desirable. The method developed in this thesis will thus be a global optimization one that will mainly solve *linear* problems and that will be able to treat *trigonometric functions* in an appropriate way.

Note that for our numerical experiments presented in Chapters 5 to 8, the size of the *TVC* problems treated has been limited (maximum 10 nodes and 10 branches) in order to easily test the developed method.

Chapter 3

An outer approximation method based on special ordered sets

In the second chapter, we have presented the *TVC* problem which motivates the development of a new optimization method and we have considered different methods to solve it. We have mainly focused on the SOS approximation method recently employed by Martin *et al.* to solve mixed integer nonlinear and nonconvex problems arising in the gas network management (see [80]). However, if the quality of the approximation problem built by this method is not good enough, the solution of this problem can be useless to solve the nonlinear problem under study, as mentioned earlier. Therefore, in this chapter, we modify the SOS approximation method to make it more robust and to ensure the computation of an optimum if the nonlinear problem is feasible, and even more, the global one.

More precisely, we propose to *globalize* the linear approximation problem in such a way that it corresponds to a linear *outer approximation* problem for the nonlinear one, that is, any feasible point x for the nonlinear problem is also feasible for the linear approximation problem and can produce (among others) an objective value for this problem equal to the one that it generates for the nonlinear problem. Note that here, the notion of outer approximation does not refer to the outer approximation methods described in Section 1.3.1. We use this term to highlight the fact that we start from an approximation problem and modify it in such a way that it becomes an outer approximation problem in the sense explained above. To this aim, each nonlinear function appearing in the problem is no longer replaced by a linear function, but included in a *linear domain based on SOS*. Doing so, the feasible domain of the linear approximation problem increases. Therefore, its solution can also be very far from the solution of the considered nonlinear problem. Accordingly, the approximation problem must be refined, which can be done through a *branch-and-bound* tree (see Section 1.2.4).

This chapter begins by highlighting the features of the SOS approximation method which should be improved in order for the resulting method to be efficient on the *TVC* problem. The limitation of the size of the linear problem is one of these features. To this goal, in our work, we limit ourselves to SOS of one and two dimensions. The reasons of this choice and the way to decompose functions of three dimensions and more in *components of one or two dimensions* are explained. In order to bound these components, bound propagation techniques are developed. The specificities of the problem are also exploited in order to limit the size of the outer approximation problem. In the latter problem, each nonlinear component is replaced by a piecewise linear domain based on SOS and that includes it. The determination of this domain

passes through the computation of the *maximum approximation errors* generated by the SOS approximation. The analytical expression of these errors is established for the three kinds of components of one or two dimensions that appear in the *TVC* problem, which are square, bilinear and trigonometric functions (sine and cosine). Since the domain of possible values for the outer approximation problem is a lot larger than the one of the nonlinear problem, its solution is possibly not relevant for the nonlinear problem. Therefore, the approximation problem is refined thanks to a branch-and-bound tree as explained in Section 3.3, where the general process of the method and some comments about its convergence are given.

Note that the continuous case is first treated and, from Section 3.3.3, the necessary adaptations to take discrete restrictions into account are detailed. Finally, we conclude this chapter by illustrating the proposed method on a simple example.

3.1 Motivation

We start this chapter by motivating the development of a new method for solving the *TVC* problem through the necessity to globalize the method, the interest of limiting the size of the approximation problem and finally the advantage of using an approach based on SOS instead of on big-M constraints.

3.1.1 Globalization of the method

As we pointed out in Chapter 2, the SOS approximation method was promising to solve our problem since it has been used to solve efficiently other mixed integer nonlinear and nonconvex problems. We have thus applied this method on the *TVC* problem (without using the techniques cited at the end of Section 2.2.3 to improve the speed of convergence), but it did not converge. Indeed, the linear approximation problem was infeasible because it was not tight enough to obtain a solution for the nonlinear problem which is itself feasible. As the *TVC* problem is subject to *equality constraints*, a solution can be more easily discarded if the approximation is not tight enough. An illustration of a linear approximation not sufficiently accurate is given in Figure 3.1, where a point x is assumed to be feasible if it satisfies the equality constraint $c2(x) - c1(x) = 0$. Accordingly, the feasible domain of the nonlinear problem is not empty. If we employ the SOS approximations of $c1$ and $c2$ represented in the figure, the feasible domain of the approximation problem is empty because the SOS approximation of $c1$ never intersects the one of $c2$. To avoid this, the approximation could be improved by employing more breakpoints for the SOS approximation problem or by refining the approximation during the branch-and-bound process by adding new breakpoints. But there is no guarantee that the generated approximation problem will be tight enough to be feasible, and if it is, nothing ensures that the solution of the linear approximation problem will be relevant to find the global optimum of the nonlinear problem. Moreover, for the *TVC* problem, the *satisfaction of the constraints is crucial* because they translate physical realities. In these conditions, the solution of a linear approximation problem seems to be insufficient. Nevertheless, this solution could be employed as a starting point for a nonlinear solver. But again, there is no guarantee to converge.

Therefore, it is crucial to transform the approximation problem in such a way that it becomes an *outer approximation problem* for the nonlinear one, in the sense given at the begin of this chapter.

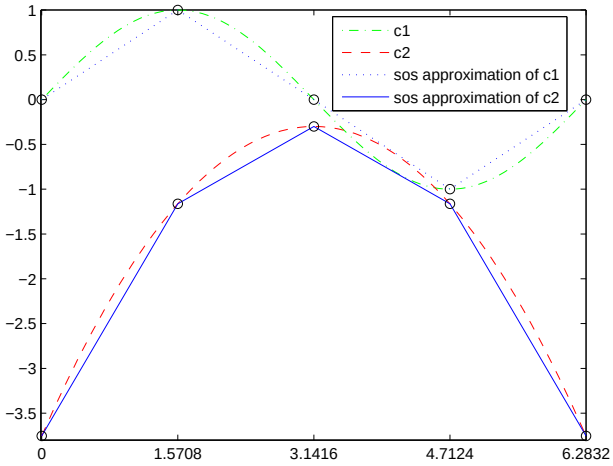


Figure 3.1: Empty feasible domain for the linear approximation problem while it is not for the nonlinear problem.

3.1.2 Size of the linear approximation problem

Another drawback of the SOS approximation method presented in Chapter 2 is the size of the linear approximation problem. In the *TVC* problem, we can consider that we have quadrilinear products $x_1x_2x_3x_4$. If we use p breakpoints in each dimension, the number of new variables λ_i introduced in the linear approximation problem is equal to p^4, by (2.23). As a consequence, the number of variables in the linear approximation problem can significantly increase with regard to the number of variables in the nonlinear problem. To reduce the number of variables λ_i introduced, we propose to decompose each nonlinear function of the problem in nonlinear components of one or two variables by means of a so-called *computational graph*.

We first illustrate this decomposition by an example. In the *TVC* problem, the definition (2.4) of the real power on a simple branch can be seen as a constraint of the form:

$$x_1 = ax_2^2 + bx_2x_3\cos(x_4), \tag{3.1}$$

where x_i, $1 \leq i \leq 4$, are the variables and a and b are parameters. While this constraint is nonconvex, it can be rewritten as a convex constraint involving new variables w_i, by setting:

$$x_1 = aw_1 + bw_4, \tag{3.2}$$

with:

$$w_1 = x_2^2, \tag{3.3}$$
$$w_2 = x_2x_3, \tag{3.4}$$
$$w_3 = \cos(x_4), \tag{3.5}$$
$$w_4 = w_2w_3. \tag{3.6}$$

As each constraint (3.3) to (3.6) can be approximated by using an SOS, a linear formulation can be obtained to approximate (3.1).

The formulation given by (3.2) and the linear approximations of (3.3) to (3.6) *reduces the number of variables* λ_i introduced in the problem. Suppose for example that we want to approximate a quadrilinear component $x_1x_2x_3x_4$ by using the same number, p, of breakpoints in each dimension. The total number of breakpoints used to approximate this component in a space of four dimensions is equal to p^4. Now, if we decompose this product in two products, $w_{12} = x_1x_2$ and $w_{34} = x_3x_4$, and then multiply the obtained products w_{12} and w_{34}, we only need to use $3p^2$ breakpoints since this modelling uses 3 products defined on a bidimensional space. If p is equal to 3, the quadrilinear product uses 81 breakpoints without decomposition, while this number is reduced to 27 if we use the decomposition in two products. If we take 5 breakpoints in each dimension, we must compare the numbers 625 and 75. As a variable λ_i is associated to each breakpoint, the decomposition allows us to reduce the number of variables in the linear outer approximation problem. Nevertheless, the number of constraints increases with the decomposition: by quadrilinear product, 12 constraints are introduced instead of 6 without decomposition. The advantage and drawback of such a decomposition will be discussed further in Section 3.2.1.

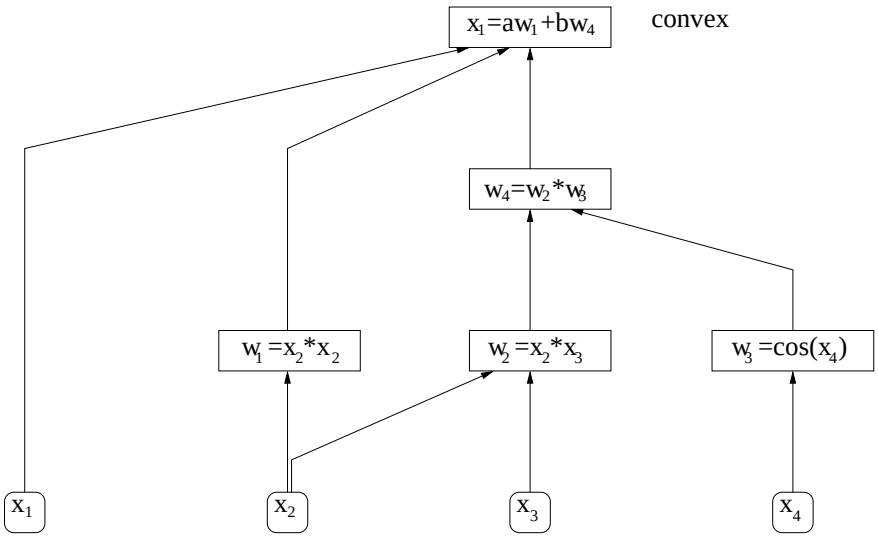


Figure 3.2: Decomposition of the constraint $x_1 = ax_2^2 + bx_2x_3\cos(x_4)$ through its computational graph.

Note that the *computational graph* of the constraint (3.1) is given in Figure 3.2. On this figure, the bottom nodes correspond to the original variables x_i, while the top node represents the constraint to decompose. The other nodes are associated to new variables w_j allowing us to decompose the constraint. Each directed edge means that the source node is directly involved in the unary functions or in the bilinear product represented at the target node.

3.1.3 SOS versus big-M approach

The approach proposed by Polisetty and Gatzke in [96] is close to ours since it is also a linear outer approximation method, but based on big-M constraints. We show here that the linear

outer approximation method based on SOS that we propose can lead to tighter approximations than the ones based on big-M constraints. To highlight this, we consider the approximation of the function $-x^2$ on $[0,3]$ by using three pieces of equal size: $[0,1]$, $[1,2]$ and $[2,3]$. Without loss of generality, we focus on the underestimation of this function. By applying (1.28), (1.29) and (1.31), where $u_i(x)$ is the linear function joining the two extreme points of the i^{th} piece for the big-M formulation, and (2.16) to (2.19) for the SOS approximation (which is underestimating in this case), we find that the underestimation of the function $-x^2$ is given by the same piecewise linear function which is represented on Figure 3.3. This underestimation, which is

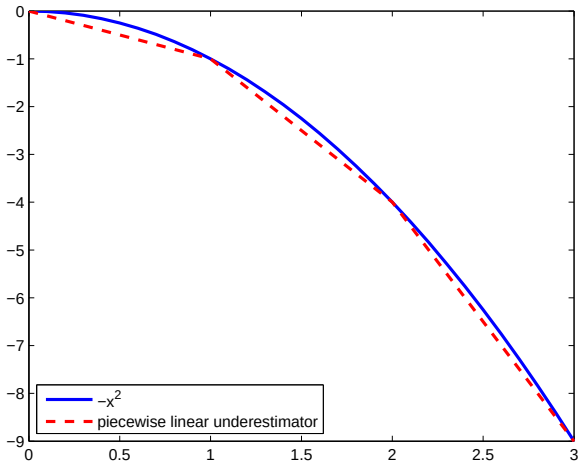


Figure 3.3: Piecewise linear underestimator obtained with a big-M formulation or an SOS formulation for $-x^2$ on $[0,3]$ by using three pieces of same size.

quite tight, is obtained by assuming the satisfaction of the binary conditions on the variables b_i for the big-M formulation and that of the SOS condition for the SOS approximation. However, in practice, these conditions are not directly fulfilled. Therefore, we must also compare the underestimations obtained with both formulations without explicitly imposing these conditions.

If the SOS condition is not satisfied, the less tight underestimator generated by the system (2.16) to (2.18) corresponds to the linear function joining (0,0) to (3,-9), which amounts to the tightest convex underestimator for $-x^2$ on $[0,3]$, as shown on Figure 3.4. Let us show that the big-M formulation can generate a worse underestimator when the binary conditions on the variables b_i are not fulfilled. We first determine a value for the parameter M. To guarantee that the functions $u_i(x) - M$ are underestimating for $-x^2$ on each piece i, M must be sufficiently large. However, to have underestimations as tight as possible, M must be chosen as small as possible. As the best linear piecewise underestimators of $-x^2$ on $[0,3]$ correspond to the linear functions joining the two extreme points of a piece, they are given by:

$$\begin{array}{rcl} u_1(x) & = & -x, \\ u_2(x) & = & -3x+2, \\ u_3(x) & = & -5x+6. \end{array} \tag{3.7}$$

By (1.28), (1.31) and the binary conditions on the variables b_i, M must satisfy:

$$u_i(x) - M \leq -x^2 \quad \forall x \in [0,3],\ \forall i = 1,..,3. \tag{3.8}$$

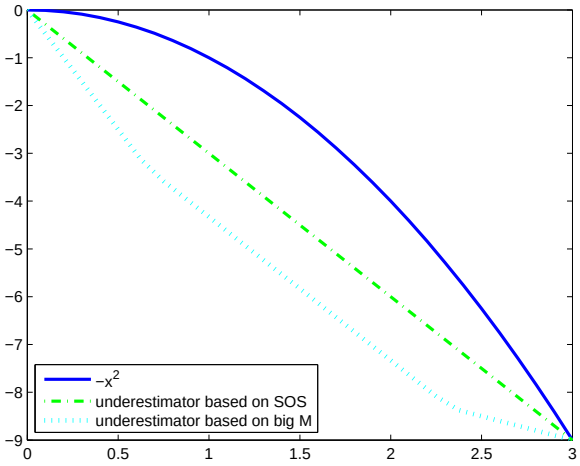


Figure 3.4: The worst piecewise linear underestimators for $-x^2$ on $[0, 3]$ by using three pieces with a big-M formulation ($M = 6$) or an SOS formulation for which the binary and the SOS conditions are not satisfied.

It can be shown that 6 is the smallest value in order for M to fulfill these inequalities (this corresponds to the difference at zero between $u_3(x)$ and $-x^2$, or equivalently, to that at three between $u_1(x)$ and $-x^2$). Therefore, by (1.28), (1.29), (1.31) and (3.7), the underestimator w must be such that:

$$\begin{cases} -x - 6(1 - b_1) \leq w, \\ -3x + 2 - 6(1 - b_2) \leq w, \\ -5x + 6 - 6(1 - b_3) \leq w, \\ \\ x \leq 1 + 6(1 - b_1), \\ 1 - 6(1 - b_2) + \delta \leq x, \\ x \leq 2 + 6(1 - b_2), \\ 2 - 6(1 - b_3) + \delta \leq x, \\ \\ b_1 + b_2 + b_3 = 1, \\ 0 \leq b_i \quad \forall i = 1, .., 3, \\ \\ 0 \leq x \leq 3. \end{cases} \tag{3.9}$$

For each value of x in $[0, 3]$, the smallest value of the underestimation w can be found by minimizing w subject to the constraints of (3.9). Doing this, we obtain the underestimator represented on Figure 3.4 which is clearly less tight than the one based on the SOS formulation. Moreover, in the present case, we have taken the smallest possible value for M. But usually, in big-M methods, a "large" value is arbitrarily chosen for M, which leads to a worse underestimator than the one represented on Figure 3.4. As a consequence, using an approach based on an SOS instead of on a big-M formulation generates tighter underestimators (possibly even much tighter).

3.2 An outer approximation problem based on SOS

As highlighted in Section 3.1.1, feasible solutions can be missed by using the SOS approximation method since there is no guarantee that the optimum solution of the nonlinear problem belongs to the domain of the linear approximation problem. It would be preferable that the linear problem is not only an approximation problem but an *outer approximation problem* for the nonlinear one. To this aim, each nonlinear function is *no longer replaced by its linear approximation but included in a linear domain*. This linear domain is based on SOS approximations and notably, on the maximum approximation errors generated by these approximations. So, in

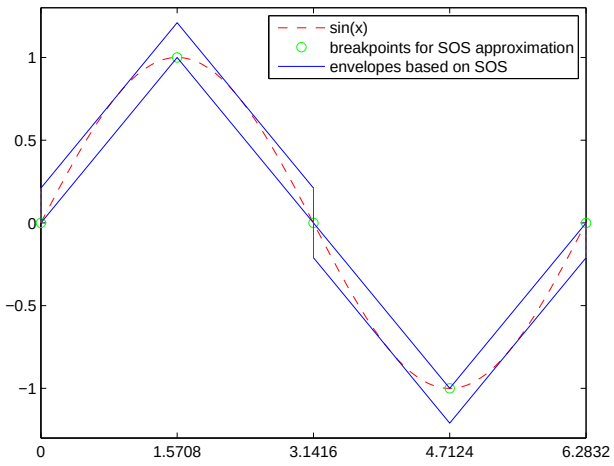


Figure 3.5: Piecewise linear approximation domain for $\sin(x)$ on $[0, 2\pi]$ obtained when the SOS condition is satisfied.

Figure 3.5, the area delimited by continuous lines, referred to as piecewise envelopes, represent the piecewise linear domain based on SOS to approximate $\sin(x)$ on $[0, 2\pi]$ when the SOS condition is fulfilled. The SOS approximation of $\sin(x)$ corresponds to the sides of the envelopes which link the images of two consecutive breakpoints.

We denote ϵ_L (respectively ϵ_U), the maximum overestimation approximation error (respectively the maximum underestimation approximation error) which can arise on an interval $[l_x, u_x]$ by replacing a function $f(x)$ by its SOS approximation, $\tilde{f}(x)$, that is,

$$\epsilon_L = \max_{x \in [l_x, u_x]} (0, \tilde{f}(x) - f(x)),$$
$$\epsilon_U = \max_{x \in [l_x, u_x]} (0, f(x) - \tilde{f}(x)).$$

Therefore, these definitions imply that the maximum approximation errors ϵ_L and ϵ_U are always positive. In the outer approximation problem, the variable w_f used to approximate the value of a function f at a point x is required to belong to the domain defined by:

$$\tilde{f}(x) - \epsilon_L \leq w_f \leq \tilde{f}(x) + \epsilon_U. \tag{3.10}$$

In this context, the *SOS condition is only imposed to determine the analytical expression of the maximum overestimation and underestimation errors, otherwise, that is, during the process of*

the algorithm developed in this section, it is no longer required. In fact, we will see that we exploit the features of the SOS approximations, since they correspond to good approximations, in order to obtain tight outer approximations but without imposing SOS conditions. The reasons of this choice will be more detailed in Section 3.2.5.

While the SOS approximations can be given by an analytical formulation common for each kind of functions (constraints (2.24) and (2.25)), the mathematical expression of the outer approximation defined in (3.10) must be treated differently according to the approximated functions because it needs the computation of the maximum approximation errors, ϵ_L and ϵ_U, which depend on the approximated functions. Therefore, for each kind of functions involved in the problem, we must compute the associated errors. In the *TVC* problem, we have several kinds of nonlinear functions: square functions (ν_i^2), trigonometric functions ($\sin(\zeta_j + \theta_i - \theta_k), \cos(\zeta_j + \theta_i - \theta_k)$), bilinear functions ($\nu_i\nu_k$), trilinear functions ($\nu_i\nu_k R_j$), products of a trilinear function and a trigonometric one ($\nu_i\nu_k R_j \sin(\zeta_j + \theta_i - \theta_k)$), etc. To limit the number of new sets λ introduced in the formulation of the outer approximation problem and to exploit the specificities of the nonlinear problem, we choose to only work with *nonlinear components of one or two variables* (see Section 3.1.2). Functions of higher dimensions are decomposed in such components in the way explained in the following section.

3.2.1 Decomposition of nonlinear functions into nonlinear components of one or two variables

As highlighted in Section 3.1.2, the decomposition of the nonlinear functions into nonlinear components of smaller dimension reduces the number of variables in the linear problem. In this thesis, we decompose each nonlinear function in components of one or two variables. Moreover, the arguments of a component implying two variables are not allowed to be functions themselves but simple variables (possibly new ones which replace some functions), that is, each product of functions is transformed into a product of variables. For example, in the computational graph presented in Figure 3.2, $\cos(x_4)$ has been approximated by w_3 before considering its product with w_2. This allows us to develop a general framework based on the SOS approximations of a *limited number of functions* (unary functions and bilinear product), which makes the method more easily applicable to a large number of problems. Indeed, the proposed outer approximation method needs the determination of the analytical expression of the maximum approximation errors generated by the SOS approximation of each function appearing in the problem. The transformation of products of two functions into products of variables allows us to use the same formulation, namely the SOS approximation of a bilinear product, to treat all products of two functions. This avoids to determine the analytical expression of the approximation errors generated by the SOS approximation of every product of any functions. With this strategy, only three kinds of components must be examined for the *TVC* problem : *square, bilinear and trigonometric functions* ($\sin(x)$ and $\cos(x)$).

Note that the decomposition is not unique. For the example presented in Figure 3.2, we could have chosen to take $w_2 = \cos(x_4)$, $w_3 = x_2 w_2$ and $w_4 = x_3 w_3$, for example. The decomposition of the problem obviously has an impact on the results. Therefore, it must be treated carefully in order to exploit the specificities of the problem. Suppose, for instance, that a bilinear product x_1x_2 appears in two trilinear products: $x_1x_2x_3$ and $x_1x_2x_4$. The decomposition $w_1 = x_1x_2$, $w_2 = x_3w_1$ and $w_3 = x_4w_1$ is better than the following one: $w_1 = x_1x_3$,

$w_2 = x_2 w_1$, $w_3 = x_2 x_4$ and $w_4 = x_1 w_3$ since it needs one new variable w_i less, and, as a consequence, one set λ less.

In the software (see Section 5.1.3) developed to test the proposed outer approximation method, the decomposition of the problem into components of one or two variables is done by hand. But there exist techniques to derive automatically a computational graph from a nonlinear function, as it is used in *automatic differentiation* developed by Griewank [53] (see also the library VGTL [106] which allows us to analyze and treat graphs and which is employed for global optimization in the COCONUT project [107]). A natural extension of our software would be to introduce such techniques in order to reduce the modelling work of the user.

The reasons for decomposing the nonlinear functions of components of one or two variables are triple. First of all, as explained above, this allows us to develop a *general framework* able to treat a large number of problems and based on the SOS approximations of a limited number of functions. Secondly, this reduces the number of variables introduced in the problem (see Section 3.1.2). Finally, the decomposition can *exploit the multiple presence* in the problem of a same nonlinear component in components of higher dimensions. For example, in the *TVC* problem, the bilinear product $\nu_i \nu_k$ appears in products of this bilinear product with two different trigonometric functions. The decomposition in components of one or two variables allows us to exploit this observation since the same linear approximation can be used to approximate the bilinear product in the different cases. Conversely, without decomposition, new breakpoints must be added each time the bilinear product intervenes. Suppose that we want to approximate $x_1 x_2$ and $x_1 x_2 x_3 x_4$, the decomposition in components of two dimensions introduces p^2 breakpoints for $x_1 x_2$, p^2 breakpoints for $x_3 x_4$ and again p^2 breakpoints for the product of the two. Accordingly, $3p^2$ breakpoints are needed. Without decomposing, we must use p^2 breakpoints for $x_1 x_2$ and p^4 breakpoints for $x_1 x_2 x_3 x_4$, thus $p^2(1 + p^2)$ breakpoints. If the number of breakpoints used in each dimension is equal to 3 (respectively to 5), the difference between both techniques corresponds to 63 breakpoints, i.e., 27 instead of 90 (respectively 575, i.e., 75 instead of 650). Again, the decomposition allows us to reduce the number of variables λ_i used in the problem but it needs two additional constraints (12 instead of 10). Furthermore, by decomposing, the value of the approximation of a same component appearing in components of higher dimension is equal everywhere because, in each case, the same set λ is used to approximate it. This is not ensured without decomposition as long as the SOS conditions are not fulfilled, because we use different sets λ to approximate the same component. Therefore, by exploiting the specificities of the problem, the decomposition technique can discard a part of the infeasible domain with regard to the technique without decomposition (see Section 3.2.3 for more details).

Nevertheless, the decomposition of a function into components of smaller dimensions also *reduces the accuracy of the approximation* since it is based on less breakpoints. For methods like the one developed by Martin *et al.* [80] where the linear SOS approximation problem is completely solved and never refined, as explained in Section 2.2.3, it is crucial to have the best approximations as possible since the quality of the solution depends on the quality of these approximations. Therefore, in this case, the functions are not decomposed. In the method developed here, the approximation problem is refined by using a branch-and-bound process (see Section 3.3), which allows us to produce tight approximations. Accordingly, in our method, an outer approximation problem less accurate at the root node will not prevent from finding the global optimum and the linear problems will be solved quicker as they imply less variables. A more extensive study should be done to compare the results obtained with an outer approximation problem of less variables and able to exploit the specificities of the problem to discard some

parts of the outer approximation domain, or with an outer approximation problem of larger size but more accurate in the sense that is based on more breakpoints. In the following, a method with decomposition has been preferred for the three reasons detailed above.

To build linear approximations given by (2.24), the variables appearing in the approximated components must be bounded. While we assume that the variables appearing in the nonlinear problem are bounded, bounds on new variables replacing a component of one or two variables and intervening themselves in a nonlinear component are not directly available. Therefore, the bounds on the original variables need to be propagated into bounds on new variables.

3.2.2 Propagation of the bounds through the computational graph

As the bounds on a component depend on the bounds of its argument(s) and on the function which links them together, we distinguish the propagation of the bounds according to the different components of one or two variables which appear in the *TVC* problem.

1. $\boxed{x^2}$
   The square function is first studied. If x belongs to the interval $[l_x, u_x]$, the lower and upper bounds on x^2, l_{x^2} and u_{x^2}, can be updated like this:

$$\begin{array}{rcl} l_{x^2} & := & \begin{cases} \min(l_x^2, u_x^2) & \text{if } 0 \notin [l_x, u_x], \\ 0 & \text{if } 0 \in [l_x, u_x], \end{cases} \\ u_{x^2} & := & \max(l_x^2, u_x^2). \end{array}$$

   For the lower bound on x^2, the results must be distinguished depending if zero belongs or not to the interval $[l_x, u_x]$.

2. $\boxed{xy}$
   If x belongs to the interval $[l_x, u_x]$ and y to $[l_y, u_y]$, the following formulas are used to update the lower and upper bounds, l_{xy} and u_{xy}, on the bilinear product xy:

$$\begin{array}{rcl} l_{xy} & := & \min(l_x l_y,\ l_x u_y,\ u_x l_y,\ u_x u_y), \\ u_{xy} & := & \max(l_x l_y,\ l_x u_y,\ u_x l_y,\ u_x u_y). \end{array}$$

3. $\boxed{\text{Trigonometric functions}}$
   For these functions, we assume, without loss of generality, that the definition domain, $[l_x, u_x]$, belongs to $[0, 2\pi]$. Indeed, it will be shown in Section 3.3.2 that the problems with a domain for trigonometric functions not contained in the interval $[0, 2\pi]$ can be transformed to be defined on an interval belonging to $[0, 2\pi]$ by means of an additional integer variable and a new linear constraint. The determination of the bounds on $\sin(x)$ and $\cos(x)$ strongly depends on the interval $[l_x, u_x]$. The lower and upper bounds on these trigonometric functions are given by the values of these functions at the extreme points of the interval, i.e., at l_x and u_x, unless an extremum is reached on the considered interval. Therefore, we have for the $sine$ function:

$$l_{\sin(x)} := \begin{cases} -1 & \text{if } l_x \leq \frac{3\pi}{2} \leq u_x, \\ \min(\sin(l_x), \sin(u_x)) & \text{otherwise}, \end{cases}$$

$$u_{\sin(x)} := \begin{cases} 1 & \text{if } l_x \leq \frac{\pi}{2} \leq u_x, \\ \max(\sin(l_x), \sin(u_x)) & \text{otherwise}, \end{cases}$$

and for the $cosine$ function:

$$l_{\cos(x)} := \begin{cases} -1 & \text{if } l_x \leq \pi \leq u_x, \\ \min(\cos(l_x), \cos(u_x)) & \text{otherwise,} \end{cases}$$

$$u_{\cos(x)} := \max(\cos(l_x), \cos(u_x)).$$

3.2.3 Exploiting common subexpressions

Another way to limit the number of variables in the problem consists in exploiting the specificities of the problem, as previously mentioned. Firstly, if a same nonlinear function with the same argument appears more than once in the problem, only one set λ is associated to it. It would be useless but also less efficient to employ several λ in this case. Indeed, by using different sets λ, different values for a same approximated function can be obtained as long as the SOS condition is not checked. Using the same λ to approximate the same function with the same argument ensures to have an identical value for all the approximations of this function, which allows us to reduce the feasible domain of the linear approximation problem, and also, the number of variables and constraints introduced in $(\tilde{P})$.

In the same way, the *multiple presence* of a variable in the nonlinear problem can sometimes be exploited to save the number of variables and constraints introduced in the problem. Suppose, for example, that a variable x appears in the square function x^2 and in the bilinear product xy. Assume furthermore that three breakpoints are used in each dimension. The linear approximation of x^2 is given by:

$$\begin{aligned} \widetilde{x^2} &= \lambda_1 x_1^2 + \lambda_2 x_2^2 + \lambda_3 x_3^2, \\ x &= \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3, \\ \sum_{i=1}^{3} \lambda_i &= 1, \quad 0 \leq \lambda_i, \end{aligned} \tag{3.11}$$

while the one associated to xy is given by:

$$\begin{aligned} \widetilde{xy} &= \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda'_{i,j} x_i y_j, \\ x &= \sum_{j=1}^{3} \lambda'_{1,j}\, x_1 + \sum_{j=1}^{3} \lambda'_{2,j}\, x_2 + \sum_{j=1}^{3} \lambda'_{3,j}\, x_3, \\ y &= \sum_{i=1}^{3} \lambda'_{i,1}\, y_1 + \sum_{i=1}^{3} \lambda'_{i,2}\, y_2 + \sum_{i=1}^{3} \lambda'_{i,3}\, y_3, \\ \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda'_{i,j} &= 1, \quad 0 \leq \lambda_{i,j}. \end{aligned} \tag{3.12}$$

These two approximations introduce twelve variables in the problem (three variables λ_i for x^2 and nine variables $\lambda'_{i,j}$ for xy). However, by expressing in (3.11) the variables λ_i in function of variables $\lambda'_{i,j}$ like:

$$\lambda_1 = \sum_{j=1}^{3} \lambda'_{1,j},\ \lambda_2 = \sum_{j=1}^{3} \lambda'_{2,j} \text{ and } \lambda_3 = \sum_{j=1}^{3} \lambda'_{3,j}, \tag{3.13}$$

the total number of additional variables necessary in the formulation can be reduced to nine. The number of constraints also decreases because the system (3.11) can be reduced by using (3.13) to:

$$\widetilde{x^2} = \sum_{j=1}^{3} \lambda'_{1,j} x_1^2 + \sum_{j=1}^{3} \lambda'_{2,j} x_2^2 + \sum_{j=1}^{3} \lambda'_{3,j} x_3^2,$$

since the two other constraints are already imposed in system (3.12). Again, it appears that a good use of the sets λ can limit the number of introduced variables and also reduce the feasible domain of the linear approximation problem since the approximations of the two functions are linked together. Note that this substitution is made possible because, in case of SOS approximations, the quantities multiplying the breakpoints x_i must be identical for the SOS approximations of x^2 and xy in order to satisfy the SOS condition. Actually, we employ a feature of the SOS approximations to refine an approximation which is not subject to the SOS condition since we know that the SOS approximations correspond to good approximations.

The specificities of the problem can also be exploited to reformulate the problem differently in order to reduce the number of sets λ used in the approximation problem. For example, in the *TVC* problem detailed in Chapter 2, the following nonlinear functions appear:

$$\begin{aligned} &\cos(\zeta_j + \theta_i - \theta_k), \\ &\cos(\zeta_j + \theta_k - \theta_i), \\ &\sin(\zeta_j + \theta_i - \theta_k), \\ &\sin(\zeta_j + \theta_k - \theta_i), \end{aligned} \tag{3.14}$$

where θ_i and θ_k are the variables while ζ_j is a parameter. To approximate these functions, only one set λ can be used instead of four. To this aim, we introduce a new variable η_j such that:

$$\eta_j = \zeta_j + \theta_i - \theta_k.$$

By substituting this variable in the nonlinear functions of (3.14), we obtain:

$$\begin{aligned} &\cos(\eta_j), \\ &\cos(2\zeta_j - \eta_j), \\ &\sin(\eta_j), \\ &\sin(2\zeta_j - \eta_j), \end{aligned} \tag{3.15}$$

respectively. By using the trigonometric formulas, (3.15) becomes:

$$\begin{aligned} &\cos(\eta_j), \\ &\cos(2\zeta_j)\cos(\eta_j) + \sin(2\zeta_j)\sin(\eta_j), \\ &\sin(\eta_j), \\ &\sin(2\zeta_j)\cos(\eta_j) - \cos(2\zeta_j)\sin(\eta_j). \end{aligned} \tag{3.16}$$

Accordingly, the nonlinear functions of (3.14) have been transformed into nonlinear functions of one same variable η_j. The linear approximation of (3.16) (equivalent to (3.14)) obtained by

using p breakpoints $\eta_{j,i},\ 1 \leq i \leq p$, can be written as:

$$\begin{aligned}
\widetilde{\cos}(\eta_j) &= \sum_{i=1}^{p} \lambda_i \cos(\eta_{j,i}), \\
\widetilde{\sin}(\eta_j) &= \sum_{i=1}^{p} \lambda_i \sin(\eta_{j,i}), \\
\widetilde{\cos}(2\zeta_j - \eta_j) &= \cos(2\zeta_j)\ \widetilde{\cos}(\eta_j) + \sin(2\zeta_j)\ \widetilde{\sin}(\eta_j), \\
\widetilde{\sin}(2\zeta_j - \eta_j) &= \sin(2\zeta_j)\ \widetilde{\cos}(\eta_j) - \cos(2\zeta_j)\ \widetilde{\sin}(\eta_j), \\
\eta_j &= \sum_{i=1}^{p} \lambda_i \eta_{j,i}, \\
\sum_{i=1}^{p} \lambda_i &= 1,\ 0 \leq \lambda_i,\ 1 \leq i \leq p.
\end{aligned} \tag{3.17}$$

By exploiting the trigonometric formulas and the fact that the functions *sine* and *cosine* have the same argument, we have finally managed to approximate four nonlinear functions by introducing only one set λ and six linear constraints.

We now focus on the way of building the linear outer approximation domains. As shown in inequalities (3.10), the determination of these domains passes through the computation of approximation errors, which is the object of the next section.

3.2.4 Maximum errors generated by SOS approximations

As the *TVC* problem can be expressed as a problem involving three different kinds of nonlinear components, the analytical expression of the approximation errors generated by their SOS approximation must be determined for each of them. The approximation errors are computed on each piece of the domain which corresponds to a segment joining two consecutive breakpoints in case of one dimension and to a triangle defined by three adjacent breakpoints in case of two dimensions. Among the overestimation (respectively underestimation) approximation errors computed on the pieces of the domain, the maximum overestimation (respectively underestimation) approximation error used in (3.10) corresponds to the largest one.

1. $\boxed{x^2}$

Theorem 3.1 *Let $[x_i, x_{i+1}]$ be a piece of the domain where the function x^2 is replaced by its SOS approximation. The maximum overestimation and underestimation approximation errors, $\epsilon_{x^2,L}$ and $\epsilon_{x^2,U}$ respectively, generated by the SOS approximation of x^2 on this piece are such that:*

$$\epsilon_{x^2,L} = \frac{(x_{i+1} - x_i)^2}{4} \quad \textit{and} \quad \epsilon_{x^2,U} = 0.$$

Proof
Since the function x^2 is convex, its SOS approximation on $[x_i, x_{i+1}]$, which corresponds

to the segment joining (x_i, x_i^2) and (x_{i+1}, x_{i+1}^2), always overestimates it. Therefore, the underestimation error, $\epsilon_{x^2,U}$, generated by the SOS approximation of x^2 is always equal to zero. It remains thus to establish the expression of the overestimation error. On the interval $[x_i, x_{i+1}]$, it can be easily shown that the analytical expression of the SOS approximation of x^2 is given by $\tilde{f}(x) = ax + b$, where $a = x_{i+1} + x_i$ and $b = -x_i x_{i+1}$. Accordingly, the overestimation approximation error at x is equal to:

$$e(x) = \tilde{f}(x) - f(x) = (x_{i+1} + x_i)x - x_i x_{i+1} - x^2, \tag{3.18}$$

and is always positive. As this error is equal to zero, and thus minimized, at the bounds of the interval $[x_i, x_{i+1}]$, the derivative of $e(x)$, that is,

$$e'(x^*) = x_{i+1} + x_i - 2x^*,$$

must be zero at the point x^* which maximizes the error on this interval. Hence,

$$x^* = \frac{x_{i+1} + x_i}{2}.$$

The maximum overestimation error is thus reached at the half of the interval $[x_i, x_{i+1}]$. By evaluating the equation (3.18) at x^* and factorizing the obtained expression, the value of the maximum overestimation error generated by the SOS approximation of x^2 on the piece $[x_i, x_{i+1}]$ can be written as:

$$\epsilon_{x^2,L} = \frac{(x_{i+1} - x_i)^2}{4}.$$

□

The maximum overestimation approximation error produced on a piece thus depends on the length of this piece. To take the right approximation errors into account on the corresponding pieces, binary variables y_i satisfying (2.20), like in the lambda method, should be introduced in the formulation to specify which piece is considered. So, in the outer approximation problem, by (3.10), the approximation w_{x^2} of x^2 should satisfy:

$$\sum_{i=1}^{p} \lambda_i x_i^2 - \sum_{i=1}^{p-1} \epsilon_{x^2,L_i}\, y_i \leq w_{x^2} \leq \sum_{i=1}^{p} \lambda_i x_i^2, \tag{3.19}$$

where ϵ_{x^2,L_i} is the maximum overestimation error produced on the piece $[x_i, x_{i+1}]$. However, in practice, to avoid to introduce binary variables in the problem, we relax (3.19) in:

$$\sum_{i=1}^{p} \lambda_i x_i^2 - \max_{i=1,p-1} \epsilon_{x^2,L_i} \leq w_{x^2} \leq \sum_{i=1}^{p} \lambda_i x_i^2. \tag{3.20}$$

To obtain approximations w_{x^2} as tight as possible, the maximum on the quantities ϵ_{x^2,L_i} should be minimized. As the errors ϵ_{x^2,L_i} depend on the length of the piece, this amounts to minimize the length of the pieces, and thus, for a fixed number of breakpoints, to choose *equally spaced breakpoints*. In this case, each piece $[x_i, x_{i+1}]$ has the same length,

denoted Δ_x. Therefore, the maximum overestimation error generated by the SOS approximation of x^2 is equal on each piece and is such that:

$$\epsilon_{x^2,L} = \frac{\Delta_x^2}{4}. \tag{3.21}$$

By assuming that p breakpoints are used, Δ_x can be expressed in function of the range $[l_x, u_x]$ of the variable x like:

$$\Delta_x = \frac{u_x - l_x}{p-1},$$

which allows us to write (3.21) as:

$$\epsilon_{x^2,L} = \frac{(u_x - l_x)^2}{4(p-1)^2}. \tag{3.22}$$

Accordingly, by applying the inequalities (3.10) to the function x^2, the approximation w_{x^2} of x^2 in the outer approximation problem, based on p equally spaced breakpoints, must satisfy the conditions (2.13) and:

$$\sum_{i=1}^{p} \lambda_i x_i^2 - \frac{(u_x - l_x)^2}{4(p-1)^2} \leq w_{x^2} \leq \sum_{i=1}^{p} \lambda_i x_i^2. \tag{3.23}$$

We recall that the *SOS condition is not imposed* for this outer approximation.

Note[(1)] that the underestimation used in (3.20) could be strengthened. Indeed, it is sometimes too strong to remove the maximum approximum error whatever the considered piece, like in (3.20). Because it is too expensive to introduce binary variables like in (3.19) to determine which piece is considered, we work on the variables λ_i instead. Since the sum on the λ_i must be equal to one and since if the SOS condition is satisfied, only λ_i and λ_{i+1} can be nonzero if the i^{th} piece is considered, we should remove an error equal to $(\lambda_i + \lambda_{i+1})\epsilon_{x^2,L_i}$ for the i^{th} piece. However, λ_i $(i \neq 1, p)$ can be nonzero if we consider the $(i-1)^{th}$ or the i^{th} piece. Therefore, for these λ_i, to avoid to remove twice an error for the same λ_i and to be sure to obtain an underestimating function, we remove the maximum between the two errors $\epsilon_{x^2,L_{i-1}}$ and ϵ_{x^2,L_i}. That is, (3.20) would be replaced by:

$$\lambda_1\big(x_1^2 - \epsilon_{x^2,L_1}\big) + \sum_{i=2}^{p-1} \lambda_i\big(x_i^2 - \max(\epsilon_{x^2,L_{i-1}}, \epsilon_{x^2,L_i})\big) + \lambda_p\big(x_p^2 - \epsilon_{x^2,L_{p-1}}\big) \leq w_{x^2}. \tag{3.24}$$

In this case, the total removed error is given by a convex combination of the errors ϵ_{x^2,L_i} produced on the pieces, which is always smaller or equal to the maximum of these errors. However, this underestimation remains a relaxation for the underestimation of (3.19) when the variables y_i are binary, because the removed error does not necessarily correspond to the error associated to the piece to which x belongs. As the underestimation (3.24) is expected to be better than that of (3.20), it would be worth to employ it for further researches.

[(1)] Following a suggestion of Sven Leyffer in his report on the present work.

2. $\boxed{xy}$

The case of a bilinear product is now examined. For each dimension, we choose a fixed number of breakpoints and build from them a grid composed of rectangles, which must be again decomposed into triangles to satisfy the SOS type 3 condition. Assume that we divide each rectangle of the grid in two triangles in the way used in Figure 2.5. Two types of triangles are obtained.

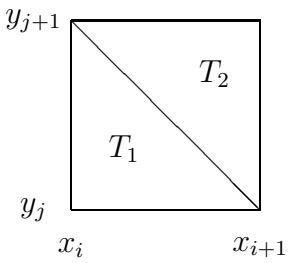


Figure 3.6: Decomposition of a rectangle of the grid into two triangles.

Each triangle is defined by three breakpoints: (x_i, y_j), (x_i, y_{j+1}) and (x_{i+1}, y_j) for T_1 and (x_{i+1}, y_{j+1}), (x_i, y_{j+1}) and (x_{i+1}, y_j) for T_2. The maximum approximation errors are first computed for triangles like T_1.

Theorem 3.2 *Let T_1 be a triangle defined by the breakpoints (x_i, y_j), (x_i, y_{j+1}) and (x_{i+1}, y_j). The SOS approximation of the bilinear product xy on this triangle is given by:*

$$\tilde{f}(x, y) = xy_j + yx_i - x_i y_j,$$

and the maximum overestimation and underestimation approximation errors, $\epsilon_{xy,L}$ and $\epsilon_{xy,U}$ respectively, satisfy:

$$\epsilon_{xy,L} = 0 \quad \textit{and} \quad \epsilon_{xy,U} = \frac{(x_{i+1} - x_i)(y_{j+1} - y_j)}{4}.$$

Proof

As explained in Section 2.2.3, the SOS approximation of xy on a triangle T_1 corresponds to the plan joining the three points $(x_i, y_j, x_i y_j)$, $(x_i, y_{j+1}, x_i y_{j+1})$ and $(x_{i+1}, y_j, x_{i+1} y_j)$ which is defined by the following equation:

$$\tilde{f}(x, y) = xy_j + yx_i - x_i y_j. \tag{3.25}$$

Indeed, it can be easily shown that the three points, which are not collinear by construction, satisfy (3.25). Since McCormick has shown in [81] that, on a rectangle $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$, the inequality:

$$xy \geq xy_j + yx_i - x_i y_j,$$

is always valid, we can conclude that the SOS approximation $\tilde{f}(x, y)$ underestimates the function xy on T_1. The associated overestimation approximation error is thus equal to zero. It remains to compute the maximum underestimation approximation error. The underestimation approximation error, which is always positive, is given by:

$$e(x, y) = f(x, y) - \tilde{f}(x, y) = xy - xy_j - yx_i + x_i y_j, \tag{3.26}$$

and its gradient by:

$$\nabla e(x, y) = \begin{pmatrix} y - y_j \\ x - x_i \end{pmatrix}.$$

This gradient is zero at (x_i, y_j) which is precisely a point defining the triangle. At this point, the approximation error is equal to zero. Therefore, a minimum has been found since the underestimation approximation error is always positive. As a consequence, this error must be reached on a side of the triangle. Three cases must be considered:

- 1^{st} *case:* Side of T_1 defined by $x = x_i$. On this side, by (3.26), the underestimation approximation error is such that:

  $$e(x_i, y) = x_i y - x_i y_j - y x_i + x_i y_j = 0.$$

  The bilinear function and its SOS approximation coincide on this side.

- 2^{nd} *case:* Side of T_1 defined by $y = y_j$. On this side, the underestimation approximation error can be computed as:

  $$e(x, y_j) = x y_j - x y_j - y_j x_i + x_i y_j = 0.$$

  Again, the bilinear function and its SOS approximation coincide.

- 3^{rd} *case:* Since the approximation error is equal to zero on the two other sides of the triangle, the maximum underestimation approximation error must be reached on the hypotenuse. As shown in Figure 3.6, the hypotenuse of T_1 is the line joining (x_i, y_{j+1}) to (x_{i+1}, y_j). Therefore, it can be derived that it is defined by:

  $$y = \frac{y_{j+1} - y_j}{x_i - x_{i+1}}\, x + \frac{x_i y_j - x_{i+1} y_{j+1}}{x_i - x_{i+1}}. \tag{3.27}$$

  By expressing y in function of x in (3.26) and simplifying the obtained expression, the underestimation approximation error produced on the hypotenuse is such that:

  $$e(x, y(x)) = \frac{y_{j+1} - y_j}{x_i - x_{i+1}}\big(x^2 - (x_i + x_{i+1})x + x_i x_{i+1}\big). \tag{3.28}$$

  As the first derivative of this error with respect to x is given by:

  $$e'(x, y(x)) = \frac{y_{j+1} - y_j}{x_i - x_{i+1}}\,(2x - x_i - x_{i+1}),$$

  and its second derivative by:

  $$e''(x, y(x)) = 2\,\frac{y_{j+1} - y_j}{x_i - x_{i+1}},$$

  which is negative since the breakpoints are increasingly ordered, the point x^* for which the first derivative is zero, that is,

  $$x^* = \frac{x_i + x_{i+1}}{2},$$

corresponds to the maximum that we are looking for. By replacing this value in (3.27), it can be obtained after some manipulations that the associated value of y is:

$$y^* = \frac{y_j + y_{j+1}}{2}.$$

The point (x^*, y^*) which maximizes the underestimation approximation error thus corresponds to the half point of the hypotenuse. By evaluating the equation (3.26) at this point, the expression of the maximum underestimation approximation error can be deduced:

$$\epsilon_{xy,U} = e(x^*, y^*) = \frac{(x_{i+1} - x_i)(y_{j+1} - y_j)}{4}. \tag{3.29}$$

□

By using a similar reasoning, we can obtain similar conclusions for triangles of type T_2.

Corollary 3.3 *Let T_2 be a triangle defined by the breakpoints (x_{i+1}, y_{j+1}), (x_i, y_{j+1}) and (x_{i+1}, y_j). The SOS approximation of the bilinear product xy on this triangle is given by:*

$$\tilde{f}(x, y) = xy_{j+1} + yx_{i+1} - x_{i+1}y_{j+1},$$

and the maximum overestimation and underestimation approximation errors, $\epsilon_{xy,L}$ and $\epsilon_{xy,U}$ respectively, satisfy:

$$\epsilon_{xy,L} = 0 \quad \textit{and} \quad \epsilon_{xy,U} = \frac{(x_{i+1} - x_i)(y_{j+1} - y_j)}{4}.$$

The approximation errors have the same expression for triangles of type T_1 and T_2. By taking equally spaced breakpoints, the maximum underestimation approximation error is identical for each piece and minimized on the entirety of the approximation domain, like for the square function. It is equal to:

$$\epsilon_{xy,U} = \frac{\Delta_x \Delta_y}{4}, \tag{3.30}$$

where Δ_x (respectively Δ_y) is the discretization step for x, respectively for y, i.e., the length between two breakpoints. Expressing this error according to the range of variables, by taking p_x equally spaced breakpoints for the x-dimension and p_y for the y-one, we obtain:

$$\epsilon_{xy,U} = \frac{(u_x - l_x)(u_y - l_y)}{4(p_x - 1)(p_y - 1)}. \tag{3.31}$$

Observe that in order to approximate the bilinear product by its SOS approximation, the approximation domain can be decomposed into triangles in the way shown in Figure 2.5, as well as by using the cutting presented in Figure 3.7. By using this cutting, it can be demonstrated by a similar proof to the one of Theorem 3.2 that the SOS approximation $\tilde{f}(x, y)$ of xy defined on triangles like in Figure 3.7 satisfies:

$$xy \leq \tilde{f}(x, y),$$

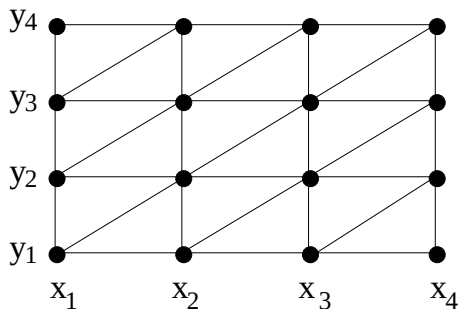


Figure 3.7: Converse cutting of the grid into triangles.

and that the maximum approximation errors, $\epsilon_{xy,L}$ and $\epsilon_{xy,U}$, are such that:

$$\epsilon_{xy,L} = \frac{(u_x - l_x)(u_y - l_y)}{4(p_x - 1)(p_y - 1)} \quad \text{and} \quad \epsilon_{xy,U} = 0. \tag{3.32}$$

Therefore, the bilinear product xy is overestimated by its SOS approximation based on triangles like in Figure 3.7 while it is underestimated by its SOS approximation based on triangles like in Figure 2.5. Since we do not impose the SOS condition for the proposed outer approximations, any triangle like in Figure 2.5 or Figure 3.7 can be generated since no $\lambda_{i,j}$ is imposed to be equal to zero. As a consequence, there always exists a convex combination of $\lambda_{i,j}$ satisfying conditions (2.26) to (2.29) and which produces an overestimating SOS approximation of xy and another one for which the associated SOS approximation is underestimating. In addition, Theorem 3.4 demonstrates that for any point (x, y, xy), there exists a unique combination of $\lambda_{i,j}$ which satisfies conditions (2.26) to (2.29) and such that $\tilde{f}(x, y) = f(x, y) = xy$. This amounts to show that any point (x, y, xy) can be written as a unique convex combination of points $(x_i, y_j, x_i y_j)$ where (x_i, y_j) are breakpoints.

Theorem 3.4 *Each feasible point (x, y, xy) with $l_x \leq x \leq u_x$ and $l_y \leq y \leq u_y$ can be written as a unique convex combination of the four points $(l_x, l_y, l_x l_y)$, $(l_x, u_y, l_x u_y)$, $(u_x, l_y, u_x l_y)$ and $(u_x, u_y, u_x u_y)$.*

Proof
To demonstrate the theorem, we need to show that there exist some positive reals $a,\ b,\ c$ and d such that:

$$\begin{pmatrix} x \\ y \\ xy \end{pmatrix} = a \begin{pmatrix} l_x \\ l_y \\ l_x l_y \end{pmatrix} + b \begin{pmatrix} l_x \\ u_y \\ l_x u_y \end{pmatrix} + c \begin{pmatrix} u_x \\ l_y \\ u_x l_y \end{pmatrix} + d \begin{pmatrix} u_x \\ u_y \\ u_x u_y \end{pmatrix}, \tag{3.33}$$

and

$$a + b + c + d = 1. \tag{3.34}$$

Since the variable x is defined on the interval $[l_x, u_x]$, it can be expressed as a convex combination of the extremities of this interval, l_x and u_x, i.e., there exists some μ belonging to $[0, 1]$ such that:

$$x = (1 - \mu)\, l_x + \mu\, u_x. \tag{3.35}$$

In the same way, there exists some ν belonging to $[0, 1]$ such that:

$$y = (1 - \nu)\, l_y + \nu\, u_y. \tag{3.36}$$

By using the expressions of x and y in function of μ and ν, the bilinear product xy can be transformed into:

$$xy = (1 - \mu)(1 - \nu)\, l_x l_y + (1 - \mu)\nu\, l_x u_y + \mu(1 - \nu)\, u_x l_y + \mu\nu\, u_x u_y. \tag{3.37}$$

It can be shown that the values:

$$\begin{array}{l} a = (1 - \mu)(1 - \nu), \\ b = (1 - \mu)\nu, \\ c = \mu(1 - \nu) \text{ and} \\ d = \mu\nu, \end{array}$$

satisfy conditions (3.33), (3.34) and the positivity condition on these reals. Indeed, by replacing in (3.33), a, b, c and d by the values suggested above and simplifying the obtained expressions, the equalities on the three components amount to (3.35), (3.36) and (3.37) respectively, which proves their validity. The satisfaction of condition (3.34) can be easily obtained by summing the values of the reals a, b, c and d and factorizing. The positivity constraint on a, b, c and d also holds since μ and ν belong to the interval $[0, 1]$. Finally, the convex combination is unique since it can be easily shown that the four vectors:

$$\begin{pmatrix} l_x \\ l_y \\ l_x l_y \\ 1 \end{pmatrix}, \begin{pmatrix} l_x \\ u_y \\ l_x u_y \\ 1 \end{pmatrix}, \begin{pmatrix} u_x \\ l_y \\ u_x l_y \\ 1 \end{pmatrix} \text{ and } \begin{pmatrix} u_x \\ u_y \\ u_x u_y \\ 1 \end{pmatrix}, \tag{3.38}$$

used in the system composed of (3.33) and (3.34) are linearly independent.

□

Note that four points (and not three) are necessary to generate any point (x, y, xy) because we impose that (x, y, xy) is written as a convex combination of the reference points. Therefore, four equations must be satisfied, that is, the system (3.33) and (3.34), which explains the fact that four points are needed. Note that this theorem is not trivial since a positivity constraint is imposed on the coefficients of a convex combination.

As a direct consequence of this theorem, for each point (x, y) there exists a convex combination fulfilling conditions (2.27) to (2.29) for which the bilinear product and its linear approximation are equal at this point as long as the variables $\lambda_{i,j}$ associated to the four extreme breakpoints are not set to zero. This is the case since the SOS conditions are no longer imposed. Accordingly, we do not have to introduce approximation errors in inequalities (3.10) to guarantee that the correct value of the bilinear product xy at a feasible point (x, y) can be produced in the linear outer approximation problem. Consequently, for the bilinear product, the inequalities (3.10) can be replaced by the equality:

$$w_{xy} = \sum_{i=1}^{p_x} \sum_{j=1}^{p_y} \lambda_{i,j} x_i y_j, \tag{3.39}$$

which amounts to condition (2.26) applied to the bilinear function. Note that this reasoning cannot be applied to the function x^2 since the value (x, x^2) cannot be generated by a convex combination of the values (x_i, x_i^2) (where x_i is a breakpoint) unless in the particular case where x is a breakpoint.

3. Trigonometric functions

While the approximation errors for the square and bilinear functions depend on the size of the pieces more than on the approximation interval, the approximation errors for trigonometric functions have a different analytical expression according to the piece where the function is approximated. Again, the domain where the trigonometric functions are approximated is assumed to belong to the interval $[0, 2\pi]$. The computation of the approximation errors for the functions $sine$ and $cosine$ must be distinguished. The analysis begins with the $sine$ function.

a. $\sin(x)$

Theorem 3.5 *Let $[x_i, x_{i+1}] \subseteq [0, 2\pi]$ be a piece of the domain where the function $\sin(x)$ is replaced by its SOS approximation $\tilde{f}(x)$ which is defined by $\tilde{f}(x) = a_i x + b_i$, where:*

$$a_i = \frac{\sin x_{i+1} - \sin x_i}{x_{i+1} - x_i} \quad \textit{and} \quad b_i = \frac{x_{i+1} \sin x_i - x_i \sin x_{i+1}}{x_{i+1} - x_i}. \tag{3.40}$$

Then, the maximum overestimation and underestimation approximation errors, ϵ_{sin,L_i} and ϵ_{sin,U_i} respectively, generated by the SOS approximation of $\sin(x)$ on the interval $[x_i, x_{i+1}]$ are such that:

$$\epsilon_{sin,L_i} = \begin{cases} a_i(2\pi - \arccos(a_i)) + b_i & \textit{if } 2\pi - \arccos(a_i) \in [x_i, x_{i+1}], \\ \quad + \sin(\arccos(a_i)) & \\ 0 & \textit{otherwise,} \end{cases}$$

$$\epsilon_{sin,U_i} = \begin{cases} -a_i \arccos(a_i) - b_i + \sin(\arccos(a_i)) & \textit{if } \arccos(a_i) \in [x_i, x_{i+1}], \\ 0 & \textit{otherwise.} \end{cases}$$

Proof

On the piece $[x_i, x_{i+1}]$, the SOS approximation of $\sin(x)$ corresponds to the segment joining $(x_i, \sin(x_i))$ to $(x_{i+1}, \sin(x_{i+1}))$ which is defined by $\tilde{f}(x) = a_i x + b_i$, where a_i and b_i are given in (3.40). The definition of the approximation error for $\sin(x)$ can thus be written as:

$$e(x) = f(x) - \tilde{f}(x) = \sin(x) - a_i x - b_i. \tag{3.41}$$

At points where this approximation error is positive, the SOS approximation underestimates the $sine$ function while it overestimates it if $e(x)$ is negative. Therefore, we are looking for the optima of $e(x)$, minima or maxima. Since the approximation error vanishes at the breakpoints x_i and x_{i+1}, the optima that interest us must be reached in $]x_i, x_{i+1}[$ at points where the derivative of $e(x)$ is equal to zero. As the derivative of $e(x)$ is given by:

$$e'(x) = \cos(x) - a_i,$$

the points where $e'(x)$ is zero on $[0, 2\pi]$ are, by inverting the $cosine$ function,

$$x_a^* = \arccos(a_i),$$

or

$$x_b^* = 2\pi - \arccos(a_i).$$

Since the value of $\arccos(x)$ is always in the interval $[0, \pi]$, it is also the case for x_a^* while x_b^* belongs to $[\pi, 2\pi]$. These two points correspond to optima for the approximation error on $[x_i, x_{i+1}]$ if they belong to this interval. We discuss three cases.

- If $x_{i+1} \leq \pi$, the maximum approximation error on $[x_i, x_{i+1}]$ is reached at x_a^* (because x_b^* is not in the studied interval which must contain an optimum), and it is an underestimation error because $\sin(x)$ is concave on the considered interval. In this case, the SOS approximation always underestimates $\sin(x)$ and the overestimation approximation error vanishes.
- If $x_i \geq \pi$, the maximum approximation error is reached at x_b^* and it is an overestimation error since $\sin(x)$ is convex on the studied domain and, as a consequence, its SOS approximation always overestimates it. The underestimation error generated by the SOS approximation is thus equal to zero.
- If $x_i < \pi$ and $x_{i+1} > \pi$, the SOS approximation underestimates the function $\sin(x)$ on an interval $[x_i, x_{m_i}]$ and overestimates it on $[x_{m_i}, x_{i+1}]$. In this case, x_a^* and x_b^* must both belong to $[x_i, x_{i+1}]$ and the maximum underestimation approximation error is reached at x_a^* while the maximum overestimation approximation error arises at x_b^*.

To summarize, an underestimation approximation error arises only if x_a^* belongs to $[x_i, x_{i+1}]$. By evaluating (3.41) at this point, the maximum underestimation approximation error is equal to:

$$\epsilon_{sin,U_i} = -a_i \arccos(a_i) - b_i + \sin(\arccos(a_i)).$$

On the other hand, an overestimation approximation error is produced only if x_b^* belongs to $[x_i, x_{i+1}]$. To obtain a positive value for the overestimation approximation error, the signs in (3.41) must be inverted. By evaluating this updated equation at x_b^*, the maximum overestimation approximation error is given by:

$$\epsilon_{sin,L_i} = a_i(2\pi - \arccos(a_i)) + b_i - \sin(2\pi - \arccos(a_i)).$$

As the *sine* function is 2π-periodic and as $\sin(-x) = -\sin(x)$, the desired result can be derived.

□

As for the square function, the convex combinations of the points $(x_i, \sin(x_i))$ (where x_i is a breakpoint) cannot generate all the possible values $(x, \sin(x))$ (see Figure 2.4).

Therefore, the approximation errors must be taken into account to define the outer approximation domain. Note that, contrary to square and bilinear functions, for the trigonometric ones, the approximation errors are different in function of the pieces even if the breakpoints are equally spaced. Therefore, (3.10) becomes[(2)]:

$$\sum_{i=1}^{p} \lambda_i \sin(x_i) - \max_{i=1,p-1} \epsilon_{sin,L_i} \leq w_{sin(x)} \leq \sum_{i=1}^{p} \lambda_i \sin(x_i) + \max_{i=1,p-1} \epsilon_{sin,U_i}. \tag{3.42}$$

b. $\cos(x)$

The $cosine$ function is now considered. In this case, the analytical expression of the errors is a little bit more complicated. Indeed, by inverting the $sine$ function to determine in a similar way as in Theorem 3.5, the points for which the derivative of the approximation error is zero, there are no longer two but three solutions, as detailed in the following theorem.

Theorem 3.6 *Let $[x_i, x_{i+1}] \subset [0, 2\pi]$ be a piece of the domain where the function $\cos(x)$ is replaced by its SOS approximation $\tilde{f}(x)$ which is defined by $\tilde{f}(x) = c_i x + d_i$, where:*

$$c_i = \frac{\cos x_{i+1} - \cos x_i}{x_{i+1} - x_i} \quad \textit{and} \quad d_i = \frac{x_{i+1} \cos x_i - x_i \cos x_{i+1}}{x_{i+1} - x_i}. \tag{3.43}$$

Then, the maximum overestimation and underestimation approximation errors, ϵ_{cos,L_i} and ϵ_{cos,U_i} respectively, generated by the SOS approximation of $\cos(x)$ on the interval $[x_i, x_{i+1}]$ are such that:

$$\epsilon_{cos,L_i} = \begin{cases} c_i(\pi + \arcsin(c_i)) + d_i & \textit{if } \pi + \arcsin(c_i) \in [x_i, x_{i+1}], \\ \quad + \cos(\arcsin(c_i)) & \\ 0 & \textit{otherwise,} \end{cases}$$

$$\epsilon_{cos,U_i} = \begin{cases} c_i(\arcsin(c_i)) - d_i & \textit{if } -\arcsin(c_i) \in [x_i, x_{i+1}], \\ \quad + \cos(\arcsin(c_i)) & \\ -c_i(2\pi - \arcsin(c_i)) - d_i & \textit{if } 2\pi - \arcsin(c_i) \in [x_i, x_{i+1}], \\ \quad + \cos(\arcsin(c_i)) & \\ 0 & \textit{otherwise.} \end{cases}$$

Proof

Since the SOS approximation of $\cos(x)$ on the piece $[x_i, x_{i+1}]$ is given by the segment $\tilde{f}(x) = c_i x + d_i$ where c_i and d_i are defined in (3.43), the approximation error generated by the SOS approximation of $\cos(x)$ on this piece is given by:

$$e(x) = f(x) - \tilde{f}(x) = \cos x - c_i x - d_i. \tag{3.44}$$

A positive error corresponds to an underestimation of $\cos(x)$ by its SOS approximation while a negative error corresponds to an overestimation. As the $cosine$ function and its SOS approximation coincide at breakpoints, the points which optimize the approximation

[(2)] An alternative formulation could be obtained by using the same reasoning as for (3.24).

error and which interest us must be in $]x_i, x_{i+1}[$, and, the derivative of the error must be zero at these points. Since the derivative of $e(x)$ is given by:

$$e'(x) = -\sin x - c_i$$

and since the image of the function $\arcsin$ is defined on $[\frac{-\pi}{2}, \frac{\pi}{2}]$, the derivative of $e(x)$ may be zero at the following points on $[x_i, x_{i+1}] \subset [0, 2\pi]$:

$$\begin{aligned} x_a^* &= \arcsin(-c_i), \\ x_b^* &= \pi - \arcsin(-c_i), \\ x_c^* &= 2\pi + \arcsin(-c_i), \end{aligned}$$

depending if x_a^*, x_b^* and x_c^* belong, or not, to the considered interval $[x_i, x_{i+1}]$. As x_a^* belongs to $[\frac{-\pi}{2}, \frac{\pi}{2}]$, x_b^* to $[\frac{\pi}{2}, \frac{3\pi}{2}]$ and x_c^* to $[\frac{3\pi}{2}, \frac{5\pi}{2}]$, by distinguishing like in Theorem 3.5, between the different possibilities according to the parts of the domain where $\cos(x)$ is concave or convex, we obtain that the maximum underestimation approximation error can be reached at x_a^* or x_c^* belonging to intervals where the $cosine$ function is concave, while the maximum overestimation approximation error can be reached at x_b^* which belongs to a convex part of the domain of $\cos(x)$. Using a similar reasoning to the one used in Theorem 3.5, that is, by replacing in equation (3.44) the different values of x^* (x_a^*, x_b^* and x_c^*) if they belong to the interval $[x_i, x_{i+1}]$, by inverting the signs in this equation to obtain a positive value for ϵ_{cos,L_i} and finally by exploiting trigonometric properties ($\arcsin(-x) = -\arcsin(x)$, $\cos(-x) = \cos(x)$, $\cos(\pi + x) = -\cos(x)$ and the periodicity of $\cos(x)$), the desired result can be derived. Note that the value of ϵ_{cos,U_i} is correctly defined since by assumption, the length of the piece $[x_i, x_{i+1}]$ is smaller than 2π. As a consequence, among the three conditions giving the value of ϵ_{cos,U_i}, only one can be fulfilled.

□

To determine the outer approximation $w_{cos(x)}$ of $\cos(x)$, a similar inequality to the one used for $sine$, (3.42), can be employed[3]:

$$\sum_{i=1}^{p} \lambda_i \cos(x_i) - \max_{i=1,p-1} \epsilon_{cos,L_i} \leq w_{cos(x)} \leq \sum_{i=1}^{p} \lambda_i \cos(x_i) + \max_{i=1,p-1} \epsilon_{cos,U_i}. \qquad (3.45)$$

In summary, an SOS approximation generates:

- an overestimation error for x^2,
- an underestimation error for xy if the cutting of Figure 2.5 is employed but an overestimation error if the cutting of Figure 3.7 is used,
- overestimation and/or underestimation errors for $sine$ and $cosine$ depending on the examined interval.

[3] By using a similar reasoning as for (3.24), this outer approximation could be strengthened.

3.2.5 Expression of the outer approximation problem

We are now able to give the expression of the linear outer approximation problem $(\widetilde{OP})$ which is nearly identical to problem $(\tilde{P})$ of Section 2.2.3. In fact, the unique difference appears in the general constraints bounding the variables $w_{j_i}^i$. Indeed, in the linear approximation problem $(\tilde{P})$ a nonlinear component $g_{j_i}^i$ is replaced by a linear function while in the linear outer approximation problem $(\widetilde{OP})$, the approximation of this component is bounded by a linear domain. In the formulation of $(\widetilde{OP})$ given below, the parameters ${\epsilon_L}_{j_i}^i$ and ${\epsilon_U}_{j_i}^i$ correspond to the maximum overestimation and underestimation approximation errors done by replacing the nonlinear component $g_{j_i}^i$ by its SOS approximation on a piece of the domain and which have been computed in the previous section for square and trigonometric functions. For the bilinear products, these errors are set to zero for the reasons detailed before. Note that $(\widetilde{P})$ is a particular case of $(\widetilde{OP})$ where all the errors ${\epsilon_L}_{j_i}^i$ and ${\epsilon_U}_{j_i}^i$ are set to zero.

$$
(\widetilde{OP}) \left\{
\begin{array}{ll}
\min & f_{lin}(x) + \displaystyle\sum_{j_0=1}^{t_0} w_{j_0}^0, \\
\text{s.t.} & g_{lin}^i(x) + \displaystyle\sum_{j_i=1}^{t_i} w_{j_i}^i = 0, \quad 1 \leq i \leq m, \\
& \displaystyle\sum_{k \in I_{M_{j_i}^i}} (\lambda_{j_i}^i)_k (g_{j_i}^i)_k - {\epsilon_L}_{j_i}^i \leq w_{j_i}^i \leq \sum_{k \in I_{M_{j_i}^i}} (\lambda_{j_i}^i)_k (g_{j_i}^i)_k + {\epsilon_U}_{j_i}^i, \\
& \qquad\qquad 1 \leq i \leq m,\ 1 \leq j_i \leq t_i, \\
& \displaystyle\sum_{k \in I_{M_{j_0}^0}} (\lambda_{j_0}^0)_k (g_{j_0}^0)_k - {\epsilon_L}_{j_0}^0 \leq w_{j_0}^0, \ \ 1 \leq j_0 \leq t_0, \\
& x_{j_i}^i = \displaystyle\sum_{k \in I_{M_{j_i}^i}} (\lambda_{j_i}^i)_k (x_{j_i}^i)_k, \quad 0 \leq i \leq m,\ 1 \leq j_i \leq t_i, \\
& \displaystyle\sum_{k \in I_{M_{j_i}^i}} (\lambda_{j_i}^i)_k = 1, \quad 0 \leq (\lambda_{j_i}^i)_k, \quad k \in I_{M_{j_i}^i},\ 0 \leq i \leq m,\ 1 \leq j_i \leq t_i, \\
& x_{j_i}^i = x|_{g_{j_i}^i}, \quad 0 \leq i \leq m,\ 1 \leq j_i \leq t_i, \\
& l_x \leq x \leq u_x, \\
& x \in \mathbb{R}^n.
\end{array}
\right.
$$

Note that in case of inequality constraints or for the objective function, only the underestimation of the variables $w_{j_i}^i$ used to replace a component involved in these functions is needed.

Remember that the approximation errors for the unary functions have been computed by assuming that the SOS conditions are fulfilled, while these conditions are not imposed in the problem $(\widetilde{OP})$. The reasons of this choice are now detailed. Without taking the approximation errors into account, the linear approximation of a function f at a point x can be equal to any value $w_{f(x)}$ such that $(x, w_{f(x)})$ belongs to DPV defined by:

$$
DPV = \left\{ (x, w_{f(x)}) \mid w_{f(x)} = \sum_{i=1}^{p} \lambda_i f_i,\ x_i = \sum_{i=1}^{p} \lambda_i x_i,\ \ \sum_{i=1}^{p} \lambda_i = 1,\ \lambda_i \geq 0,\ 1 \leq i \leq p \right\}. \tag{3.46}
$$

However, the possible values $(x, w_{f(x)})$ do not always correspond to good approximations as highlighted in Figure 2.4. Among the possible approximations, we would like to keep the good ones like SOS approximations. Therefore, to guarantee that a point x feasible for the nonlinear problem belongs to the feasible domain of the outer approximation problem and can produce in this problem the same value as in the nonlinear problem for the approximated components, the approximation domain has been extended by using the maximum approximation errors generated by the SOS approximation, and not the maximum approximation errors generated on the set DPV. Indeed, the maximum approximation errors generated by the SOS approximation are smaller than the ones produced on the set DPV because the SOS approximation is only one of the numerous approximations satisfying (3.46). Using the approximation errors associated to the SOS approximation allows us to reduce the domain $DPVOA$ of possible values for the outer approximation of a function f at a point x which is given by:

$$DPVOA = \left\{ \begin{array}{l} (x, w_{f(x)}) \mid \quad \displaystyle\sum_{i=1}^{p} \lambda_i f_i - \epsilon_{f,L} \leq w_{f(x)} \leq \sum_{i=1}^{p} \lambda_i f_i + \epsilon_{f,U}, \\ \qquad\qquad x_i = \displaystyle\sum_{i=1}^{p} \lambda_i x_i, \quad \sum_{i=1}^{p} \lambda_i = 1, \quad \lambda_i \geq 0, \;\; 1 \leq i \leq p \end{array} \right\}. \quad (3.47)$$

As the errors associated to SOS approximations are used to build the outer approximations, we refer to the latter as *outer approximations based on SOS*. Note that this appellation is a little bit improper for the bilinear product since in this particular case, the approximation errors are not employed. But for the sake of generalization, we talk even though about outer approximations based on SOS.

With outer approximations, each possible value $(x, f(x))$, where f is a nonlinear component, belongs to the domain of the outer approximation $(\widetilde{OP})$. However, this does not imply that the solution of the outer approximation problem corresponds to the global optimum of the nonlinear problem since the domain of the linear outer approximation problem is larger than the one of the nonlinear problem. Indeed, each nonlinear component has been replaced by a linear domain which includes it. As a consequence, the solution of the outer approximation problem can be quite far from the one of the nonlinear problem. Accordingly, to find the global optimum of the nonlinear problem, the outer approximation problem must be refined until the approximation is sufficiently tight to guarantee that the solutions of the linear and nonlinear problems are close enough to find the solution of the nonlinear problem by starting from the one of the linear problem. This refinement is made possible by using branch-and-bound, as explained in the next section.

3.3 An outer approximation method to solve nonlinear problems

To be closer and closer to the nonlinear problem that it approximates, the linear outer approximation problem $(\widetilde{OP})$ is refined by employing a branch-and-bound tree. As we do *no longer impose the SOS condition*, the branching is not used to satisfy this conditions like in the classical SOS approximation method. This choice will be justified in Chapter 4 by demonstrating that branching on original variables and always keeping the same number of breakpoints to

approximate a same component everywhere in the tree allows us to obtain tighter approximations than the ones obtained by branching on the variables λ_i. Before going further, we illustrate this on an example.

Suppose that we branch on x_i at the middle point of its range. As the range of the variable x_i has been divided in two equal parts, to keep the same number of breakpoints, the latter must be twice as close in the two subproblems to be equally spaced. Therefore, the resulting outer approximations are better. A graphic illustration of the refinement of the outer approximation for the function x^2 on the interval $[-2, 2]$ is given in Figures 3.8, 3.9 and 3.10. The first figure represents the domain before branching of the possible values (x, w_{x^2}) for the outer approximation of x^2 based on three equally spaced breakpoints. This domain, $DPVOA$, is delimited by the continuous lines on the figure. By branching on x equal to zero, two outer approximations associated to both new subproblems are obtained, one with $x \leq 0$ and another one for which $x \geq 0$. Since, in this case, the two graphic representations are symmetric, we only consider the subproblem where $x \geq 0$. To keep constant the number of breakpoints after branching, a new breakpoint must be added. To have equally spaced breakpoints, it is introduced at $x = 1$. The obtained outer approximation is represented in Figure 3.9. It can be seen that this new outer approximation is tighter than before branching. Indeed, the upper side of the outer approximation domain after branching (the segment joining the images of the first and last breakpoints) corresponds to the SOS approximation on the piece $[0, 2]$ before branching illustrated in Figure 3.8. Figure 3.10 represents the domain of possible values obtained by branching on the variable λ. In this case, a new breakpoint cannot be added (see Section 4.2). By comparing Figures 3.8 and 3.10, we can observe that the size of the domain of possible values for the outer approximation is smaller after branching on x than on λ since the addition of a new breakpoint has allowed a refinement of the outer approximation.

Moreover, we can also note that this choice avoids, as previously explained, to use an SOS approximation error, and thus to enlarge the domain of the possible values, for the outer approximation of a bilinear product, which it is not the case when the SOS condition is imposed. Therefore, the branching is no longer done on the variables λ_i but on the *variables of the nonlinear problem*. By branching on such variables, the linear outer approximation problem can be modified since the linear outer approximations are based on the bounds of the original variables. The following section discusses these changes in the problem.

3.3.1 Refinement of the outer approximation problem due to branching

First, we recall that we treat here the case where the problem (P) is not subject to discrete restrictions. This latter case will be examined in Section 3.3.3.

After having solved the linear outer approximation problem $(\widetilde{OP})$, a variable x_i which *appears nonlinearly* in problem (P) is chosen for branching. Indeed, it is useless to branch on a variable which intervenes only linearly in the problem since it does not generate any approximation error, and branching on it would not improve the quality of the outer approximation problem. Once the branching variable x_i has been chosen (see Chapter 6 for more details), we branch on it at a point m_{x_i} belonging to the approximation domain of x_i and the current problem is divided into two subproblems by imposing for the left subproblem that:

$$x_i \leq m_{x_i}, \tag{3.48}$$

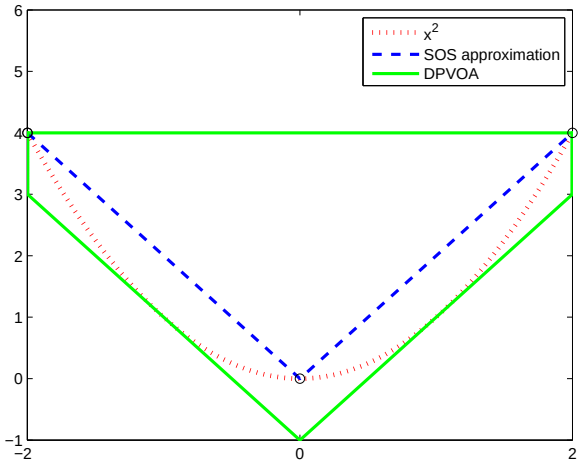


Figure 3.8: Domain before branching of the possible values (x, w_{x^2}) for the outer approximation of x^2 when $x \in [-2, 2]$.

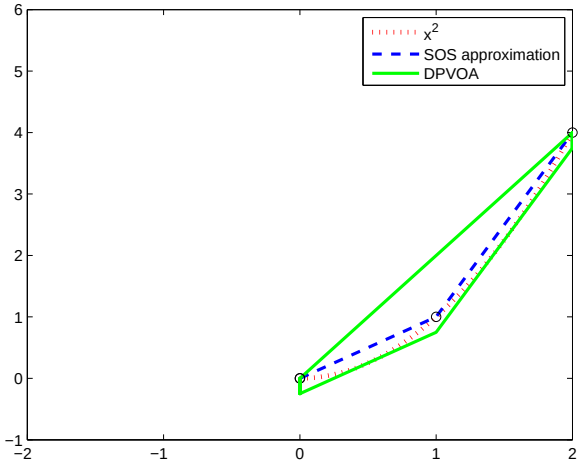


Figure 3.9: Domain after branching of the possible values (x, w_{x^2}) for the outer approximation of x^2 when $x \in [0, 2]$.

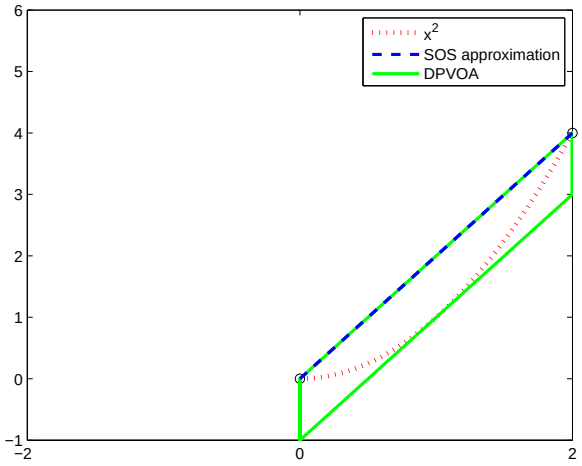


Figure 3.10: Domain after branching on λ_2 of the possible values (x, w_{x^2}) for the outer approximation of x^2 when $x \in [0, 2]$.

and for the right one that:

$$x_i \geq m_{x_i}. \tag{3.49}$$

By branching, the bounds on the variable x_i are tightened: the upper bound u_{x_i} can be updated to m_{x_i} for the left subproblem while the lower bound l_{x_i} can be set to m_{x_i} for the right one. As the range of the variable x_i has changed, the approximations of all nonlinear components in which this variable appears can be refined. We decide to always use the *same number of breakpoints* to approximate a same nonlinear component throughout the branch-and-bound tree. These breakpoints are also chosen *equally spaced* to have a uniform information on the approximation domain. Moreover, as highlighted in Section 3.2.4, this choice allows us to minimize on each piece of the domain, the approximation errors generated by the SOS approximation of the square function and thus to minimize the size of the domain $DPVOA$ (3.47) for this function.

The refinement of the outer approximation explained above needs the modification of the general constraints of the outer approximation problem $(\widetilde{OP})$ because some of them depend on breakpoints. Note that we do not limit ourselves to refine the approximation of the components of one or two variables in which the variable x_i directly appears, but if the refined components are themselves arguments of other components of one or two variables, we also refine the approximation for these components. For example, assume that we have a trilinear product $x_1x_2x_3$ decomposed in w_1 which approximates x_1x_2 and w_2 which replaces x_3w_1, and that the branching is done on x_1. Therefore, the bounds on x_1 but also on w_1 and w_2 can be updated by using the bound propagation technique detailed in Section 3.2.2. As the bounds on the variables are modified, the associated breakpoints are updated in order to keep equally spaced breakpoints. Accordingly, the maximum approximation errors must be modified since the pieces have changed. A tighter outer approximation problem is thus obtained. More schematically, after branching on a variable x_i, Algorithm 3.1 is applied for each generated subproblem.

Algorithm 3.1: Update of the outer approximation problem after branching on x_i

1. Update the *bounds* on x_i and, by bound propagation, all the bounds on components w_{k_i} involving x_i as well as the bounds on components having a component w_{k_i} as argument.
2. Update the *breakpoints* associated to a variable for which the bounds have been updated at Step 1, in order to keep constant their number throughout the branch-and-bound tree and also to keep them equally spaced.
3. Update the *general constraints* of $(\widetilde{OP})$ which depend on breakpoints which have been modified at Step 2.
4. Update the *maximum approximation errors* associated to approximations which have been modified at Step 3 and, as a consequence, update the *bounds on the associated general constraints*.

3.3.2 Scheme of the method

Since we have explained how to update the outer approximation problem $(\widetilde{OP})$ after branching on a variable, we can now detail the general process of the method that we have developed

and which can be seen as a variant of the branch-and-bound strategy detailed in Section 1.2.4. We start by building the linear outer approximation problem $(\widetilde{OP})$ from the nonlinear problem (P), as explained in Section 3.2.5, and then we solve the obtained problem. As the linear problem is an outer approximation problem with regard to the nonlinear one, the optimum value of problem $(\widetilde{OP})$ consists of a *lower bound* for the nonlinear problem. Therefore, if the linear outer approximation problem is infeasible, the nonlinear problem (P) is also infeasible. If it is not the case, the nonlinear problem is solved by starting from the solution of the linear problem. If the nonlinear solver finds an optimum value, denoted U^*, for the nonlinear problem, it consists of an *upper bound* for this problem. Indeed, there is no guarantee that the optimum found is the global one. But at this time, U^* is the best optimum value found so far. If the difference between the upper and lower bounds is smaller than a fixed small quantity ϵ, both problems can be considered to be close enough and the solution of the nonlinear problem is the global optimum within the accuracy ϵ.

However, the situation where the optimum values of the nonlinear problem and its outer approximation are directly so close seldom arises. Therefore, the outer approximation problem must be refined. In this way, a variable x_i appearing nonlinearly in problem (P) is chosen and we branch on it. To guarantee the convergence of the method, the branching variable must produce an approximation error at the current solution. Indeed, it would be useless to branch on a variable for which the approximations are exact at the current solution, for all the components in which it appears since the branching cannot improve the quality of the outer approximation problem. Different ways to choose the branching variable are tested and compared in Chapter 6. By branching on x_i like in (3.48) and in (3.49), two new linear outer approximation subproblems are created and updated by Algorithm 3.1. These subproblems are ordered in a branch-and-bound tree where they are associated to nodes. These nodes are put in the stack of nodes to treat.

One subproblem is chosen and solved. Different techniques to select this subproblem are tested and discussed in Chapter 7. If this linear outer approximation subproblem is infeasible or if its optimum value is larger than the best optimum value, U^*, found so far, the node associated to this subproblem can be fathomed, as in the classical branch-and-bound strategy (see Section 1.2.4). Otherwise, the nonlinear problem is solved by starting from the solution of the linear outer approximation subproblem. If a solution is found for (P) and if the optimum value produced is better than the best optimum value, U^*, found so far, U^* is updated. When the optimum values of the outer approximation and the nonlinear problems solved at this node are within the accuracy ϵ, the node can be fathomed since on the part of the domain examined at this node, we have found the global optimum of the nonlinear problem within the accuracy ϵ. On the other hand, if the nonlinear solver cannot find a solution for the problem (P) by starting from the solution of the linear outer approximation problem, the associated node cannot be fathomed because there is no guarantee that the nonlinear problem is infeasible on this part of the domain because the nonlinear solver is a local one. Therefore, we must pursue refining the linear outer approximation problem and we put the node associated to this subproblem in the stack of nodes.

After this, a new subproblem which needs to be treated is chosen in the tree. If the selected subproblem has been already solved, it corresponds to a subproblem which must be refined by being divided in two new subproblems. Therefore, we choose a branching variable and branch on it. This creates two new subproblems which are updated by Algorithm 3.1. Among these

two subproblems, we select one, solve it and act as explained above. The other node is put in the stack of nodes to treat. Note that if the upper bound U^* can be updated, all nodes of the stack having a lower bound on the optimum value larger than the new upper bound can be fathomed. Next, the process is repeated by choosing in the tree a subproblem to treat. As the other subproblem generated by branching has not been solved, it belongs to the set of nodes to explore. If it is chosen to be treated, it is no longer necessary to choose a branching variable since this step has already been done before, and the subproblem can be directly solved. Therefore, among the set of nodes to treat, they are two kinds of nodes: *nodes to divide* and *nodes to solve*. Finally, if the whole tree has been explored, the global optimum value of the nonlinear problem (P) corresponds to U^* unless all nonlinear problems solved during the process of the algorithm were infeasible, in which case the nonlinear problem is infeasible.

Note that we do not branch directly and do not create two subproblems after having solved successfully a linear subproblem since the associated nodes could be discarded if a better upper bound U^* was found before they are examined. In this case, we would have chosen a branching variable needlessly. We will see in Chapter 6 that this choice can be expensive. Moreover, branching only when we choose to treat the node allows us to consider and to store only one node to explore (the node to divide) instead of two, which limits the memory space needed to solve the problem.

The method proposed above is given more schematically in Algorithm 3.2 and the proof of its convergence is established in Theorem 3.7.

Theorem 3.7 *If all variables appearing nonlinearly in the nonlinear problem (P) are bounded, Algorithm 3.2 converges to the global optimum of problem (P), within the accuracy ϵ, in a finite number of iterations.*

Proof
This theorem is demonstrated in two steps: we begin by showing that a node associated to a domain which contains a better optimum for the problem (P) than the one found so far by the algorithm cannot be cut and next, we show the finiteness of the algorithm.

As the approximation problems $(\widetilde{OP}^k)$ are linear, the solution found, if the problem is feasible, is the global one on the domain explored at iteration k. Moreover, as $(\widetilde{OP}^k)$ is an outer approximation problem for the nonlinear problem (P), the global optimum of $(\widetilde{OP}^k)$ consists of a valid lower bound for the one of (P) on this domain. Therefore, a node is fathomed only if the associated feasible domain does not contain a better solution then the current one since Algorithm 3.2 fathoms nodes in the three following cases:

1. *If the linear problem $(\widetilde{OP}^k)$ is infeasible,* the node can be fathomed because any solution feasible for the problem (P) on the domain examined at iteration k is always feasible for its outer approximation problem $(\widetilde{OP}^k)$. As a consequence, the problem (P) is also infeasible on the part of the domain explored at iteration k.

2. *If the optimum value of the linear problem $(\widetilde{OP}^k)$ is larger than the best optimum value found so far for the nonlinear problem,* the node can also be discarded since the optimum value of $(\widetilde{OP}^k)$ always corresponds to a lower bound for the one of problem (P) on the domain under study. Accordingly, the domain associated to the node examined at iteration k cannot contain a better solution.

Algorithm 3.2: Outer approximation method based on SOS (continuous case)

Let ϵ be a fixed accuracy for the method.

Init: Set $U^* := +\infty$, the current optimum value for the nonlinear problem (P).
Build the linear outer approximation problem $(\widetilde{OP}^0)$ from the nonlinear problem (P) and place it at the root node of a branch-and-bound tree.
Set the index of the current iteration, k, to zero.

while (the tree has not been completely explored) **do**

1. *Choose a linear problem* to treat in the tree.
   If this problem has never been solved **then** Go to 3.

2. *Choose a branching variable* which generates an approximation error*, branch on it, create two new subproblems* and *update them*, as explained in Algorithm 3.1.
   Add the two subproblems to the branch-and-bound tree and select one of these two subproblems to solve.

3. *Solve* the selected subproblem $(\widetilde{OP}^k)$.
   Let $\tilde{x}_k$ be its solution and w_{f_k} its optimum value, if it is feasible.
   If ($(\widetilde{OP}^k)$ is infeasible) or ($w_{f_k} \geq U^* - \epsilon$) **then** Fathom the node and go to 5.

4. *Solve the nonlinear problem* (P) by starting from $\tilde{x}_k$.
   If (a solution has been found for (P) by starting from $\tilde{x}_k$) **then**
   Let x_k^* and f_k^* be the optimum solution and value for (P) obtained by starting from $\tilde{x}_k$.
   If ($f_k^* < U^*$) **then** Update the current solution by setting $x^* := x_k^*$ and $U^* := f_k^*$.
   Fathom all the nodes of the stack having a lower bound on the optimum value of (P) larger than $U^* - \epsilon$
   If ($f_k^* - w_{f_k} \leq \epsilon$) **then** Fathom the node.

5. Set $k := k + 1$.

end (while)

3. *The optimum value of the linear outer approximation problem is close, within an accuracy ϵ, to the optimum value of the nonlinear problem at the node examined at iteration k.* Therefore, the solution of the nonlinear problem (P) can be considered as the global one, within the accuracy ϵ, on the domain associated to the node. When the optimum values of problems $(\widetilde{OP}^k)$ and (P) are so close, the examined node can be cut, after having updated the best optimum value for (P) if necessary, since the value of the global optimum of (P) is known, within an accuracy ϵ, on the domain associated to the examined node.

Accordingly, a node is fathomed only when the associated domain is guaranteed not to contain a better optimum than the current one. It remains to prove that any branch can be stopped by the fathoming of a node after a finite number of iterations. If none of the three conditions above to fathom a node is fulfilled, we branch on a variable which generates an approximation error at the current solution of the linear problem with regard to the nonlinear one. By branching on such variables, their ranges are reduced and by keeping the same number of breakpoints to approximate a same component throughout the branch-and-bound tree, the outer approximation problems are tightened and become closer and closer to the nonlinear one when one goes down in the tree. Therefore, as the ranges of the variables used in the approximations are bounded by assumption, after branching a finite number of times, the outer approximation problems will be close enough to the nonlinear problem in such a way that one of the three above conditions to cut a node will be satisfied.

□

Note that this theorem requires that all variables appearing nonlinearly in the problem are bounded. This assumption is also needed to build the linear outer approximation problem since the employed linear outer approximations are based on an interval. However, in *TVC* problem, the variables θ which appear in trigonometric functions ($\cos(\zeta_j + \theta_i - \theta_k)$ for example) are unbounded. In (3.15), the quantity $\zeta_j + \theta_i - \theta_k$ has been replaced by a new variable η_j. As the variables θ are unbounded, the variable η_j is also unbounded. Nevertheless, as the trigonometric functions are 2π-periodic, the analysis of these functions can be limited to the interval $[0, 2\pi]$. The variable η_j has thus been bounded by 0 and 2π, which amounts to also bound the quantity $\zeta_j + \theta_i - \theta_k$ by these values. This could discard feasible solutions but, in practice, for classical electrical networks involved in the *TVC* problem, the difference $\theta_i - \theta_k$ is not expected to vary a lot around the parameters ζ_j. In the treated problems, the value of these parameters is such that we can impose that the quantities $\zeta_j + \theta_i - \theta_k$ belong to the interval $[0, 2\pi]$. Nevertheless, if this assumption is turned out to be too strong, it could be relaxed, even if some bounds (not necessarily tight) must still be imposed on each quantity $\eta_j = \zeta_j + \theta_i - \theta_k$. By introducing a new integer variable, t, belonging to a bounded interval and an additional constraint in the modelling, the quantity $\eta_j \in [l, u]$ can be transformed into a new variable η'_j in order that η'_j is bounded by 0 and 2π, that is:

$$\eta'_j = \eta_j - 2\pi t \ \in [0, 2\pi]. \tag{3.50}$$

As a multiple of 2π is removed from η_j, the values of the trigonometric functions at η_j and η'_j coincide because of the periodicity of these functions.

This formulation introduces an integer variable in the problem to solve. Until now, the treated problem has been assumed to depend on continuous variables only. The following section discusses the modifications necessary to take discrete variables into account.

3.3.3 Adaptation of the method to the discrete case

If the nonlinear problem to solve is subject to some discrete restrictions, Algorithm 3.2 must be adapted. This task can be realized relatively easily because the method has developed an ideal framework to treat discrete variables. Indeed, it uses a branch-and-bound process, which is often employed to solve integer programs. With discrete variables, the goal of branching is

double: to refine the outer approximations but also to satisfy the discrete restrictions. Accordingly, the candidate variables for branching are both the variables which appear *nonlinearly* in the original problem and the *discrete* ones.

In Algorithm 3.2 developed in the continuous case, the linear outer approximation and the nonlinear problems must be solved. However, in discrete case, it would be too costly to find the discrete solution of these problems at each node of the branch-and-bound tree, which would imply to solve mixed integer linear and nonlinear problems at each node. Therefore, to obtain a lower bound on the optimum value of the nonlinear problem, we limit ourselves to solve at each node the *continuous relaxation* of the mixed integer linear problem. The discrete restrictions are gradually enforced as one goes down in the branch-and-bound tree.

In the continuous case, the optimum value of the nonlinear problem (P) solved by starting from the solution of the linear outer approximation problem (if an optimum has been found) corresponds to an upper bound for the global optimum of the nonlinear problem. In the discrete case, it is not necessarily true if we solve the continuous relaxation of the nonlinear problem because the obtained solution does not usually fulfill the discrete restrictions. To use the solution of the continuous relaxed nonlinear problem as an upper bound, this solution must be guaranteed to be feasible for the discrete nonlinear problem. Therefore, we do not solve the continuous relaxation of the nonlinear problem, but we solve the nonlinear problem in which the value of each discrete variable is *fixed* to the closest discrete value to its current value at the solution of the associated linear outer approximation problem. For example, if at the solution of the linear outer approximation problem, an integer variable has a value of 6.12, the value of this variable is fixed to 6 in the associated nonlinear problem. In this way, the solution of the nonlinear problem is ensured to be feasible with regard to the discrete restrictions.

According to what precedes, Algorithm 3.2 has been adapted to the discrete case to obtain Algorithm 3.3, where the modifications are highlighted in italic. Note that Theorem 3.7 about the convergence to the global optimum and the finiteness of the method remains applicable if the discrete variables can take only a finite number of values, which is the case for the *TVC* problem.

Bound propagation

To refine the outer approximations, specific techniques can be developed to treat the discrete variables. So, the link between a discrete variable (not integer) only feasible for a finite number of equally spaced values (typically the ratio of voltage of *TVC* problem) and the integer variable allowing us to model this constraint can be exploited to establish a suitable bound propagation technique. Indeed, suppose that a discrete variable x defined on $[l_x^0, u_x^0]$ can be equal to only $k+1$ equally spaced values given by:

$$x_k = l_x^0 + z\frac{u_x^0 - l_x^0}{k} \quad \text{where } z \text{ is integer and } z \in [0, k].$$

If we branch on the integer variable z, the bounds on the variable x can also be updated from the new bounds of z given by $[l_z, u_z]$, where l_z and u_z are integer, as:

$$l_x := l_x^0 + l_z\frac{u_x^0 - l_x^0}{k} \quad \text{and} \quad u_x := l_x^0 + u_z\frac{u_x^0 - l_x^0}{k}. \tag{3.51}$$

This allows us to refine next the outer approximations of components in which the variable x intervenes. In the same way, if we branch on the variable x, the bounds on the integer variable

Algorithm 3.3: Outer approximation method based on SOS (discrete case)

Let ϵ be a fixed accuracy for the method.

Init: Set $U^* := +\infty$, the current optimum value for the nonlinear problem (P).
Build the linear outer approximation problem $(\widetilde{OP}^0)$ from the nonlinear problem (P) and place it at the root node of a branch-and-bound tree.
Set the index of the current iteration, k, to zero.

while (the tree has not been completely explored) **do**

1. Choose a linear problem to treat in the tree.
   If this problem has never been solved **then** Go to 3.

2. Choose a branching variable which generates an approximation error *or which violates a discrete restriction*, branch on it, create two new subproblems and update them as explained in Algorithm 3.1.
   Add the two subproblems to the branch-and-bound tree and select one of these two subproblems to solve.

3. Solve *the continuous relaxation* of the selected subproblem $(\widetilde{OP}^k)$.
   Let $\tilde{x}_k$ be its solution and w_{f_k} its optimum value, if it is feasible.
   If ($(\widetilde{OP}^k)$ is infeasible) or ($w_{f_k} \geq U^* - \epsilon$) **then** Fathom the node and go to 5.

4. *Fix the values of the discrete components of $\tilde{x}_k$ to the feasible discrete values closest to their values at $\tilde{x}_k$.*
   Solve the nonlinear problem (P) by starting from $\tilde{x}_k$.
   If (a solution has been found for (P) by starting from $\tilde{x}_k$) **then**
   Let x_k^* and f_k^* be the optimum solution and value for (P) obtained by starting from $\tilde{x}_k$.
   If ($f_k^* < U^*$) **then** Update the current solution by setting $x^* := x_k^*$ and $U^* := f_k^*$.
   Fathom all the nodes of the stack having a lower bound on the optimum value of (P) larger than $U^* - \epsilon$.
   If ($f_k^* - w_{f_k} \leq \epsilon$) **then** Fathom the node.

5. Set $k := k + 1$.

end (while)

z can be updated from the bounds $[l_x, u_x]$ by:

$$l_z := \left\lceil k \, \frac{l_x - l_x^0}{u_x^0 - l_x^0} \right\rceil \quad \text{and} \quad u_z := \left\lfloor k \, \frac{u_x - l_x^0}{u_x^0 - l_x^0} \right\rfloor . \tag{3.52}$$

3.3.4 Illustration of the method

Algorithm 3.3 is now applied on an example which has been chosen to be intentionally simple in order to give a graphic illustration of the method. Assume that we want to find the global optimum, within an accuracy ϵ equal to 10^{-3}, of the problem (P_{ex}) which is only subject to a discrete restriction:

$$(P_{ex}) \begin{cases} \min & 3\sin(x) + 0.2\,(x-1)^2, \\ \text{s.t.} & x \in [0, 2\pi] \text{ and } x \text{ is a multiple of } \frac{2\pi}{9}. \end{cases}$$

To model the discrete restriction on x, a new integer variable z belonging to the interval $[0, 9]$ is introduced in the problem by means of the constraint:

$$x = z\frac{2\pi}{9}.$$

The graphic representation of the continuous objective function of (P_{ex}) as a function of x only is given in Figure 3.11. The circles represent the points which satisfy the discrete restriction

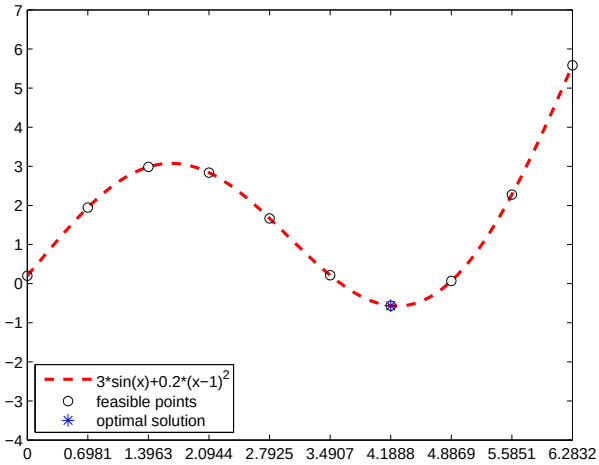


Figure 3.11: Graphic representation of problem (P_{ex}).

and the one which minimizes the objective function f, that is $x^* = 4.189$, is highlighted.

We start by building the linear outer approximation problem $(\widetilde{OP_{ex}})$ for (P_{ex}). We should mention first that in this case, an underestimation problem would be enough to solve the problem since the nonlinear functions only appear in the objective function of a minimization problem. Therefore, there is no need to overestimate them. However, to illustrate the proposed method, we build the linear outer approximation problem. To this aim, we decide to use only one set λ based on five equally spaced breakpoints x_i, that is,

$$x_i = l_x + (i-1)\,\frac{u_x - l_x}{4}, \quad 1 \le i \le 5, \tag{3.53}$$

where l_x and u_x are the current lower and upper bounds on x. Each time these bounds are modified, the breakpoints are updated. By approximating $\sin(x)$ and x^2 by two new variables $w_{\sin(x)}$ and w_{x^2} thanks to conditions (2.17), (2.18), (3.23) and (3.42), the linear outer approximation problem is given by:

$$(\widetilde{OP_{ex}}) \begin{cases} \min & 3w_{\sin(x)} + 0.2\,(w_{x^2} - 2x + 1), \\ \text{s.t.} & x = z\frac{2\pi}{9} \quad \text{with } z \text{ integer}, \\ & w_{\sin(x)} \geq \sum_{i=1}^{5} \lambda_i \sin(x_i) - \epsilon_{\sin,L}, \\ & w_{\sin(x)} \leq \sum_{i=1}^{5} \lambda_i \sin(x_i) + \epsilon_{\sin,U}, \\ & w_{x^2} \geq \sum_{i=1}^{5} \lambda_i x_i^2 - \epsilon_{x^2,L}, \\ & w_{x^2} \leq \sum_{i=1}^{5} \lambda_i x_i^2, \\ & x = \sum_{i=1}^{5} \lambda_i x_i, \\ & 1 = \sum_{i=1}^{5} \lambda_i, \\ & 0 \leq \lambda_i, \quad 1 \leq i \leq 5, \\ & l_x \leq x \leq u_x, \\ & l_z \leq z \leq u_z, \\ & l_{w_{\sin(x)}} \leq w_{\sin(x)} \leq u_{w_{\sin(x)}}, \\ & l_{w_{x^2}} \leq w_{x^2} \leq u_{w_{x^2}}. \end{cases}$$

In this problem, x, z, $w_{\sin(x)}$, w_{x^2} and λ_i, $1 \leq i \leq 5$ are the variables. We denote the objective function $w_{f(x)}$. The errors $\epsilon_{\sin,L}$ and $\epsilon_{\sin,U}$ correspond to the maximum overestimation and underestimation approximation errors done on a piece $[x_i, x_{i+1}]$ by approximating $\sin(x)$ by its SOS approximation. These two errors are computed on each piece by using Theorem 3.5 and among these errors, the maximum ones are taken. The overestimation approximation error generated by x^2, $\epsilon_{x^2,L}$, is given by the formula (3.22). The parameters l_k and u_k denote the lower and upper bounds on the variable k. Note that the bounds on $w_{\sin(x)}$ and w_{x^2} can be computed by using the bound propagation technique detailed in Section 3.2.2. The update of these bounds allows us to discard solutions where the approximation $w_{\sin(x)}$ of $\sin(x)$ does not belong to the interval $[-1, 1]$, for example. If this update of the bounds on new variables is not taken into account, the domain given by the possible values $(x, w_{f(x)})$ for the outer approximation problem $(\widetilde{OP_{ex}})$ relaxed with regard to the discrete restriction is illustrated in Figure 3.12. On this picture, the image of the five breakpoints are represented by circles and the SOS approximation of the function to minimize corresponds to the piecewise linear function which links the images of two consecutive breakpoints. Note that the expression of this piecewise linear function $w_{f(x)} = \sum_{i=1}^{5} \lambda_i (3\sin(x_i) + 0.2\,(x_i^2 - 2x + 1))$ can be written, by linearity, in function of the linear approximations of $\sin(x)$ and x^2 as:

$$3\sum_{i=1}^{5}(\lambda_i \sin(x_i)) \; + 0.2\sum_{i=1}^{5}(\lambda_i x_i^2) + \; 0.2(-2x + 1).$$

However, the constraints of problem $(\widetilde{OP_{ex}})$ are not sufficient to guarantee to have an SOS approximation since the SOS type 2 condition is not explicitly required. Therefore, all points belonging to the convex hull based on the breakpoints are feasible. This convex hull, which corresponds to the domain DPV defined in (3.46) (in fact the set of the possible values $(x, w_{f(x)})$ for the linear approximation method $(\tilde{P})$ of Section 2.2.3) is also represented in the figure. Note that the left and right sides of this convex hull correspond to a part of the SOS approximation.

Observe also that the global optimum of the nonlinear problem highlighted by a star does not belong to this convex hull. Therefore, to ensure that the optimum solution, x^*, of the nonlinear problem can produce an objective value equal to $f(x^*)$ in the approximation problem, the domain DPV has been extended by adding or removing the approximation errors ($\epsilon_{\sin,L}$, $\epsilon_{\sin,U}$ and $\epsilon_{x^2,L}$) to have an outer approximation problem. Accordingly, the obtained domain, $DPVOA$, includes the previous one, as shown in Figure 3.12.

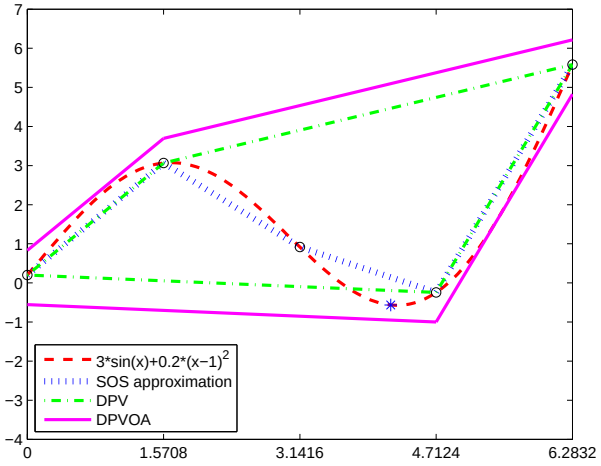


Figure 3.12: Graphic representation of the continuous relaxation of problem $(\widetilde{OP_{ex}})$ without taking the bounds on $w_{\sin(x)}$ and w_{x^2} into account.

But this domain can be a little bit tightened since it does not take the bounds on the new variables $w_{\sin(x)}$ and w_{x^2} into account. The domain obtained by discarding the parts of the domain which violate the bound constraints on $w_{\sin(x)}$ and w_{x^2} is presented in Figure 3.13. The minimum on this domain for the linear outer approximation problem relaxed with regard to the discrete restriction is highlighted by a star. The problem (P_{ex}) is now solved by using Algorithm 3.3.

1. LP1

   In the problem $(\widetilde{OP_{ex}})$, x belongs to $[0, 2\pi]$. Therefore, the bounds on the new variables are given by $l_{w_{\sin(x)}} = -1$, $u_{w_{\sin(x)}} = 1$, $l_{w_{x^2}} = 0$ and $u_{w_{x^2}} = 4\pi^2$ by using the bound propagation technique of Section 3.2.2. As five breakpoints placed at 0, $\frac{\pi}{2}$, π, $\frac{3\pi}{2}$ and 2π are used, by Theorem 3.5 and equation (3.22), the maximum approximation errors are such that $\epsilon_{\sin,L} = \epsilon_{\sin,U} = 0.211$ and $\epsilon_{x^2,L} = \frac{\pi^2}{16}$. By solving the linear outer approximation problem $(\widetilde{OP_{ex}})$ relaxed with regard to its discrete restriction, the obtained optimum value, $w^*_{f(x)}$, is equal to -0.905 and the solution is such that $x^* = 3.720$ and $z^* = 5.329$, which corresponds to the star of Figure 3.13. The solution does not satisfy the SOS type 2 condition because $\lambda = (0.211, 0, 0, 0.789, 0)$ and, logically, the integer restriction is also violated.

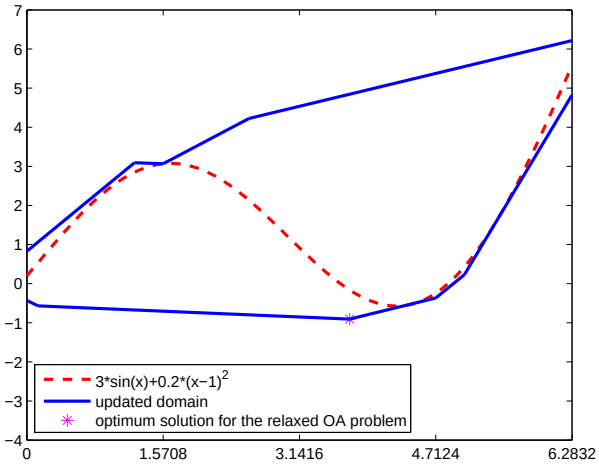


Figure 3.13: Graphic representation of the continuous relaxation of problem $(\widetilde{OP_{ex}})$ by taking the bounds on $w_{\sin(x)}$ and w_{x^2} into account.

2. NLP1
The nonlinear problem (P_{ex}) is next solved by starting from the solution of the linear outer approximation problem in which the value of the discrete variable has been fixed to its closest feasible value. As $z^* = 5.329$ at the solution of $(\widetilde{OP_{ex}})$, z is equal to 5 for the solution of the nonlinear problem. Note that for this simple problem, the nonlinear problem must not be really solved. Indeed, by fixing the value of z, the value of x is directly determined and is equal to 3.491. It remains to evaluate the objective function of (P_{ex}) at this value, which gives an optimum value, $f^*(x)$, equal to 0.215. Since it is the first feasible solution found for (P_{ex}), U^* is updated by:

$$U^* = 0.215.$$

As there is no reason to fathom the node, we choose to branch on the discrete variable z at $z^* = 5.329$, and, as a consequence, we obtain two new feasible intervals for z: $I_1 = [0, 5]$ and $I_2 = [6, 9]$. Each of these two intervals is associated to a new problem which is added to a branch-and-bound tree. This tree is represented in Figure 3.15. The way to choose the branching variable and the problem to examine first will be discussed in the following chapters. Note that in this particular case, the interval I_1 could be reduced to $[0, 4]$ because the optimum found when z is equal to 5 is guaranteed to be the global one for the problem (P_{ex}) because all variables of the problem were fixed. Nevertheless, for the illustration of the method and since this particular case does not arise with more complicated problems, we keep $I_1 = [0, 5]$.

3. LP2
Assume that the next problem to solve is the one associated to the feasible interval for z given by $I_2 = [6, 9]$. By (3.51), the variable x must belong to the interval $[4.189, 2\pi]$. By propagating these bounds on $w_{\sin(x)}$ and w_{x^2}, the bounds on these variables can be updated by $l_{w_{\sin(x)}} = -1$, $u_{w_{\sin(x)}} = 0$, $l_{w_{x^2}} = 17.546$ and $u_{w_{x^2}} = 4\pi^2$. Since the breakpoints are

based on the lower and upper bounds on x, they are also modified by (3.53) to be less spaced, which implies that the approximations are refined. The errors associated to the new SOS approximations are given by $\epsilon_{\sin,L} = 0.033$, $\epsilon_{\sin,U} = 0$ and $\epsilon_{x^2,L} = 0.069$. The graphic representation of the resulting linear outer approximation problem relaxed with regard to the discrete restriction, is given in Figure 3.14. The domain DPV is delimited by continuous lines while the domain $DPVOA$ containing all possible values $(x, w_{f(x)})$ for the outer approximation problem (by taking the bounds on $w_{\sin(x)}$ and w_{x^2} into account) is delimited by discontinuous lines. In this case, the upper side of these two domains coincide since the SOS approximation of $\sin(x)$ and x^2 never underestimate the approximated functions on the considered domain for x. Therefore, the underestimation error is equal to zero. Note that the difference between both domains becomes quite small. By solving the problem $(\widetilde{OP_{ex}})$ updated with the values given above, the optimum value

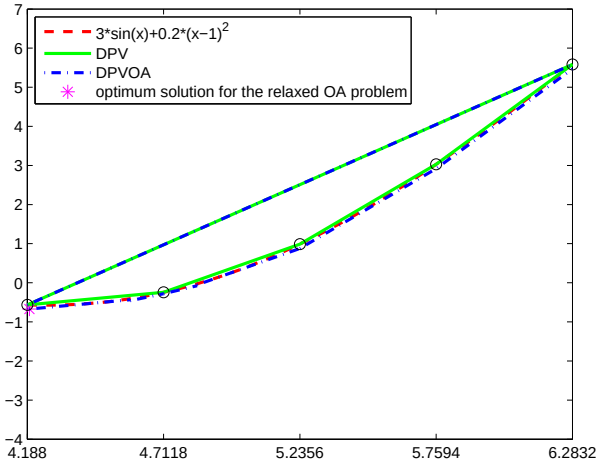


Figure 3.14: Graphic representation of the continuous relaxed problem $(\widetilde{OP_{ex}})$ if $z \in [6, 9]$.

and solution are given by $w^*_{f(x)} = -0.672$, $x^* = 4.196$ and $z^* = 6.011$. As it could be expected, the value of the objective function has increased with regard to the first linear problem solved. However, it is not high enough compared to U^* in order to fathom the node. The associated nonlinear problem must thus be solved.

4. NLP2
By fixing z to the closest feasible integer to its current value, 6.011, i.e., by fixing z to 6, the value of x is directly fixed to 4.189, which gives a better optimum value for the nonlinear problem than the one previously found. Accordingly, U^* is updated with this value:
$$U^* = -0.564.$$

5. LP3
We choose to pursue refining the linear outer approximation problem treated. Again, the problem is splitted in two new subproblems that are added to the branch-and-bound tree. We branch on $z = 6.011$, which gives two feasible "intervals" for z: $I_3 = \{6\}$ and

$I_4 = [7, 9]$. We select to solve the problem associated to I_3. For this subproblem, the value of z is fixed to 6. As a consequence, the value of x can be directly determined. All the variables of the original problem have thus a fixed value. As an outer approximation defined at a point, and not on an interval, amounts to the approximated nonlinear function, the optimum values of the problems $(\widetilde{OP_{ex}})$ and (P_{ex}) coincide. Since the global optimum of the nonlinear problem when z is equal to 6 has already been found (see NLP2), the node can be fathomed. Another way to reason would be to say that the difference between the optimum values of the linear and nonlinear problem (here equal to zero because these are the same problems as all the variables are fixed), is smaller than the fixed accuracy ϵ and that, therefore, the node can be fathomed.

6. LP4
   We now choose to examine the problem associated to the interval $I_4 = [7, 9]$. With such values for z, the variable x must belong by (3.51) to the interval $[4.887, 2\pi]$. By propagating these bounds on $w_{\sin(x)}$ and w_{x^2}, we obtain $l_{w_{\sin(x)}} = -0.984$, $u_{w_{\sin(x)}} = 0$, $l_{w_{x^2}} = 23.882$ and $u_{w_{x^2}} = 4\pi^2$. Since the breakpoints based on the lower and upper bounds of x have changed, the SOS approximation errors must also be updated by: $\epsilon_{\sin,L} = 0.014$, $\epsilon_{\sin,U} = 0$ and $\epsilon_{x^2,L} = 0.030$. By injecting these values in $(\widetilde{OP_{ex}})$ and solving its continuous relaxation, the obtained optimum value is equal to $w^*_{f(x)} = 0.068$. As this value is larger than U^*, we can conclude that we cannot find a better solution on this node, and accordingly, we can fathom it.

7. LP5
   We go up in the branch-and-bound tree to examine the last problem not yet solved, that is, the one associated to $I_1 = [0, 5]$. The update of the bounds on the variables is given by $[l_x, u_x] = [0, 3.491]$, $[l_{w_{\sin(x)}}, u_{w_{\sin(x)}}] = [-0.342, 1]$ and $[l_{w_{x^2}}, u_{w_{x^2}}] = [0, 12.185]$, while the approximation errors are given by $\epsilon_{\sin,L} = 0.001$, $\epsilon_{\sin,U} = 0.091$ and $\epsilon_{x^2,L} = 0.190$. The optimum value of the updated problem $(\widetilde{OP_{ex}})$ relaxed with regard to the discrete restriction is given by $w^*_{f(x)} = 0.159$ which is larger than U^*. As a consequence, the node can be fathomed. Since the whole tree has been explored, the solution of the problem (P_{ex}) is reached at $x^* = 4.189$ (with $z^* = 6$) and its optimum value is equal to -0.564.

3.4 Conclusion

In this chapter, we have explained the reason of the failure of the usual SOS approximation method on the *TVC* problem and we have proposed to decompose each nonlinear function of the problem in nonlinear components of one or two variables in order to reduce the size of the linear problems. We have also shown that the approximation based on SOS can be better than the one based on a big-M formulation. After having highlighting the features that should be improved in the classical SOS approximation method, we have exploited these observations and have developed a method that globalizes the SOS approximation problem by replacing each nonlinear function by a *linear domain* which includes it. This allows us to obtain an outer approximation problem. These domains are based on SOS approximations and more particularly, on the approximation errors generated by the latter. These errors have been computed for each kind of components of one or two variables appearing in the treated problem. Each function

of the problem defined on a space of more than two dimensions has indeed be decomposed into components of one or two variables, which produces a decrease in the quality of the used approximations. However, this decomposition allows us to reduce the number of variables necessary in the linear outer approximation problem, to exploit the specificities of the problem in order to refine the approximations and finally to develop a general framework easily adaptable to treat a large number of problems. The specificities of the problem have been also exploited to reduce the size of the linear outer approximation problem.

The linear outer approximation problems built to solve the nonlinear problem are no longer subject to the SOS conditions but we use the fact that good approximations fulfill these conditions, in order to refine the approximation. Without imposing these conditions, tight approximations can even though be obtained by refining the outer approximation problem through a branch-and-bound tree and updating breakpoints. In this context, the treatment of discrete restrictions is not a major difficulty. With the method developed in this chapter, the global optimum of the nonlinear problem itself is found, after a finite number of iterations, instead of the global optimum of its linear SOS approximation as it is usually the case in approximation methods.

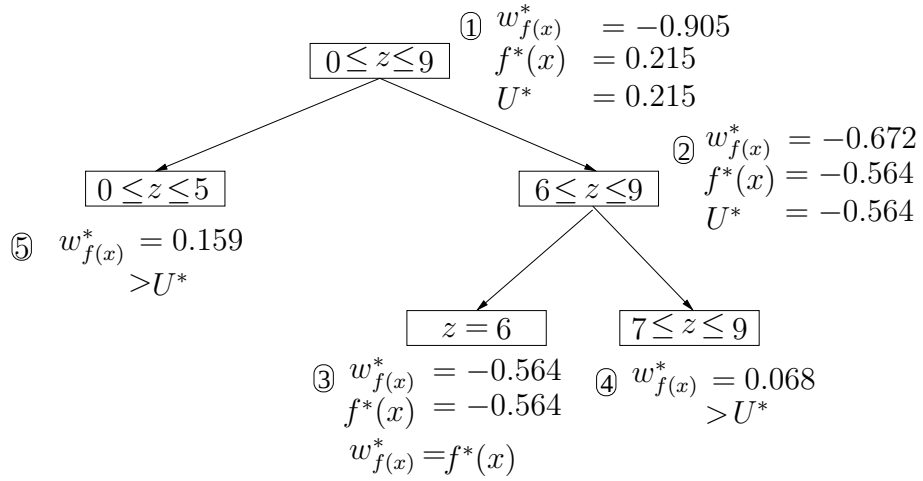


Figure 3.15: Branch-and-bound tree developed to solve the problem (P_{ex}).

Chapter 4

Theoretical considerations on the proposed method

This chapter aims at theoretically justifying the interest of the proposed method. Firstly, the outer approximations developed in Section 3.2 are compared with other outer approximation techniques employed in the literature: *tangent lines* to a convex function, *McCormick's inequalities* [81] and *convex trigonometric underestimators* for trigonometric functions (Caratzoulas and Floudas [24]). The advantages and drawbacks of these techniques and ours are highlighted and show that the outer approximations proposed in this thesis are competitive with the other ones, in particular for the *TVC* problem. Secondly, the question of the choice of the branching variable is treated. By always using the same number of breakpoints to approximate a same nonlinear component everywhere in the branch-and-bound tree, that is, by *updating breakpoints* when one goes down in this tree, we will see that branching on variables λ_i is no longer convenient and that it is preferable to branch on *original* variables. A comparison of the size of the domains of possible values for the outer approximation obtained by branching on the variables λ_i without updating breakpoints or by branching on original variables and updating breakpoints is also performed for the square and bilinear functions. The case of trigonometric functions is not considered for the reasons mentioned in Section 4.2. The performed comparison underlines the advantage of updating breakpoints and thus, of branching on original variables.

4.1 Comparison with other outer approximation techniques

As the *TVC* problem can be formulated by using three kinds of functions and as different outer approximation techniques have been developed in the literature for these functions, they are again separately considered. The analysis starts with the square function.

4.1.1 Outer approximations of x^2 based on SOS versus on tangent lines

As the square function is convex, a usual way (see Polisetty and Gatzke [96], for instance) to bound this function below by a linear function consists in bounding it by its *tangent lines* at some points of its domain, since a tangent line to a convex function always underestimates it. To bound above a convex function f defined on an interval $[x_1, x_p]$, the best linear function which can be employed corresponds to the linear function, denoted ϕ, joining $(x_1, f(x_1))$ and

$(x_p, f(x_p))$. As f corresponds to the square function in the present case, it is easy to show that ϕ is given by:

$$\phi(x) = (x_1 + x_p)\, x - x_1 x_p. \tag{4.1}$$

Note that this linear function is also the upper limit of the domain of possible values for the outer approximation of x^2 proposed in Section 3.2. Therefore, the difference between the outer approximations based on SOS or delimited below by tangent lines arises in the lower limit of the outer approximation domains. This is illustrated in Figure 4.1, which represents the domain of possible values for the outer approximation of x^2, on the left when based on SOS with four equally spaced breakpoints and on the right when defined by the tangent lines at these breakpoints and by the straight line ϕ bounding the square function above. We now show that if the set λ used in the outer approximations proposed in Section 3.2 and if the tangent lines are both based on the same number p of equally spaced breakpoints, then the two compared outer approximation techniques are *equivalent with regard to two different quality measures*: the maximum approximation errors and the area of the domain of the possible values for (x, w_{x^2}).

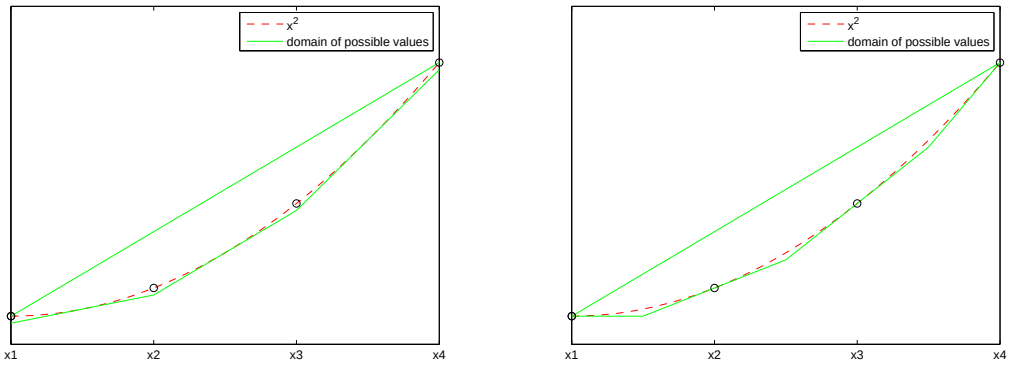


Figure 4.1: Domains of possible values for the outer approximation of x^2: on the left, based on SOS, and on the right, based on its tangent lines.

Maximum approximation errors

First of all, we examine the maximum overestimation and underestimation approximation errors that the two outer approximation techniques based on p equally spaced breakpoints can produce on the approximation domain denoted $[l_x, u_x]$. Note that the maximum overestimation approximation error is identical in both cases since the upper limit of the domain of possible values for the two approximation techniques is the same. It thus remains to compare the underestimation approximation errors.

1. *Outer approximation based on SOS*
   In Theorem 3.1, the expression of the maximum overestimation approximation error produced on a piece by the SOS approximation of x^2 has been established. To guarantee that the domain DPV defined in (3.46) does not discard any possible value (x, x^2) for the nonlinear approximated problem, this domain has been enlarged to give $DPVOA$ (see (3.47)) by bringing down the lower limit of DPV by the overestimation error defined in

Theorem 3.1. Since the lower limit of DPV, which is given by the SOS approximation of x^2, overestimates x^2 everywhere, except at breakpoints at which it is equal to x^2 (see Figure 3.8), the maximum underestimation approximation error produced by the outer approximation of x^2 based on SOS is reached at breakpoints. Moreover, it is equal to the maximum overestimation approximation error $\epsilon_{x^2,L}$ produced on a piece. Denoting $\epsilon_{\lambda(x^2),U}$ this maximum underestimation approximation error, we then have by (3.22):

$$\epsilon_{\lambda(x^2),U} = \frac{(u_x - l_x)^2}{4(p-1)^2}. \tag{4.2}$$

2. *Outer approximation based on tangent lines*
For this outer approximation technique, the lower limit of the domain of possible values is determined by tangent lines. The expression of the tangent line of x^2 at a breakpoint x_i, $1 \leq i \leq p$, and denoted T_i, is given by:

$$T_i \equiv y = 2x_i x - x_i^2. \tag{4.3}$$

Since the maximum underestimation error on a piece is reached at the intersection of two tangent lines of x^2 defined at two consecutive breakpoints, as shown on the right part of Figure 4.1, the point x^* which maximizes the underestimation error must be one of the points x_i^*, $1 \leq i \leq p-1$, that satisfy:

$$2x_i x_i^* - x_i^2 = 2x_{i+1} x_i^* - x_{i+1}^2, \tag{4.4}$$

or equivalently, by isolating x_i^*:

$$x_i^* = \frac{x_i + x_{i+1}}{2}. \tag{4.5}$$

As the underestimation approximation error, denoted $e_U(x)$, is equal at x_i^* to:

$$e_U(x_i^*) = (x_i^*)^2 - T_i(x_i^*) = (x_i^*)^2 - 2x_i x_i^* + x_i^2,$$

by replacing x_i^* by its value given in (4.5), it can be derived that:

$$e_U(x_i^*) = \frac{(x_{i+1} - x_i)^2}{4}. \tag{4.6}$$

As the breakpoints are equally spaced, the quantity $x_{i+1} - x_i$ is the same for all indices i, $1 \leq i \leq p-1$, and each point $x_i^* = \frac{x_i + x_{i+1}}{2}$ at the half of a piece produces the same value for the underestimation approximation error, that is, (4.6). Therefore, the maximum underestimation approximation error, that we denote $\epsilon_{\text{tangente}(x^2),U}$, is equal to this value. By expressing this error in function of the range of the variable x and not in function of the breakpoints, we find that:

$$\epsilon_{\text{tangente}(x^2),U} = \frac{(u_x - l_x)^2}{4(p-1)^2},$$

since $x_{i+1} - x_i = \frac{u_x - l_x}{p-1}$. We thus obtain the same maximum underestimation approximation error as in (4.2).

The maximum underestimation and overestimation approximation errors are thus equivalent for the two outer approximation techniques. Observe however that the maximum underestimation errors are reached at breakpoints for the outer approximation based on SOS while they are at the half of the pieces for the outer approximations based on tangent lines. We now demonstrate that the area of the domain of possible values is also identical for the two outer approximation techniques.

Area of the domain of possible values for the outer approximations

1. *Outer approximation based on SOS*
   We have the following result.

   Theorem 4.1 *The area of the domain of possible values for the outer approximation of x^2 based on SOS with p equally spaced breakpoints is given by:*

$$A_\lambda = \frac{\Delta_x^3}{12}\,\frac{2p^2-4p+3}{(p-1)^2},$$

   where Δ_x is the range of the variable x, that is, $\Delta_x = u_x - l_x$.

   Proof
   As presented in Figure 4.2, the domain of possible values for the outer approximation of x^2 based on SOS can be decomposed into $p-1$ trapeziums where the number of trapeziums corresponds to the number of pieces. The "height", H, of a trapezium is equal to the length of a piece, i.e., $\frac{\Delta_x}{p-1}$. The length of the bases, the parallel sides defined by $x = x_i$, $1 \leq i \leq p$, and denoted B_i, is equal to the sum of $\epsilon_{\lambda(x^2),U}$, the maximum underestimation error defined in (4.2) generated by the outer approximation of x^2 based on SOS and $\epsilon_{\text{over},x^2}(x_i)$, the maximum overestimation approximation error produced at x_i. By expressing (4.2) in function of Δ_x, $\epsilon_{\lambda(x^2),U}$ becomes:

$$\epsilon_{\lambda(x^2),U} = \frac{\Delta_x^2}{4(p-1)^2}\cdot \tag{4.7}$$

   The maximum overestimation approximation error that can be produced at x_i, $\epsilon_{\text{over},x^2}(x_i)$, is equal to the difference at point x_i between the upper limit $\phi(x)$ defined at (4.1) and the square function, that is,

$$\begin{aligned}\epsilon_{\text{over},x^2}(x_i) &= \phi(x_i) - x_i^2,\\ &= (x_1+x_p)x_i - x_1x_p - x_i^2,\\ &= (x_p - x_i)(x_i - x_1).\end{aligned}$$

   By using the fact that the breakpoints are equally spaced to write $(x_p - x_i)$ and $(x_i - x_1)$ in function of Δ_x, $\epsilon_{\text{over},x^2}(x_i)$ is equal to:

$$\epsilon_{\text{over},x^2}(x_i) = \frac{\Delta_x^2}{(p-1)^2}\,(p-i)(i-1)\cdot \tag{4.8}$$

   Since the area A_λ of the domain of possible values is equal to the sum of the areas of $p-1$ trapeziums, we have:

$$A_\lambda = \sum_{i=1}^{p-1}\frac{1}{2}H(B_i + B_{i+1}) = \sum_{i=1}^{p-1}\frac{1}{2}\frac{\Delta_x}{p-1}\,(2\epsilon_{\lambda(x^2),U} + \epsilon_{\text{over},x^2}(x_i) + \epsilon_{\text{over},x^2}(x_{i+1})).$$

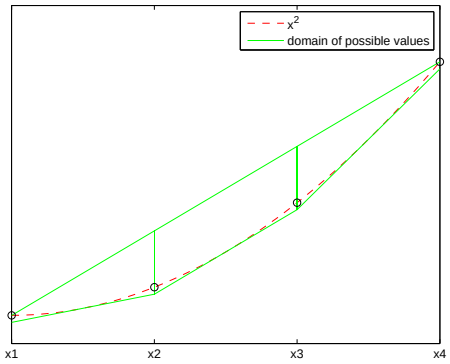


Figure 4.2: Decomposition of the domain of possible values for the outer approximation of x^2 based on SOS into $p-1$ trapeziums.

As $\epsilon_{\text{over},x^2}(x_i)$ is equal to zero at points x_1 and x_p, A_λ becomes:

$$A_\lambda = \frac{1}{2}\frac{\Delta_x}{p-1}\left(2\sum_{i=1}^{p-1}\epsilon_{\lambda(x^2),U} + 2\sum_{i=2}^{p-1}\epsilon_{\text{over},x^2}(x_i)\right).$$

By simplifying and replacing $\epsilon_{\lambda(x^2),U}$ and $\epsilon_{\text{over},x^2}(x_i)$ by their values given in (4.7) and (4.8) respectively, we have:

$$A_\lambda = \frac{\Delta_x}{p-1}\left((p-1)\frac{\Delta_x^2}{4(p-1)^2} + \frac{\Delta_x^2}{(p-1)^2}\sum_{i=2}^{p-1}(p-i)(i-1)\right).$$

By developing and applying the following properties:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \quad \text{and} \quad \sum_{i=1}^{n} i^2 = \frac{1}{6}\, n\,(n+1)(2n+1), \tag{4.9}$$

the desired result is obtained, since:

$$\begin{aligned}
A_\lambda &= \frac{\Delta_x^3}{(p-1)^3}\left(\frac{1}{4}(p-1) + \sum_{i=2}^{p-1}\Big(-p+(p+1)i-i^2\Big)\right)\\
&= \frac{\Delta_x^3}{(p-1)^3}\left(\frac{1}{4}(p-1) + (p-2)(-p) + (p+1)\left(\frac{(p-1)p}{2} - 1\right)\right.\\
&\quad \left. -\frac{1}{6}\,(p-1)\,p\,(2(p-1)+1) + 1\right)\\
&= \frac{\Delta_x^3}{(p-1)^3}\left(\frac{-4p^2+9p-1}{4} + \frac{p^3+3p^2-10p}{6}\right)\\
&= \frac{\Delta_x^3}{(p-1)^3}\,\frac{2p^3-6p^2+7p-3}{12}\\
&= \frac{\Delta_x^3}{12}\,\frac{2p^3-2p^2-4p^2+4p+3p-3}{(p-1)^3}\\
&= \frac{\Delta_x^3}{12}\,\frac{(p-1)(2p^2-4p+3)}{(p-1)^3},
\end{aligned}$$

and thus, finally:

$$A_\lambda = \frac{\Delta_x^3}{12}\frac{2p^2-4p+3}{(p-1)^2}.$$

□

2. *Outer approximation based on tangent lines*

Theorem 4.2 *The area of the domain of possible values for the outer approximation of x^2 based on tangent lines defined at p equally spaced breakpoints is also given by:*

$$A_{tang} = \frac{\Delta_x^3}{12}\frac{2p^2-4p+3}{(p-1)^2}.$$

Proof
As shown in Figure 4.3, the domain of the possible values for the outer approximation of x^2 based on tangent lines can be decomposed into 2 triangles and $p-2$ trapeziums. The

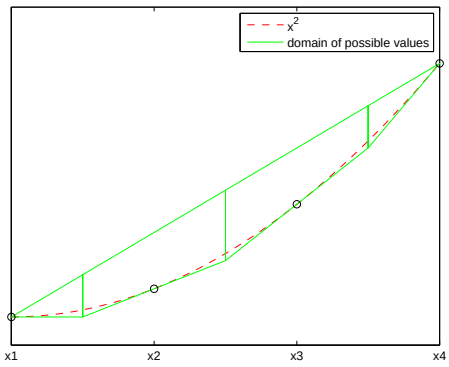


Figure 4.3: Decomposition of the domain of possible values for the outer approximation of x^2 based on tangent lines into 2 triangles and $p-2$ trapeziums.

"height" of the i^{th} trapezium, denoted H_{trapez_i}, corresponds to the distance between the points $\frac{x_i+x_{i+1}}{2}$ and $\frac{x_{i+1}+x_{i+2}}{2}$, or equivalently to the length of a piece since the breakpoints are equally spaced. Therefore,

$$H_{\text{trapez}_i} = \frac{\Delta_x}{p-1}.$$

The length of the bases, the parallel sides defined at points $\frac{x_i+x_{i+1}}{2}$, $1 \leq i \leq p-1$ and denoted B_{trapez_i}, is equal to the distance at $\frac{x_i+x_{i+1}}{2}$ between ϕ and the tangent T_i. By using (4.1) and (4.3), we have:

$$\begin{aligned}
B_{\text{trapez}_i} &= \phi(\tfrac{x_i+x_{i+1}}{2}) - T_i(\tfrac{x_i+x_{i+1}}{2}) \\
&= (x_1+x_p)\frac{x_{i+1}+x_i}{2} - x_1x_p - 2x_i\frac{x_{i+1}+x_i}{2} + x_i^2 \\
&= \frac{x_1x_i}{2} + \frac{x_ix_p}{2} + \frac{x_{i+1}x_p}{2} + \frac{x_1x_{i+1}}{2} - x_1x_p - x_ix_{i+1} \\
&= \frac{1}{2}\Big((x_p-x_i)(x_{i+1}-x_1) + (x_p-x_{i+1})(x_i-x_1)\Big).
\end{aligned}$$

Since the breakpoints x_i are equally spaced, B_{trapez_i} can be rewritten in term of Δ_x as:

$$\begin{aligned} B_{\text{trapez}_i} &= \frac{1}{2}\left(\frac{\Delta_x^2}{(p-1)^2}(p-i)i + \frac{\Delta_x^2}{(p-1)^2}(p-i-1)(i-1)\right) \\ &= \frac{1}{2}\frac{\Delta_x^2}{(p-1)^2}\Big((p-i)i + (p-i)i - (p-i) - (i-1)\Big) \\ &= \frac{1}{2}\frac{\Delta_x^2}{(p-1)^2}\Big(2(p-i)i - p + 1\Big). \end{aligned} \tag{4.10}$$

Accordingly, the area of the i^{th} trapezium is given by:

$$\begin{aligned} A_{\text{trapez}_i} &= \tfrac{1}{2}H_{\text{trapez}_i}(B_{\text{trapez}_i} + B_{\text{trapez}_{i+1}}) \\ &= \frac{1}{2}\frac{\Delta_x}{p-1}\left(\frac{\Delta_x^2}{2(p-1)^2}(2(p-i)i - p + 1 + 2(p-i-1)(i+1) - p + 1)\right), \\ &= \frac{1}{4}\frac{\Delta_x^3}{(p-1)^3}(2(p-i)i - 2p + 2 + 2(p-i)i - 2(i+1) + 2(p-i)), \\ &= \frac{1}{4}\frac{\Delta_x^3}{(p-1)^3}(4(p-i)i - 4i), \\ &= \frac{\Delta_x^3}{(p-1)^3}(-i^2 + i(p-1)). \end{aligned}$$

We now consider the area of the triangles. The height of the first triangle (the one defined by using (x_1, x_1^2)) is given by:

$$H_{\text{triangle}_1} = \frac{x_2 - x_1}{2}. \tag{4.11}$$

Since the breakpoints x_i are equally spaced, H_{triangle_1} can be rewritten as:

$$H_{\text{triangle}_1} = \frac{\Delta_x}{2\,(p-1)}.$$

The basis of the first triangle, B_{triangle_1}, corresponds to the basis at $\frac{x_1+x_2}{2}$ of the first trapezium. By evaluating (4.10) with $i = 1$, we find:

$$B_{\text{triangle}_1} = \frac{\Delta_x^2}{2(p-1)}.$$

Therefore, the area of the first triangle is equal to:

$$A_{\text{triangle}_1} = \frac{1}{2}H_{\text{triangle}_1}B_{\text{triangle}_1} = \frac{\Delta_x^3}{8\,(p-1)^2}.$$

It can be shown by a similar reasoning that the area of the second triangle is identical. By summing the areas of the two triangles and the $p-2$ trapeziums, we obtain the total area A_{tang} of the domain of possible values for the outer approximations of x^2 based on the

tangent lines defined at the p breakpoints:

$$\begin{aligned}
A_{tang} &= 2\,\frac{\Delta_x^3}{8\,(p-1)^2} + \sum_{i=1}^{p-2} \frac{\Delta_x^3}{(p-1)^3}(-i^2 + i(p-1)) \\
&= \frac{\Delta_x^3}{(p-1)^2}\left(\frac{1}{4} + \frac{1}{p-1}\sum_{i=1}^{p-2}\Big(-i^2 + i(p-1)\Big)\right) \\
&= \frac{\Delta_x^3}{(p-1)^2}\left(\frac{1}{4} + \frac{1}{p-1}\left(-\frac{1}{6}\,(p-2)(p-1)(2(p-2)+1)\right.\right. \\
&\qquad \left.\left. +(p-1)\frac{(p-2)(p-1)}{2}\right)\right) \\
&= \frac{\Delta_x^3}{(p-1)^2}\left(\frac{1}{4} + (p-2)\left(\frac{3-2p}{6} + \frac{p-1}{2}\right)\right) \\
&= \frac{\Delta_x^3}{12}\,\frac{2p^2-4p+3}{(p-1)^2},
\end{aligned}$$

where we have used (4.9).

□

The area of the domain of possible values (x, w_{x^2}) is thus identical for the outer approximations based on SOS or on tangent lines. As the maximum overestimation and underestimation approximation errors are also the same, the quality of the two outer approximation techniques can be considered as *equivalent*. The difference between these techniques appears in the number of constraints and variables needed to build the outer approximations, but also in the way to exploit the knowledge of the problem.

First, with respect to the number of variables, the outer approximation technique based on SOS is more expensive than the outer approximation technique based on tangent lines because it needs the introduction of p variables λ_i. On the other hand, with respect to the number of constraints, the less costly technique depends on the number of breakpoints used. If the tangent lines are based on p breakpoints, the associated outer approximation uses $p+1$ constraints (p to bound the domain below and one to bound it above). To the contrary, the outer approximation of x^2 based on SOS requires four constraints, whatever the number of breakpoints. Accordingly, the outer approximation based on SOS is more expensive with regard to the number of constraints if less than three breakpoints are used, while it is cheaper if more than three breakpoints are employed. Note also that this outer approximation technique can exploit the fact that a variable appears in different functions of the problem in order to reduce the domain of possible values for the outer approximation, as explained in Section 3.2.3.

Observe now that the bounds on the value of the approximation w_{x^2} can be tightened by applying the bound propagation developed in Section 3.2.2. Therefore, some values belonging to the domain of possible values for the outer approximation of x^2 may possibly be rejected by the updated bounds on w_{x^2}. When zero is not strictly inside the approximation interval for x, the bounds of the interval of the possible values for w_{x^2} correspond by bound propagation, to the squares of the lower and upper bounds on x. Accordingly, bound propagation can allow us to refine the outer approximation around a bound of the approximation interval for x. As by construction, these bounds are breakpoints, the bound propagation allows us to refine the outer

approximation around a breakpoint. For the outer approximation technique based on SOS, the underestimation approximation error is maximized at breakpoints while it is equal to zero at these points if the outer approximation technique is based on tangent lines (see Figure 4.1). Therefore, refining the domain around a breakpoint reduces more the domain for the outer approximation based on SOS, as illustrated in Figure 4.4. The left part of this figure corresponds to the outer approximation based on SOS while the right one represents the outer approximation based on tangent lines. As shown in the figure, only the outer approximation based on SOS can be tightened. In this case, the refinement of the domain of possible values is obtained thanks to the lower bound on the square function which is given by the square of the lower bond of the approximation interval. Note that we have deliberately omitted to treat the case where zero is strictly inside the approximation interval for x, in which case, the discussion is different. This is motivated by the fact that in the *TVC* problem, the arguments of square functions are always positive.

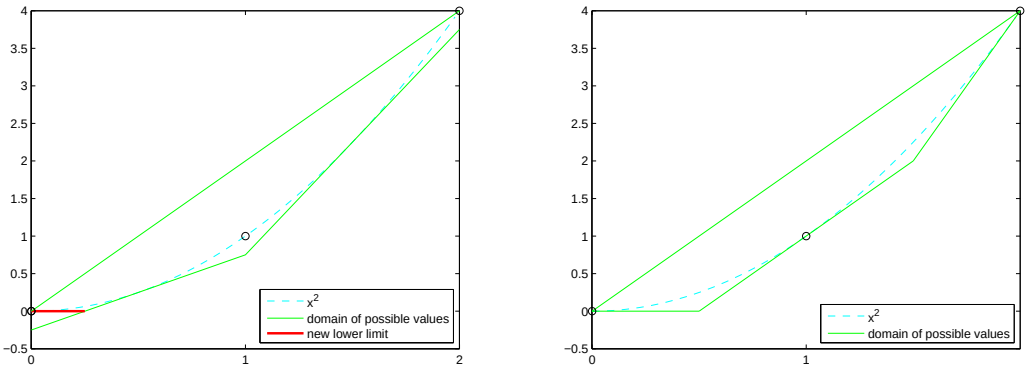


Figure 4.4: Domains of possible values for the outer approximation of x^2: on the left, based on SOS, and on the right, based on its tangent lines.

Since the outer approximations based on SOS can be more refined by bound propagation than the ones based on tangent lines and since the variables which are arguments of square functions generally also appear in other functions in the *TVC* problem, the use of outer approximations based on SOS can be justified to solve this problem.

4.1.2 Outer approximations of xy based on SOS versus on McCormick's inequalities

A popular way (see Adjiman *et al.* [9], Sahinidis and Tawarmalani [109]) to outer approximate a bilinear product xy on a rectangle $[l_x, u_x] \times [l_y, u_y]$ consists in using McCormick's inequalities developed in [81]. These inequalities bound xy below and above in this way:

$$(McCormick) \begin{cases} xy \leq l_x y + u_y x - l_x u_y, \\ xy \leq u_x y + l_y x - u_x l_y, \\ xy \geq l_x y + l_y x - l_x l_y, \\ xy \geq u_x y + u_y x - u_x u_y. \end{cases} \tag{4.12}$$

We will show in this section that the outer approximation generated by these inequalities has exactly the same domain of possible values as the outer approximation based on SOS defined by the conditions (2.27), (2.28), (2.29) and (3.39). To establish this result, a definition is firstly introduced.

Definition 4.1 *Let f: $X \to \mathbb{R}$ be a continuous function defined on a non empty convex set, X, of $\mathbb{R}^2$. The convex envelope of $f(x,y)$ over X is a function $C_{vex}(x,y)$ which satisfies:*

- *$C_{vex}(x,y)$ is convex on X,*
- *$\forall (x,y) \in X$, $C_{vex}(x,y) \leq f(x,y)$,*
- *If $h(x,y)$ is a convex function defined on X such that $\forall (x,y) \in X$, $h(x,y) \leq f(x,y)$, then $\forall (x,y) \in X$, $h(x,y) \leq C_{vex}(x,y)$.*

The convex envelope is thus the tightest convex underestimator of the function f on the set X. A similar definition can be given for the concave envelope, $C_{cave}(x,y)$, of a function f defined on a set X in such a way that the concave envelope is the tightest concave overestimator of f on X. In the case of a bilinear product xy defined on a rectangle, Al Khayyal and Falk have shown in [11] that the convex and concave envelopes are given by McCormick's inequalities (4.12), as stated by the following theorem.

Theorem 4.3 *The convex and concave envelopes of the biliner product xy defined on the rectangle $[l_x, u_x] \times [l_y, u_y]$ are given by:*

$$C_{vex}(x,y) = \max\{l_x y + l_y x - l_x l_y,\ u_x y + u_y x - u_x u_y\}, \tag{4.13}$$

$$C_{cave}(x,y) = \min\{l_x y + u_y x - l_x u_y,\ u_x y + l_y x - u_x l_y\}. \tag{4.14}$$

Let us now introduce the set C_{env} as:

$$C_{env} = \{(x,y,z) |\ C_{vex}(x,y) \leq z \leq C_{cave}(x,y),\ l_x \leq x \leq u_x, l_y \leq y \leq u_y\}. \tag{4.15}$$

Using Theorem 4.3, the set C_{env} for a bilinear product xy defined on a rectangle can be rewritten by using the definitions (4.13) and (4.14) of the convex and concave envelopes as:

$$\begin{array}{rl} C_{env} = \{(x,y,z)| & \max\{l_x y + l_y x - l_x l_y,\ u_x y + u_y x - u_x u_y\} \leq z, \\ & z \leq \min\{l_x y + u_y x - l_x u_y,\ u_x y + l_y x - u_x l_y\}, \\ & l_x \leq x \leq u_x, l_y \leq y \leq u_y\}. \end{array} \tag{4.16}$$

The third component of the points belonging to C_{env} is thus bounded as the bilinear product in McCormick's inequalities (4.12).

We now compare C_{env} and the set $DPVOA$ which contains all the possible values (x, y, w_{xy}) for the outer approximation of xy based on SOS. To this aim, note that since $C_{vex}(x,y)$ and $C_{cav}(x,y)$ are respectively the tightest convex underestimator and concave overestimator of a function $f(x,y)$ (here, xy) on $[l_x, u_x] \times [l_y, u_y]$, it is easy to show that C_{env} is the convex hull (see Section 1.1.3) of points $(x, y, f(x,y))$ $((x,y) \in [l_x, u_x] \times [l_y, u_y])$. We now recall in Theorem 4.4, a known result (see Hiriart-Urruty and Lemaréchal [59]).

Theorem 4.4 *The convex hull of points $(x, y, f(x, y))$ is the set of the convex combinations of these points.*

Therefore, according to what precedes and by particularizing the previous theorem to $f(x, y) = xy$, the following theorem can be derived.

Theorem 4.5 *C_{env} is the set of the convex combinations of (x, y, xy) with $l_x \leq x \leq u_x$ and $l_y \leq y \leq u_y$.*

Using this result, Theorem 4.6 can now be shown.

Theorem 4.6 *For a bilinear product defined on a rectangle $[l_x, u_x] \times [l_y, u_y]$, the set C_{env} of the convex combinations of points (x, y, xy) coincides with the set $DPVOA$ defined in (3.47).*

Proof
The equivalence is demonstrated by using two inclusions. First, we show that:

$$DPVOA \subseteq C_{env}.$$

Indeed, by definition, each point of $DPVOA$ is a convex combination of points $(x_i, y_j, x_i y_j)$ where (x_i, y_j) are breakpoints, since the approximation errors $\epsilon_{xy,L}$ and $\epsilon_{xy,U}$ used in (3.47) are equal to zero for the reasons detailed in Section 3.2.4. As these breakpoints belong to $[l_x, u_x] \times [l_y, u_y]$, any point of $DPVOA$ belongs to C_{env}, by Theorem 4.5.

Secondly, we prove the inverted inclusion:

$$C_{env} \subseteq DPVOA.$$

By Theorem 3.4, any point (x, y, xy) such that $l_x \leq x \leq u_x$ and $l_y \leq y \leq u_y$, can be rewritten as a convex combination of the four extreme points $(l_x, l_y, l_x l_y)$, $(l_x, u_y, l_x u_y)$, $(u_x, l_y, u_x l_y)$ and $(u_x, u_y, u_x u_y)$, where (l_x, l_y), (l_x, u_y), (u_x, l_y) and (u_x, u_y) are breakpoints. Accordingly, any point of C_{env}, a convex combination of points (x, y, xy) by Theorem 4.5, can be expressed as a convex combination of the four extreme points given above, and thus belongs to $DPVOA$.

□

By combining the previous results, one can derive Theorem 4.7.

Theorem 4.7 *The possible values (x, y, w_{xy}) for the outer approximation of the bilinear product xy on $[l_x, u_x] \times [l_y, u_y]$ are the same for an outer approximation of xy satisfying McCormick's inequalities and for an outer approximation of xy based on SOS.*

Proof
By Theorem 4.6, the domain of possible values for the outer approximation of xy on a rectangle based on SOS coincides with the set C_{env}. This set being defined by (4.16), it is obvious by using (4.12) that the desired result is valid.

□

As the quality of the two outer approximation techniques is the same with respect to the size of the domain of possible values, a natural question is to ask if there exist some reasons to

employ the formulation based on SOS while it introduces more variables in the problem than McCormick's inequalities. Again, the use of outer approximations based on SOS can produce tighter outer approximations if one argument of the bilinear product appears in another function of the problem. Suppose that a variable x intervenes twice in the problem in x^2 and xy. By applying the modelling of Section 3.2.5 which uses (3.23) and (3.39), the outer approximation based on SOS with three equally spaced breakpoints in each dimension is given for these functions by:

$$\begin{cases} w_{x^2} \leq \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{ij}\ {x_i}^2, \\ w_{x^2} \geq \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{ij}\ {x_i}^2 - \epsilon_{x^2,L}, \\ w_{xy} = \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{ij}\ x_i y_j, \\ x = \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{ij}\ x_i, \\ y = \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{ij}\ y_j, \\ \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{ij} = 1, \quad 0 \leq \lambda_{ij},\ 1 \leq i,j \leq 3. \end{cases}$$

Assume also that the approximation of x^2 and the constraints of the problem imply that $x = x_2$ with $\lambda_{11} = \lambda_{12} = \lambda_{13} = \lambda_{31} = \lambda_{32} = \lambda_{33} = 0$. Hence,

$$\begin{aligned} w_{xy} &= \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{ij}\ x_i y_j \\ &= x_2 \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{ij}\ y_j \\ &= xy. \end{aligned}$$

In this case, the outer approximation of xy is exact. We now examine the outer approximation based on McCormick's inequalities. As $x = x_2 = \frac{l_x+u_x}{2}$ since the breakpoints are equally spaced and by using Δ_x to denote the quantity $u_x - l_x$, it is easy to show that McCormick's inequalities given in (4.12) can be expressed as:

$$\begin{aligned} xy &\leq l_x y + \tfrac{u_y}{2}\Delta_x, \\ xy &\leq u_x y - \tfrac{l_y}{2}\Delta_x, \\ xy &\geq l_x y + \tfrac{l_y}{2}\Delta_x, \\ xy &\geq u_x y - \tfrac{u_y}{2}\Delta_x. \end{aligned} \tag{4.17}$$

For example, if $x \in [-1, 2]$ and $y \in [-3, 0]$, the value of all parameters appearing in the right terms of inequalities (4.17) can be replaced and these inequalities become:

$$\begin{aligned} xy &\leq -y, \\ xy &\leq 2y + \tfrac{9}{2}, \\ xy &\geq -y - \tfrac{9}{2}, \\ xy &\geq 2y. \end{aligned}$$

Moreover, assume that $y = -\frac{3}{2}$. Accordingly, McCormick's inequalities require that the product xy is bounded by:

$$-3 \leq xy \leq \frac{3}{2}.$$

As a consequence, McCormick's inequalities are weaker in this case than the outer approximation based on SOS which produces an exact approximation. An optimization problem illustrating this observation is given in Appendix A. When arguments of the bilinear product appear in other functions of the problem, the outer approximation based on SOS may bring tighter outer

approximations than McCormick's inequalities (the inverse being not true). This is due to the fact that the same λ is used to approximate different functions (see Section 3.2.3) with the outer approximation based on SOS. Therefore, the outer approximations of the functions using the same λ are linked together. This imposes an implicit constraint on the problem, which allows us to reject some values of the domain of possible values as shown in the previous example. Since arguments of the bilinear product in the decomposition of the *TVC* problem can also appear in square functions, we think that the choice of the outer approximations based on SOS can be justified, even if it is more expensive with respect to the number of variables.

For the sake of easiness, we have handled each bilinear product in the same way, that is, by using its outer approximation based on SOS. However, in some cases, other formulations should be preferred. So, when the arguments of the bilinear product do not appear in unary functions, McCormick's inequalities should be used instead since they do not introduce variables λ in the problem. Finally, the specificities of the problem can also be exploited to reformulate it in a better way. For example, the *TVC* problem involves a function $a_i\nu_i^2$ where a_i is binary, which can be modelled, after decomposition, as a bilinear product xy where y is binary. For this particular product, an outer approximation is not needed since $w = xy$ can be rewritten as four linear inequalities by exploiting the fact that y is binary. Indeed, if $y = 0$ then $w = y = 0$, otherwise, $y = 1$ and then $w = x$. This can be expressed by using a big-M formulation:

$$\begin{array}{rcl} w & \geq & -My, \\ w & \leq & My, \\ w & \geq & z - M(1-y), \\ w & \leq & z + M(1-y), \end{array} \tag{4.18}$$

which is exactly equivalent to $w = xy$ when the binary condition on y is satisfied and for $M > 0$ sufficiently large.

4.1.3 Outer approximations of trigonometric functions based on SOS versus on convex trigonometric underestimators

The determination of outer approximations appropriate for trigonometric functions has not been treated much in the literature. To our knowledge, the only work on the matter is due to Caratzoulas and Floudas. In [24], they propose to underestimate a nonconvex trigonometric function by a *convex trigonometric* one. Since bounding above a function f amounts to bounding below its opposite (which can be written as a trigonometric function by means of trigonometric properties if f is a trigonometric function) and taking the opposite of the obtained underestimator, we limit the study of the outer approximations of trigonometric functions to their underestimations. Although the underestimators built by Caratzoulas and Floudas are convex but not necessarily linear, linear underestimations can also be obtained from them by taking the tangent lines of these *convex* trigonometric functions at some points of their domain. But here, we consider the convex trigonometric underestimators, which are given in [24] for $\sin(x)$ and $\cos(x)$ on $[0, 2\pi]$ by:

$$U_{\sin} = -15.72 \sin\left(\frac{1}{6}(x + 2\pi)\right) + 13.61,$$

and

$$U_{\cos} = -16.99 \sin\left(\frac{1}{6}(x + 2\pi)\right) + 15.72.$$

Figure 4.5 compares these underestimations with the ones based on SOS and shows that the latter are globally tighter.

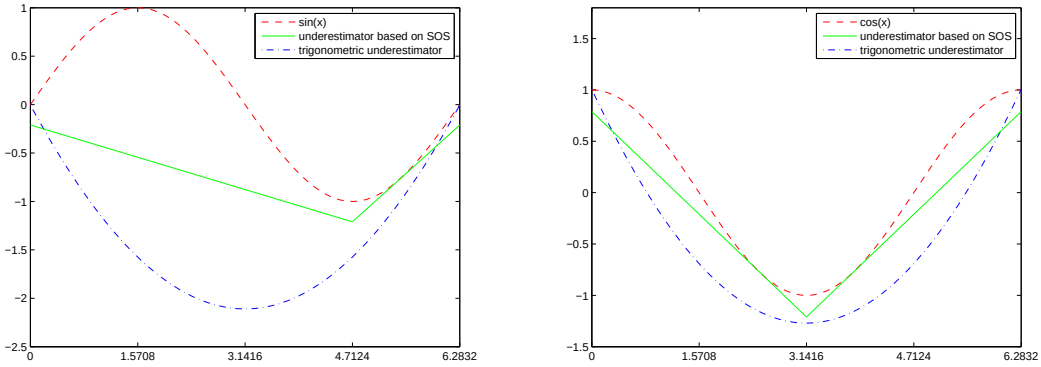


Figure 4.5: Underestimations of $\sin(x)$ and $\cos(x)$ on $[0, 2\pi]$ based on SOS or given by a convex trigonometric function.

Moreover, Caratzoulas and Floudas have shown that the maximum underestimation error grows linearly with the size of the domain. This is not the case with the underestimation based on SOS where the underestimation error is always smaller than the maximum underestimation error reached on $[0, 2\pi]$ augmented by one, since the periodicity of trigonometric functions is exploited, as detailed in Section 3.3.2. A graphic illustration of the underestimation of $\cos(x)$ on $[0, 10\pi]$ is presented in Figure 4.6. On this domain, the convex trigonometric underestimator established in [24] is given by:

$$U_{cos} = -47.91 \sin\left(\frac{2\pi}{188.48}(x + 31.41)\right) + 42.49.$$

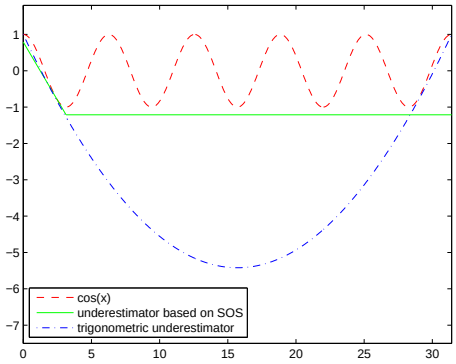


Figure 4.6: Underestimators of $\cos(x)$ on $[0, 10\pi]$ based on SOS or given by a convex trigonometric function.

Again, as highlighted by the figure, the underestimator based on SOS is globally tighter than the convex trigonometric underestimator. Note that, when the length of the approximation domain is smaller than π, the trigonometric function may be convex on this domain. In this case, the convex trigonometric underestimator is better than the underestimator based on SOS because it corresponds to the trigonometric function itself. Finally, when the trigonometric function is defined on a domain where it is concave, the underestimator used by Caratzoulas and Floudas is not a trigonometric function but the straight line joining the extreme points of the interval, exactly like the underestimator based on SOS.

Two underestimation techniques with very different goals have been compared. Caratzoulas and Floudas employ a twice continuously differentiable convex underestimator, not necessarily linear, while the underestimation used in the method proposed therein is linear. Both techniques have advantages and drawbacks: tighter on some domains, less accurate on other ones. The underestimator based on SOS needs additional variables contrary to the convex trigonometric underestimator but the latter produces nonlinear problems which must be solved by a nonlinear solver while the underestimator based on SOS generates linear problems. For these reasons, we think that the use of outer approximations based on SOS for trigonometric functions can be considered as a valid alternative with regard to the existing technique.

4.2 Branching on original variables or on variables λ_i

Since the previous section has highlighted that for the *TVC* problem, the outer approximations based on SOS are competitive with regard to the usually employed outer approximations, the use of the outer approximations based on SOS can be theoretically validated. The present section now focuses on the way to refine the outer approximations. For classical SOS approximations methods (not outer), the approximation problem is refined by branching on the variables λ_i (see Martin [80], Möller [83], Tomlin [110]) in order to satisfy the SOS conditions. However, the method developed in this thesis needs more than the satisfaction of the SOS conditions to guarantee the convergence to the global optimum of the problem, as seen in Section 3.1.1. Indeed, we must be able to refine the approximations as much as necessary. We cannot limit ourselves to the SOS approximation of the problem which is possibly not enough accurate if the chosen number of breakpoints is too small. In order to refine sufficiently the approximations, the proposed method uses a *constant number of breakpoints* to approximate a same nonlinear component on a given interval everywhere in the branch-and-bound tree. This introduction of new breakpoints when one goes down in the tree allows us to produce *tighter and tighter linear outer approximations* since the latter are built on smaller and smaller pieces, which implies that the approximation errors and as a consequence, the domain of possible values, decrease. This is the key for the convergence of the method.

We now explain how to update breakpoints when one goes down in the tree. After branching, the range of the branching variable is divided in two intervals. Therefore, the number of remaining breakpoints in each of these two intervals is smaller than before branching. To keep constant the number of breakpoints on each of these two intervals, new equally spaced breakpoints denoted x_i' or $\bar{x}_i$ are introduced in the subproblems, as shown in Figure 4.7, where five breakpoints are used. In this figure, the white dots correspond to the previous breakpoints which do no longer belong to the approximation interval for the branching variable after branching. Note that the updated breakpoints are chosen equally spaced for the reasons mentioned in Sec-

tion 3.3.1.

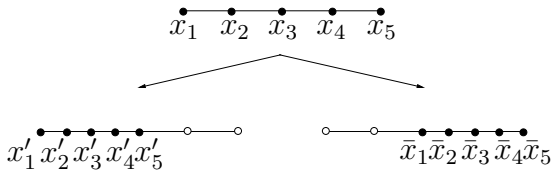


Figure 4.7: Update of breakpoints after branching.

By modifying the breakpoints when one goes down in the tree, the branching on variables λ_i is no longer valid, as illustrated by the branching represented in Figures 4.7 and 4.8. Indeed, before branching, a variable λ_i is associated to each breakpoint x_i. After branching, which is realized on λ_3 in this case, to each of the new breakpoints x'_i or $\bar{x}_i$ is again associated a variable λ_i. The same variables λ_i as before branching are employed to avoid to increase the number of variables in the problem. This situation is represented in Figure 4.8. However, these variables

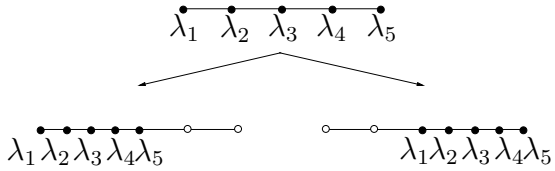


Figure 4.8: Update of variables λ_i after branching.

have another meaning after branching since they are associated to other breakpoints. Therefore, we no longer can use the branching constraints (2.21) and (2.22), which would have required in the present case that $\lambda_4 = \lambda_5 = 0$ and $\lambda_1 = \lambda_2 = 0$, respectively. But we have to employ $x \leq x_3$ and $x \geq x_3$ instead, where x_3 is a parameter equal to the value before branching of the third breakpoint, that is, the one on which we branch. Thus, *when the breakpoints are modified, branching on the variables λ_i is no longer relevant, while branching on the original variables allows us to avoid this problem.* Note that in the one-dimensional case, branching on $x = x_i$ by updating breakpoints or on λ_i without modifying breakpoints allows us to discard exactly the same part of the domain, as highlighted by comparing the approximation intervals for the subproblems of Figures 4.7 and 4.9, the latter figure representing a classical branching at λ_3. However, by updating breakpoints like in Figure 4.7, the outer approximations on the

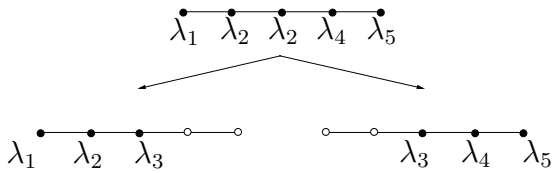


Figure 4.9: Branching on variables λ_i.

remaining subdomains can be refined since the breakpoints are less spaced and thus, the outer

approximations become tighter. This remark is also true in two dimensions as long as the branching is done vertically or horizontally, as explained in Section 4.2.2.

This section presents the reduction in the size of the domain of possible values by always keeping the number of breakpoints (used to approximate a same component on a given interval) constant and by branching on original variables instead of on the variables λ_i. This reduction is analytically computed for square and bilinear functions. We do not consider the reduction for trigonometric functions since, as seen in Section 3.2.4, the approximation errors for this kind of functions depend on the approximation interval and not only on the size of this domain. Therefore, the expression of the area of $DPVOA$ will vary with respect to the approximation interval and the place where we branch in this interval. As the goal of this section is to give an idea of the reduction in the size of the domain and not to develop heavy analytical expressions, we limit our analysis to square and bilinear functions.

4.2.1 Outer approximation of x^2

To correctly compare the two branching choices, we assume that the branching is done at the same place in both cases. For branching on variables λ_i, the branching is done on λ_j while for branching on original variables, the branching is realized on x_j, the associated breakpoint. Without loss of generality, we are going to study the area of $DPVOA$ defined in (3.47) for the right subproblems (see below) generated by branching. The expression of the area associated to the left subproblems can be derived in a similar way. For the branching on the variable λ_j, the following constraint must be added to the right subproblem:

$$\sum_{i=j}^{p} \lambda_i = 1, \quad \text{or equivalently } \lambda_i = 0, \ 1 \leq i \leq j-1,$$

and for the branching on the variable x, we require that:

$$x \geq x_j.$$

Theorem 4.8 establishes the difference and the ratio between the areas of $DPVOA$ for the outer approximation of x^2 after a branching on λ_j without modifying breakpoints, or obtained by branching on $x = x_j$ and keeping the same number of equally spaced breakpoints for the outer approximation before and after branching.

Theorem 4.8
Let Δ_x be the range of the variable x before branching and p be the number of equally spaced breakpoints. Then, the area of $DPVOA$ for the outer approximation of x^2 for the right subproblem (R) obtained by branching on the variable λ_j without updating the breakpoints is given by:

$$A_{\lambda_R} = \frac{\Delta_x^3}{12} \frac{(p-j)}{(p-1)^3} (1 + 2(p-j)^2),$$

while that obtained by branching on the variable x at the j^{th} breakpoint and by keeping a constant number of equally spaced breakpoints is given by:

$$A_{x_R} = \frac{\Delta_x^3}{12} \frac{(p-j)^3}{(p-1)^5} (2p^2 - 4p + 3).$$

The area associated to the branching on the variable λ_j is thus always larger and the difference between the two areas is equal to:

$$A_{\lambda_R} - A_{x_R} = \frac{\Delta_x^3}{12} \frac{(p-j)}{(p-1)^5} (j-1)(2p-j-1) > 0,$$

while their ratio is given by:

$$\frac{A_{\lambda_R}}{A_{x_R}} = \frac{(p-1)^2}{(p-j)^2} \frac{2p^2-4pj+2j^2+1}{2p^2-4p+3} > 1.$$

Proof
We begin by computing the area A_{λ_R} of $DPVOA$ for the outer approximation of x^2 associated to the branching on the variable λ_j. In the same way as in Theorem 4.1 which established the area of $DPVOA$ before branching, the area A_{λ_R} obtained after branching can be seen as the sum of the areas of $p-j$ trapeziums (since it remains $p-j$ pieces after branching). The "height", H, of a trapezium is equal to the length of a piece, i.e., $\frac{\Delta_x}{p-1}$. The length of the bases, the parallel sides at $x = x_i$, $j \leq i \leq p$, and denoted B_i, is equal to the sum of $\epsilon_{\lambda(x^2),U}$, the maximum underestimation error given in (4.7) generated by the outer approximation based on SOS and $\epsilon_{\text{over},x^2}(x_i)$, the maximum overestimation approximation error which can be produced at x_i for the right subproblem obtained after branching on λ_j. The error $\epsilon_{\text{over},x^2}(x_i)$ is equal to the difference at point x_i between the straight line joining (x_j, x_j^2) to (x_p, x_p^2) and the square function, that is,

$$\begin{aligned} \epsilon_{\text{over},x^2}(x_i) &= x_j^2 + (x_p + x_j)(x_i - x_j) - x_i^2 \\ &= (x_p - x_i)(x_i - x_j). \end{aligned}$$

By using the fact that the breakpoints are equally spaced to write $(x_p - x_i)$ and $(x_i - x_j)$ in function of Δ_x, $\epsilon_{\text{over},x^2}(x_i)$ is equal to:

$$\epsilon_{\text{over},x^2}(x_i) = \frac{\Delta_x^2}{(p-1)^2} (p-i)(i-j). \tag{4.19}$$

Therefore, the area associated to the branching on λ_j can be computed as follows:

$$A_{\lambda_R} = \sum_{i=j}^{p-1} \frac{1}{2} H(B_i + B_{i+1}) = \sum_{i=j}^{p-1} \frac{1}{2} \frac{\Delta_x}{p-1} (2\epsilon_{\lambda(x^2),U} + \epsilon_{\text{over},x^2}(x_i) + \epsilon_{\text{over},x^2}(x_{i+1})).$$

Since the overestimation approximation error $\epsilon_{\text{over},x^2}(x_i)$ is equal to zero at x_j and x_p, A_{λ_R} becomes:

$$A_{\lambda_R} = \frac{1}{2} \frac{\Delta_x}{p-1} \left(2 \sum_{i=j}^{p-1} \epsilon_{\lambda(x^2),U} + 2 \sum_{i=j+1}^{p-1} \epsilon_{\text{over},x^2}(x_i) \right).$$

Using (4.7) and (4.19), we then find:

$$A_{\lambda_R} = \frac{\Delta_x}{p-1} \left(\frac{\Delta_x^2}{4(p-1)^2}(p-j) + \frac{\Delta_x^2}{(p-1)^2} \sum_{i=j+1}^{p-1} (p-i)(i-j) \right).$$

Using (4.9), we get:

$$\begin{aligned}
A_{\lambda_R} &= \frac{\Delta_x^3}{(p-1)^3}\left(\frac{1}{4}(p-j)+\sum_{i=j+1}^{p-1}(pi-i^2+ij-pj)\right)\\
&= \frac{\Delta_x^3}{(p-1)^3}\left(\frac{1}{4}(p-j)+(p-j-1)(-pj)+(p+j)\left(\frac{p(p-1)}{2}-\frac{j(j+1)}{2}\right)\right.\\
&\quad \left.-\frac{p-1}{6}\,p\,(2(p-1)+1)+\frac{j}{6}(j+1)(2j+1)\right)\\
&= \frac{\Delta_x^3}{(p-1)^3}\left(\frac{1}{4}(p-j)+(p-j-1)(-pj)+\frac{(p+j)^2}{2}(p-j-1)\right.\\
&\quad \left.-\frac{p-1}{6}\,p\,(2(p-1)+1)+\frac{j}{6}(j+1)(2j+1)\right)\\
&= \frac{\Delta_x^3}{(p-1)^3}\left(\frac{1}{4}(p-j)+(p-j-1)\frac{(p^2+j^2)}{2}-\frac{p^2-p}{6}(2p-1)\right.\\
&\quad \left.+\frac{j^2+j}{6}(2j+1)\right)\\
&= \frac{\Delta_x^3}{(p-1)^3}\left(\frac{1}{4}(p-j)+\frac{p^3-p-3p^2j+3pj^2-j^3+j}{6}\right)\\
&= \Delta_x^3\,\frac{(p-j)}{(p-1)^3}\left(\frac{1}{4}+\frac{p^2+j^2-2pj-1}{6}\right)\\
&= \frac{\Delta_x^3}{12}\,\frac{(p-j)}{(p-1)^3}\,(1+2p^2+2j^2-4pj)\\
&= \frac{\Delta_x^3}{12}\,\frac{(p-j)}{(p-1)^3}\,(1+2(p-j)^2),
\end{aligned}$$

which is the desired result. The area associated to the branching on the variable x is now computed. Since no variable λ_i is compelled to be equal to zero, Theorem 4.1 can be applied with the range of the variable x obtained after branching which is given by:

$$\Delta_{x_R} = \Delta_x\,\frac{p-j}{p-1}.$$

Therefore, by Theorem 4.1,

$$\begin{aligned}
A_{x_R} &= \frac{\Delta_{x_R}^3}{12}\,\frac{2p^2-4p+3}{(p-1)^2}\\
&= \frac{\Delta_x^3}{12}\,\frac{(p-j)^3}{(p-1)^5}\,(2p^2-4p+3),
\end{aligned}$$

which is again the desired result. By removing now A_{x_R} from A_{λ_R}, we obtain that:

$$A_{\lambda_R}-A_{x_R} = \frac{\Delta_x^3}{12}\,\frac{(p-j)}{(p-1)^3}\left(1+2(p-j)^2-\frac{(p-j)^2}{(p-1)^2}(2p^2-4p+3)\right) \tag{4.20}$$

$$= \frac{\Delta_x^3}{12}\,\frac{(p-j)}{(p-1)^3}\,\alpha, \tag{4.21}$$

where we define α as:

$$\alpha = 1+2(p-j)^2-\frac{(p-j)^2}{(p-1)^2}(2p^2-4p+3).$$

As $\frac{\Delta_x^3}{12}\,\frac{(p-j)}{(p-1)^3}$ is always positive since $1 < j < p$, it remains to show that α is also positive. By employing some basic manipulations, the desired result can be obtained. Indeed, we have that:

$$\begin{aligned}
\alpha &= \frac{(p-1)^2(1+2(p-j)^2) - (p-j)^2(2p^2-4p+3)}{(p-1)^2} \\
&= \frac{(p-1)^2 + (p-j)^2\,(2(p-1)^2 - (2p^2-4p+3))}{(p-1)^2} \\
&= \frac{(p-1)^2 - (p-j)^2}{(p-1)^2} \\
&= \frac{(j-1)\,(2p-1-j)}{(p-1)^2} > 0,
\end{aligned}$$

because $1 < j < p$. Therefore, the difference between A_{λ_R} and A_{x_R} is always positive and is equal to:

$$A_{\lambda_R} - A_{x_R} = \frac{\Delta_x^3}{12}\,\frac{(p-j)}{(p-1)^5}\,(j-1)\,(2p-1-j).$$

Finally, the ratio $\frac{A_{\lambda_R}}{A_{x_R}}$ can be derived and is equal to:

$$\frac{A_{\lambda_R}}{A_{x_R}} = \frac{(p-1)^2}{(p-j)^2}\,\frac{2p^2-4pj+2j^2+1}{2p^2-4p+3}.$$

□

Note that taking $j = 1$ in the expression of A_{λ_R} amounts to the formula of the area of $DPVOA$ before branching established in Theorem 4.1. Table 4.1 gives some examples of values associated to the right subproblems obtained with different values for the number p of breakpoints and for the place indexed by j where the branching is performed. This table, in particular the ratio $\frac{A_{\lambda_R}}{A_{x_R}}$, confirms the interest of using the suggested branching technique instead of branching on the variables λ_i. (Similar results can also be derived for the left subproblems. In this case, each occurrence of $(p-j)$ in Theorem 4.8 must be replaced by $(j-1)$).

The question of the place of branching is also important. Theorem 4.9 shows that branching at the half of the range of x is the best way to reduce the sum on the areas of $DPVOA$ for the outer approximation of x^2 for the right and left subproblems.

Theorem 4.9 *The best choice to minimize the sum of the areas of the domain of possible values for the outer approximation of x^2 for the left and right subproblems is to branch at the half of the range of the variable x, if we suppose that the p breakpoints used to build the outer approximation are equally spaced.*

Proof

By Theorem 4.8, the area of $DPVOA$ for the right subproblem obtained by branching on x at x_j is equal to:

$$A_{x_R} = \frac{\Delta_x^3}{12}\,\frac{(p-j)^3}{(p-1)^5}\,(2p^2-4p+3).$$

p	j	A_{λ_R}	A_{x_R}	$A_{\lambda_R} - A_{x_R}$	A_{λ_R}/A_{x_R}
3	2	$\frac{1}{32}\Delta_x^3$	$\frac{3}{128}\Delta_x^3$	$\frac{1}{128}\Delta_x^3$	$\frac{4}{3}$
5	2	$\frac{19}{256}\Delta_x^3$	$\frac{297}{4096}\Delta_x^3$	$\frac{7}{4096}\Delta_x^3$	$\frac{304}{297}$
5	3	$\frac{3}{128}\Delta_x^3$	$\frac{11}{512}\Delta_x^3$	$\frac{1}{512}\Delta_x^3$	$\frac{12}{11}$
5	4	$\frac{1}{256}\Delta_x^3$	$\frac{11}{4096}\Delta_x^3$	$\frac{5}{4096}\Delta_x^3$	$\frac{16}{11}$

Table 4.1: Some examples of values associated to the right subproblem for different values for the number p of breakpoints and for the place indexed by j where the branching is performed.

By using a similar reasoning as in Theorem 4.8, it can be shown that the area of $DPVOA$ for the left subproblem is equal to:

$$A_{x_L} = \frac{\Delta_x^3}{12}\,\frac{(j-1)^3}{(p-1)^5}\,(2p^2-4p+3).$$

Accordingly, the sum of the areas of $DPVOA$ for the two subproblems obtained after branching on x_j is given by:

$$A(j) = \frac{\Delta_x^3}{12}\,\frac{2p^2-4p+3}{(p-1)^5}\,((p-j)^3+(j-1)^3)\cdot$$

Since the derivative of the sum of the areas with respect to j must be equal to zero for the value of j which minimizes this sum, we compute j for which:

$$A'(j) = \frac{\Delta_x^3}{12}\,\frac{2p^2-4p+3}{(p-1)^5}(-3(p-j)^2+3(j-1)^2) = 0,$$

that is:

$$(j-1)^2-(p-j)^2 = 0.$$

By factorizing, we find that j must satisfy:

$$(p-1)(2j-p-1) = 0.$$

Since it can be easily shown that the second derivative of $A(j)$ is positive, the value:

$$j^* = \frac{p+1}{2}$$

minimizes the sum of the areas. As the breakpoints are equally spaced between x_1 and x_p, x_{j^*} (which is not necessarily a breakpoint since j is not an integer if p is even) corresponds to the point at the half of the range of the variable x.

□

4.2.2 Outer approximation of xy

After having determined the reduction in $DPVOA$ obtained for the outer approximation of the square function by keeping constant the number of breakpoints and branching on the original variables instead of on the variables λ_i, we now analyze the case of a bilinear function. As explained in Section 2.2.3, there exist several ways to branch on the variables $\lambda_{i,j}$, contrary to the one-dimensional case: vertically, horizontally or diagonally. Therefore, different comparisons with these three branching techniques must be performed. The comparison basis will be the *volume* of $DPVOA$ for the right subproblems generated by both kinds of branching variables (on $\lambda_{i,j}$ or on original ones). We show in this section that the best way to reduce the sum of the volumes of $DPVOA$ for the left and right subproblems consists in branching on original variables at the half of their range, as for the square function.

Volume of $DPVOA$ before branching
We start by determining the volume of $DPVOA$ before branching for the bilinear product xy.

Theorem 4.10 *The volume of $DPVOA$ for the outer approximation of xy on $[l_x, u_x] \times [l_y, u_y]$ is given by:*

$$V_0 = \frac{1}{6}(u_x - l_x)^2(u_y - l_y)^2. \tag{4.22}$$

Proof
Theorem 4.6 has established that, for a bilinear product defined on a rectangle $[l_x, u_x] \times [l_y, u_y]$, the set $DPVOA$ is equivalent to the set C_{env} where the possible values for the approximation of xy are delimited by the convex and concave envelopes of xy on this rectangle. The volume of $DPVOA$ thus corresponds to the volume between these convex and concave envelopes defined by (4.13) and (4.14), respectively. To compute the volume between these envelopes, we need to precisely determine on each part of the domain which function maximizes (respectively minimizes) the right term of (4.13) (respectively (4.14)). To this aim, the approximation domain is divided in four parts like in Figure 4.10.

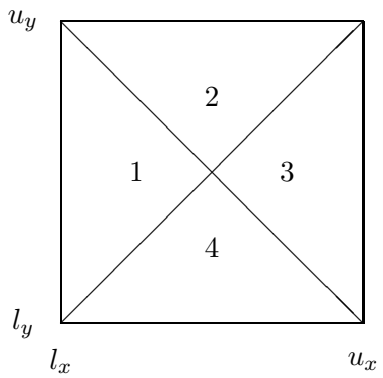


Figure 4.10: Decomposition of the approximation domain for xy following its convex and concave envelopes.

Let us consider the concave envelope of xy on parts 1 and 2. On these parts of the domain, y is bounded below by the straight line joining (l_x, l_y) and (u_x, u_y), and thus satisfies:

$$y \geq \frac{u_y - l_y}{u_x - l_x}(x - l_x) + l_y, \quad \forall x, y, \tag{4.23}$$

or equivalently, since $u_x - l_x > 0$:

$$(u_x - l_x)y + (l_y - u_y)x - u_x l_y + l_x u_y \geq 0, \quad \forall x, y. \tag{4.24}$$

Since this expression can be rewritten as:

$$u_x y + l_y x - u_x l_y \geq l_x y + u_y x - l_x u_y, \quad \forall x, y,$$

the concave envelope (4.14) on parts 1 and 2 is precisely given by:

$$l_x y + u_y x - l_x u_y.$$

By similar reasonings, it can be established that the convex and concave envelopes are defined on the four parts of Figure 4.10 as:

on part 1:
$C_{vex}(x, y) = l_x y + l_y x - l_x l_y,$
$C_{cave}(x, y) = l_x y + u_y x - l_x u_y,$

on part 2:
$C_{vex}(x, y) = u_x y + u_y x - u_x u_y,$
$C_{cave}(x, y) = l_x y + u_y x - l_x u_y,$

on part 3:
$C_{vex}(x, y) = u_x y + u_y x - u_x u_y,$
$C_{cave}(x, y) = u_x y + l_y x - u_x l_y,$

on part 4:
$C_{vex}(x, y) = l_x y + l_y x - l_x l_y,$
$C_{cave}(x, y) = u_x y + l_y x - u_x l_y.$

In order to compute the volume of $DPVOA$, each triangle of Figure 4.10 is now divided in two parts, as presented in Figure 4.11. We first compute the volume between the concave and convex envelopes on part 1_a. On this part, x can vary from l_x to $m_x = \frac{l_x + u_x}{2}$, while y is bounded above by $m_y = \frac{l_y + u_y}{2}$ and below by (4.23). Accordingly, the volume of $DPVOA$ can be computed on 1_a as in the following.

$$
\begin{aligned}
V_{1_a} &= \int_{l_x}^{\frac{l_x+u_x}{2}} \int_{\frac{u_y-l_y}{u_x-l_x}(x-l_x)+l_y}^{\frac{l_y+u_y}{2}} (C_{cave}(x,y) - C_{vex}(x,y))\ dydx \\
&= \int_{l_x}^{\frac{l_x+u_x}{2}} \int_{\frac{u_y-l_y}{u_x-l_x}(x-l_x)+l_y}^{\frac{l_y+u_y}{2}} (l_x y + u_y x - l_x u_y - l_x y - l_y x + l_x l_y)\ dydx \\
&= \int_{l_x}^{\frac{l_x+u_x}{2}} \int_{\frac{u_y-l_y}{u_x-l_x}(x-l_x)+l_y}^{\frac{l_y+u_y}{2}} (u_y - l_y)(x - l_x)\ dydx \\
&= (u_y - l_y) \int_{l_x}^{\frac{l_x+u_x}{2}} (x - l_x)[y]_{\frac{u_y-l_y}{u_x-l_x}(x-l_x)+l_y}^{\frac{l_y+u_y}{2}}\ dx, \\
&= (u_y - l_y) \int_{l_x}^{\frac{l_x+u_x}{2}} (x - l_x) \left(\frac{u_y - l_y}{2} - \frac{u_y - l_y}{u_x - l_x}(x - l_x) \right)\ dx \\
&= (u_y - l_y)^2 \int_{l_x}^{\frac{l_x+u_x}{2}} \left(\frac{1}{2}(x - l_x) - \frac{1}{u_x - l_x}(x - l_x)^2 \right) dx \\
&= (u_y - l_y)^2 \left[\frac{1}{2} \left(\frac{x^2}{2} - x l_x \right) - \frac{1}{u_x - l_x} \frac{(x - l_x)^3}{3} \right]_{l_x}^{\frac{l_x+u_x}{2}} \\
&= (u_y - l_y)^2 \left(\frac{1}{4} \left(\frac{l_x + u_x}{2} - l_x \right) \left(\frac{l_x + u_x}{2} + l_x \right) - \frac{1}{2}\, l_x \left(\frac{l_x + u_x}{2} - l_x \right) \right. \\
&\qquad \left. - \frac{1}{3} \frac{1}{u_x - l_x} \left(\frac{l_x + u_x}{2} - l_x \right)^3 \right) \\
&= (u_y - l_y)^2 \left(\frac{1}{4} \frac{u_x - l_x}{2}\ \frac{3l_x + u_x}{2} - \frac{1}{2}\, l_x \frac{u_x - l_x}{2} - \frac{1}{3} \frac{1}{u_x - l_x} \frac{(u_x - l_x)^3}{8} \right) \\
&= \frac{1}{4}\, (u_y - l_y)^2\, (u_x - l_x)\ \left(\frac{3l_x + u_x}{4} - \ l_x - \frac{1}{6}(u_x - l_x) \right) \\
&= \frac{1}{48}\, (u_y - l_y)^2\, (u_x - l_x)^2.
\end{aligned}
$$

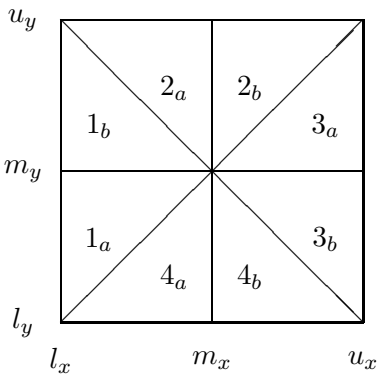


Figure 4.11: Decomposition of the approximation domain to compute the volume of $DPVOA$.

By using similar computations, it can be shown that the volume on each of the eight triangles of Figure 4.11 is equal to V_{1_a}. Therefore, as the volume of $DPVOA$ is equal to the sum of the

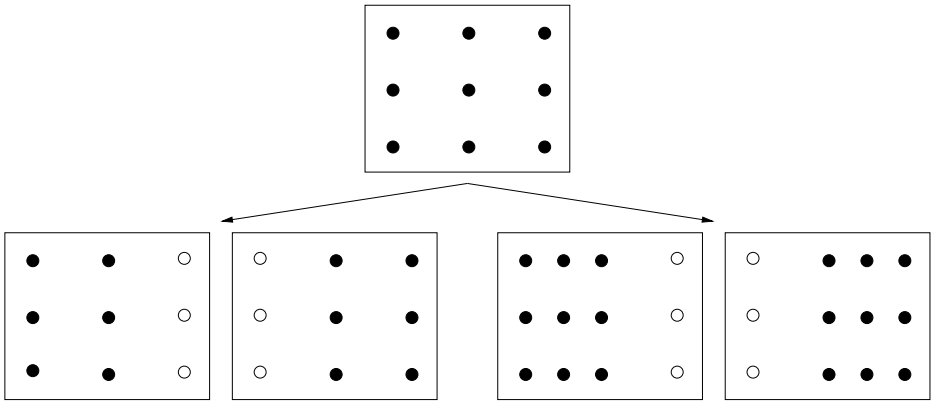
volumes on the eight triangles, we have that:

$$V_0 = \frac{1}{6}(u_y - l_y)^2(u_x - l_x)^2.$$

□

Vertical branching

Let us consider the modification in the volume obtained by branching. Suppose that p_x equally spaced breakpoints are used for x and p_y for y, which gives $p_x p_y$ breakpoints. Also assume that the branching is vertically performed at the same place, either on the variable $\lambda_{j,\cdot}$ or on the breakpoint x_j depending on the branching variable. The representation of breakpoints before and after branching (in case of three breakpoints in each dimension before branching) is given in Figure 4.12. The left part of the figure represents the breakpoints associated to the branching on the variables $\lambda_{i,j}$ given by equations (2.31) and (2.32) while the right one, the breakpoints employed for the branching on x. For the latter, new breakpoints are created. The previous breakpoints that are no longer used (since they do not belong to the updated approximation domain) are represented by white dots.



Branching on $\lambda_{i,j}$ Branching on x

Figure 4.12: Breakpoints for a vertical branching phase (three breakpoints in each dimension before branching).

For both branching techniques, the volume of $DPVOA$ after branching is the same. Indeed, by Theorem 4.10, the volume of $DPVOA$ for xy on a rectangle only depends on the bounds of this rectangle, which are in the present case identical for both kinds of branching. As by branching on $\lambda_{j,\cdot}$ or on x_j, the feasible range of x for the right subproblem is reduced to be equal to $\frac{p_x - j}{p_x - 1}(u_x - l_x)$, the volume of $DPVOA$ for the right subproblem is, by applying Theorem 4.10 with this new range, such that:

$$V_{vert} = \frac{1}{6}\frac{(p_x - j)^2}{(p_x - 1)^2}(u_x - l_x)^2(u_y - l_y)^2. \tag{4.25}$$

Horizontal branching

We now examine the case of an horizontal branching at $\lambda_{\cdot,k}$ or at breakpoint y_k, depending if the

branching is done on variables $\lambda_{i,j}$ (as defined in (2.33) and (2.34)) or on the original variables. The configuration of breakpoints before and after branching is given in Figure 4.13.

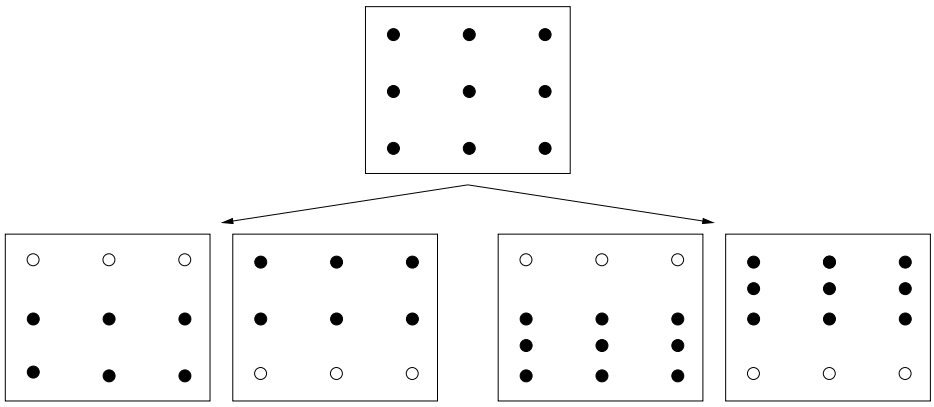


Figure 4.13: Breakpoints for an horizontal branching phase (three breakpoints in each dimension before branching).

A similar reasoning as for vertical branching can be held to obtain that the volume of $DPVOA$ is identical by branching on variables $\lambda_{.,k}$ or on y_k. Moreover, the volume of $DPVOA$ for the right subproblem obtained after having branched on $\lambda_{.,k}$ or on $y = y_k$ is, by applying Theorem 4.10, equal to:

$$V_{horiz} = \frac{1}{6}\frac{(p_y - k)^2}{(p_y - 1)^2}(u_x - l_x)^2(u_y - l_y)^2.$$

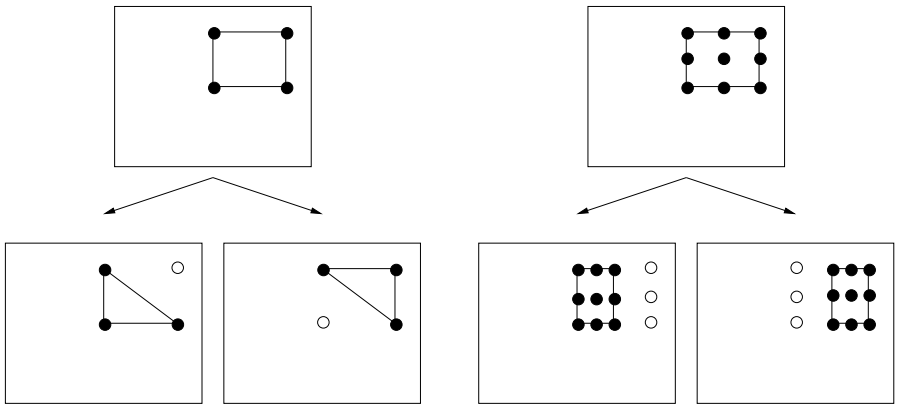
Diagonal branching

We finally consider the third way of branching on the variables $\lambda_{i,j}$, that is, the diagonal branching detailed in Section 2.2.3. We first focus on the branching on variables $\lambda_{i,j}$. Assume that after having branched several times, it is no longer possible to vertically or horizontally branch and that it remains only four breakpoints associated to a possibly nonzero $\lambda_{i,j}$. Therefore, the SOS condition (2.30) is not necessarily satisfied. In order to impose this condition, a diagonal branching given by (2.35) and (2.36) is performed. We recall that the diagonal branching consists in dividing the rectangle formed by the four breakpoints to which are associated a nonzero $\lambda_{i,j}$ in two triangles according to its diagonal. In this way, the SOS condition is necessarily fulfilled for the left and right generated subproblems and the bilinear product is thus replaced by a plan, its SOS approximation. As a consequence, the approximation is no longer outer. Indeed, Theorem 3.4 cannot be applied since only three breakpoints can be associated to nonzero $\lambda_{i,j}$ while four of them are necessary in order that the approximation is outer. Accordingly, in order to guarantee that any possible value (x, y, xy) is also feasible for the SOS approximation, an approximation error must be introduced to enlarge the domain of possible values.

To avoid this, when we branch on original variables, we prefer to pursue branching vertically or horizontally to always keep a rectangle as approximation domain, since in this case, there is *no need to use an approximation error* by Theorem 3.4. Furthermore, a diagonal branching

cannot be performed on original variables by simply modifying their bounds, as it is the case with the variables $\lambda_{i,j}$. Accordingly, a diagonal branching on original variables would imply to adopt a branch-and-cut technique instead of a branch-and-bound one, and thus to add new constraints when going down in the tree.

Since the branching on variables $\lambda_{i,j}$ or on original variables does not divide the approximation domain in the same way, the resulting approximation domains are different. Therefore, the sum of the volumes of $DPVOA$ for the right and left subproblems could be better by branching on $\lambda_{i,j}$ but we will see that it is not the case. Figure 4.14 represents the approximation domains after branching which are delimited by triangles in case of branching on variable $\lambda_{i,j}$ or by rectangles in case of branching on original variables. Here, the situation before branching is not the same depending if the branching is performed on variables $\lambda_{i,j}$ or on original variables. This is due to the fact that this situation results from previous branching phases. During the latter, the number of breakpoints associated to a nonzero $\lambda_{i,j}$ has been reduced by branching on $\lambda_{i,j}$ while it has not changed by branching on original variables because we have chosen to update breakpoints.



Branching on $\lambda_{i,j}$ Branching on x

Figure 4.14: Breakpoints for a diagonal branching phase if we branch on $\lambda_{i,j}$ and for a vertical branching phase if we branch on x (three breakpoints used in each dimension for the first outer approximation problem).

With a branching on variables $\lambda_{i,j}$, the approximation (which is an SOS approximation in this case) given by the convex combination $\sum_i \sum_j \lambda_{i,j} x_i y_j$ of (3.47) does not generate any volume because only the three breakpoints associated to the remaining nonzero $\lambda_{i,j}$ can be used and only one plan can be defined with three noncollinear points. But, as explained above, this approximation is no longer outer and an approximation error must be introduced in the formulation of the linear approximation. This error has been established in Theorem 3.2 for a particular kind of triangle. The expression of the error for other types of triangle is also given in Section 3.2.4. Therefore, the outer approximation on the triangle defined by the three breakpoints to which are associated a nonzero $\lambda_{i,j}$ must satisfy:

$$\sum_i \sum_j \lambda_{i,j} x_i y_j \leq w_{xy} \leq \sum_i \sum_j \lambda_{i,j} x_i y_j + \epsilon_{xy,U}, \tag{4.26}$$

or

$$\sum_i \sum_j \lambda_{i,j} x_i y_j - \epsilon_{xy,L} \leq w_{xy} \leq \sum_i \sum_j \lambda_{i,j} x_i y_j, \tag{4.27}$$

depending on the shape of the triangle formed by the breakpoints associated to the nonzero $\lambda_{i,j}$. We recall that the errors $\epsilon_{xy,L}$ and $\epsilon_{xy,U}$ produced by the SOS approximation of xy on a triangle corresponding to one half of the rectangle $[l_x, u_x] \times [l_y, u_y]$ are identical and are equal by Theorem 3.2 to:

$$\epsilon_{xy} = \frac{(u_x - l_x)(u_y - l_y)}{4}. \tag{4.28}$$

Theorem 4.11 establishes the expression of the volume of $DPVOA$ for the outer approximation of xy on a triangle given by constraints (4.26) or (4.27). In this theorem, l_x, u_x, l_y and u_y denote the bounds on the approximation domain before the last branching phase.

Theorem 4.11 *Let $Trig$ be a triangle corresponding to one half of the rectangle $[l_x, u_x] \times [l_y, u_y]$. The volume of the domain of possible values for the outer approximation of xy on triangle $Trig$ for constraints (4.26) or for constraints (4.27), depending on the shape of $Trig$, is given by:*

$$V_{diag} = \frac{1}{8}(u_x - l_x)^2 (u_y - l_y)^2.$$

Proof
The volume V_{diag} corresponds to the volume delimited by constraints (4.26) or (4.27) imposed on the triangle $Trig$, that is, the volume of an hyper-triangle having the triangle $Trig$ as basis and for which the height is equal, by subtraction of the part of the constraint ((4.26) or (4.27)) bounding w_{xy} below from the other part of the constraint bounding w_{xy} above, to:

$$\sum_i \sum_j \lambda_{i,j} x_i y_j + \epsilon_{xy} - \sum_i \sum_j \lambda_{i,j} x_i y_j,$$

since $\epsilon_{xy,L} = \epsilon_{xy,U} = \epsilon_{xy}$. By hypothesis, the basis of triangle $Trig$ is equal to $u_x - l_x$ while its height is equal to $u_y - l_y$. By applying the formula of the volume of an hyper-triangle, we thus find that:

$$V_{diag} = \frac{1}{2}(u_x - l_x)(u_y - l_y)\epsilon_{xy}.$$

Replacing ϵ_{xy} by its value given in (4.28), the desired result is finally obtained.

□

Coming back to the branching on original variables for which the number of breakpoints used remains constant, we decide, without loss of generality, to vertically branch on a breakpoint $\bar{x}_j$ which has been added after a previous branching phase. Formula (4.25) can thus be applied to determine the volume of $DPVOA$ for the outer approximation of xy. Therefore, the volume is equal to:

$$V_{vert} = \frac{1}{6}\frac{(p_x - j)^2}{(p_x - 1)^2}(u_x - l_x)^2 (u_y - l_y)^2.$$

Summary of the results
A summary of the obtained results is given in Table 4.2 for p_x equally spaced breakpoints for

x and p_y for y. For the sake of clarity, the quantities $u_x - l_x$ and $u_y - l_y$ giving here the length of the approximation interval before branching have been replaced by Δ_x and Δ_y, respectively. The diagonal branching is assumed to be performed if vertical or horizontal branching can no longer be applied. Note that if each occurrence $(p_x - j)$ is replaced by $(j-1)$, and $(p_y - k)$ by $(k-1)$, we obtain the results for the left subproblems.

	Branching on variables $\lambda_{i,j}$	Branching on original variables
Volume before branching	$\frac{1}{6}\Delta_x^2\Delta_y^2$	$\frac{1}{6}\Delta_x^2\Delta_y^2$
Volume after a vertical branching (on $\lambda_{j,\cdot}$ or on x_j)	$\frac{1}{6}\frac{(p_x-j)^2}{(p_x-1)^2}\Delta_x^2\Delta_y^2$	$\frac{1}{6}\frac{(p_x-j)^2}{(p_x-1)^2}\Delta_x^2\Delta_y^2$
Volume after an horizontal branching (on $\lambda_{\cdot,k}$ or on y_k)	$\frac{1}{6}\frac{(p_y-k)^2}{(p_y-1)^2}\Delta_x^2\Delta_y^2$	$\frac{1}{6}\frac{(p_y-k)^2}{(p_y-1)^2}\Delta_x^2\Delta_y^2$
Volume after a diagonal branching on $\lambda_{i,j}$ or after a vertical branching on x_j	$\frac{1}{8}\Delta_x^2\Delta_y^2$	$\frac{1}{6}\frac{(p_x-j)^2}{(p_x-1)^2}\Delta_x^2\Delta_y^2$

Table 4.2: Comparison of the volumes of $DPVOA$ for the outer approximation of xy for the right subproblems obtained by different types of branching.

As for the square function, it is easy to show, by employing a similar reasoning, that in every case, the best place to branch on in order to reduce the sum of the volumes for the left and right subproblems corresponds to the half of the interval of the original variables. Doing this, the formula in the last column and last row of Table 4.2 can be replaced by:

$$\frac{1}{24}\Delta_x^2\Delta_y^2, \tag{4.29}$$

since $(p_x - j)$ is, in this case, the half of $(p_x - 1)$. Note that by branching at the half of the range of the original variables, the volume of $DPVOA$ for the left and right subproblems is identical.

By comparing the results according to the branching variable, we see that the difference only appears when a diagonal branching is performed. By (4.29), we can see moreover that it is again preferable to branch on the original variables (at the half of their range) since in this way, the volume of $DPVOA$ is divided by three with respect to the branching on variables $\lambda_{i,j}$.

4.3 Conclusion

This chapter has presented a comparison of the outer approximations based on SOS and proposed in Section 3.2 with other outer approximations employed in the literature. It has been highlighted that the outer approximations based on SOS are more expensive in terms of the number of variables since they need variables λ_i. However, these additional variables allow us to produce tighter outer approximations by exploiting the multiple presence of a same variable in the problem. As the quality of the outer approximations based on SOS is competitive with the other examined outer approximation techniques, notably for the features of the *TVC* problem, we consider that the technique suggested in this thesis offers an interesting alternative to the existing methods.

After having underlined the interest of the outer approximation method based on SOS, our choice to branch on the original variables has been justified. To guarantee the convergence to the global optimum, the breakpoints are updated when one goes down in the tree in such a way that a same component is always approximated by using the same number of equally spaced breakpoints. In this way, the pieces become smaller and smaller when one goes down in the tree, and the outer approximations are, therefore, tighter and tighter. In this context, branching on variables λ_i is no longer convenient since the meaning of these variables varies from an iteration to another. Accordingly, branching is done on original variables. Finally, an analytical comparison of the volumes of $DPVOA$ given in (3.47) has been presented for the outer approximation of square and bilinear functions, by branching on variables λ_i without modifying the breakpoints and by branching on original variables with an update of the breakpoints. The obtained results allow us to quantify the improvement and show that the latter technique leads to tighter approximation domains.

Chapter 5

Presolve and range reduction

This chapter and the next two are devoted to choices to improve a branch-and-bound process: *presolve and range reduction*, *variable choice* and *node selection*. The goal of presolve and range reduction amounts to *reduce the range of variables*, and as a consequence, to refine the outer approximations. We refer to this bound tightening as *presolve* or *preprocessing* when it is performed at the top of the branch-and-bound tree and as *range reduction* when it is realized during the exploration of this tree. The variable selection is also important because the choice of branching variables determines in some way the depth of the tree where a node can possibly be cut. Since earlier the nodes are cut in the tree, better the results, the branching variable must be chosen in such a way that branching on it produces a sufficiently large modification for the two resulting subproblems compared to the parent problem. A branching rule is thus necessary to choose the *best* branching variable. At last, the order in which the branch-and-bound tree is explored must be treated carefully as it allows us to find more or less quickly good upper bounds to cut nodes. We show that a convenient way to treat these three features can improve the results a lot. In each of the three chapters, we propose and test several variants on a collection of twenty problems for the continuous case but also for the discrete one. These problems are presented in Section 5.1.2.

In a lot of methods, an appropriate presolve can improve the results. In this chapter, we show that it is also the case for our method. As the speed of convergence of the outer approximation method based on SOS strongly depends on the ranges of the variables, these ranges must be reduced as much as possible. We begin the discussion about presolve by analyzing the impact of a basic presolve in the sense explained below. Two kinds of presolve are suggested, tested and discussed: one is cheaper, the other is more expensive. An improved version of the preprocessing obtained by propagating the modified bounds into the whole approximation problem is then considered. The importance of the choice of the branching variable is also highlighted. Furthermore, a more elaborate preprocessing technique to decide when the presolve phase must be stopped is developed. Finally, all these ideas are applied not only at the beginning of the algorithm but also during its process.

5.1 Comparison basis

This section presents the *basic method* to which the possible improvements will be compared as well as the test problems used for this comparison. Some details about the software

developed to test these possible improvements are also given and the ways chosen to graphically represent the results are finally explained.

5.1.1 Basic method

In what follows, some alternatives are proposed to improve the method which is given by Algorithm 3.2 for the continuous case and by Algorithm 3.3 for the discrete one. To build the linear outer approximation problems used in these algorithms, we have decided to base the outer approximations on a number of breakpoints that we think appropriate to catch the nonlinear behaviour of the approximated functions: three breakpoints for the square functions and five for the trigonometric ones. For the bilinear products, four breakpoints (two in each dimension) are used. Indeed, by Theorem 3.4, only four breakpoints are needed to approximate a bilinear product. However, if one or both arguments of the bilinear product also appear in a square function, nine breakpoints (three in each dimension) are employed. In this way, the same set λ can be used to handle the variable appearing in the square function and in the bilinear product in order to refine the outer approximations of these functions, as highlighted in Appendix A.

The methods given by Algorithms 3.2 or 3.3 allow other degrees of freedom (node selection, branching variable choice, etc.). For the basic method, we have chosen to use *no presolve or range reduction*, to employ a *depth-first search with backtracking* and to use a branching rule based on the variable with the *largest range*, as detailed below. As no presolve or range reduction is used for the basic method, nothing particular is done to reduce the range of the variables, which is the simplest choice that we can take. In a branch-and-bound process, a node selection is needed to determine the order in which the nodes are treated. The depth-first search proposed by Dakin [28] and Little *et al.* [76] explores the tree in a depth-first manner. This means that the deepest nodes in the tree are first treated. Therefore, as long as we can go down in the tree, we do it. When a node is fathomed, the next node to be examined is the last one which has been created and which has not yet been explored. This is the principle of backtracking.

Once the node to refine has been determined, the branching variable must be chosen. The branching rule based on the largest range naturally requires to branch on the variable having the largest range among the variables which, at the current solution, are not at their nonlinear value for the outer approximations of all nonlinear components in which they appear, or which do not satisfy the discrete restrictions. Indeed, it would be useless to refine a variable which generates right values for the original problem and does not violate the discrete restrictions, since such a variable cannot improve the quality of the outer approximation problem by branching on it. Moreover, we decide to branch at a different place of the approximation interval depending if the variable is continuous or discrete. If the variable is discrete, we choose to branch at its *current value* in the solution in order to discard this current solution. Indeed, by branching on an integer variable x at its current noninteger value, denoted x', we tighten the bounds of the approximation interval by imposing $x \leq \lfloor x' \rfloor$ for the left subproblem and $x \geq \lceil x' \rceil$ for the right one. Therefore, x' is no longer feasible for both subproblems. For a continuous variable, the branching at the current solution does not always discard this solution *with regard to the original variables* because the outer approximation is possibly not sufficiently refined. As a consequence, if the variable is continuous, the branching is realized to improve the quality of the outer approximation. Since it has been shown in Chapter 4 that the *middle of the range* of the branching variable is the best place for branching to reduce the sum of the sizes of the domain

of possible values for the left and right subproblems for the square and bilinear functions, we branch at this place for the continuous variables.

By branching on a variable, two subproblems are generated. We decide to first solve the one having the *most feasible domain* with regard to the current solution. More precisely, if we branch on a continuous variable, the range of this one is partitioned in two parts. The value of the branching variable at the current solution before branching thus belongs to one of the domains of the two generated subproblems. The subproblem with this domain is thus first treated since it can be expected that this problem is more feasible than the other one and that the value of the objective function does not increase much compared to its current value. Indeed, the value of the original variables at the current solution remains valid with respect to the bound constraints of this subproblem. Now, if the branching is performed on an integer variable x, the current value of this variable, denoted x', is discarded, as explained above. Therefore, the same method as for continuous variables is no longer applicable. We then choose to firstly examine the subproblem having the closest domain to the current value of the branching variable. Thus, if x' is smaller than $\frac{\lfloor x' \rfloor + \lceil x' \rceil}{2}$, the left subproblem is treated before the right one, otherwise, it is the contrary.

In the next sections, several techniques related to presolve and range reduction, are proposed in order to accelerate the speed of convergence of this basic method. When a modification allows us to improve the method, it is validated and the basic method is updated to take this modification into account. Therefore, the *basic method evolves* with the experiments.

5.1.2 Test problems

To test the impact of our different ideas, a collection of twenty test problems has been built. Each of them is used in a continuous and a discrete version. The latter is simply obtained from the continuous one by imposing that some variables take discrete values (typically, binary values for the variables a_i and discrete ones for the ratio of voltage R_j of the *TVC* problem). In order to model the discrete restrictions different from binary restrictions, new integer variables are introduced in the problem, as detailed in Section 2.1.2. The number of variables in the discrete problems is thus equal to the number of variables in the associated continuous problems augmented by the number of discrete variables which are not binary.

All the problems have particularities of the *TVC* problem presented in Section 2.1.3. They comprise trigonometric, square, bilinear, trilinear functions and also products of a trigonometric function with a trilinear one. The first class of problems named *pb* can be seen as *toy problems*. They have in general less variables than the other ones, called *TVC*, and always less constraints. Problems *TVC* correspond to the *TVC* problem of Chapter 2 applied to different sets of data. The expression of each problem *pb* is given in Appendix B. In fact, we have only seven different problems of type *pb*. As for the *TVC* problems, we employ the structure of the problems on different data (two for each problem *pb*). The differences can appear in the bounds of the variables and constraints, in the coefficients of the problem as well as in the choice of the discrete variables. Table 5.1 gives some information about the problems: the number of variables and constraints in the nonlinear continuous problem, the number of variables and constraints in the linear outer approximation continuous problems $(\widetilde{OP})$, the number of sets λ used in these linear outer approximation problems based on SOS, the number of discrete variables in the discrete version of the problems, the optimum value of the problem without or with discrete restrictions, within an accuracy ϵ on the value of the objective function and finally, the optimum value of

the first linear outer approximation problem. For the numerical experiments, the accuracy ϵ used in Algorithms 3.2 and 3.3 has been set to 10^{-3}. Note that the problems are also more or less ordered according to their difficulty. Problems *pb0* to *pb5* are generally the easiest to solve while the *TVC* problems are the most difficult.

	#var	#cons	#var in OA	#cons in OA	#sets λ	#disc	f^*_{cont}	f^*_{disc}	1^{st} f_{LP}
pb0	4	2	44	33	6	1	-3.0070	-2.9144	-3.3348
pb1	4	2	44	33	6	1	-1.8875	-1.8186	-2.0000
pb2	6	2	41	32	5	1	0.0000	0.0000	-0.2500
pb3	6	2	41	32	5	1	0.2500	0.2500	-0.2321
pb4	12	4	97	75	11	2	0.0248	0.0342	-0.2578
pb5	12	4	97	75	11	2	11.6073	11.6528	6.7186
pb6	12	4	143	101	19	3	0.0081	0.0400	-0.4100
pb7	12	4	143	101	19	3	0.4337	0.4370	-1.2325
pb8	12	4	119	81	14	2	0.0366	0.0900	-0.3100
pb9	12	4	119	81	14	2	7.8094	8.2900	0.3452
pb10	10	4	111	74	13	2	0.0423	0.0900	-0.3100
pb11	10	4	111	74	13	2	7.8356	7.9485	1.4924
pb12	24	8	275	195	40	6	1.7612	2.0855	-2.1758
pb13	24	8	275	195	40	6	0.5163	0.5483	-2.4008
TVC1	16	9	269	210	39	6	5.6514	5.6606	0.0000
TVC2	18	9	275	214	40	6	2.3796	2.3840	1.3145
TVC3	27	15	431	331	61	9	5.3187	5.3301	0.6231
TVC4	27	15	431	331	61	9	1.0123	1.0295	0.3063
TVC5	37	21	602	472	87	13	1.1654	1.1865	0.0000
TVC6	38	21	635	496	92	14	0.0910	0.0993	0.0000

Table 5.1: Brief description of the test problems.

5.1.3 Some details about the implementation

The software developed to test the possible improvements for the proposed method has been written in *Fortran 95* [82]. As explained in Chapter 3, this method needs to solve linear and nonlinear problems. To handle the linear problems, the commercial solver *Cplex* [4] has been employed. The method used to solve the problems is based on a simplex algorithm (see Chapter 1). In order to solve the nonlinear problems, the nonlinear solver *filterSQP* developed by Fletcher and Leyffer [44] has been employed. This solver implements a sequential quadratic programming method using a filter to guarantee the convergence to a (local) optimum (see again Chapter 1 for more details about these notions).

Our software is not optimized with regard to the CPU time. For example, the update of the linear problems could be improved and the solution of a linear problem could be accelerated by exploiting the information obtained during the previous solutions of the linear problems. A natural extension of our work would be to improve the software in this way.

A maximum number of linear problems allowed to solve has been fixed to 500.000. If this number is exceeded on a problem, the method is considered not to converge in a reasonable time on the treated problem and the algorithm stops.

5.1.4 Graphic representation of the results

To analyze the impact of the different suggested variants on the basic method, two kinds of graphs are used: *performance profiles* and *graphs showing the improvement or degradation percentage* obtained for each problem with the new alternative versus the basic method. Note that in addition to these graphic representations, the full numerical results are given in Appendix C.

Performance profiles

The performance profiles have been introduced by Dolan and Moré [34] to display in a single graph some information about results obtained with two or more methods on a set of problems. In particular, the graph highlights the efficiency and the robustness of each method with regard to the other ones. The basis of comparison can be CPU time, number of function evaluations, number of subproblems solved, etc. To build the graph, *performance ratios* are computed for each problem p and for each method m:

$$r_{pm} = \frac{q_{pm}}{\min\{q_{pm'} | m' \in M\}},$$

where M is the set of compared methods and q_{pm} are the quantities on which we base the comparison. As we want to minimize the quantities q_{pm} (CPU time, function evaluations, etc.), the best technique on problem p is the one which minimizes the performance ratio r_{pm} among all techniques $m \in M$. For this particular technique, the ratio is equal to one. If the method m fails on problem p, r_{pm} is set to infinity. The definition of *performance profile* for the method m is given by:

$$p_m(\sigma) = \frac{\#\{p | r_{pm} \leq \sigma\}}{\#\{p\}}, \quad \sigma \geq 1.$$

The performance profile gives the proportion of problems for which the method m has a performance within a factor σ of the best observed performance.

The graph showing performance profiles has as many curves as compared techniques. The variable σ is represented in abscissa and the performance profiles in ordinate. The efficiency of a method can be seen on the left of the graph. Indeed, the value of $p_m(1)$ gives the percentage of problems for which the method m is the best. The robustness is given on the right of the graph: when σ tends to infinity, $p_m(\sigma)$ gives the percentage of problems that the technique m can solve. In summary, the higher the curve, the better the technique.

To compare the different techniques, we base the performance profiles of Chapters 5 to 7 on the number of linear problems solved and also on the CPU time for Chapter 8.

Improvement percentage

Performance profiles present the results obtained on average with different methods, without detailing them for each problem. Accordingly, each problem is supposed to have the same weight. But amongst our test problems, these called *TVC* have more importance than the other ones because they correspond more closely to the real problems that we want to solve and because they are also the largest ones. Therefore, it is interesting to distinguish the results obtained *on each problem*. As the number of linear problems solved can strongly vary from one problem to another, the *improvement percentages* are used instead of the numbers themselves. The improvement percentage obtained with the method m on the problem p with respect to the

reference method r is expressed by:

$$i_{pm} = \begin{cases} 100\left(\frac{n_{pr}}{n_{pm}} - 1\right) & \text{if } n_{pr} \geq n_{pm}, \\ -100\left(\frac{n_{pm}}{n_{pr}} - 1\right) & \text{if } n_{pr} \leq n_{pm}, \end{cases} \tag{5.1}$$

where n_{pm} is the number of linear problems solved by the method m to find the global solution of problem p. A positive value for i_{pm} means that the method m is better than the reference method r while a negative value means the inverse. Note also that a deterioration and an improvement of same magnitude produce the same absolute value for i_{pm}.

An example of a graph representing improvement percentages can be found on Figure 5.1. On this graph, the improvement percentages i_{pm} are represented in ordinate while the abscissa give the names of the problems. The numbers 0 to 13 are respectively associated to problems *pb0* to *pb13* while the notations $T1$ to $T6$ correspond to the six *TVC* problems. Moreover, this graph always comprises an horizontal line at $y = 0$ that corresponds to the reference technique. For each problem and for each method compared to the reference technique is associated a bar showing the magnitude of the improvement or the deterioration. For each problem, the higher the bar, the better the technique. If for a problem, all the bars are drawn below zero, the reference technique is the best.

5.2 Basic presolve

After having presented the basic method and the comparison basis, we now discuss the presolve itself. As we already mentioned, the smallest the domain of a nonlinear component, the best its outer approximation based on SOS. Therefore, it is desirable to strengthen the bounds on the variables. However, we only focus on the original variables and on the ones which replace a nonlinear component in the outer approximation formulation (the variables $w^i_{j_i}$ of problem $(\widetilde{OP})$ detailed in Section 3.2.5). Indeed, we are not interested in tightening the bounds on the variables λ_i because they are updated to 0 and 1 as soon as the associated breakpoints are modified, as explained in Section 4.2, that is, when the bounds on the original variables to which are associated these breakpoints are tightened. Accordingly, the strengthening of the bounds on variables λ_i is expected not to be durable.

As the tightening of bounds has a cost (see below), we do not try to reinforce bounds which could be relaxed later. That is, we strengthen the bounds on variables only if they remain valid in the whole branch-and-bound tree. Furthermore, we neither consider the bounds on continuous variables appearing only linearly in the problem, because the tightening of such bounds cannot improve the quality of the outer approximation problem. Indeed, in the linear outer approximation problem, a linear component is approximated by itself, and thus, does not produce an approximation error. To summarize, the candidate variables for bound strengthening are the *variables of the original problem which appear nonlinearly in the problem or which are discrete, but also the variables* $w^i_{j_i}$ introduced to approximate a nonlinear component.

For each variable x candidate for bound tightening, two linear problems are solved (see also Adjiman [8]):

$$(T_x) \begin{cases} \min / \max & x, \\ \text{s.t.} & \text{constraints of the linear outer approximation problem } (\widetilde{OP}). \end{cases}$$

These optimization problems thus have the same structure as the ones solved during the process of the algorithm except for the objective function. Therefore, the complexity of solving one of the problems (T_x) or one problem $(\widetilde{OP})$ can be considered as more or less equivalent. The optimum value of the minimization problem of (T_x) gives the smallest feasible value for x, that is, its lower bound, while the one of the maximization problem is equal to its upper bound.

This basic presolve has been applied to our collection of twenty problems presented in Section 5.1.2 at the top of the tree, before solving the first linear outer approximation problem. The following alternatives have been compared: *no presolve, presolve with one sweep* and *full presolve*. For presolve with one sweep, the minimization and maximization problems (T_x) are solved once for each candidate variable for bound strengthening. With full presolve, this operation is repeated (multiple sweeps), until for each variable, the bound strengthening can no longer produce an improvement in the range of the variable larger than the fixed accuracy, ϵ, required for the optimum value (here 10^{-3}). This improvement is obviously compared with respect to the range of the variable before the last sweep. Presolve with one sweep thus corresponds to a cheap presolve but it does not possibly exploit all the benefits of a presolve. To the contrary, the full presolve tries to strengthen the bounds on the variables as much as possible but the improvements generated by this kind of presolve can, in some cases, be small compared to its cost. Note that when it is detected that a bound on a variable can be tightened by solving a problem (T_x), this bound is updated in the bound constraints of the outer approximation problem $(\widetilde{OP})$ and, thus, also in the bound constraints of problems (T_x) for all variables x for which we must still try to improve the bounds. As a consequence, with regard to the situation without presolve, *only the bounds on some variables* are modified in the outer approximation problem $(\widetilde{OP})$ after using the kind of presolve explained in this section.

The two types of presolve (one sweep or full) have been tested on the twenty problems presented earlier. They have produced for the continuous case only two improvements in terms of the number of linear problems to solve and, for the discrete version, three improvements (one sweep presolve) and two improvements (full presolve). All the other results were less good than without presolve. Therefore, we consider that the presolve detailed above is not efficient since it generally deteriorates the results. However, the idea of this presolve is quite simple. The following section focuses on a more sophisticated presolve technique.

5.3 Presolve with propagation of the tightened bounds in the problem

The bound tightening is important because the outer approximations based on SOS are built by using the bounds on the variables. For the presolve of the previous section, the bounds on the variables have been strengthened but this bound tightening has not been exploited to refine the outer approximations. Indeed, these are always the outer approximations based on the bounds on the variables before preprocessing which are employed, even if they are now defined on smaller intervals. We could gain a lot by *updating the outer approximations of* $(\widetilde{OP})$ *with the bounds tightened by presolve*.

We are thus interested in the update of the linear outer approximation problem $(\widetilde{OP})$ due to the strengthening of bounds on some variables. In Section 3.3.1, we have explained how to update the problem after branching on a variable. For the subproblems, the impact of branching

can be assimilated to bound tightening. Therefore, the same update as for branching can be employed for bound tightening. This update summarized in Algorithm 3.1 is crucial to converge to the solution. Briefly, the bounds on the branching variable (here, the strengthened variable) are modified and the outer approximations of the nonlinear components based on this variable are updated. Finally, the bounds associated to the variables $w^i_{j_i}$ which replace these nonlinear components are also improved, if possible (see Algorithm 3.1 for more details). Note that we have decided to use this update not only after presolve but also *during the presolve* once a bound has been tightened in order to refine the outer approximation problem $(\widetilde{OP})$ as soon as possible. As a consequence, the constraints of problems (T_x) are also modified for all variables x which are still candidate for bound tightening, by hoping to produce tighter bounds on the variables x in this way. In summary, the constraints of (T_x) (general constraints and bound constraints) thus vary each time we can strengthen the bounds on a variable and before trying to tighten another bound.

When it is used in the case of a presolve allowing us to tighten the bounds on new variables $w^i_{j_i}$, the update of Algorithm 3.1 must be carefully handled with regard to the bound propagation technique of Section 3.2.2 which is employed to propagate bounds on some variables to bounds on other ones. Indeed, we must take care that the bound propagation does not produce a worse bound than the current one. For example, suppose that a bilinear product xy appears in a problem with $x \in [-2, 2]$ and $y \in [1, 4]$. By bound propagation, the lower bound on the outer approximation w_{xy} of xy is equal to -8 and its upper bound to 8. Imagine furthermore that the presolve has strengthened the lower bound on w_{xy} to -6. In this case, we do not update the lower bound by the value given by the bound propagation technique and keep a lower bound equal to -6 because -6 is tighter than -8. Accordingly, the bound propagation is used to update the bounds *only if it does not relax the current bounds*.

In case of a presolve allowing us to strengthen the bounds on new variables $w^i_{j_i}$, the bound propagation of Section 3.2.2 can be improved. Indeed, this section focuses on the bound propagation only from original variables into new variables replacing nonlinear components involving these original variables. With the considered presolve, the bounds on the new variables can be also tightened. As a consequence, the bound propagation can be done in the opposite direction: from new variables replacing nonlinear components into their arguments. Again, we detail this bound propagation according to the three nonlinear components appearing in the *TVC* problem.

1. $\boxed{x^2}$
   Suppose that the bounds on w_{x^2} have been updated to be equal to l_{x^2} and u_{x^2}, the bounds l_x and u_x on x can be modified like this:

$$
\begin{array}{l}
\text{if } (l_x \geq 0) \text{ then} \\
\quad l_x = \max(l_x,\ \sqrt{l_{x^2}}) \\
\quad u_x = \min(u_x,\ \sqrt{u_{x^2}}) \\
\text{else if } (u_x \leq 0) \text{ then} \\
\quad l_x = \max(l_x,\ -\sqrt{u_{x^2}}) \\
\quad u_x = \min(u_x,\ -\sqrt{l_{x^2}}) \\
\text{else} \\
\quad l_x = \max(l_x,\ -\sqrt{u_{x^2}}) \\
\quad u_x = \min(u_x,\ \sqrt{u_{x^2}})
\end{array}
$$

The conditions are used to determine which, among the negative or positive square roots, must be taken into account. The maximum taken for the lower bound and the minimum for the upper bound aim at preventing from relaxing the bounds, as explained above.

2. xy

If the bounds l_{xy} and u_{xy} on w_{xy} have been tightened, the following rule can be applied to strengthen the bound on $x \in [l_x, u_x]$ from l_{xy} and u_{xy} and from the bounds l_y and u_y on y, if the latter are different from zero:

$$\begin{array}{l}\text{if } \big((l_y > 0) \text{ or } (u_y < 0)\big) \text{ then} \\ \qquad l_x = \max\left(l_x, \min\left(\frac{l_{xy}}{l_y}, \frac{l_{xy}}{u_y}, \frac{u_{xy}}{l_y}, \frac{u_{xy}}{u_y}\right)\right) \\ \qquad u_x = \min\left(u_x, \max\left(\frac{l_{xy}}{l_y}, \frac{l_{xy}}{u_y}, \frac{u_{xy}}{l_y}, \frac{u_{xy}}{u_y}\right)\right)\end{array}$$

Indeed, the lower and upper bounds l_{xy} and u_{xy} on xy are necessarily produced by an extreme point (l_x, l_y), (l_x, u_y), (u_x, l_y) or (u_x, u_y). If zero does not belong to the interval $[l_y, u_y]$, for l_y, u_y, l_{xy} and u_{xy} fixed, the lower and upper bounds on x cannot take infinite values and thus, must belong to the set of the four values obtained by dividing the extreme values for xy by the extreme values for y. The maximum taken for the lower bound and the minimum for the upper bound prevent from generating worse bounds than the current ones. The condition on the bounds of y avoids to treat situations where zero belongs to $[l_y, u_y]$ which need a more sophisticated updating rule obtained by discussing the values of l_{xy}, u_{xy}, l_y and u_y. Since in the *TVC* problems, the arguments of the bilinear components are positive in most cases, we limit ourselves to the rule presented above. Note that by symmetry, the same reasoning can be used to strengthen the bounds on y from these on x if zero does not belong to $[l_x, u_x]$.

3. Trigonometric functions

For the trigonometric functions, we again assume without loss of generality that the approximation domain $[l_x, u_x]$ is included in $[0, 2\pi]$. The analysis starts with the $sine$ function. The bound propagation aims here at propagating the bounds $l_{\sin}$ and u_{sin} on the outer approximation $w_{\sin(x)}$ into the bounds l_x and u_x. We first check if $\sin(l_x)$ is in $[l_{\sin}, u_{\sin}]$. If it is the case, the lower bound on x cannot be tightened since the current lower bound produces a value for the $sine$ function belonging to the feasible interval for this component. When $\sin(l_x)$ is outside this interval, the values of x belonging to $[0, 2\pi]$ and producing the extreme points $l_{\sin}$ and $u_{\sin}$ are computed by means of the function $\arcsin$. As $\arcsin(x)$ always belongs to $[\frac{-\pi}{2}, \frac{\pi}{2}]$ and as x must be inside $[0, 2\pi]$, not only $\arcsin(l_{\sin})$ and $\arcsin(u_{\sin})$ (which belongs to $[-\frac{\pi}{2}, \frac{\pi}{2}]$) must be evaluated, but also $2\pi + \arcsin(l_{\sin})$ and $2\pi + \arcsin(u_{\sin})$ (which belongs to $[\frac{3\pi}{2}, \frac{5\pi}{2}]$) and, finally, $\pi - \arcsin(l_{\sin})$ and $\pi - \arcsin(u_{\sin})$ (which belongs to $[\frac{\pi}{2}, \frac{3\pi}{2}]$). Among these six values, the lower bound on x corresponds to the smallest one comprised between l_x and u_x. The same reasoning can be used to update the upper bound, if necessary. This bound propagation technique is schematically given below.

For the $cosine$ function, a similar reasoning can be held. As the image of the function $\arccos$ is defined on $[0, \pi]$, only four values instead of six can produce the extreme points: $\arccos(l_{\cos})$, $\arccos(u_{\cos})$, $2\pi - \arccos(l_{\cos})$ and $2\pi - \arccos(u_{\cos})$. The bound propagation technique associated to the $cosine$ function is given below.

Bound propagation from $l_{\sin}$ and $u_{\sin}$ to l_x and u_x
if $\big((l_{\sin} > \sin(l_x))$ or $(\sin(l_x) > u_{\sin})\big)$ then
$l = 10^6$
if $(l_x \leq \arcsin(l_{\sin}) \leq u_x)$ then $l = \arcsin(l_{\sin})$
if $(l_x \leq \arcsin(u_{\sin}) \leq u_x)$ then $l = \min(\arcsin(u_{\sin}), l)$
if $(l_x \leq 2\pi + \arcsin(l_{\sin}) \leq u_x)$ then $l = \min(2\pi + \arcsin(l_{\sin}), l)$
if $(l_x \leq 2\pi + \arcsin(u_{\sin}) \leq u_x)$ then $l = \min(2\pi + \arcsin(u_{\sin}), l)$
if $(l_x \leq \pi - \arcsin(l_{\sin}) \leq u_x)$ then $l = \min(\pi - \arcsin(l_{\sin}), l)$
if $(l_x \leq \pi - \arcsin(u_{\sin}) \leq u_x)$ then $l = \min(\pi - \arcsin(u_{\sin}), l)$
$l_x = \max(l, l_x)$
if $\big((l_{\sin} > \sin(u_x))$ or $(\sin(u_x) > u_{\sin})\big)$ then
$u = -10^6$
if $(l_x \leq \arcsin(l_{\sin}) \leq u_x)$ then $u = \arcsin(l_{\sin})$
if $(l_x \leq \arcsin(u_{\sin}) \leq u_x)$ then $u = \max(\arcsin(u_{\sin}), u)$
if $(l_x \leq 2\pi + \arcsin(l_{\sin}) \leq u_x)$ then $u = \max(2\pi + \arcsin(l_{\sin}), u)$
if $(l_x \leq 2\pi + \arcsin(u_{\sin}) \leq u_x)$ then $u = \max(2\pi + \arcsin(u_{\sin}), u)$
if $(l_x \leq \pi - \arcsin(l_{\sin}) \leq u_x)$ then $u = \max(\pi - \arcsin(l_{\sin}), u)$
if $(l_x \leq \pi - \arcsin(u_{\sin}) \leq u_x)$ then $u = \max(\pi - \arcsin(u_{\sin}), u)$
$u_x = \min(u, u_x)$

The strategy applied by the full presolve described in this section is summarized in Algorithm 5.1. The method related to a one sweep presolve is obtained from this algorithm if the loop "repeat" is only done once. The results obtained by using these two kinds of presolve which propagate the tightening of the bounds in the whole outer approximation problem are presented in Figure 5.1, which shows the improvement percentage i_{pm} defined by (5.1) and obtained by using a one sweep presolve or a full presolve compared to the method without presolve. Since presolve handles continuous and discrete variables in the same way, the results obtained in both cases are represented on the same graph. The left part of the graph shows the results for the continuous problems ordered like in Table 5.1 while the right part presents the result for the discrete version. For this part, the problems are placed in the reverse order in such a way that the *TVC* problems, continuous and then discrete, are in the center of the graph. Therefore, these are the results at the center of the graph on which we focus the most. For some problems, the method does not converge within the number of linear problems allowed to solve. A dot is used to represent such a situation. When the dot is placed at $y = 0$, the two compared methods fail to converge. When it is placed at $y > 0$, the method with presolve converges while the method without presolve does not. It is exactly the contrary when the dot is situated at $y < 0$. Sometimes, the method with presolve allows us to obtain more than 1000% improvement, or equivalently, the number of linear problems solved is divided by 11. In order to have a readable graph, such a situation is represented by a star. For example, since the number of

linear problems solved for the discrete version is divided more or less by thirteen for *pb5* and by twenty-two for *TVC3* with a full presolve, stars are used for these problems. The complete results for the three compared methods in term of number of linear and nonlinear problems solved can be found in Tables 8.5, 8.6 and 8.7 of Appendix C.

Bound propagation from $l_{\cos}$ and $u_{\cos}$ to l_x and u_x

if $\big((l_{\cos} > \cos(l_x))$ or $(\cos(l_x) > u_{\cos})\big)$ then
  $l = 10^6$
  if $(l_x \leq \arccos(l_{\cos}) \leq u_x)$ then $l = \arccos(l_{\cos})$
  if $(l_x \leq \arccos(u_{\cos}) \leq u_x)$ then $l = \min(\arccos(u_{\cos}), l)$
  if $(l_x \leq 2\pi - \arccos(l_{\cos}) \leq u_x)$ then $l = \min(2\pi - \arccos(l_{\cos}), l)$
  if $(l_x \leq 2\pi - \arccos(u_{\cos}) \leq u_x)$ then $l = \min(2\pi - \arccos(u_{\cos}), l)$
  $l_x = \max(l, l_x)$
if $\big((l_{\cos} > \cos(u_x))$ or $(\cos(u_x) > u_{\cos})\big)$ then
  $u = -10^6$
  if $(l_x \leq \arccos(l_{\cos}) \leq u_x)$ then $u = \arccos(l_{\cos})$
  if $(l_x \leq \arccos(u_{\cos}) \leq u_x)$ then $u = \max(\arccos(u_{\cos}), u)$
  if $(l_x \leq 2\pi - \arccos(l_{\cos}) \leq u_x)$ then $u = \max(2\pi - \arccos(l_{\cos}), u)$
  if $(l_x \leq 2\pi - \arccos(u_{\cos}) \leq u_x)$ then $u = \max(2\pi - \arccos(u_{\cos}), u)$
  $u_x = \min(u, u_x)$

Algorithm 5.1: Full presolve

Set $C = \{x_i : x_i$ appears nonlinearly in (P) or x_i is discrete$\} \cup \{w_{j_i}^i\}_{i=0,..,m,\ j=1,..,t_i}$.

REPEAT

For all $x \in C$:

1. Solve (T_x) subject to the current constraints of $(\widetilde{OP})$.
2. **If** (the two problems (T_x) are feasible) **then**
   Set $x^*_{\min}$ and $x^*_{\max}$, the optimum values of these problems
   else
   Stop: the problem (P) is infeasible
3. **If** $((x^*_{\min} > l_x)$ or $(x^*_{\max} < u_x))$ **then** Apply Algorithm 3.1 to the variable x.

UNTIL (the improvement in the range of each variable is smaller than 10^{-3})

In Figure 5.1, we observe that the three methods converge only for the problem *TVC3* among the *TVC* problems for the continuous version, as highlighted by dots. For the discrete version, the methods with presolve allow us to obtain better results on these problems. Indeed, they converge on five *TVC* problems instead of two without presolve. Furthermore, the improvement obtained for *TVC3* is significant: more than 2100% improvement with the full presolve (see

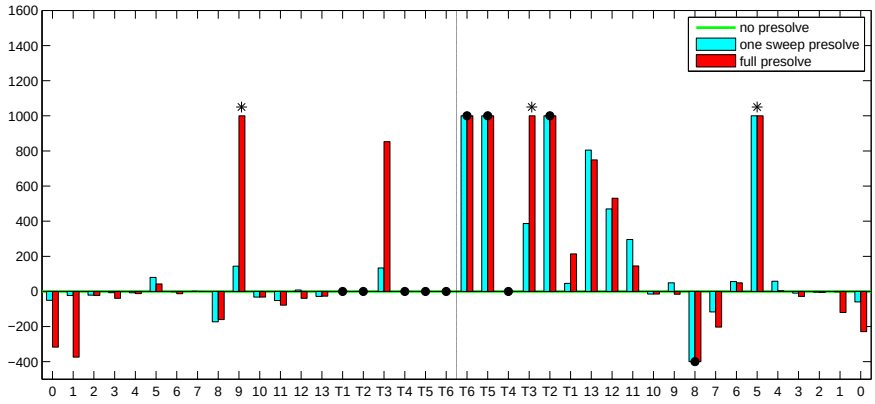


Figure 5.1: Comparison of the number of linear problems solved with a one sweep or a full presolve with regard to no presolve.

tables). In fact, the methods with presolve mainly improve results for problems at the center of the graph, that is, for the most difficult problems while they deteriorate results for the smallest problems (in the sense that a small number of linear problems is solved for such problems). This result could be expected because of the cost of the presolve which must be amortized in order for the presolve to be efficient. This is in general only possible for not too small problems. For the smallest problems, the number of problems solved during the presolve is too large with regard to the total number of problems solved during the process of the algorithm. Therefore, the presolve is too expensive with respect to the improvement that it generates. Observe also that the full presolve produces results which are in majority more extreme than the ones obtained with a one sweep presolve. When the results are improved with presolve, the obtained improvements are often larger with a full presolve than with a one sweep presolve, but when the results are deteriorated, the full presolve produces worse results than the one sweep presolve. Note finally that our experiments (that we do not detail here) have shown that if we only do a bound propagation from original variables into new ones and not in the opposite direction, the results are generally deteriorated for problems which are not too small. A complete bound propagation phase is thus useful.

5.4 Dependency of the results of presolve on the branching rule

The improvement and deteriorations obtained in the results are not only due to the solution of LPs during the presolve. Indeed, the choice of the branching variable also influences a lot the results, as shown in this section. Therefore, in order to highlight more the effect of preprocessing, we try to reduce the impact of the selection of the branching variable. In order to show the impact of the branching variable, we now focus on the reduction percentage obtained in the range of the original variables after presolve. This percentage is computed as:

$$100\left(1-\frac{\sum \text{ranges of the original variables after presolve}}{\sum \text{ranges of original variables before presolve}}\right). \tag{5.2}$$

The reduction percentages obtained for each problem by using a one sweep or a full presolve are respectively reported in Tables 8.6 and 8.7 while Figure 5.2 graphically presents them. This fig-

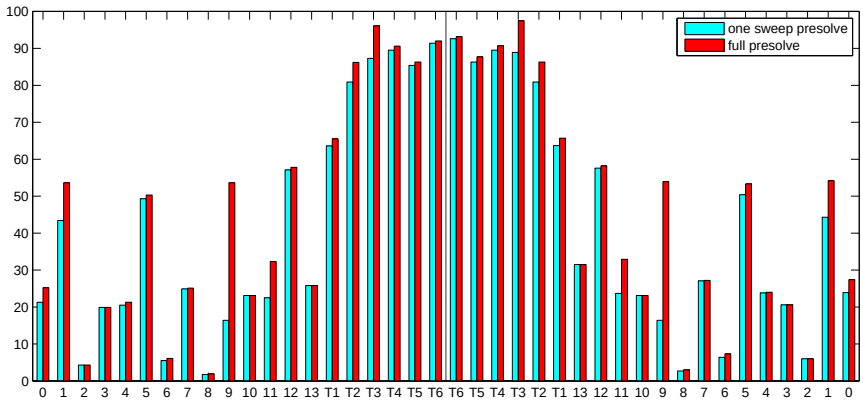


Figure 5.2: Reduction percentages in the range of the original variables.

ure shows that the range reduction generated by presolve can be quite substantial. For instance, for all the *TVC* problems, the presolve allows us to reduce the sum of the variable ranges by at least 60%, which can explain the good results of the methods with presolve on *TVC* problems in the discrete case. However, the reduction percentages cannot always justify the results obtained by using a presolve. So, for *pb8*, the full presolve reduces the sum of the variable ranges by 1.9% only, in the continuous case, but the number of linear problems to solve increases by 161%. The method with full presolve needs to solve 85229 LPs (366 during presolve) while only 32673 LPs are solved by the method without presolve. This result appears to be quite surprising because the presolve does not modify the problem much.

In fact, the deterioration in the results can be explained because the presolve *changes the order* in which we branch on the variables. For *pb8*, the majority of the variables have the same range before presolve. Without preprocessing, among the variables having the same largest range, we begin by branching on the first one, then on the second one, etc. By using a presolve, the range of three variables (on twelve) is modified for *pb8* and in particular, the range of the second considered variable is reduced by 3%. The problem has not changed much, but instead of branching on the second variable early in the tree because of our branching rule based on the largest range, we branch on this variable after having examined nearly all the other ones. However, this variable is precisely a variable which can eliminate important parts of the domain by branching on it. For comparison purposes, if we modify the order of the variables which have the same range before presolve by treating the second variable in last, the results without presolve are very different: 96809 LPs to solve instead of 32763. The results are thus influenced a lot by the order of branching on the variables. This shows the necessity of having a sophisticated technique to choose the best branching variable.

In order to compare the methods with or without presolve on a more consistent basis, we have tried to reduce the impact of the order in which we branch on the variables. Instead of considering the variables from 1 to n (which is random) and branching on the first one with the largest range, we *sort them in the order of their range after a full presolve* and examine them in this order. Nevertheless, we always branch on the variable which has the largest range. In

fact, we have only removed a part of random in the data. Note that the impact of this trick will be more important on *pb* problems because for these problems, the ranges of the variables before presolve are often identical, contrary to the *TVC* problems. For example, the number of linear problems to solve without presolve for *pb8* is equal to 111721 in the continous case by using such a trick. This value is quite far from the one obtained without this specific way for considering the variables (32763 LPs).

We have then considered the results produced by the method without presolve by using the particular ranking mentioned above and have compared them with the two methods with presolve. The new results for the method without presolve are given in Table 8.8. In the following, these results will be always used for the method without presolve instead of these of Table 8.5 since they have been obtained by trying to limit the impact of the choice of the branching variable in order to highlight the effect of presolve. We can also note that the results of these two tables can strongly differ ($pb8$ in the continuous case for example). This shows that the choice of the branching variable is very important.

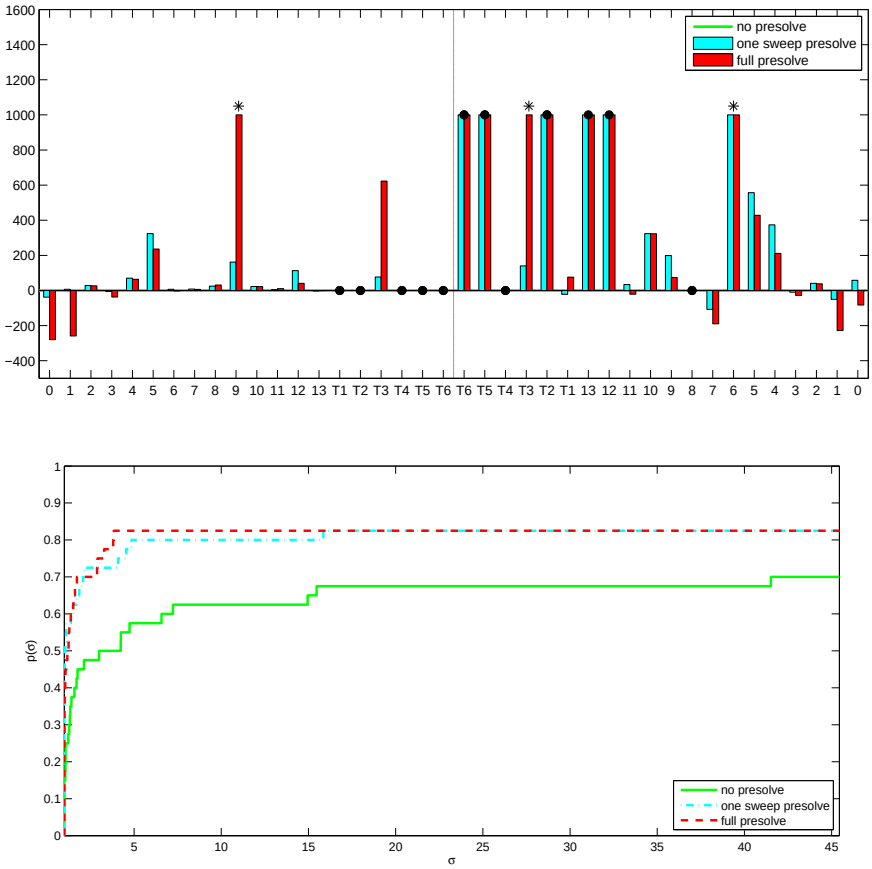


Figure 5.3: Comparison of the number of linear problems solved with a one sweep or a full presolve with regard to no presolve but by using a specific order for considering the variables.

A graphic comparison of the different proposed methods is given in Figure 5.3. The representation of the improvement percentage for the one sweep and for the full presolves is also

given separately by the two first pictures of Figure 5.4. The top picture of Figure 5.3 represents the improvement percentage in the number of LPs for a one sweep or a full presolve, compared to the situation without presolve while the bottom figure shows the associated performance profiles. It is clear by observing these figures that the methods with presolve are better than the one without presolve since they are more efficient and more robust. Note that on Figure 5.1, the benefit of using a presolve was not so obvious since it deteriorated the results for a larger number of problems, due to the modification of the order to branch on the variables.

To conclude this section, the use of a presolve can be validated since the number of linear problems solved to find the global solution is generally better with presolve and since the methods with presolve are able to solve a larger number of problems, as shown in Figure 5.3. Note finally that the best method among both with presolve depends on the problem treated. On the smallest problems (in the sense that a small number of problems must be solved), a one sweep presolve is better than a full presolve. But on larger problems, the full presolve is generally better. However, the difficulty of solving the problem is not a priori known and as a consequence, the best presolve technique cannot be determined. For the tested problems, it seems nevertheless to be correlated with the number of variables but it is not always the case.

5.5 Adaptable presolve

Since the best technique among the one sweep or the full presolve is problem dependent, we have developed a new presolve able to combine the advantages of both presolve and to give good results on all problems. As seen earlier, the efficiency of a presolve on a problem mainly depends on the complexity of the problem and on the total cost of the method. However, this information is not a priori known. Therefore, we have tried to find a trade-off between both presolve techniques by tightening the variables as long as a one sweep presolve produces a sufficient reduction in the *sum* of the ranges of the original variables. We refer to this kind of presolve as an *adaptable presolve*. In practice, this latter strengthens the bounds on original variables until the total reduction percentage (5.2) for the sum of the ranges of these variables obtained in a sweep is smaller than 20% [(1)]. If this percentage is larger than 20%, we pursue preprocessing by trying to tighten the bounds on the new variables $w^i_{j_i}$, otherwise, we stop the presolve phase. This is motivated by our numerical experiments which have shown that the improvement due to the tightening of new variables is a lot weaker than the one obtained with original variables. It can however improve results on large problems. This is why bounds on new variables are reinforced only if the tightening of the bounds on original variables has allowed us to sufficiently decrease their range. After having tried to tighten bounds on the new variables, the presolve phase is repeated on original variables and so on. Schematically, Algorithm 5.2 is applied.

The obtained results are graphically presented in Figure 5.4 (see also Table 8.9). On the three top figures, the improvement percentages (5.1) have been detailed for each kind of presolve (one sweep, full or adaptable) with regard to the method without presolve. The last figure compiles all the obtained results to have a more global sight on them. It can be observed that the adaptable presolve consists of a trade-off between the results obtained with a one sweep and a full presolve

[(1)] The best results have been obtained by using this value.

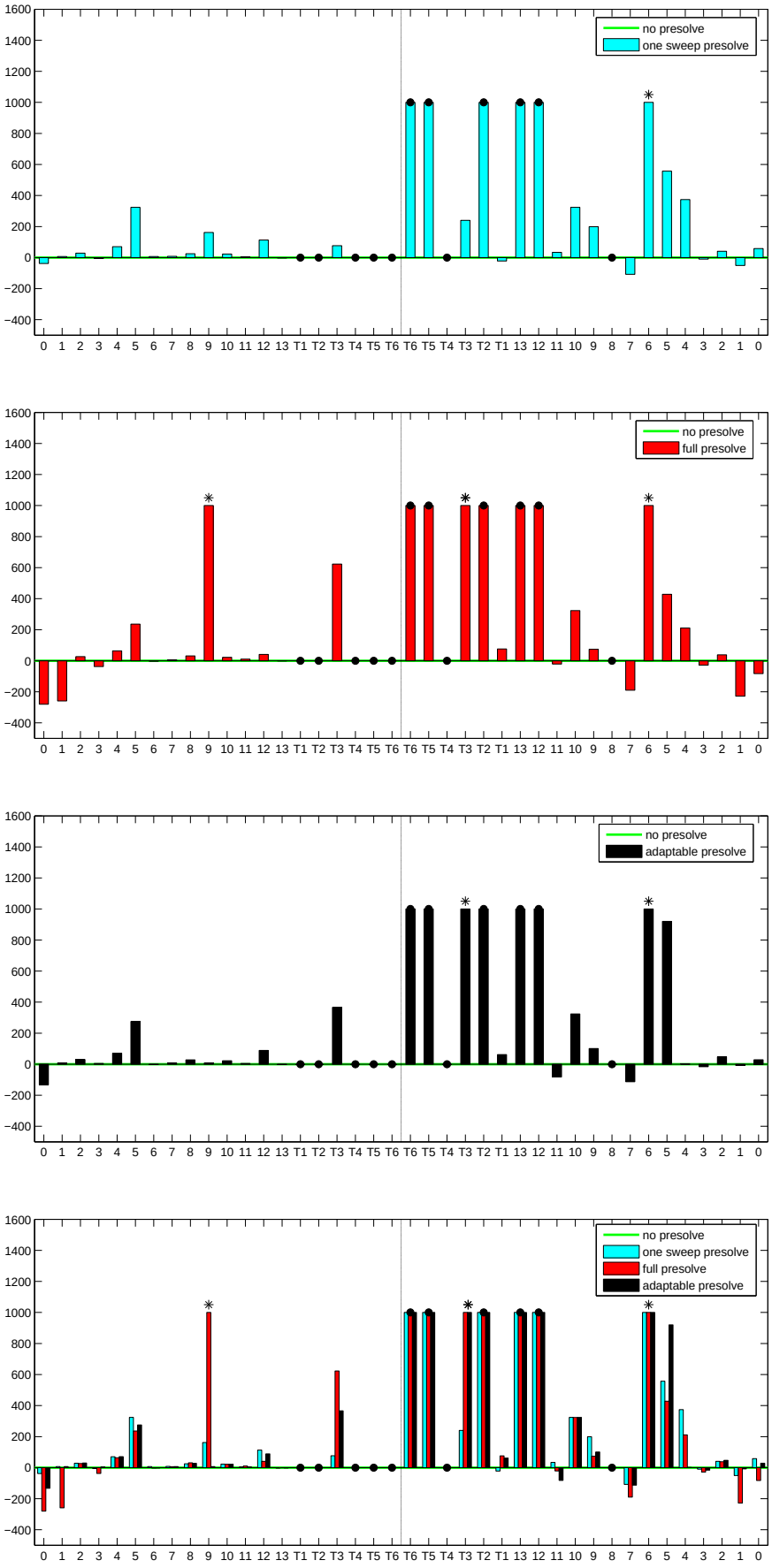


Figure 5.4: Comparison of the number of linear problems solved.

Algorithm 5.2: Adaptable presolve

REPEAT

1. Tighten the bounds on original variables by steps 1 to 3 of Algorithm 5.1.
2. Compute by (5.2) the reduction percentage RP in the sum of the ranges of the original variables with regard to its value before the last sweep.
3. **If** (RP is larger than 20 %) **then** tighten the bounds on new variables $w_{j_i}^i$ by steps 1 to 3 of Algorithm 5.1.

UNTIL (RP is smaller than 20 %)

for the majority of problems. Indeed, it strengthens the bounds more than the one sweep presolve when a sufficient improvement in the range of the variables is observed but it does not tighten the bounds as much as the full presolve since the stopping criterion is less strict (since it is based on the sum of the ranges of the variables instead of on the maximum tightening of only one variable). Note that the deterioration of the results with an adaptable presolve compared to a full presolve is more pronounced for the continuous case. Furthermore, the adaptable presolve is more efficient on a larger number of problems than the other ones as shown by Figure 5.5 which gives the performance profiles for the three methods with presolve and the method without preprocessing. The bottom figure corresponds to a zoom of the performance profiles around $\sigma = 1$. However, for some problems, the convergence of the method with adaptable presolve is considerably slower than with full presolve (39 times slower for *pb9* in the continuous case). For these problems, the first sweep on original variables does not bring much reduction in the range of the original variables and as a consequence, the preprocessing is stopped. But the repetition of presolve phases can finally strongly decrease the variable range. However, such a situation cannot be detected a priori. Nevertheless, as the method with adaptable presolve is more efficient and allows us to solve the same number of problems than the two other methods with presolve, we think that it consists of a good alternative between the one sweep and the full presolves. Therefore, we pursue using it in the following.

5.6 Range reduction

The previous experiments have shown that the presolve can improve a lot the results (see *TVC* problems in the discrete case for example). In this section, we show that the ideas used for the presolve can also be employed during the process of the algorithm to improve its perfomance. When the presolve technique is employed in the tree, we refer to it as *range reduction* instead of preprocessing. Since the presolve phase is quite expensive (2 LPs to solve for each variable candidate for bound tightening), a range reduction phase is not performed at each node. Accordingly, the following strategy is employed: a number k is fixed and a range reduction phase is performed for each node at a level of the tree equal to a multiple of this number k. The scheme of this technique is given in Algorithm 5.3.

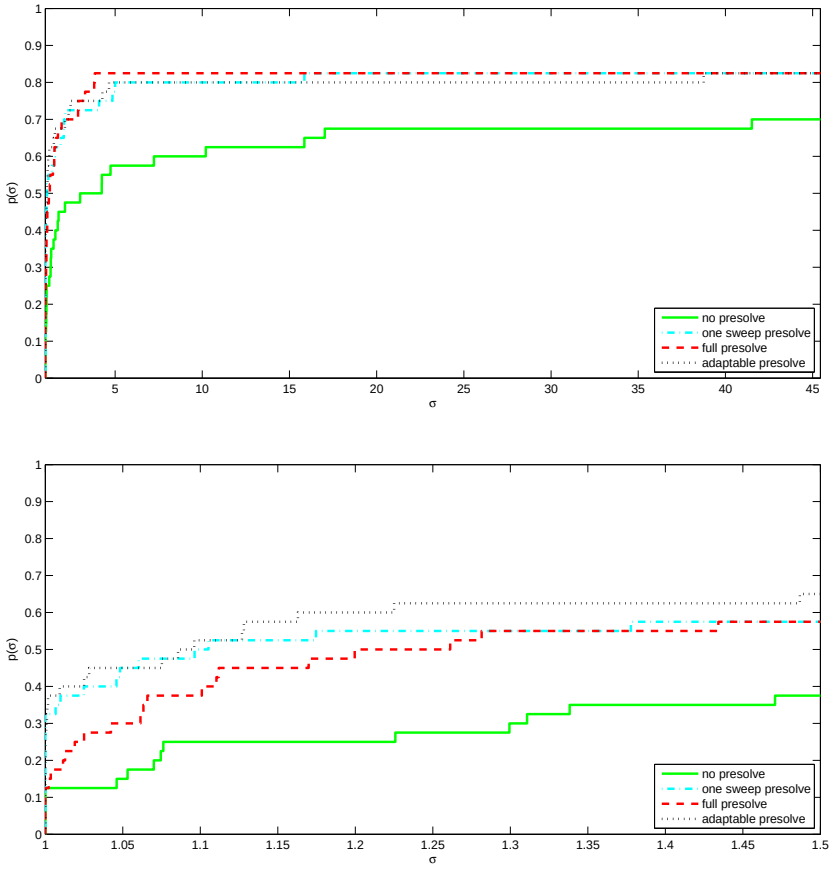


Figure 5.5: Performance profiles of methods with different kinds of presolve and its zoom around $\sigma = 1$.

Algorithm 5.3: Range reduction

Let $level$ be the level in the tree of the examined node and k be a fixed number.

1. If ($level$ = multiple of k) then tighten the bounds on original variables.
2. Proceed to the general iteration of Algorithm 3.2 or 3.3, update $level$ and go to 1.

The level of a node in the tree is used to determine when to proceed to a range reduction instead of the number of iterations in order to avoid risking tighten the bounds for two consecutive nodes. After a vertical moving of k levels in the tree, the problem is expected to be sufficiently modified with regard to the last range reduction on the same branch in such a way that a bound tightening is useful. For our numerical experiments, two different values have been used for k depending on the impact of the presolve on the range of the original variables. If the presolve needs at least two sweeps and that the second sweep allows us to reduce the range of the variables by at least 2%, k is set to 4, otherwise, a range reduction phase is performed less often by setting k to 8[(2)]. For a range reduction phase, the bound tightening is limited to the

[(2)] Among the different alternatives and values tested, these ones have produced the best results.

original variables since the strengthening of bounds on new variables generally produces small modifications only. Moreover, only one sweep is performed contrary to the presolve for which bound tightening is repeated until the sum of the reductions in the range of the original variables is too small. In case of a range reduction, the repetition of consecutive bound tightening would be too costly with regard to the improvement that it would generate. Indeed, with the presolve, the update of the bounds can be applied on the whole tree while the range reduction modifies the bounds only for the subtree (which can be possibly rapidly discarded) generated from the node where the range reduction is performed. Note that Algorithm 3.1 is again used to update the outer approximation problem each time a bound can be tightened during the range reduction phase.

We present on Figure 5.6 the results obtained with an adaptable presolve using, or not, a range reduction compared to the method without presolve. Firstly, we can observe that the

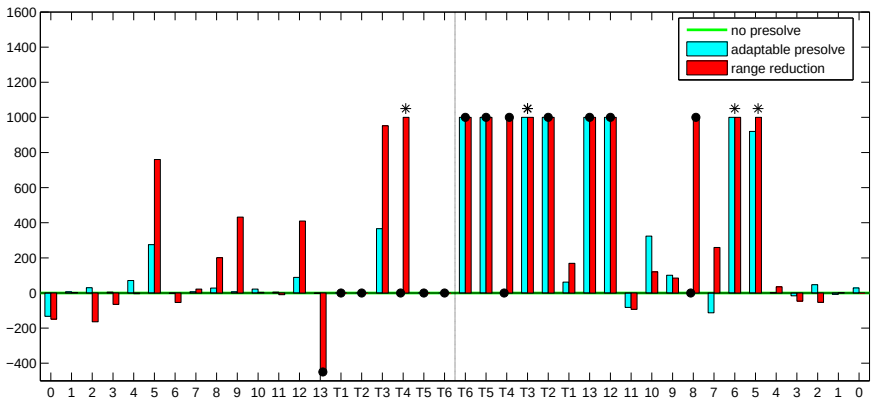


Figure 5.6: Comparison of the number of linear problems solved with an adaptable presolve using or not a range reduction phase compared to the method without presolve.

range reduction allows us to produce the best results on the *TVC* problems among the three compared methods (see Table 8.10 for the detail of the results). With range reduction, the method converges for the first time on *TVC4* for the continuous and discrete cases, as shown by the dots. It also converges for problem *pb8* in the discrete version contrary to the other methods, but it does not find a solution for *pb13* in the continuous case while this problem is solved with the two other methods. However, if we increase the number of linear problems allowed to solve, a solution is found for this problem after solving 735760 LPs, which corresponds to 54% deterioration with regard to the method with adaptable presolve. On the other hand, we can note on Tables 8.9 and 8.10 that the improvements due to the range reduction phases compared to the method with adaptable presolve only, are generally larger than the deteriorations that it generates. The improvements are concentrated on the largest problems while the deteriorations more often arise on the smallest ones, which is logical. Indeed, for the latter on which the presolve is already too expensive, such results could be expected. To the contrary, for the largest problems, the cost of range reduction can be amortized, which produces good results. We now focus on the performance profiles from Figure 5.7 that clearly show which of the three methods (no presolve, adaptable presolve or adaptable presolve combined with range reduction) is the

best since the method with range reduction is both the most efficient and robust.

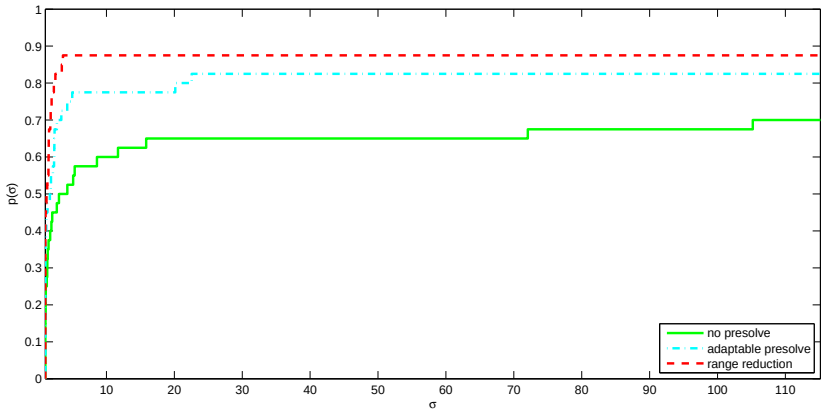


Figure 5.7: Performance profiles related to the comparison of the number of linear problems solved with an adaptable presolve using or not a range reduction phase compared to the method without presolve.

Finally, we can note in Tables 8.9 and 8.10 that the method with range reduction does not only allow us to reduce the number of linear problems solved but also the number of nonlinear problems, in the sense that it *reduces the ratio between the number of NLPs and of LPs solved*. Indeed, during the general process of the algorithm, an NLP is associated to each LP successfully solved unless the related node can be cut (see Algorithm 3.2 or 3.3). On the other hand, no NLP is associated to the LPs solved during the presolve and the range reduction phases. Accordingly, the ratio between the number of NLPs and of LPs decreases by using a presolve and even more with range reduction. These ratios can be found in Tables 8.8, 8.9 and 8.10 for the three methods compared in Figure 5.7. Finally, as the number of LPs solved decreases by using bound tightening, the number of nonlinear problems also decreases. For all the reasons mentioned above, the method with range reduction will be always applied in the following.

5.7 Conclusion

This chapter has shown that presolve can significantly improve the results if the problem to solve is not too small. Otherwise, its cost is too high compared to the improvement that it generates. In all cases, a trade-off must be found between these two aspects. To this aim, an adaptable presolve based on the improvement percentage in the sum of the ranges of the original variables has been proposed. Another presolve depending on the size of the problem could also be considered since for the treated problems, the efficiency of the presolve on a problem seems to be correlated to the size of this problem.

Moreover, the bound tightening is not only efficient at the beginning of the algorithm but also during its process. With such a technique, the results have been a lot improved and the resulting method has allowed us to converge on some problems which could not be solved before, within the number of linear problems allowed (even though, some problems still remain

unsolved). Finally, we have highlighted in this chapter that the obtained results are strongly dependent on the order in which we branch on the variables. Therefore, a suitable technique should be developed to choose the best branching variable.

Chapter 6

Variable selection

In this chapter, the question of the choice of the *branching variable* is examined. As the goal of branching is to refine the outer approximation problem and also, in the discrete case, to satisfy the discrete restrictions, the choice of the branching variable must be treated carefully. Indeed, it determines the way of building the tree, and thus the speed of convergence of the outer approximation method, as explained in Section 1.2.4. For the experiments presented in the previous chapter, a very simple branching rule which consists to branch on the variable having the largest range has been used. With this choice, the method proposed in Chapter 5 was not converging on some problems before the maximum number of linear problems allowed to solve was reached. We show here that the convergence can be obtained for these problems by using a more clever *branching rule*. Several ways to choose the branching variable among the variables of the original problem are proposed, tested and discussed: by basing the choice on the *approximation errors* (with two different ways to compute these errors), by adapting to our outer approximation problem the *strong branching* method suggested by Applegate *et al.* in [12] as well as the *pseudocost* technique of Bénichou *et al.* [23] developed in mixed integer linear programming, and finally by combining pseudocosts with strong branching iterations. As shown by the numerical experiments, this latter is the best choice amongst the proposed strategies. We also highlight the interest of "branching again" on variables which have allowed us to cut nodes in a close past, and the importance of weighting the approximation errors used in branching rules by the coefficients of the original problem.

For all the numerical experiments presented below, the adaptable presolve and range reduction techniques developed in Sections 5.5 and 5.6 respectively, have been applied since it has been shown that they generally improve the results. Therefore, the basic method on which the possible improvements are tested and quantified is the one detailed in Section 5.1.1 and modified by the addition of an adaptable presolve and a range reduction technique.

6.1 Preliminary note on branching rules

Before developing and testing branching rules, we first give some comments about their applicability and present a trick allowing us to generally improve them.

6.1.1 Applicability of branching rules

Branching rules aim at choosing among candidate variables for branching, the one which is the best to branch on according to some criterion defined by the branching rule. As branching is used to make the outer approximation problem as close as possible to the nonlinear approximated problem and also to satisfy the discrete restrictions (if the problem is subject to such restrictions), the candidate variables for branching are the *variables of the original problem which appear nonlinearly in the problem or which are discrete*.

Because of the goal of branching, the variable selected by a branching rule is used provided, at the current solution, it is not at its nonlinear value for the outer approximations of all components in which it appears or, provided it does not satisfy the discrete restrictions. Otherwise, it would be useless to branch on this variable, since the branching could not bring any improvement in the quality of the outer approximation problem. For the same reason, we do neither branch on variables appearing only linearly in the nonlinear problem. Indeed, each linear component of a nonlinear problem remains unchanged in the linear outer approximation problem. By summary, the branching variable is the first variable selected by the branching rule *producing an approximation error or violating a discrete restriction*.

6.1.2 Exploiting the best candidate for branching

We pursue this discussion on branching by doing a general note: nodes can sometimes be cut rapidly due to the branching on *only one* variable, always the same, among those involved in the branching. Branching on other variables would not allow us to fathom nodes (at least after a limited number of branching on the same branch), but if we branch on this particular variable, the generated nodes can be immediately cut. Detecting such a situation is desirable because it would be interesting to directly branch on this particular variable instead of branching on the variable selected by some branching rule. Therefore, in practice, we consider the branching which has been done just before the two last nodes that have been fathomed. If the branching has been performed on the same variable for both, we can assume that the branching on this particular variable will again allow us to cut nodes in the next branching phases. Accordingly, we do not employ any branching rule but we branch again on this variable. If this allows us to cut the two generated nodes, we pursue branching on it, otherwise, we come back to the branching rule under use. We refer to this technique as the *exploitation of the best candidate for branching*.

Note that this technique is a trick added to improve a branching rule, and not a branching rule itself since it can be employed in few cases. In fact, it is only used when it is expected to allow a faster convergence of the method. We consider that the fathoming of only one node (left or right) obtained just after branching does not produce a sufficient improvement to use this trick. We would like to fathom both nodes instead. To this aim, in order to determine if a best candidate for branching exists, we only focus on the branching which has been performed just before the two last nodes which have been cut by *going down* in the branch-and-bound tree. The branching realized before a node fathomed by going up in the tree is not considered. Indeed, in the case of a depth-first search with backtracking (see Section 5.1.1), a node examined by going up in the tree implies that the other node generated by branching was feasible. Accordingly, in such a situation, we are not interested in branching again on the variable which has been used for this branching since it has not allowed us to cut both nodes. To the contrary, if a node

explored by going down in the tree has been cut, we can think that the other node generated by branching will be also fathomed since the first subproblem to be treated among the left and the right is the one which is expected to be the most feasible, as explained in Section 5.1.1.

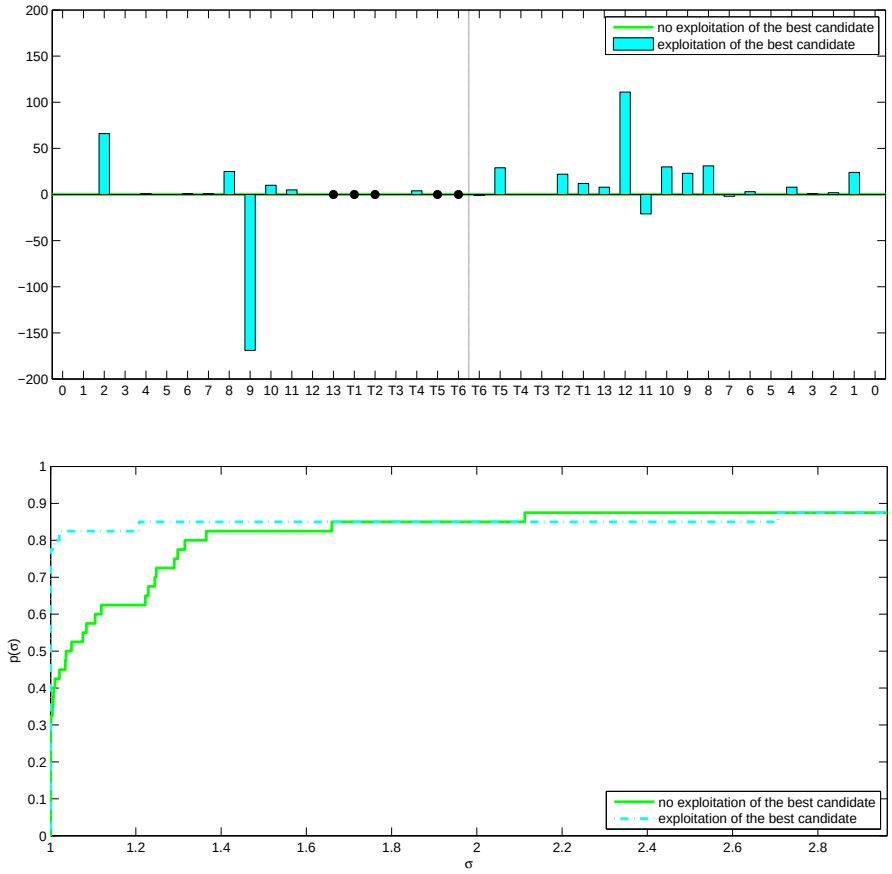


Figure 6.1: Comparison of the number of linear problems solved with or without exploiting the best candidate for branching, if it exists.

This trick has been applied to the basic method developed before with the branching rule based on the largest range. The results obtained with or without exploiting the best candidate for branching, if it exists, are graphically presented on Figure 6.1 (see also Tables 8.10 and 8.11 for more details). The top figure represents the improvement percentages i_{pm} defined in (5.1) and produced by exploiting the best candidate for branching, while the bottom figure gives the performance profiles. Remember that the interpretation of these graphs has been detailed in Sections 5.1.4 and 5.3. The dots at $y = 0$ in the top figure show that both compared methods do not converge on five of the largest problems in the continuous case. Nevertheless, the exploitation of the best candidate for branching allows us to produce twenty-one improvements in the results, ten equalities and only four deteriorations with regard to the basic method. One of these deteriorations is however quite important (160% deterioration for *pb9* in the continuous case) with regard to the improvements produced by this trick, as shown in the top graph of Figure 6.1 or in the right part of the performance profiles. In fact, all branching rules being subject to a random part out of control, good techniques give sometimes worse results on some problems

than other generally less good methods, the latter having the chance to branch on a "good variable" at a "good time". As the method proposed in this section generally improves the results while the deteriorations are seldom, we validate this trick and add it to the basic method for the next experiments.

6.2 Maximum approximation error branching

Instead of branching on the variable with the largest range, the branching rule proposed in this section is based on the *approximation errors*. Indeed, to produce a relevant solution for the nonlinear problem, the linear outer approximation problems must be close to the nonlinear problem. To this goal, we aim at refining, among all the outer approximations of the nonlinear components used in the problem (see Section 3.2), the one which produces the maximum approximation error. This idea has been used by Adjiman *et al.* [8]. We have applied this concept by computing the errors in two ways: as the *theoretical approximation errors* but also as the *approximation errors really produced* at the current solution. Initially, we only focus on the continuous problems.

For each approximated nonlinear component, the maximum error (theoretical or real) generated by its outer approximation is computed. Once these errors are determined, the nonlinear component associated to the largest error is selected. However, as we want to branch on original variables only, we must decide how to choose the branching variable from this component. In the developed method, the components can be functions of one or two variables which can be themselves original or new, as explained in Section 3.2.1. If the selected component is a function of one variable and if this variable belongs to the original problem, this variable is chosen for branching. To the contrary, if the variable is a new variable introduced by the linear outer approximation formulation, it is used to replace a nonlinear component. Consequently, this new component becomes the one to refine and the process is repeated in order to determine the variable of the original problem which contributes the most to the approximation error associated to this component. Now, if the nonlinear component is a function of two variables, more precisely a bilinear function in the employed formulation, the variable to branch on must be selected among both. In this case, the variable with the largest range is chosen since it is the one which contributes the most to the volume of the domain of possible values for the outer approximation of a bilinear product (see (4.22)). If this variable belongs to the original problem, it is selected for branching, otherwise the associated component becomes the component to refine and we look for the variable which contributes the most to the associated approximation error. This process is repeated until finding a variable of the original problem to branch on.

To illustrate this strategy, the following example is considered: suppose that the nonlinear function:

$$x_1 x_2 \sin(x_3)$$

appears in a problem and that it is approximated by its linear outer approximation w_3 such that:

$$\begin{aligned} w_1 &\approx \sin(x_3), \\ w_2 &\approx x_1 x_2, \\ w_3 &\approx w_1 w_2, \end{aligned}$$

where $x \approx y$ means that x is the linear outer approximation of y. Assume also that the maximum approximation error in the problem occurs for w_3. As w_3 approximates a component of

two variables (w_1 and w_2), we determine the one which contributes the most to the error, i.e., the one with the largest range. Suppose that it is w_1. Since w_1 is not an original variable, we look for the variable which contributes the most to the approximation error associated to w_1. As w_1 replaces a function of one variable only, x_3, which belongs to the original problem, x_3 is finally selected for branching.

On the other hand, when discrete restrictions are imposed in the nonlinear problem, new possibilities are introduced for the solution of the linear outer approximation problem to produce an error for the mixed integer nonlinear problem. Indeed, the solution of the outer approximation problem does not necessarily satisfy the discrete restrictions. Therefore, in addition to the approximation errors related to the nonlinear components, we must also take the errors associated to discrete restrictions into account to determine the variable which can make the outer approximation problem the closest to the original one by branching on it. We must thus select the maximum one among the errors related to the approximation of a nonlinear component and the errors corresponding to the violation of a discrete restriction. In the following, we improperly assimilate each discrete restriction to a nonlinear component and the error produced by the violation of a discrete restriction to an approximation error. As a consequence, in the discrete case, the number of components to take into account in order to determine the one associated to the most important error is larger than in the continuous case.

This branching rule is summarized in Algorithm 6.1. The results obtained by applying this algorithm are now considered by using two different definitions for the approximation errors.

Algorithm 6.1: Maximum approximation error branching

1. For each nonlinear component and for each discrete restriction, *compute the error* produced by its approximation or by its violation, respectively.
2. Choose among the computed errors the *maximum* one provided that a positive error occurs at the current solution for the component $w^i_{j_i}$ or for the discrete restriction on y_k associated to the error.
3. **Do while** (the branching variable has not been found)
   If (the error is related to a *discrete restriction* on y_k) **then**
   Select y^k as branching variable and **stop**
   else if ($w^i_{j_i}$ is an *unary* function) **then**
   if ($w^i_{j_i}$ is a function of an original variable x_k) then
   Select this variable x_k for branching and **stop**
   else
   Set $w^i_{j_i}$ to its argument
   else (choose among the *two arguments* of $w^i_{j_i}$ the one with the *largest range*)
   if (this argument is an original variable) **then**
   It is selected for branching and **stop**
   else
   Set $w^i_{j_i}$ to this argument

6.2.1 Maximum theoretical approximation errors

The first examined errors are the maximum *theoretical* approximation errors, that is, for each nonlinear component, the maximum approximation error (underestimation or overestimation) is theoretically computed. We distinguish here the errors produced by an outer approximation or by a discrete restriction.

Maximum theoretical approximation error generated by an outer approximation
If $f(x)$ is approximated by $w_{f(x)}$ on $[l_x, u_x]$, the maximum theoretical approximation error is given by:

$$e_f = \max_{x\in[l_x,u_x]} |f(x) - w_{f(x)}|.$$

More precisely, in the present case, for each nonlinear component f of the problem, we must compute the maximum error between the function f and any of its linear outer approximations based on SOS, $w_{f(x)}$, satisfying (3.47). Accordingly, e_f can be written as:

$$\begin{aligned} e_f = &\max_{x\in[l_x,u_x],\ \lambda_i} \left(\max\left(f(x) - \sum_{i=1}^{p} \lambda_i f(x_i) + \epsilon_{f,L}, \sum_{i=1}^{p} \lambda_i f(x_i) + \epsilon_{f,U} - f(x)\right)\right) \\ &\text{with} \quad x = \sum_{i=1}^{p} \lambda_i x_i, \quad \sum_{i=1}^{p} \lambda_i = 1, \quad \lambda_i \geq 0, \quad 1 \leq i \leq p. \end{aligned} \tag{6.1}$$

In this formulation, the errors $\epsilon_{f,L}$ and $\epsilon_{f,U}$ correspond respectively to the maximum overestimation and underestimation errors produced by the SOS approximation of f and used in (3.47). On the other hand, the error e_f is computed without assuming that the SOS condition is satisfied because no requirement is imposed to this goal in the linear outer approximation problems $(\widetilde{OP})$ of Section 3.2.5. Accordingly, by (6.1), we must determine the maximum approximation error (underestimation or overestimation) produced between f and all possible values for its linear outer approximation based on SOS, $w_{f(x)}$, satisfying (3.47). Like in the previous chapters, the errors are separately computed for the three kinds of nonlinear components appearing in the decomposition of the *TVC* problem.

1. $\boxed{x^2}$
   The domain of possible values for the outer approximation of x^2 defined in (3.47) is represented in Figure 6.2 in case of an outer approximation based on three breakpoints. It is clear on this figure that the maximum approximation error (underestimation or overestimation) between x^2 and an outer approximation satisfying (3.47) is reached for (x_2, x_1^2) or, equivalently, for (x_2, x_p^2), (where p, the number of breakpoints is equal to 3 in Figure 6.2). It is an overestimation error which is equal to the maximum overestimation approximation error produced by the SOS approximation of x^2 on the piece $[x_1, x_p]$. Therefore, by Theorem 3.1 applied to the piece $[x_1, x_p]$, the maximum theoretical approximation error is equal to:
   $$\frac{(x_p - x_1)^2}{4}. \tag{6.2}$$
   Note that this error can be easily obtained from the maximum overestimation approximation error $\epsilon_{x^2,L}$ (3.22) used in (3.47) and which has already been computed to build the

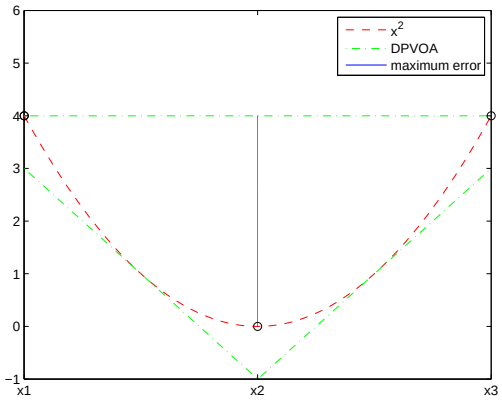


Figure 6.2: Maximum approximation error on $DPVOA$ for x^2.

outer approximation problem $(\widetilde{OP})$. Indeed, by multiplying (3.22) by $(p-1)^2$, we obtain (6.2) since $l_x = x_1$ and $u_x = x_p$.

2. xy

We now switch to the case of a bilinear product. As explained in Section 3.2.4, the approximation errors $\epsilon_{f,L}$ and $\epsilon_{f,U}$ used in (6.1) are set to zero for this function. Furthermore, Theorem 4.6 has stated that, when (x, y) is defined on a rectangle $[x_1, x_{p_x}] \times [y_1, y_{p_y}]$, the set of possible values for the linear outer approximations of xy defined in (3.47) coincides with the set of convex combinations of points (x, y, xy) where (x, y) belongs to the rectangle. Combining this result with Theorem 4.5, we obtain that the set of the linear outer approximations of xy is delimited by the convex and concave envelopes of the bilinear product. Determining the maximum approximation error between the bilinear product and any linear outer approximation satisfying (3.47) thus amounts to compute the maximum error, in absolute value, between the bilinear product and its concave or convex envelope. The expression of these envelopes on a rectangle established in [11] by Al Khayyal and Falk have been reported in Theorem 4.3. Moreover, it has been shown in the proof of Theorem 4.10 that these convex and concave envelopes on a rectangle are defined on triangles corresponding to the half of the considered rectangle $[x_1, x_{p_x}] \times [y_1, y_{p_y}]$. As it can be easily derived that these envelopes coincide with the SOS approximations of the bilinear product on such triangles (they are both linear functions defined by the three same points), the maximum theoretical approximation error that we are looking for corresponds to the maximum approximation error generated by the SOS approximation of xy on these triangles. Therefore, formula (3.30) can be applied with $\Delta_x = x_{p_x} - x_1$ and $\Delta_y = y_{p_y} - y_1$ to find that the maximum theoretical approximation error (6.1) for a bilinear product is equal to:

$$\frac{(x_{p_x} - x_1)(y_{p_y} - y_1)}{4}. \tag{6.3}$$

3. Trigonometric functions

For these functions, we first consider the maximum underestimation and overestimation

approximation errors generated by any linear approximation of f (not outer), that is, by any linear approximation belonging to DPV defined in (3.46). This amounts to set in (6.1) the approximation errors $\epsilon_{f,L}$ and $\epsilon_{f,U}$ to zero. Once these maximum underestimation and overestimation approximation errors are determined on DPV, the maximum theoretical approximation error e_f, that is, the maximum error in absolute value on $DPVOA$ defined in (3.47), can be obtained by adding the approximation errors $\epsilon_{f,L}$ and $\epsilon_{f,U}$ to the underestimation and overestimation approximation errors produced on DPV respectively, and by taking the maximum among both. Indeed, as the overestimation error $\epsilon_{f,L}$ is removed from the linear approximation in (3.47) and as the underestimation error $\epsilon_{f,U}$ is added, the underestimation and overestimation errors on $DPVOA$ are the corresponding errors on DPV increased by $\epsilon_{f,L}$ and $\epsilon_{f,U}$ respectively. This is summarized in Theorem 6.1 which is also valid for other functions than trigonometric ones.

Theorem 6.1 *Let e_U^* and e_O^* be the maximum underestimation and overestimation approximation errors reached on DPV. The maximum theoretical approximation error e_f, that is, the maximum approximation error in absolute value produced on $DPVOA$, is equal to:*

$$e_f = \max(e_U^* + \epsilon_{f,L}, e_O^* + \epsilon_{f,U}),$$

where $\epsilon_{f,L}$ and $\epsilon_{f,U}$ are respectively the maximum overestimation and underestimation approximation errors produced by the SOS approximation of f and used in the definition of $DPVOA$.

We first compute the maximum overestimation and underestimation approximation errors produced on DPV. However, the determination of the domain DPV is not obvious. Indeed, by definition, DPV corresponds to the set of convex combinations of points $(x_i, f(x_i))$ where x_i are breakpoints but the breakpoints which really define this set depend on the studied interval. For example, for $\sin(x)$, if the linear approximations are built by using five equally spaced breakpoints, DPV is defined on $[0, 2\pi]$ by $x_1,\ x_2,\ x_4$ and x_5 (thus not x_3) while it is by all breakpoints on $[0, \pi]$, as shown on Figures 6.3 and 6.4. On these figures, breakpoints are highlighted by circles and the domain DPV based on these breakpoints is represented by the rectangle comprising them for $[0, 2\pi]$ and by the pentagon for $[0, \pi]$.

We start by examining $\sin(x)$. The error at a point x between this function and a linear approximation belonging to DPV must arise on a side of DPV and not at a point inside it. Indeed, the maximum error between a function and a point of a domain limited by linear constraints is reached on one of these constraints, in the present case, on a side of DPV. For example, on Figure 6.4, the maximum approximation error can be reached on the segments defining DPV, that is, on the segments joining $(x_1, \sin(x_1))$ to $(x_2, \sin(x_2))$, $(x_2, \sin(x_2))$ to $(x_3, \sin(x_3))$, $(x_3, \sin(x_3))$ to $(x_4, \sin(x_4))$, $(x_4, \sin(x_4))$ to $(x_5, \sin(x_5))$ and, finally, $(x_1, \sin(x_1))$ to $(x_5, \sin(x_5))$. Observe that, on $[0, \pi]$, the maximum overestimation approximation error on DPV is equal to zero since any linear approximation belonging to DPV underestimates $\sin(x)$ on this interval, while the maximum underestimation error is reached on the straight line joining $(x_1, \sin(x_1))$ to $(x_5, \sin(x_5))$ at $x = x_3 = \frac{\pi}{2}$ and is equal to $\sin(\frac{\pi}{2}) - 0 = 1$. We now detail how to compute these errors analytically. To this aim, the maximum overestimation and underestimation approximation errors between the function $\sin(x)$ and a segment joining

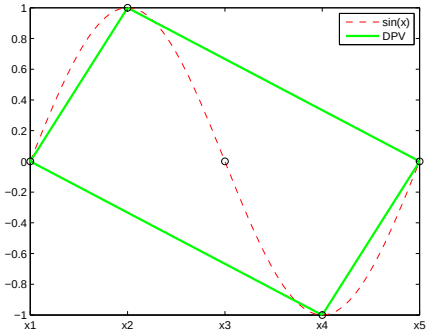


Figure 6.3: DPV for $\sin(x)$ on $[0, 2\pi]$ defined by five equally spaced breakpoints.

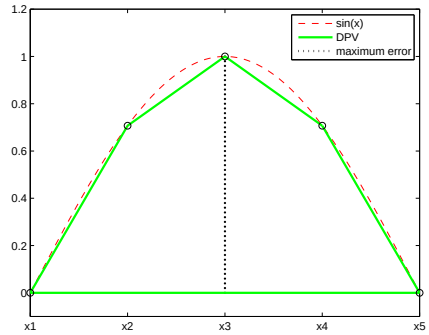


Figure 6.4: DPV for $\sin(x)$ on $[0, \pi]$ defined by five equally spaced breakpoints.

$(x_i, \sin(x_i))$ to $(x_j, \sin(x_j))$ are first determined. Such a segment can be seen as the SOS approximation of $\sin(x)$ on the piece $[x_i, x_j]$. Therefore, applying Theorem 3.5 on the piece $[x_i, x_j]$ allows us to directly determine the maximum overestimation and underestimation errors that we are looking for on this piece.

Theorem 3.5 gives the expression of the maximum underestimation and overestimation approximation errors between $\sin(x)$ and the straight line joining $(x_i, \sin(x_i))$ to $(x_j, \sin(x_j))$. For some couples (x_i, x_j), this line defines a side of DPV, as explained earlier. To determine the maximum underestimation and overestimation approximation errors produced on DPV, Theorem 3.5 must be applied to each side of DPV. Among all the computed overestimation (respectively underestimation) approximation errors, the largest one corresponds to the maximum overestimation (respectively underestimation) error that we are looking for.

By a similar reasoning for the function $\cos(x)$, Theorem 3.6 can be applied to determine the maximum underestimation and overestimation errors on DPV for this function. By adding the maximum underestimation and overestimation errors on DPV computed by using Theorems 3.5 and 3.6 to the approximation errors $\epsilon_{f,L}$ and $\epsilon_{f,U}$ as explained in Theorem 6.1, the maximum approximation error on $DPVOA$, or equivalently e_f, is finally obtained.

Maximum theoretical approximation error produced by a discrete restriction
Let us now consider the maximum theoretical approximation error generated by the violation of a discrete restriction. Remember first that, in Section 2.1.2, all the discrete restrictions have been modelled as *integer restrictions*. The analysis can thus be limited to errors produced by the violation of an integer restriction.

Without loss of generality, we assume that the variable x subject to an integer restriction is defined on $[\bar{l}_x, \bar{u}_x]$ where $\bar{l}_x$ and $\bar{u}_x$ are two integers such that $\bar{u}_x > \bar{l}_x$. The considered error, denoted $e_{int}(x)$, produced for an integer variable at a point x in $[\bar{l}_x, \bar{u}_x]$ corresponds to the minimum distance between x and the nearest integer. Therefore, $e_{int}(x)$ is defined by:

$$e_{int}(x) = \min(x - \lfloor x \rfloor, \lceil x \rceil - x). \tag{6.4}$$

The maximum theoretical approximation error for an integer restriction is obtained by taking the maximum on x in equation (6.4), that is,

$$\max_{x \in [\bar{l}_x, \bar{u}_x]} e_{int}(x).$$

It can be easily derived that this maximum is reached at $x = \frac{\lfloor x \rfloor + \lceil x \rceil}{2}$ and at this value, $e_{int}(x)$ is equal to 0.5. Therefore, the maximum theoretical approximation error for an integer restriction is always constant.

Usefulness of weighting the errors
In a first attempt to use the branching rule based on the maximum theoretical approximation errors on the test problems of Section 5.1.2, only one of the five problems unsolved by the basic method developed at this stage could be solved. Comparing with the branching rule based on the largest range, it has been observed that the best results were shared between both methods depending on the problems treated. For more details about the numerical results, we refer the reader to Table 8.12.

The disappointing results produced by the branching rule based on the maximum theoretical approximation errors can be explained by the fact that the branching has been performed on the variable contributing the most to the approximation error for one component but without exploiting the knowledge of the problem. Indeed, a component can be subject to a large approximation error, but if its coefficient in the nonlinear problem is very small, the branching on the associated variable will not make the linear outer approximation problem much closer to the nonlinear one. To remedy this problem, the approximation errors (including also the violations of integer restrictions) have been *weighted* by the absolute value of the coefficients of the associated nonlinear components appearing in the original problem. If a nonlinear component appears more than once in the problem, the largest coefficient (in absolute value) associated to this component is chosen to weight the error[(1)].

Figure 6.5 compares the results obtained with or without weighting the maximum theoretical approximation errors and shows that the majority of the results is improved by weighting (see also Tables 8.12 and 8.13). The method with weighting converges for all *TVC* problems, in

[(1)] An alternative that would be worth to be tested would be to take into account the number of times that a variable appears in the problem by summing all the (weighted) errors that it generates in the problem.

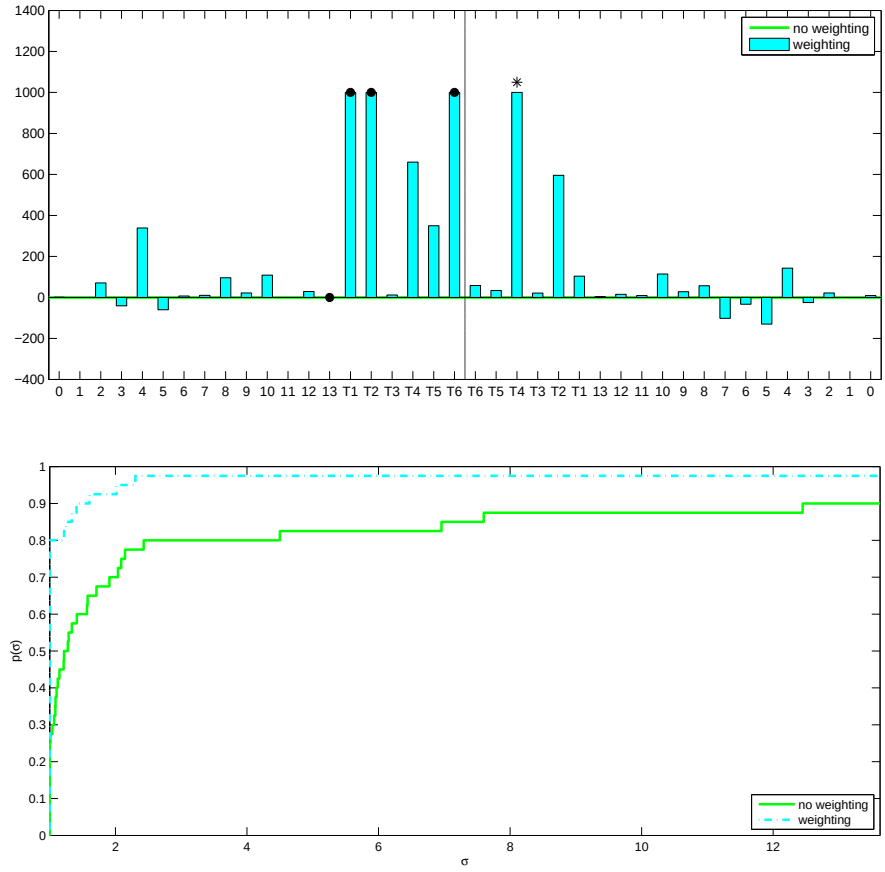


Figure 6.5: Comparison of the number of linear problems solved with or without weighting the maximum theoretical approximation errors.

particular for *TVC1*, *TVC2* and *TVC6* in the continuous case, which were unsolved so far within the maximum number of linear problems allowed to solve. For the other problems, only *pb13* in the continuous case remains unsolved. These facts are represented by dots on the top figure of Figure 6.5 (see Sections 5.1.4 and 5.3 for a detailed interpretation of the graph). Furthermore, the improvements obtained with weighting can be important with regard to the produced deteriorations. More precisely, the number of linear problems solved for *TVC4* is approximatively divided by eight in the continuous case and by twelve for the discrete case when the errors are weighted (that is 660% and 1145% improvement, respectively). On the other hand, the largest deterioration in the results is equal to 130% and occurs for *pb5* in the discrete case. Such a deterioration in the results can be explained by the fact that, sometimes, the choice of the branching variable without weighting is better because it allows us to divide early the range of a variable which can improve the speed of convergence by branching on it (for instance, because branching on it increases the value of the objective function which is the goal of strong branching detailed in Section 6.3). As a consequence, it happens that weighting does not always give better results. However, since the method with weighting is more robust and efficient than the method without, as shown in the performance profiles of Figure 6.5 and since the deteriorations obtained with weighting mainly arise for smallest problems (the ones at the extremities of the

top graph) while the largest improvements are obtained for the most difficult problems (the ones at the center of the top graph), this modification of the basic method can be considered as an appreciable step and we pursue using this idea below.

Note that we can a priori be afraid that the maximum theoretical branching rule together with the fact that the number of breakpoints is fixed introduces a bias in the method by preferring branching on a particular kind of function. In practice, such a drawback does not happen due to the fact that the ranges of the variables are different and that the theoretical approximation errors are weighted by the coefficients of the original problem.

Branching rule based on weighted maximum theoretical errors versus on largest range
The results obtained when the branching rule is based on the weighted theoretical approximation errors are now compared with the results when branching on the variable having the largest range. The graphic comparison is presented in Figure 6.6 while the extensive numerical results are given in Tables 8.11 and 8.13. It can be observed from the top picture of Figure 6.6 that

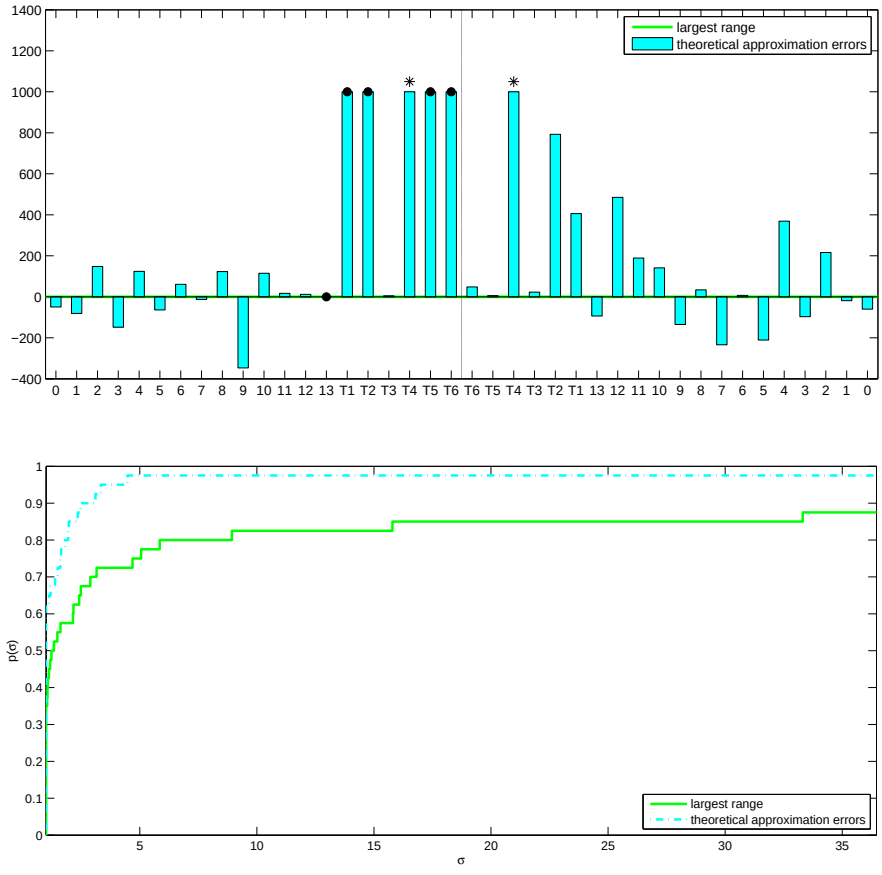


Figure 6.6: Comparison of the number of linear problems solved by basing the branching rule on the maximum weighted theoretical approximation errors or on the largest range.

for problems *pb*, the best results are shared between both methods. The results really depend whether the variables with the largest range at the top of the tree are variables which allow a quick convergence by branching on them. If they are, the method based on the largest range

can be better because it branches early in the tree on important variables for the convergence. If they are not, the method based on the approximation errors is generally better. The success of the method with the largest range is thus more random because it depends on the range of the variables at the top of the tree. For *TVC* problems, the use of weighted theoretical errors improves the results since it allows us to get convergence for all of these problems and notably for four problems that did not converge with the branching rule based on the largest range. Moreover, for *TVC4* which can be solved by the two methods, branching according to the approximation errors divides the number of linear problems solved by thirty-three in the continuous case and by sixteen in the discrete one, which explains the two stars in the graph. Employing weighted maximum theoretical approximation errors thus allows us to obtain a more efficient and robust method, as shown in the performance profiles of Figure 6.6. As a consequence, we validate this technique with regard to the branching rule based on the largest range and adopt it in the basic method. A more elaborate variable selection can thus significantly improve the results.

6.2.2 Approximation errors really produced at the current solution

We now investigate the case where we branch on the variable producing the maximum *real* approximation error at the current solution. So, if a nonlinear component $f(x)$ is approximated by $w_{f(x)}$ and if x_0 is the value of x at the current solution, the real approximation error produced by $w_{f(x)}$ is given by:

$$e_f(x_0) = |f(x_0) - w_{f(x_0)}|.$$

If the problem is subject to integer restrictions, the associated errors are obtained by evaluating $e_{int}(x)$ defined in (6.4) at the current value of the integer variables.

Only the computation of the errors is modified with regard to the technique based on maximum theoretical approximation errors. For the numerical experiments, the computed approximation errors have still been weighted by their associated coefficients in the nonlinear problem. Figure 6.7 compares the results obtained by basing the branching rule on real approximation errors or on maximum theoretical approximation errors (see also Tables 8.13 and 8.14). By comparing these methods, we note that the best method between both is strongly problem dependent. By considering the top graph of Figure 6.7, it is not totally clear which method is the best. According to the performance profiles, the branching based on theoretical approximation errors is more efficient since it produces the best results for 24 problems (on 40). However, the largest improvement is generated by the branching rule based on the real approximation errors, as shown in the right part of the performance profiles of Figure 6.7. Note that this improvement (220%) is obtained for *pb3* in the continuous case, which is a small problem.

Moreover, the cost of the two methods is not too different to prefer one method instead of the other. Indeed, in each case, approximation errors associated to the nonlinear components or to the discrete restrictions appearing in the problem must be computed. As explained in Section 6.2.1, the maximum theoretical approximation errors can be obtained for square functions by multiplying by a constant (depending on the number of breakpoints) the approximation errors $\epsilon_{x^2,L}$ defined in (3.22) and already computed to build the outer approximation problem. By (6.3), the computation of the maximum theoretical approximation error associated to a bilinear product is also relatively simple. For trigonometric functions, we need to apply Theorem 3.5 or 3.6 to compute the maximum overestimation and underestimation errors on each possible sides

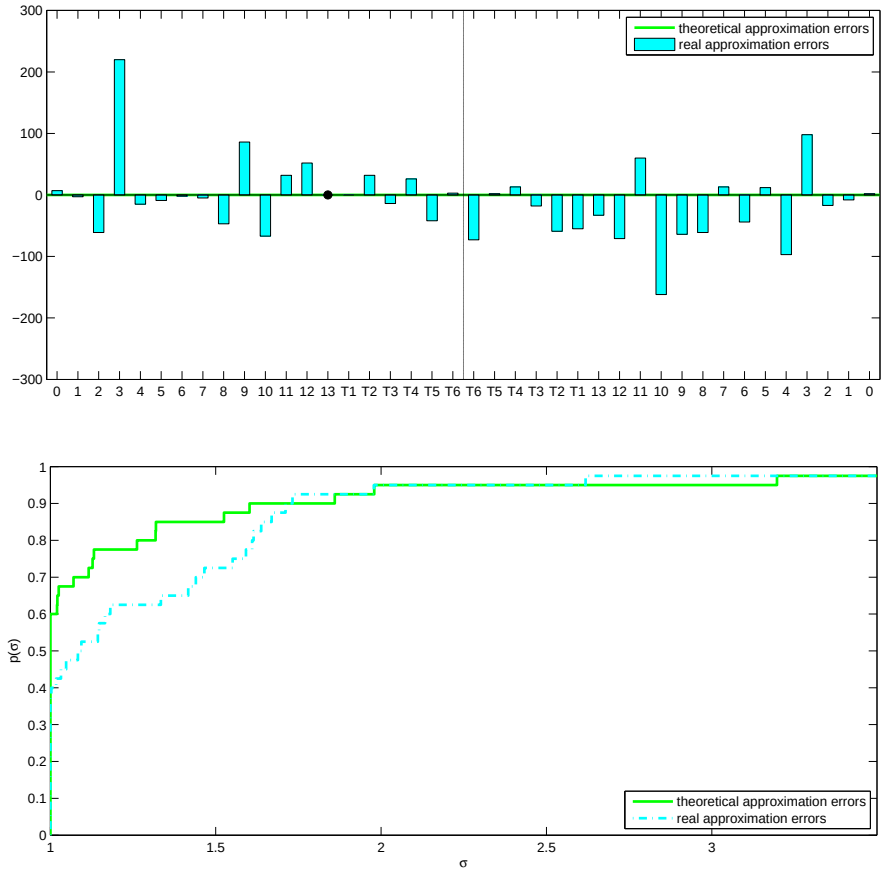


Figure 6.7: Comparison of results obtained by basing the branching rule on maximum theoretical or real approximation errors (weighted in both cases).

of DPV. Among these, the maximum ones must be determined and by Theorem 6.1, these values must be summed with the maximum approximation errors $\epsilon_{f,L}$ and $\epsilon_{f,U}$ used in (3.47) and which have been already computed to build the outer approximation problem. The determination of the theoretical errors associated to trigonometric functions is thus a little bit more expensive than for square and bilinear functions. For an integer restriction, the maximum theoretical approximation error is a constant. On the other hand, to compute the real approximation errors, the different nonlinear components appearing in the problems (x^2, xy and trigonometric functions) must be evaluated at the current solution. The resulting values are then compared with the values of their approximations which have been computed during the solution of the linear outer approximation problem. For the integer restrictions, the real approximation errors are given by the function $e_{int}(x)$ evaluated at the current solution. Finally, the approximation errors obtained are weighted in the same way in the two cases. As the computational cost of the errors in both cases is not dramatically different and remains cheap compared to the total cost of the method, no method can be preferred.

However, the method based on theoretical approximation errors is better for a larger number of problems, and also for a larger number of *TVC* problems (7 on 12). As we are mainly interested in solving these problems, we pursue using the branching rule based on theoretical

approximation errors in what follows.

6.3 Strong branching

Instead of basing the choice of the branching variable on approximation errors, we could decide to branch on the variable that produces, by branching on it, the *largest increase in the value of the objective function* for the two generated subproblems. Doing so, we hope to cut nodes more rapidly (see Section 1.2.4). This idea, known as *strong branching*, has been developed by Applegate *et al.* [12].

The application of strong branching requires the solution of *two* linear problems for each candidate variable for branching at each iteration. In fact, strong branching simulates to branch in turn on each of these variables, and for each of them, the two resulting subproblems are solved in order to know how the value of the objective function would be modified with such a branching. In the following, we refer to this technique as the *strong branching iteration.* This branching is purely *informative* and not actual. Its goal is to obtain information about the variation in the value of the objective function, and not directly to refine the outer approximation problem. As a consequence, one does not go down in the tree during a strong branching iteration.

Strong branching aims at branching on the variable which provides the *largest increase* in the value of the objective function for the two subproblems generated by branching. However, the variation in the value of the objective function usually differs for the right and left subproblems. Therefore, to choose the branching variable, the obtained values for both subproblems are combined through a *score function* (Eckstein [36]) defined by:

$$\text{score}(x, y) = (1 - \mu)\min(x, y) + \mu \max(x, y), \tag{6.5}$$

where $\mu \in [0, 1]$, is a weighting parameter. Since, in our case, the score function is applied to the variations of f for the left and right subproblems, the strong branching technique computes, for each variable x_i candidate for branching:

$$s_i = \text{score}(f_i^+ - f, f_i^- - f), \tag{6.6}$$

where:

- f is the solution of the linear outer approximation problem before branching,
- f_i^+ is the solution of the right subproblem obtained after branching on x_i,
- f_i^- is the solution of the left subproblem obtained after branching on x_i.

As we want to increase as much as possible the value of f in order to fathom nodes, we choose to branch on the variable x_j producing the largest change in the objective function value, that is, the variable x_j such that:

$$s_{max} = s_j = \max_i s_i.$$

Note that if the maximum score function is reached for several variables, the branching variable is, among these variables, the first one which has been examined. Therefore, it corresponds to the one with the smallest index i since the variables are treated in the order of their indices.

The motivation of using a score function is based on the fact mentioned by Linderoth and Savelsbergh in [75] that it is better to branch on a variable producing a good increase in f for the two resulting subproblems instead of on a variable which brings an important increase for one subproblem and to a weak (or no) increase for the other. Achterberg *et al.* [6] have proposed to use a weighting parameter μ equal to $\frac{1}{6}$. Our experiments with different values of μ have confirmed that it is a good choice.

After having branched on x_j, the first subproblem (left or right) to be examined is the one with the most promising value for the objective function. It thus corresponds to the subproblem for which f_i^+ or f_i^- is the smallest, that is, the subproblem for which the value of the objective function is minimized.

This strong branching technique has been applied to our test problems, but the method did not converge on eleven of the *TVC* problems (continuous and discrete cases) before the maximum number of linear problems allowed to solve was reached. This can be explained by the fact that with strong branching, we try to increase the value of the objective function but without taking the quality of the outer approximation problem into account. Therefore, it arises that we try to increase the value of the objective function of an outer approximation problem which is quite far from the original one. A possible way to improve the convergence would be to also branch to refine the quality of the outer approximation problem.

6.3.1 Strong branching based on the quality of the outer approximation problem

Since our goal in branching is not only to increase the value of the objective function but also to make the outer approximation problem closer to the nonlinear one, the improvement in the *quality of the approximation* should be also taken into account in the branching rule. As a consequence, we must first quantify the quality of an approximation. To this aim, we measure the difference (in the sense explained below) between a linear outer approximation problem and the original approximated problem. Accordingly, smaller this difference, better the associated outer approximation. In practice, the difference, denoted d, is computed as the sum of the maximum errors produced with regard to the original problem. That is, we compute d as:

$$\begin{aligned} d = & \sum_{i=0}^{m} \text{errors generated by the approximation of the constraint } g^i \\ & + \sum_{i=1}^{b} \text{errors generated by the violation of the } i^{th} \text{ integer restriction}, \end{aligned} \tag{6.7}$$

where the objective function of the nonlinear problem has been assimilated to g^0 by sake of easiness and b is the number of discrete restrictions in the nonlinear problem. In (6.7), the errors generated by the violation of an integer restriction correspond to the weighted maximum theoretical approximation errors associated to each discrete variable and detailed in Section 6.2.1. We now describe how to compute the errors generated by the approximation of a constraint used in definition (6.7). Since each constraint $g^i,\ 0 \leq i \leq m$, can be decomposed as:

$$g^i = g^i_{lin} + \sum_{k_i} L_{k_i} f_{k_i}, \tag{6.8}$$

where g^i_{lin} is the linear part of the constraint g^i, f_{k_i} are the nonlinear components involved in g^i and L_{k_i} are the coefficients of these components, we define the approximation error associated to g^i by:

$$\sum_{k_i} |L_{k_i}| e_{k_i}. \tag{6.9}$$

In this formula, the quantities e_{k_i} are the maximum *theoretical* approximation errors computed in Section 6.2.1 and generated by the outer approximations based on SOS of the nonlinear components f_{k_i}. Theoretical approximation errors are used instead of real ones because they lead to a more intuitive definition of the *quality* of the approximation. Indeed, intuitively, the quality of an approximation problem must be improved by branching since the problem is refined. As the maximum theoretical approximation errors are reduced by branching since the outer approximations are refined, the quantity (6.9) decreases for at least one index i, and thus (6.7) also decreases, which corresponds to an improvement in the quality of the approximation. To the contrary, this is not guaranteed with real approximation errors in the sense that after branching, the real approximation errors may increase and thus, increase the quantity (6.9) and thereafter (6.7), which would produce a decrease in the quality of the approximation according to our definition. Moreover, in the following, we will need to average the different variations obtained in the tree. By allowing improvements and deteriorations in the quality of the approximations, we risk to obtain completely irrelevant values. Therefore, we prefer using theoretical errors since they produce monotonous variations. Note that with such a definition for d, the value of d is decreasing when one goes down in the tree (while that of f is increasing).

Since we have determined how to quantify the quality of an approximation, we can now measure the improvement in the quality of the approximation obtained by branching. To combine these improvements for the left and right subproblems generated by branching on a variable x_i, the score function defined in (6.5) is again used:

$$t_i = \text{score}(d - d_i^+, d - d_i^-), \tag{6.10}$$

where

- d measures the quality of the linear outer approximation problem before branching,
- d_i^+ measures the quality of the right subproblem obtained after branching on x_i,
- d_i^- measures the quality of the left subproblem obtained after branching on x_i.

6.3.2 Scheme of the method

Before giving the scheme of our adaptation of strong branching, we first discuss the treatment of infeasible problems. As seen previously, strong branching is based on the variations in the objective function value and in the quality of the approximation obtained by branching. Sometimes however, these variations cannot be determined. Indeed, to compute the increase in the value of the objective function obtained by branching on a variable, we must solve two linear problems by candidate variable for branching. It can obviously happen that one or both subproblems are infeasible. If after branching on a variable x_i, the two generated subproblems are infeasible, the node for which we look for the best branching variable can be directly fathomed.

Conversely, if only one subproblem is infeasible, the interval of x_i associated to the infeasible subproblem can be discarded and as a consequence, the bounds on x_i can be tightened for the node for which we look for the best branching variable. In these cases, *strong branching thus allows us to reduce the range of the variables and thus, to refine the outer approximation problem*. Therefore, the interest of strong branching is not only to choose the branching variable but also to strengthen the outer approximation problem.

When one problem is found infeasible after branching on x_i during a strong branching iteration, the determination of the variation in f generated by this branching is not possible because the value f_i^- or f_i^+ (depending if it is the left or the right subproblem which is infeasible) is not available to evaluate the score function (6.6). However, as one subproblem has been discarded because it was infeasible, the range of the variable x_i can be tightened to be equal to the range obtained for the variable x_i for the other subproblem, the feasible one. Consequently, this is only the variation associated to this subproblem that interests us. In this case, we set the score function to this variation without combining with another variation computed for the infeasible problem, that is, we set $s_i = f_i^+ - f$ if it is the left subproblem which is infeasible and $s_i = f_i^- - f$ if it is the right one.

After having explained some features about strong branching applied to outer approximations, we now summarize the process of this technique. *Two goals*, which must be combined[(2)], are taken into account in the branching rule: increase the value of the objective function and improve the quality of the outer approximation problem with respect to the nonlinear one. Note that our numerical experiments have shown that it is better to give priority to the increase of f. Therefore, we choose to base our choice on the objective function as often as possible and try to improve the quality of the approximation only when the branching based on the objective function is expected to produce a small (or no) increase in f, or to be irrelevant because only a few number of score functions are nonzero, which can be an indication that the outer approximation problem is not close enough to the nonlinear one since its feasible domain is too large.

In practice, if the value of the maximum score function associated to the increase in f is smaller than a fixed quantity ϵ_1 or if the proportion of zero score functions is larger than another fixed parameter, denoted ϵ_2, we try to improve the quality of the outer approximation problem. However, unless the proportion of zero score functions is larger than the parameter ϵ_2, we do not branch twice consecutively to improve the quality of the outer approximations in order to avoid to refine too much an approximation problem that could possibly be discarded if the value of f would a little bit increase. Indeed, in some cases, small increases in the value of the objective function can even though help to cut nodes. More schematically, we apply Algorithm 6.2 presented below. Note that this algorithm is well defined since, if the proportion of score functions s_i equal to zero is too large, it is always possible to find a variable which allows us to improve the quality of the approximation according to our definition (6.7).

By applying Algorithm 6.2 to choose the branching variable with $\epsilon_1 = 0.01$ and $\epsilon_2 = 0.75$[(3)], the method converges for all problems, even for problem *pb13*. On Figure 6.8, we compare the results obtained using strong branching (see also Table 8.15) with the ones related to the branching rule based on the weighted theoretical approximation errors (see Table 8.13) presented in Section 6.2.1.

[(2)] In this work, we have chosen to try to improve one goal or the other, but not a combination of both by means of some merit function. It would be worth to also test this alternative.

[(3)] Among the tested values for parameters ϵ_1 and ϵ_2, Algorithm 6.2 has produced the best results with these particular values.

The top figure again represents the improvement percentage defined in (5.1) obtained by using strong branching with regard to the branching rule based on the weighted theoretical approximation errors. On this figure, we observe that the strong branching method is very competitive because it improves the majority of the results and sometimes significantly. For nine problems highlighted by stars, the number of linear problems to solve is divided by more than ten with strong branching. The best improvement is obtained for *pb8* in the discrete case with a number of linear problems to solve divided by 44 (see Tables 8.13 and 8.15 for more details). These results explain the particular performance profiles of Figure 6.8: since strong branching is better on a larger number of problems and since it generates improvements in the results notably larger than the deteriorations that it produces, the curve representing its performance nearly merges with the axis $\sigma = 1$ and thus, cannot be seen on the figure.

Algorithm 6.2: Strong branching: choice of branching variable

Set $C = \{i \mid x_i \text{ is candidate for branching}\}$,
$\epsilon_1, \epsilon_2 =$ fixed real parameters.

1. For each variable x_i ($i \in C$), *solve the two linear problems* generated by branching on it and *compute the associated score functions* s_i by (6.6).
   Set s_{max} the maximum score function and x_j the variable for which the maximum is reached. If there exist several variables x_j for which the score function is maximized, choose among them the one with the smallest index j.
2. **If** $\big((s_{max} < \epsilon_1$ and the branching for the parent node did not aim at improving the quality of the approximation) or (the proportion of zero score functions $s_i > \epsilon_2)\big)$ **then**
   For all $i \in C$, *compute the score functions* t_i defined in (6.10) and based on the improvement of the quality of the approximation.
   Choose the variable x_j associated to the maximum score function t_j.
   If there exist several variables x_j producing the maximum value for this score function, choose among them the one with the smallest index j.
3. *Branch* on x_j.

The substantial improvements can be justified by the fact that, for some problems, better results can be obtained by mainly branching on variables which rapidly increase the value of the objective function. Strong branching can detect these variables and branch on them. Methods based on approximation errors do not branch so often on such variables because, with repeated branching on them, the associated approximation errors become too small with regard to the errors generated by the other variables, which are, as a consequence, chosen for branching. Strong branching is also particularly efficient since it allows us to discard some parts of the feasible domain, as explained above. Especially for some problems *pb*, it reduces a lot the feasible domain in the first levels of the tree, and thus, avoids to branch numerous times in the tree to discard infeasible nodes. Indeed, earlier the outer approximation problem can be refined in the tree, smaller the size of the branching tree.

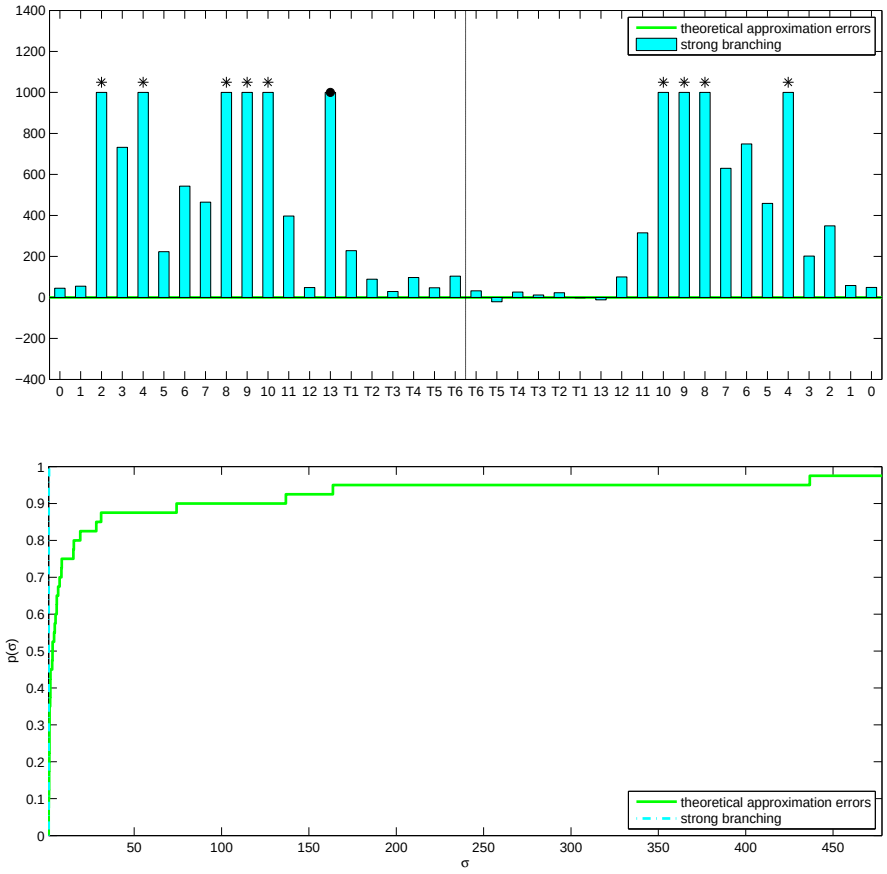


Figure 6.8: Comparison of results obtained with strong branching with regard to the branching rule based on the weighted theoretical approximation errors.

However, strong branching can also deteriorate the results. This is the case for two of the *TVC* problems in the discrete case (*TVC1, TVC5*) and also for *pb13*. This can be explained by the cost of strong branching which is relatively important. Indeed, strong branching needs to solve two linear problems for each candidate variable for branching at each iteration. On our problems, the number of linear problems solved during the strong branching iterations is generally larger than the half of the total number of linear problems solved (see the sixth column of Table 8.15 which gives the percentage of linear problems solved during strong branching iterations). By comparing the cost of using strong branching with the one of the approximation error method, it is obvious that strong branching is more expensive because it requires the solution of linear problems while the branching rule based on theoretical approximation errors only needs to compute approximation errors by applying the formula or theorem detailed in Section 6.2.1. Nevertheless, the use of strong branching generally allows us to decrease the *total number* of linear problems to solve with regard to the other branching rule. Therefore, the cost of using strong branching can be amortized since the cost associated to the solution of a linear problem for a general iteration or during a strong branching iteration can be considered to be similar.

Note furthermore that no nonlinear problem is associated to the linear problems solved during strong branching iterations. Since the use of strong branching reduces the number of general

iterations, the number of nonlinear problems to solve also decreases. For all the reasons mentioned above, strong branching can be validated with regard to the approximation error methods. However, in order to reduce its cost, another strategy is investigated.

6.4 Pseudocosts

An alternative of strong branching called *pseudocosts*, has been developed by Bénichou *et al.* [23]. The goal of pseudocosts is to collect information similar to the one obtained by strong branching, but without systematically computing a sequence of linear problems. In fact, pseudocosts try to extract information from linear problems *solved* during the general branch-and-bound process. This information is stored and used in order to *predict* the increase in f obtained by branching on a particular variable. For each candidate variable for branching, pseudocosts scale, that is, compute by unity of reduction in the range of the variable, the increase in the value of the objective function (for the left and right subproblems) that one can expect by branching on it. Such a scaling allows us to predict the increase in f everywhere in the tree and whatever the range of the variable is. Indeed, for a same variable, if we branch at the middle of a small interval, the increase in f is expected to be smaller than if we branch at the middle of a large interval. Scaling pseudocosts allows us to take this dependence on the range of the variables into account. To predict the increase in f for a subproblem (left or right) obtained after branching on a variable, the pseudocosts associated to this variable are multiplied by the range of the variable discarded by this subproblem.

Since the increase in f is generally different for the left and right subproblems, two kinds of pseudocosts are used for each candidate variable for branching: the *up pseudocosts* which are associated to the right subproblems and the *down pseudocosts* which are associated to the left ones. Let us focus to the up pseudocosts which are indexed by $+$, to distinguish them from the down pseudocosts, indexed by $-$. According to what precedes, and in order that the definition of pseudocosts is also applicable for continuous variables, we define the up pseudocosts associated to a variable x_i by:

$$\zeta_i^+ = \frac{f_i^+ - f}{m_{x_i} - l_{x_i}}, \tag{6.11}$$

where:

- f is the value of the linear outer approximation problem before branching on x_i. At this level, x_i belongs to $[l_{x_i}, u_{x_i}]$,
- f_i^+ is the value of the right subproblem obtained after branching on x_i. At this level, x_i belongs to $[m_{x_i}, u_{x_i}]$,
- m_{x_i} is the value where we branch on x_i when $x_i \in [l_{x_i}, u_{x_i}]$,
- $m_{x_i} - l_{x_i}$ corresponds to the range of the variable x_i discarded by branching.

The value (6.11) must be still multiplied by the length of the part of the domain discarded by the right subproblem to predict the increase in the value of f. Note that to compute the up pseudocost, we divide the variation in the value of the objective function obtained for the *right* subproblem by the size of the interval remaining for the *left* subproblem, that is, by the length

of the interval which has been discarded. Indeed, this is the size of the discarded interval which has the largest impact on the variation of the objective value. So, the variation in the value of the objective function for two subproblems can be expected to be similar if parts of same size of the range of the same branching variable are discarded. This is not the case if these are the two remaining intervals which have the same size because if for one problem, the discarded interval is a lot larger than for the other one, the variation in the objective function is expected to be larger for the first one.

Each time new information is available, pseudocosts are *updated* and *averaged* in order to be as close as possible to the values that they predict. To this goal, we index by k the pseudocosts ζ_i^+ and all the quantities involved in (6.11) to mean that they are related to the k^{th} branching on x_i. Therefore, (6.11) becomes:

$$\zeta_{i,k}^+ = \frac{f_{i,k}^+ - f_k}{m_{x_{i,k}} - l_{x_{i,k}}}.$$

We also introduce η_i^+, the number of times that we have branched on the variable x_i, solved the associated right subproblem and that this latter was feasible. We then evaluate the sum σ_i^+ of all up pseudocosts computed for the variable x_i:

$$\sigma_i^+ = \sum_{k=1}^{\eta_i^+} \zeta_{i,k}^+. \tag{6.12}$$

By dividing this sum by η_i^+, we obtain Ψ_i^+, the *averaged up pseudocosts* associated to the variable x_i:

$$\Psi_i^+ = \frac{\sigma_i^+}{\eta_i^+}. \tag{6.13}$$

Finally, the predicted increase in the value of f for the right subproblem obtained by branching on the variable $x_i \in [l_{x_i}, u_{x_i}]$ at point m_{x_i}, can be obtained by multiplying the averaged up pseudocost by the range of the variable x_i discarded by the right subproblem, that is, $m_{x_i} - l_{x_i}$. Mathematically, the predicted increase is equal to:

$$(m_{x_i} - l_{x_i})\Psi_i^+.$$

Similar definitions can be established for the down pseudocost $\zeta_{i,k}^-$, and for η_i^-, σ_i^- and Ψ_i^-.

As for strong branching, the predicted increases for the left and right subproblems must be combined. Again, the score function (6.5) is used to this aim. For each variable x_i candidate for branching at point m_{x_i} ($x_i \in [l_{x_i}, u_{x_i}]$), the score function associated to pseudocosts Ψ_i^+ and Ψ_i^- and denoted ps_i is computed as:

$$ps_i = \text{score}\left((m_{x_i} - l_{x_i})\Psi_i^+, (u_{x_i} - m_{x_i})\Psi_i^-\right). \tag{6.14}$$

We then choose to branch on the variable x_j such that:

$$ps_{max} = ps_j = \max_i ps_i.$$

Once the branching variable has been selected, the next subproblem (left or right) to be examined is the one with the most promising pseudocost, that is, the one which predicts the

smallest increase in f. Here, it corresponds to the subproblem associated to the smallest quantity between $(m_{x_i} - l_{x_i})\Psi_i^+$ and $(u_{x_i} - m_{x_i})\Psi_i^-$.

As for strong branching iterations, we also want to improve the quality of the outer approximation problem. Therefore, we define pseudocosts associated to this quality. These parameters, denoted Φ_i, are scaled exactly as Ψ_i but are defined for $d_k - d_{i,k}^+$ and $d_k - d_{i,k}^-$, where the index k means that these values are associated to the k^{th} branching on the variable x_i, d_k measures the quality of the approximation problem before the k^{th} branching, as defined in (6.7), while $d_{i,k}^+$ (respectively $d_{i,k}^-$) measures the quality of the right (respectively left) subproblem obtained after the k^{th} branching on x_i. By assuming that after this branching, the variable x_i belongs to the interval $[m_{x_{i,k}}, u_{x_{i,k}}]$ for the right subproblem, we thus need to compute for the up pseudocost:

$$\beta_{i,k}^+ = \frac{d_k - d_{i,k}^+}{m_{x_{i,k}} - l_{x_{i,k}}}, \tag{6.15}$$

$$\theta_i^+ = \sum_{k=1}^{\delta_i^+} \beta_{i,k}^+, \tag{6.16}$$

$$\Phi_i^+ = \frac{\theta_i^+}{\delta_i^+}. \tag{6.17}$$

In this formulation, δ_i^+ corresponds to the number of times that we have branched on the variable x_i and solved the associated right subproblem (even if this problem was infeasible, contrary to η_i^+). Again, we can give similar definitions for $\beta_{i,k}^-$, δ_i^-, θ_i^- and Φ_i^- and also combine averaged pseudocosts Φ_i^+ and Φ_i^- through a score function.

6.4.1 Handling of pseudocosts

Update of pseudocosts

When a linear outer approximation problem $(\widetilde{OP})$ defined in Section 3.2.5 is solved during the process of the algorithm, the quantities Ψ_i^- and Φ_i^- or Ψ_i^+ and Φ_i^+ (depending if we examine a left or a right child) associated to the variable x_i on which we have branched before are updated. In order to update the averaged up pseudocost Ψ_i^+ for the $\eta_i^+ + 1$ times, from the current pseudocost denoted $\Psi_{i,\eta_i^+}^+$ and from the new pseudocost $\zeta_{i,\eta_i^+ + 1}$, the following formula is employed:

$$\Psi_{i,\eta_i^+ + 1}^+ = \frac{\eta_i^+ \Psi_{i,\eta_i^+}^+ + \zeta_{i,\eta_i^+ + 1}}{\eta_i^+ + 1}.$$

Similar formula are employed to update Ψ_i^-, Φ_i^+ and Φ_i^-.

Initialization of pseudocosts

The pseudocosts are thus updated as soon as a linear outer approximation problem $(\widetilde{OP})$ is solved. However, at the top of the tree, we do not have any information to initialize pseudocosts, which prevents from choosing the branching variable correctly. The initialization of the parameters Ψ_i [(4)] is thus a crucial phase. To this goal, a *strong branching iteration* is performed

[(4)]Note that for the sake of clarity, we use Ψ_i to refer to as Ψ_i^- as well as Ψ_i^+. A similar notation Φ_i is also employed for Φ_i^- and Φ_i^+.

as suggested by Linderoth and Savelsbergh in [75]. Sometimes, however, this is not enough to obtain relevant pseudocosts Ψ_i associated to the objective function. Indeed, as explained for strong branching, we can have situations where the value of the objective function is not modified by branching on a variable x_i and, as a consequence, the associated pseudocosts vanish. For pseudocosts Φ_i, such a problem does not arise, because they depend on the quality of the approximation (6.7) which is based on the theoretical approximation errors. Since at least one of these errors is reduced by branching, the pseudocosts Φ_i cannot vanish, which is not the case for the pseudocosts Ψ_i associated to the increase in f. If the pseudocost Ψ_i associated to the variable x_i is equal to zero, this variable is never chosen to branch on when the branching rule is based on the increase in the value of the objective function. Indeed, the variable associated to the largest pseudocost is chosen for branching. However, by using branch-and-bound, the outer approximation problem is refined and thus modified. Therefore, it happens that a variable which could not produce a modification in the value of the objective function at the top of the tree (and thus, associated to an initial zero pseudocost) has more influence in a deeper level of the tree and can lead to an important increase in the value of the objective function. Accordingly, branching on such a variable is useful.

To this aim, during the process of the algorithm, we try to *update the zero pseudocosts* Ψ_i. More precisely, for each variable x_i candidate for branching and associated to a zero pseudocost Ψ_i, two linear problems are solved, as in strong branching, to know the modification in the value of f obtained by branching. We refer to this step as *reduced strong branching iteration* since it implies a limited number of variables. Nevertheless, it would be too expensive to do this for every general iteration. Accordingly, in practice, a reduced strong branching iteration is performed after $4Z$ [(5)] general iterations, where Z is the number of reduced strong branching iterations already realized. The call to reduced strong branching iterations is thus more and more spaced in order not to repeat useless iterations to update pseudocosts on some variables which will never allow us to increase the value of the objective function by branching on them. We also take care to avoid to repeat reduced strong branching iterations for two consecutive levels (parent and child nodes) on the same branch because it can be expected that, after branching only once, the problem will not have changed enough to obtain a nonzero value for the zero pseudocosts. Therefore, in practice, we require to go down of three[(6)] levels before repeating a reduced strong branching iteration on a same branch. Note that the goal of these reduced strong branching iterations is only to update the zero pseudocosts and not to choose the branching variable like in strong branching since we have real information for a limited number of variables only.

Case of infeasible problems

When a linear outer approximation problem $(\widetilde{OP})$ is detected infeasible during a general iteration of Algorithm 3.2 or 3.3, the update (or initialization) of the associated pseudocost Ψ_i^+ or Ψ_i^- is impossible because there is no optimum value f_i^+ or f_i^- for the linear problem to update the pseudocosts. As a consequence, the pseudocost linked to an infeasible problem remains unchanged. Note that we do not increase the value of such a pseudocost, although that the associated variable seems to be a good candidate for branching since it has allowed us to cut a part of the domain, and this, in order to avoid to give too much weight to the associated variable.

[(5)] Among the different tested values, this one has produced the best results.

[(6)] This value of parameter seems to be a good choice according to our numerical experiments.

Indeed, it is possible that this variable produces a very small variation for the other subproblem generated by branching and that the associated node cannot be rapidly cut. Therefore, branching on it is not so desirable.

6.4.2 Scheme of the method

All the steps detailed before are now combined to give the method based on pseudocosts to choose the branching variable. Pseudocosts are first initialized by doing a strong branching iteration. At this stage, the branching variable is chosen exactly as for strong branching because the same (and correct) information is available. Next, the choice of the branching variable is based on pseudocosts unless more than a fixed proportion of pseudocosts, given by ϵ_2, is equal to zero. In this case, we consider that we do not have a good representation of the problem and the choice of branching variable is still based on strong branching. Strong branching iterations are then performed with the hope of updating the different pseudocosts quickly. When a complete strong branching iteration is realized, the choice of branching variable is based on the obtained information, as detailed in Algorithm 6.2. Moreover, as long as *all* pseudocosts have not a nonzero value, we try to update them by doing reduced strong branching iterations in the way and for the reasons detailed earlier.

As for strong branching, branching is performed to increase the value of the objective function but also to improve the quality of the linear outer approximation problem with regard to the original one. If the maximum score function (6.14) related to the predicted increase in f is larger than a fixed accuracy ϵ_1, the branching is performed to increase the value of the objective function. Otherwise, branching is realized to improve the quality of the approximation, unless the branching on the parent node was already based on the quality of the approximation, exactly like for strong branching. The method to choose the branching variable is schematically presented in Algorithm 6.3.

Figure 6.9 compares the results obtained using strong branching with these obtained by applying Algorithm 6.3 with $\epsilon_1 = 0.01$ and $\epsilon_2 = 0.5$[(7)]. By analyzing these graphs, we can see that the best method with regard to the number of linear problems solved is problem dependent since the best results are shared between the two methods. However, it appears on the performance profiles of Figure 6.9 that strong branching is more efficient than pseudocosts since it gives the best results on 26 problems (instead of on 16 problems) and that the deteriorations produced by using pseudocosts are larger than the improvements that they generate.

These results can be justified by the fact that pseudocosts try to predict the information computed by strong branching. Sometimes it works, but it happens that pseudocosts do not give relevant information and that the number of linear problems to solve increases with regard to strong branching. For example, if a pseudocost has a very small nonzero value at its initialization, we branch on it very far in the tree. But this variable could possibly have more importance from some level where the outer approximation problem has been refined and branching on it would be relevant. The difficulty comes from the fact that the variations in the value of the objective function obtained by branching on a variable (scaled by unity of range reduction) can be different depending on the level of the tree and the refinement of the linear outer approximation

[(7)] According to our numerical experiments, the value of these parameters can be considered as adapted to our algorithm.

Algorithm 6.3: Pseudocosts: choice of branching variable

Set $C = \{i \mid x_i \text{ is candidate for branching}\}$,
$\epsilon_1, \epsilon_2 =$ fixed real parameters.

0. Do a strong branching iteration to *initialize* pseudocosts.
   Choose the branching variable as in Algorithm 6.2 and branch on it.
   Set to 1 the index Z of phases to update zero pseudocosts and set to 1 the number $laststrong$ of the last strong branching iteration (reduced or not). Set to 1 the index k of the current iteration.
   Choose the next subproblem to solve between the two generated ones.
1. Set $k = k + 1$. *Solve* the linear outer approximation subproblem.
   If (this problem is infeasible) **then go to 3**
2. *Update* Ψ_i^+ and Φ_i^+ like in (6.13) and (6.17), or Ψ_i^- and Φ_i^- depending if a right or a left child is examined, with the information obtained after solving the linear outer approximation problem.
   Set K= $\#\{i \in C \mid \Psi_i^+ > 0 \text{ and } \Psi_i^- > 0\}$ and J= $\#\{i \in C \mid \Psi_i^+ > 0 \text{ or } \Psi_i^- > 0\}$.
3. Choose a problem not yet explored. **If** (all the tree has been explored) **then stop**.
   If (the selected node is a node to solve (see Section 3.3.2)) **then**
   go to 1 since the branching variable has already been chosen.
4. **If** $(J < \epsilon_2 \times \#C)$ **then**
   Do a *strong branching iteration*, choose the branching variable x_j as in detailed in Algorithm 6.2, set $laststrong = k$, *update pseudocosts* with the obtained information and **go to 6**
   else
   Base the score functions on pseudocosts
5. *Choose the variable* x_j which maximizes the score functions (6.14).
   Set ps_{max} the maximum score function.
   If $(($$ps_{max} < \epsilon_1$) and (the choice of the branching variable for the parent node was not based on the quality of the approximation problem)$)$ **then**
   Base the score functions on the quality of the approximation
   and *choose the variable* x_j which maximizes these score functions.
6. *Branch on* x_j, create two subproblems, choose one of them to solve and put the other one in the stack of nodes to treat.
7. **If** $\big($$(K \neq \#C)$ and $(k - laststrong \geq 4 \times Z)$ and (the number of levels on this branch before the last strong branching iteration (reduced or not) is larger than 3)$\big)$**then**
   Do a strong branching iteration for each variable x_i such that $i \notin K$,
   set $Z = Z + 1$ and $laststrong = k$ and **go to 1**

problem. Pseudocosts cannot really take this into account, since the same scaled estimation of the increase in the objective function by branching on a particular variable is used, as long

as a new problem is not solved after branching on this variable. Strong branching avoids this difficulty because it computes the real variation at each iteration. Therefore, it always gives a relevant information everywhere in the tree. But the quality of this information has a cost.

Nevertheless, on some *TVC* problems (especially in the discrete case), pseudocosts improve the results. As a consequence, we cannot directly reject this method. The advantage of pseudocosts on strong branching for these problems must be even though put into perspective. Indeed, strong branching iterations reduce the ratio between the number of nonlinear and of linear problems solved. Therefore, the number of nonlinear problems solved with pseudocosts is generally larger (see Tables 8.15 and 8.16) than with strong branching. However, if we do not consider the impact of the different branching rules on the method, the cost of using pseudocosts is smaller than the one related to strong branching since it does not need to solve at each iteration two linear problems for each candidate variable for branching.

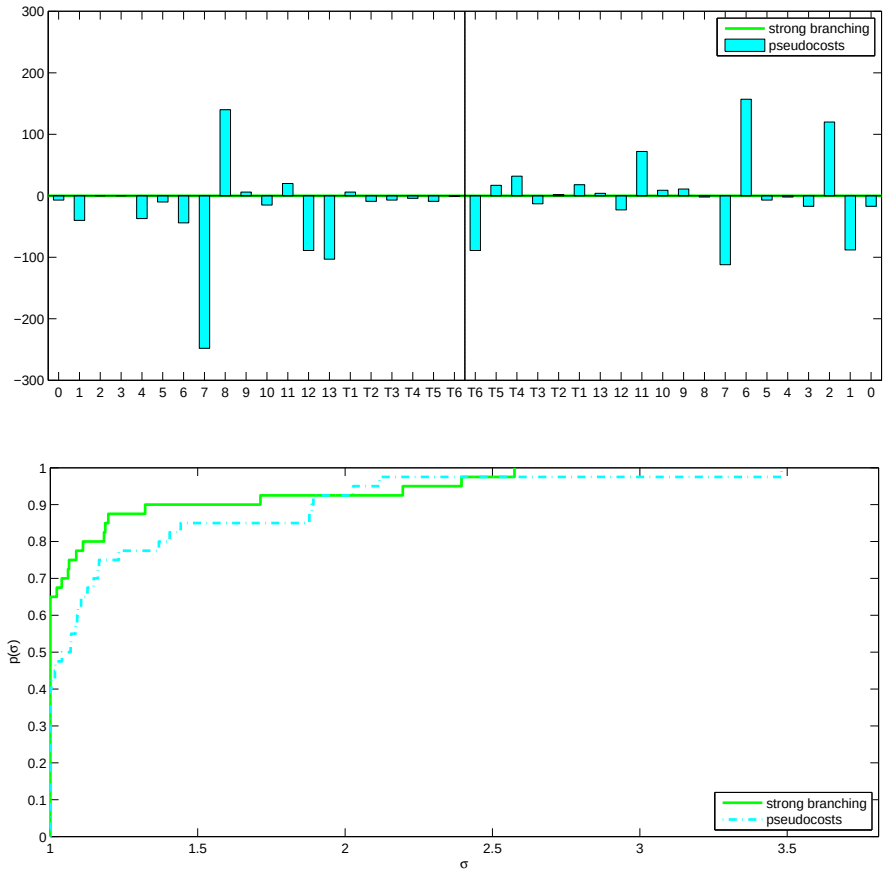


Figure 6.9: Comparison of results obtained by using pseudocosts with regard to strong branching.

6.5 Pseudocost technique with strong branching iterations

The idea of this section is to take advantage of strong branching (relevant information) but also of pseudocosts (cheaper technique), by combining both methods. To have more realistic pseudocosts at our disposal, we decide to do strong branching iterations for the four[(8)] first levels of the tree. Indeed, the choices at the top of the branch-and-bound tree are the most important. Therefore, we choose to select there the branching variable as better as possible even if it is expensive. This step also allows us to initialize pseudocosts on a more extensive basis than by doing only one strong branching iteration. During the process of the method, in order to keep realistic pseudocosts, we also repeat strong branching iterations for each candidate variable for branching every k levels in the tree. In practice, we have chosen $k = 4$[(9)]. In this way, pseudocosts are regularly updated for all variables, which allows us to obtain a better information on the variables able, or not, to increase the value of the objective function by branching on them. Furthermore, these strong branching iterations can sometimes cut some parts of the feasible domain and, thus, refine the outer approximation problem earlier in the tree than if we had waited to branch actually on the variables allowing us to discard these parts of the domain.

When a strong branching iteration is performed, the branching variable is obviously chosen following the information given by strong branching and pseudocosts are updated and averaged with the new values obtained during the strong branching iteration[(10)]. In this way, pseudocosts are expected to be more consistent with the real variations. Note that, with this method, the call to reduced strong branching iterations to update the zero pseudocosts (step 7 of Algorithm 6.3) can be removed as strong branching iterations are regularly repeated during the process of the algorithm.

The results obtained with this method are shown on Figure 6.10 and compared with results due to strong branching and pseudocosts. Results given by the combination of these two methods are generally intermediate between results obtained with strong branching and pseudocosts (see Tables 8.15, 8.16 and 8.17 for more details). Sometimes the improvement is larger and the results are better than for the two previous methods (for example, in the continuous case, *pb0*, *pb10*, *TVC1*, *TVC2*, *TVC3*, *TVC5* and *TVC6*). In these cases, we have benefited from advantages of both methods. We think that the proposed method is a good compromise between strong branching and pseudocosts. However, the combination of both methods can also give results which are worse than the two combined methods separately taken: *pb13*, *TVC1* and *TVC2* for the discrete case, for instance. Such results can be explained by the fact that the two methods separately taken can give good results. By adding strong branching iterations to pseudocosts, the number of linear problems to solve increases but the improvement provided by these iterations is not sufficient to balance their cost. Moreover, for some problems, strong branching can cut numerous parts of the feasible domain. This allows us to reduce the feasible domain early in the tree. By adding strong branching iterations to pseudocosts, we hope to obtain the same effect. But, as the strong branching iterations are spaced in this case, the outer approximation problems cannot be refined so early and we sometimes do expensive strong branching iterations too deeply in the tree. This justifies the few deteriorations of the results.

According to the performance profiles of Figure 6.10, the combination of strong branch-

[(8)] This parameter has provided the best results among the tested values.

[(9)] This particular value has been selected among the different tested values since it has produced the best results.

[(10)] Another possibility would consist to reset the pseudocosts with the values obtained during the strong branching iteration.

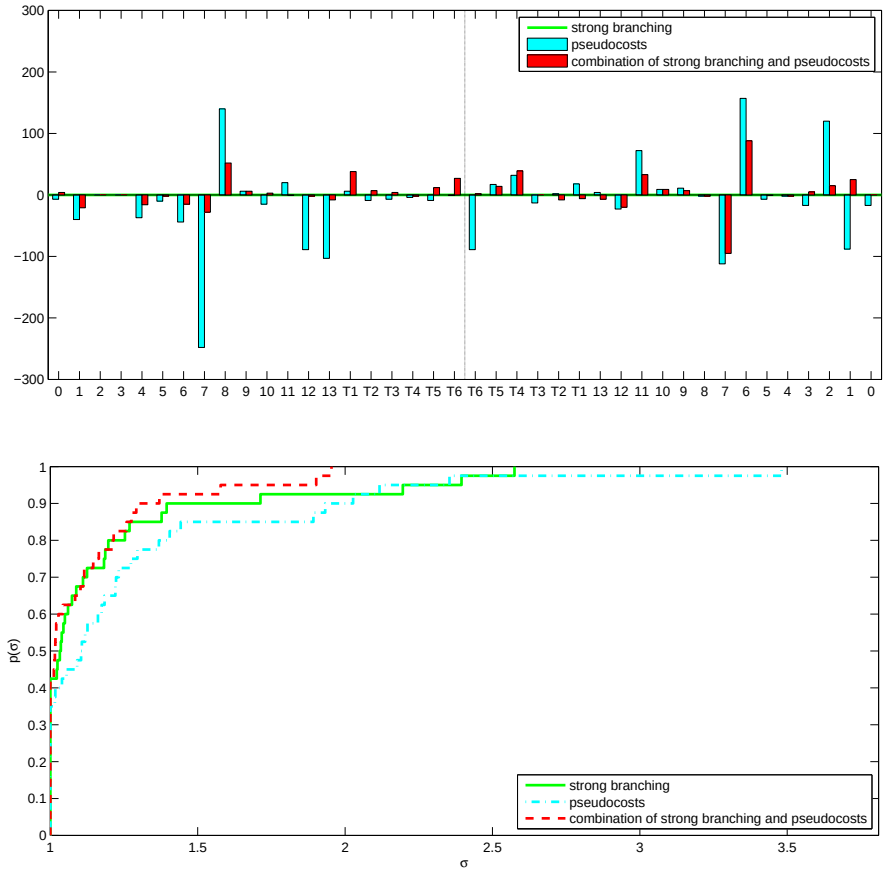


Figure 6.10: Comparison of results obtained by using strong branching, pseudocosts and pseudocosts combined with strong branching.

ing and pseudocosts is better than the two previous ones since it generally corresponds to the higher curve. However, the difference with strong branching is weak. We now compare the strong branching technique with the combination of strong branching and pseudocosts since they are the two best methods. The performance profiles related to this comparison are given in Figure 6.11. They show that the combination of both methods is more efficient than strong branching while this latter technique produces the better improvement generated in the results (problem *pb7* in the discrete case). On this figure, we also compare the improvement percentage obtained with the combination of both methods with regard to strong branching. In fact, it is the same figure as Figure 6.10 in which the results related to pseudocosts have been removed. Since, in addition, the combination of both method gives the best results on the largest problems (on the center of the graph), and notably, on *TVC* problems (eight improvements, one equality and three deteriorations), we validate the combination of pseudocosts and strong branching. Note however that the ratio of nonlinear problems and of linear ones is still smaller for strong branching than for the method which combines it with pseudocosts. But for the treated problems, the cost of the nonlinear problems with regard to the linear outer approximation ones is not high enough to change the conclusions on the largest problems if we take the number of nonlinear problems solved into account. We thus prefer the branching rule combining pseudocosts and

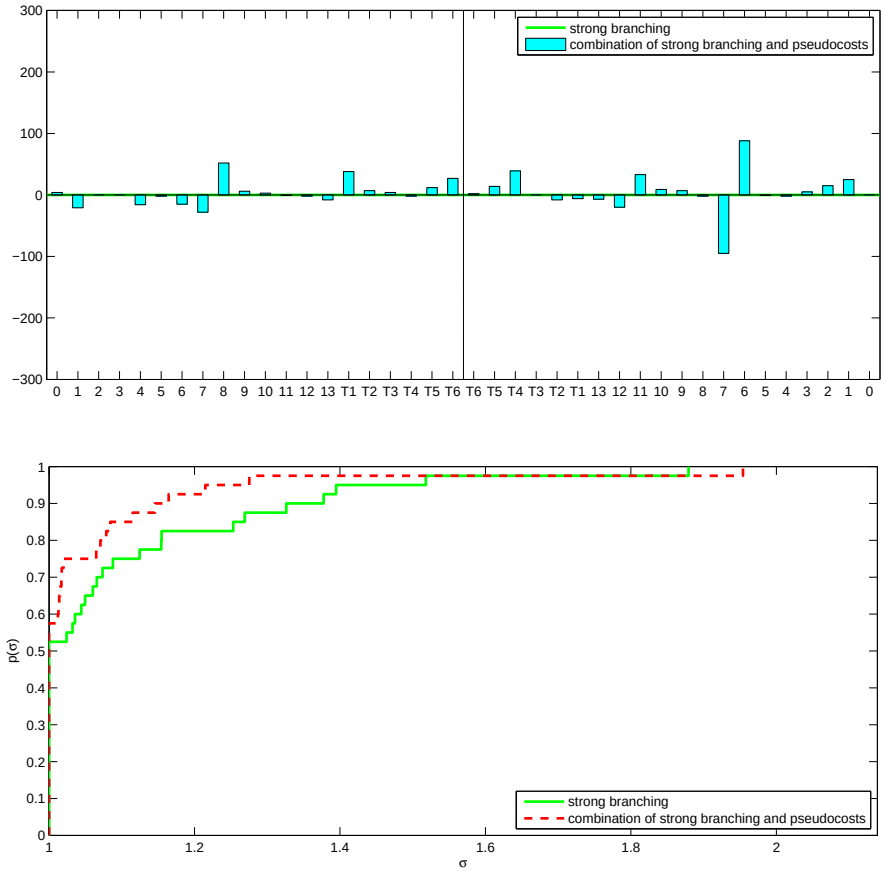


Figure 6.11: Comparison of results obtained by using strong branching and pseudocosts combined with strong branching.

strong branching instead of the strong branching technique although these two techniques do not give results dramatically different.

As the proposed technique is less good than strong branching on some problems, we can wonder if the conclusions obtained in Section 6.3.2 with regard to the branching rule based on the weighted theoretical approximation errors remain valid. The answer is obvious by considering Figure 6.12 which compares the technique proposed in the present section with the branching rule based on the weighted theoretical approximation errors. The top figure gives the improvement percentage in the number of linear problems to solve obtained with the combination of strong branching and pseudocosts. For a quarter of the problems, this latter method divides by more than ten the number of linear problems to solve, as highlighted by stars. However, on *TVC1*, *TVC5* and *pb13* in the discrete case, the branching rule based on the theoretical approximation errors is better than the last proposed technique, like for strong branching. Note nevertheless that the deteriorations produced by the method combining strong branching and pseudocosts are small with regard to the improvements that it generates. Moreover, the bottom figure which represents the performance profiles is similar to the one of Figure 6.8: the curve associated to the combination of strong branching and pseudocosts nearly merges with the axis $\sigma = 1$, and thus, cannot be seen on Figure 6.12. This is an indication that the proposed tech-

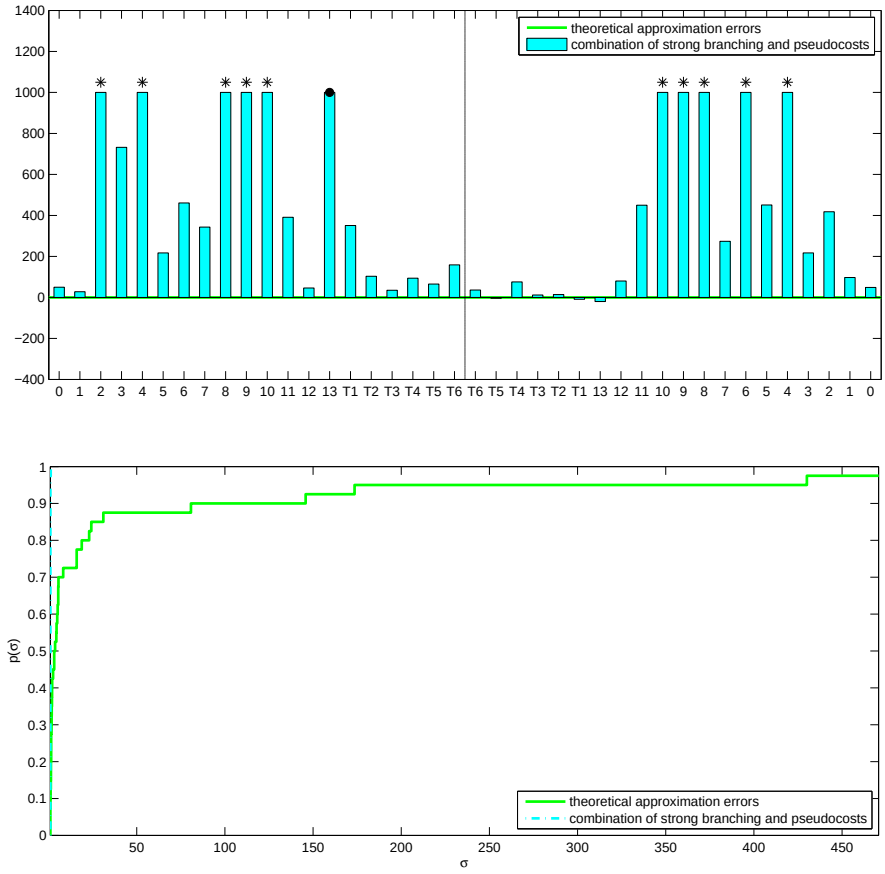


Figure 6.12: Comparison of results by using pseudocosts combined with strong branching iterations and theoretical approximation error branching.

nique is better than the method based on theoretical approximation errors. Therefore, the basic method is modified by basing the branching rule on the combination of pseudocosts with strong branching.

6.6 Conclusion

In this chapter, several branching rules have been tested. We have begun by highlighting that the results could be improved by branching again on a variable which has previously allowed us, by branching on it, to fathom the last nodes that have been cut, instead of branching on the variable selected by the usual branching rule. Then, we have shown that the branching rules based on the approximation errors are better than the simple branching rule based on the largest range of the variables. Indeed, they allow us to solve all problems *TVC*. For these methods, the interest of weighting the variables to take the knowledge of the problem into account has been underlined. The best approximation error branching method (based on theoretical or on real approximation errors) depends on the problem, but we prefer theoretical errors because they produce slightly better results for the largest problems.

However, these rules are themselves not as good as the branching rules based on the increase in the value of the objective function, like strong branching or pseudocosts. These techniques developed in mixed integer programming have been adapted to be efficient on our outer approximation problems. Indeed, in case of outer approximation, it is also desirable to make the linear outer approximation problem closer to the original one in addition to increase the value of the objective function. Therefore, this increase is no longer the only criterion to choose the variable. In some cases, branching to try to improve the quality of the outer approximation problem with regard to the original problem is preferred. Strong branching updated in this way has produced good results. The success of this method to solve outer approximation problems can be explained by the fact that it allows us not only to choose a good branching variable but also, in some cases, to reduce the range of the variables. Therefore, strong branching refines the outer approximation problem and can also be seen as a strategy of range reduction. Moreover, using strong branching iterations allows us to reduce the number of general iterations, and therefore, the number of nonlinear problems solved. Since the cost of strong branching is significant, a technique based on pseudocosts has also been developed. We have shown that the initialization and update of pseudocosts is not as easy as in the integer case. Indeed, the increase in the value of the objective function can often vanish by branching on continuous variables and these increases computed by unity of change in the variables can be very different depending on the depth of the tree and the degree of refinement of the outer approximation problem.

To work with pseudocosts as realistic as possible, we have combined pseudocosts with strong branching iterations. The goal of this technique is twofold since it amounts to reduce the cost of strong branching while trying to obtain realistic pseudocosts by exploiting as much as possible the information given by strong branching. The combination of pseudocosts and strong branching seems to be a good compromise between both methods. As this technique gives the best results on the most difficult problems, we think that it can be considered as an interesting alternative and we adopt it for the following.

Chapter 7

Node selection

This third chapter about choices related to a branch-and-bound process is concerned with the *node selection* that determines which node must be treated next. Like the issues studied in the two previous chapters, the node selection is very important since it can strongly increase the speed of convergence of the proposed method. Indeed, the way used to explore the tree allows us to find more or less rapidly a feasible solution for the nonlinear problem, and as a consequence, an *upper bound on the value of its objective function* or better, the optimum value itself. Better this upper bound, quicker nodes can be fathomed, and thus higher the speed of convergence. Therefore, it is desirable to search the tree in order to find a good upper bound on the value of the objective function as fast as possible (in the sense that it allows us to discard nodes).

For the numerical experiments presented in the previous chapters, a *depth-first search* has been used to explore the branch-and-bound tree, as explained in Section 5.1.1. This chapter now focuses on two other node selection rules: the *best-first search* and the *best-estimate* criterion adapted to our outer approximations. In Section 3.3.2, we have seen that we consider two kinds of nodes when we must choose a node to examine: the *nodes to divide* and the *nodes to solve*. The first class of nodes contains the nodes associated to subproblems that have been solved and which must be divided in two new nodes by branching. When a node to divide is selected to be treated, two new nodes are generated and one of them is solved. With the branching rule used (pseudocosts combined with strong branching iterations developed in Section 6.5), the selected node among both corresponds to the one leading to the smallest increase predicted by pseudocosts, as explained in Section 6.4. The other node which remains unsolved belongs to the second class of nodes, i.e., the nodes to solve. Both kinds of nodes belong to the set of nodes to explore, also called *open nodes*.

For the best-first search and the best-estimate criterion, quantities related to nodes are computed in order to determine the best node to examine. To compute these quantities, we only focus on the nodes to divide. Indeed, the quantities related to nodes to solve are given by the quantities computed for their parent nodes since they quantify the ability of their children to be promising nodes and in addition, no other information is available for a node to solve.

7.1 Depth-first search

In order to motivate the use of a different node selection rule, some features of the depth-first search with backtracking employed so far are recalled. With a depth-first search, one goes down in the tree as long as possible, that is, as long as the treated node cannot be cut. With backtracking, when a node is fathomed, the next node to be examined is the last one which has been generated. The set of open nodes can thus be seen as a *LIFO* (last in first out) stack since the last generated node is the first one to be treated. As a consequence, the linear problems associated to two nodes consecutively examined are generally quite close, which can be exploited to solve more rapidly the linear problems by using the solution associated to the last examined node (solution itself, Jacobian, Hessian) to solve the linear problem associated to the next treated node. Moreover, the depth-first search is also expected to quickly find feasible solutions for the nonlinear problem, and thus upper bounds on its objective value, since these solutions are generally in the depth of the tree. These are the main advantages of a depth-first search. However, if a bad direction is taken in the tree, a lot of useless iterations can be performed before finding the optimum value or even an upper bound on it. Indeed, in this case, numerous linear problems associated to nodes which would have been fathomed if a better bound had been known when they were examined can possibly be solved. This is why another node selection rule is considered.

7.2 Best-first search

The *best-first search* allows us to avoid the drawback related to a depth-first search mentioned above since it selects, among the set of open nodes, the one with the *smallest lower bound* on the value of the objective function of the nonlinear problem. We recall that, for each node of the branch-and-bound tree, the lower bound on the value of this objective function is given by the optimum value of the linear outer approximation problem solved at this node, if it is a node to divide, or at its parent, if it is a node to solve. With such a choice, no problem with a lower bound on the optimum value of the nonlinear problem larger than the actual optimum value (a priori not known) is solved. Therefore, we never treat problems associated to superfluous nodes, that is, nodes which would have been fathomed if a better upper bound had been available when they would have been selected, which is not the case with a depth-first search. Roughly speaking, the depth-first search explores the tree in a vertical way while the best-first one searches it in a more horizontal way.

Let us now consider the drawbacks of a best-first search which mainly come from the *number of open nodes*, possibly large, that must be stored in the stack of open nodes. Indeed, if a good upper bound to discard node is not found rapidly for the best-first search, the number of open nodes can become significantly large. This problem does not arise with a depth-first search. More precisely, in the worst case, at the k^{th} level of the branch-and-bound tree, 2^{k-1} nodes are open with a best-first search while this number is reduced to k when a depth-first search is employed. For large size problems, this can produce problems of memory. The search of the node associated to the smallest lower bound on the optimum value is also needed to choose the most promising node in the stack. This operation can be expensive if the stack comprises a lot of nodes. Moreover, the local behaviour of the depth-first search is lost and as a consequence, the information coming from the last solved problem cannot be exploited to solve

the next problem treated.

Instead of a depth-first search, a best-first search has been used in the method proposed in Section 5.1.1 and improved by the modifications validated in Chapters 5 and 6. This node selection rule chooses to treat the open node with the smallest lower bound on the optimum value of the nonlinear problem. If several nodes minimize this lower bound, we choose to examine, among them, the deepest one in the tree, and if again several candidates remain, the last of them which has been created is chosen. This choice tries to limit the size of the tree by going down in this tree with the hope of discarding nodes. The results obtained with this best-first search are given in Table 8.18 and must be compared with the ones produced with a depth-first search already mentioned in Table 8.17.

The graphic comparison of these results is presented in Figure 7.1. As previously, the top graph represents the improvement percentage i_{pm} in the number of linear problems solved defined in (5.1) and obtained by using a best-first search instead of a depth-first one, while the bottom graph gives the performance profiles related to these methods. We refer the reader to Sections 5.1.4 and 5.3 for a more detailed explanation about the interpretation of these graphs.

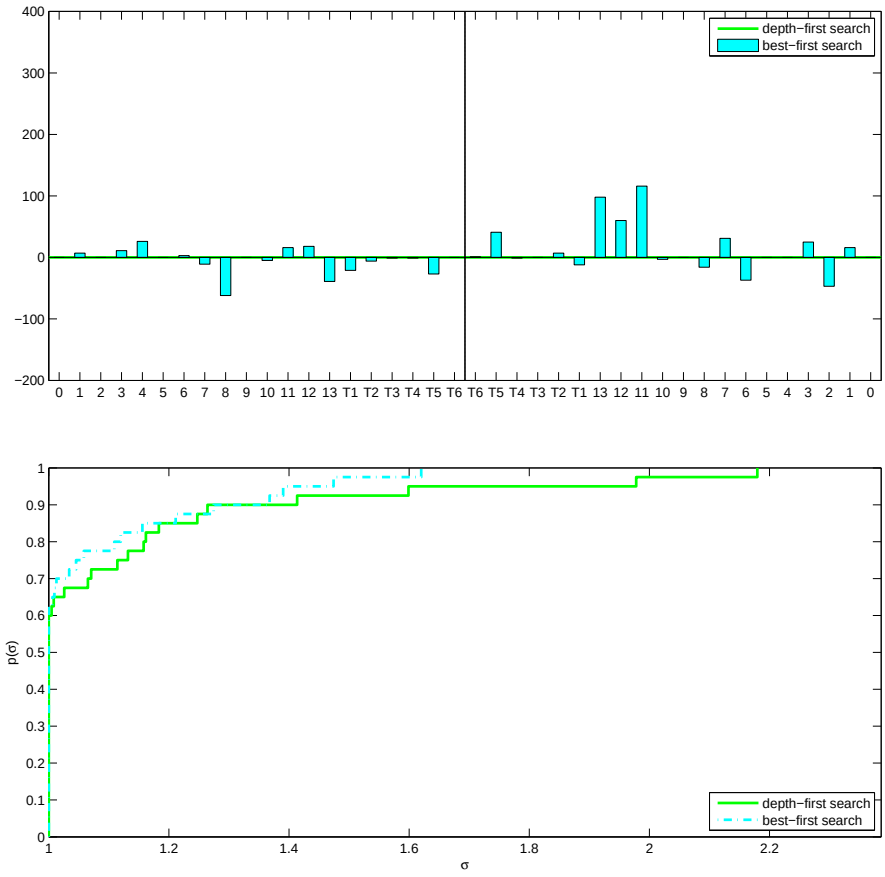


Figure 7.1: Comparison of results obtained by using a best-first search instead of a depth-first one.

The results presented in Figure 7.1 are quite surprising, because the best-first search is not

significantly better than the depth-first one. We observe 16 improvements, 9 equalities and 15 deteriorations although the best-first search is guaranteed not to solve problems associated to superfluous nodes, to the contrary of the depth-first search. Observe also that better results are obtained for the discrete case than for the continuous one. This can be explained by the fact that, with the data used for the problems treated, the optimum value of the original problem is rapidly found with a depth-first search for the continuous version of the problems. In this case, the number of superfluous nodes examined with a depth-first search can be considered as insignificant. As a consequence, for these problems, there is no reason that the best-first search produces better results.

Note furthermore, that with the branching rule employed (pseudocosts combined with strong branching iterations developed in Section 6.5), the node selection strategy does not only modify the way of exploring the tree, but also the branching rule. Indeed, as the tree is searched in a different order, the pseudocosts are *differently updated* and the selected branching variables can vary with regard to the method using a depth-first search. Therefore, at the end of the branch-and-bound process, the generated branching tree will be different depending on the node selection strategy employed. One can expect that an update of pseudocosts (see (6.13)) by passing through the tree in a depth-first manner is better than in a best-first one. This is justified by the fact that the quantity of increase in the value of the objective function computed at a node by unity of reduction in the range of the variable, that is, the pseudocost, can strongly vary according to the place of the considered node in the tree and the degree of refinement of the outer approximation problem. Locally, however, as the outer approximation is not modified much, these quantities and thus the pseudocosts, do not differ much. To exploit this in order to have relevant pseudocots, it is preferable to update and average pseudocosts on a more local basis, that is, in a depth-first manner instead of updating them in a best-first way. This reasoning is confirmed by the results. Indeed, for five of the *TVC* problems in the continuous case, the best-first search gives less good results than the depth-first one. For these problems, since the optimum value is rapidly found, as explained above, it is more the impact of a different order to update pseudocosts which is highlighted in the results than the ability of the node selection to discard nodes more or less rapidly.

Nevertheless, for problems for which the optimum is not rapidly found, the drawback of updating the pseudocosts in a best-first way is exceeded by the benefit obtained by avoiding to examine superfluous nodes. This can be observed for the discrete problems represented as usually in the right part of the top graph of Figure 7.1. For these problems, 9 improvements are produced by employing a best-first search, 5 equalities and 6 deteriorations. On the performance profiles of Figure 7.1, it appears that the best-first search is slightly more efficient and also allows us to generate improvements generally larger than the produced deteriorations. However, on larger problems than the ones treated here, the size of the stack of open nodes for a best-first search could generate problems of memory.

7.3 Best-estimate criterion

In order to benefit from the advantages of depth and best-first searches while reducing their drawbacks, some node selection rules consisting in a compromise between these two techniques have been proposed in the literature (see Breu and Burdet [21], Bénichou *et al.* [23] or Gauthier and Ribière [48], for instance). We are interested here in the *best-estimate criterion* introduced

by Bénichou *et al.* in [23] for a branching on integer variables. This particular way of exploring the tree can be motivated by the noting that the best-first search selects the node associated to the smallest lower bound on the optimum value but without taking care that a good feasible value for the original problem (in the sense that it allows us to discard nodes) is, or not, reachable from this node. In fact, it would be more interesting to examine nodes that are expected to *lead to good upper bounds*, which is the idea of best estimates. For each node not yet treated, the value of the objective function for the nonlinear problem is *predicted* for the best feasible solution reachable from this node. This amounts to estimate the increase in the value of the objective function in order to obtain a feasible solution for the nonlinear problem by starting from the solution of the linear outer approximation problem. To this goal, the pseudocosts Ψ_i defined in Section 6.4 are employed.

For integer restrictions, the determination of the moving in the range of the integer variable from its current value to reach an integer feasible value is obvious. More precisely, if at the current solution, an integer variable x_i is equal to x_i^*, the quantity of moving necessary to reach an integer solution is equal to $\lceil x_i^* \rceil - x_i^*$ or $x_i^* - \lfloor x_i^* \rfloor$, that is, we need to discard the part $[l_i, \lceil x_i^* \rceil[$ or $]\lfloor x_i^* \rfloor, u_i]$, respectively, from the current approximation interval $[l_i, u_i]$ for x_i. According to our definition of pseudocosts given in (6.11), the *smallest predicted increase* in the value of the objective function produced when the value of a variable x_i, equal to x_i^* at the current solution, becomes integer is given by:

$$\min\left((u_i - \lfloor x_i^* \rfloor)\Psi_i^-,\ (\lceil x_i^* \rceil - l_i)\Psi_i^+\right). \tag{7.1}$$

Indeed, as explained in Section 6.4, the predicted increase in the value of the objective function for a subproblem obtained by branching is computed by multiplying the associated pseudocost by the range of the variable discarded by branching. The use of the minimum in (7.1) is motivated by the fact that when one goes down in the tree, we have two possibilities: examining the left or the right subproblem. To estimate the smallest increase in the value of the objective function to obtain a feasible solution, the node leading to the smallest predicted increase in f is obviously chosen.

In the case of outer approximations of nonlinear components, the determination of the quantity of moving in the range of the variable x_i in order that the value of the outer approximations of the components involving x_i coincides with the value of the approximated components, is not obvious. Indeed, for each value of x_i, the outer approximations can produce the correct values (since they are outer) but this is only guaranteed when the approximation interval is reduced to one point. Therefore, the predicted increase in the value of the objective function in such a way that the current value of x_i given by x_i^* produces the same value for the components in which it is involved, in the outer approximation problem and in the original problem could be expressed like:

$$\left((u_i - x_i^*)\Psi_i^- + (x_i^* - l_i)\Psi_i^+\right), \tag{7.2}$$

since the parts $[l_i, x_i^*[$ and $]x_i^*, u_i]$ must be removed from the approximation interval in order to reduce it to $\{x_i^*\}$. However, this increase in f is computed in the worst case and is therefore, generally overestimating. Indeed, it is not always necessary to refine the variable so much in order that at the current solution, a variable produces the same values in the outer approximation and in the nonlinear problems. Such solutions can also arise with variables defined on more or less large intervals. Moreover, our numerical experiments have shown that formula (7.2) does not lead to good results. For these reasons, we have defined differently the predicted increase

in f:

$$\min\left((u_i - x_i^*)\Psi_i^-, (x_i^* - l_i)\Psi_i^+\right). \tag{7.3}$$

This new estimation of the increase in f is expected to be underestimating since it allows us to only remove one infeasible part ($[l_i, x_i^*[$ or $]x_i^*, u_i]$) from the approximation interval in order that the outer approximations of the nonlinear components involving x_i produce exact values. But there is no guarantee in this way. The motivation of using definition (7.3) is twofold. Firstly, this definition is built on the same basis as (7.1). Indeed, in case of an outer approximation, x_i^* can be considered as the feasible value to reach and for which we want to obtain an outer approximation producing correct values. Secondly, by discarding the part $]x_i^*, u_i]$ or $[l_i, x_i^*[$ from the approximation interval $[l_i, u_i]$, x_i^* becomes a bound of the remaining approximation interval. Remember that the outer approximations based on SOS of some particular nonlinear components are *exact at the bounds* of the approximation interval. This is the case for the lower and upper bounds on the arguments of a bilinear product since its outer approximation based on SOS is defined by (2.27), (2.28), (2.29) and (3.39). By using bound propagation of Section 3.2.2, it is also true for the square function, more particularly, for the bound on the interval corresponding to the square root of the lower bound on this square function (when zero is not strictly inside the approximation interval, as explained at the end of Section 4.1.1, see Figure 4.4). This also holds for the lower or upper bound of trigonometric functions if these ones are defined on particular intervals where their outer approximations based on SOS are always overestimating or underestimating and if the considered bound produces an extremum for the trigonometric function. For all the cases mentioned above, the outer approximations based on SOS defined on $[l_i, x_i^*]$ or on $[x_i^*, u_i]$ are exact at x_i^*. Therefore, the predicted increase (7.3) can be considered as valid since it is enough to remove one part $]x_i^*, u_i]$ or $[l_i, x_i^*[$ to obtain an exact approximation in x_i^* .

We can now determine the quantity f_{est}^t estimating the best possible value for the objective function of the nonlinear problem which can be reached from a node t to divide. This best value is computed by starting from f_L^t, the optimum value of the linear outer approximation problem at node t and by adding to this value, the increase predicted by pseudocosts for each nonlinear and integer variables to obtain a feasible solution for the nonlinear problem. Note that this definition does not take the possible correlations between the variables into account and suppose that all variables are independent. Grouping together f_L^t and the increases in this value generated by the attempts to satisfy the integer restrictions (7.1) and to obtain outer approximations at some points producing correct values (7.3), the estimated value f_{est}^t for the objective function associated to the best feasible solution reachable from a node t is defined by:

$$f_L^t + \sum_{i \in N_{nl}} \min\left((u_i^t - x_i^t)\Psi_i^{-,t}, (x_i^t - l_i^t)\Psi_i^{+,t}\right) + \sum_{i \in N_{int}} \min\left((u_i^t - \lfloor x_i^t \rfloor)\Psi_i^{-,t}, (\lceil x_i^t \rceil - l_i^t)\Psi_i^{+,t}\right), \tag{7.4}$$

where
- f_L^t is the optimum value of the linear outer approximation problem at node t,
- N_{nl} is the set of variables appearing nonlinearly in the nonlinear problem,
- N_{int} is the set of integer variables of the problem,
- $\Psi_i^{+,t}$ and $\Psi_i^{-,t}$ are the up and down pseudocosts associated to the variable x_i at node t,
- l_i^t and u_i^t are respectively the lower and upper bounds on the variable x_i at node t,
- x_i^t is the current value of the variable x_i at the solution the linear outer approximation problem at node t.

Our node selection rule inspired by best estimates chooses to treat the open node with the smallest predicted value f_{est}^t for the objective function of the nonlinear problem. If several nodes produce the same smallest value, we choose to examine, among them, the one which is the deepest in the tree, and if several nodes are always candidate, we select the last of them which has been created. This node selection rule has been applied to the method developed so far instead of a depth or a best-first search. The obtained results are given in Table 8.19 and are graphically compared with the ones produced by a depth or a best-first search on Figure 7.2. It can be observed on Tables 8.17, 8.18 and 8.19 and on this figure that the number of linear

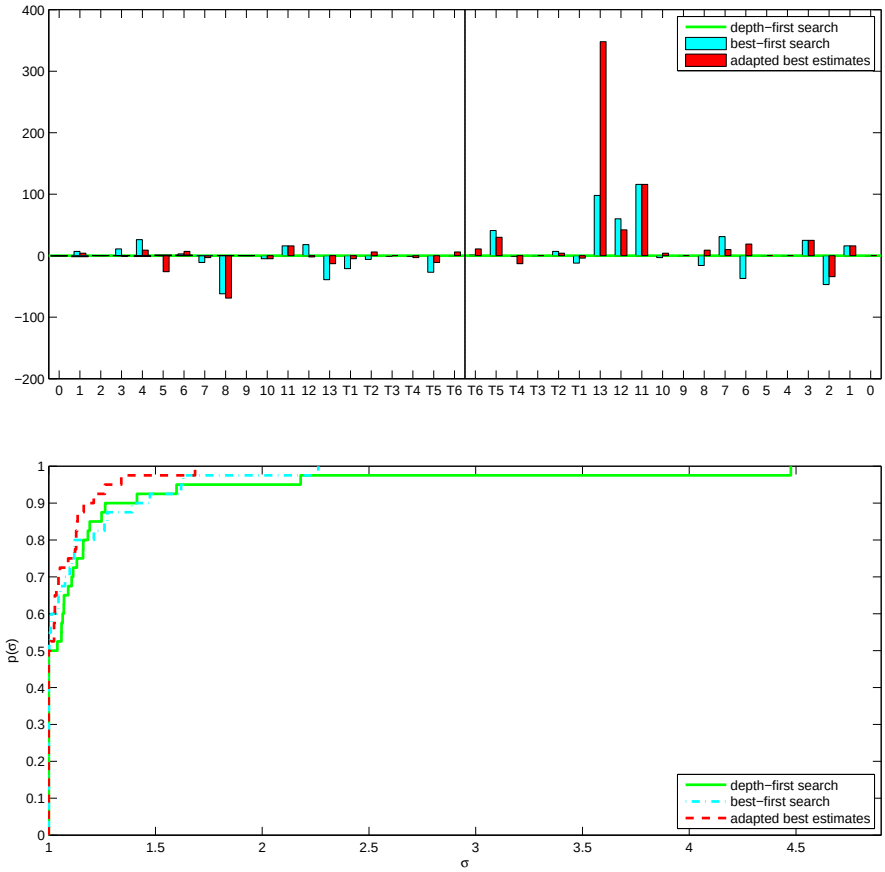


Figure 7.2: Comparison of results obtained by using a best-first search or an adaptation of best estimates instead of a depth-first search.

problems solved with the adaptation of the best estimates is generally intermediary between the ones produced for a depth and a best-first search, even if it is closer to the number of linear problems obtained with a best-first search. This result could be expected because of the definitions of the three compared node selection rules. Note however that the adaptation of best estimates can also produce results better than the two other ones (for the discrete case: *pb6*, *pb8*, *pb10*, *pb13* and *TVC6*), and sometimes significantly (*pb13*). In these cases, it has allowed us to reduce the drawbacks of the best and depth-first searches. According to the performance profiles of Figure 7.2, the adaptation of best estimates is generally better than the depth and best-first searches. Indeed, the performance profile associated to the adaptation

of best estimates is generally higher than the two other performance profiles. The adaptation of best estimates allows us to produce improvements larger than the generated deteriorations. However, with regard to the efficiency, the best-first search is the best method among the three compared according to the performance profiles of Figure 7.2, but if we only compare the best-first search and the adaptation of best estimates, this is the latter node selection rule which is the most efficient, as shown in Figure 7.3. This result about the efficiency can be explained by the fact that the adaptation of best estimates provides results generally indermediary between the depth and best-first searches, which prevents it from being the most efficient if we compare the three methods together. However, if we compare it only with the best-first search, it is the most efficient since it is also usually better on the problems for which the depth-first search was the best method.

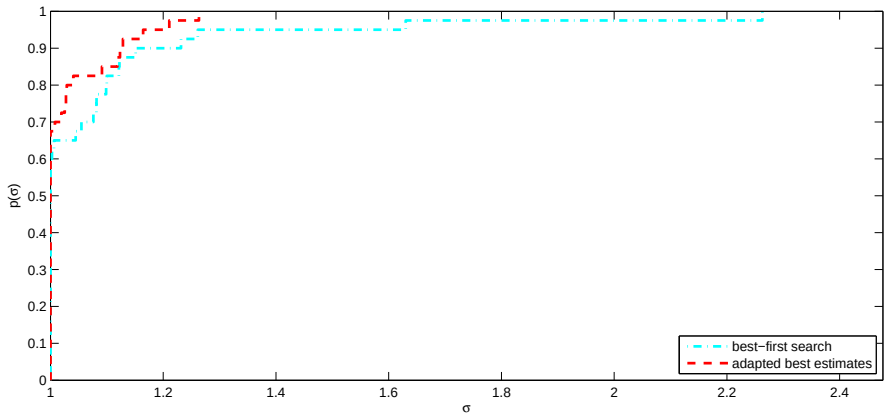


Figure 7.3: Performance profiles related to a best-first search and to an adaptation of best estimates.

With regard to the maximum number of nodes stored in the stack of open nodes, the adaptation of best estimates is also intermediary between the best and depth-first searches, as represented in Figure 7.4. This figure shows the percentage of increase in the maximum number of nodes stored in the stack for a best-first search and for the adaptation of best estimates compared to a depth-first search. The scale has been chosen to be intentionaly large to focus on the largest increases in the size of the stack, since these are the ones which can be problematic. The maximum number of nodes stored in the stack during the process of the method is also given for each problem in Tables 8.17, 8.18 and 8.19. The fact that the adaptation of best estimates produces better results than best-first search on average (according to the performance profiles of Figure 7.3 while it needs less storage, is an indication that our adaptation of best estimates allows us to suitably detect the nodes able to generate good upper bounds.

For all the reasons mentioned above, we think that the adaptation of best estimates corresponds to a good compromise between the depth and best-first searches and surely, to the best technique among the three node selection rules tested. Note nevertheless that it is more expensive than the two other techniques since it needs to evaluate and store the best predicted value (7.4) for the objective function reachable from each open node and, once a node has to be selected, it needs to choose in the stack of open nodes, the one producing the smallest value.

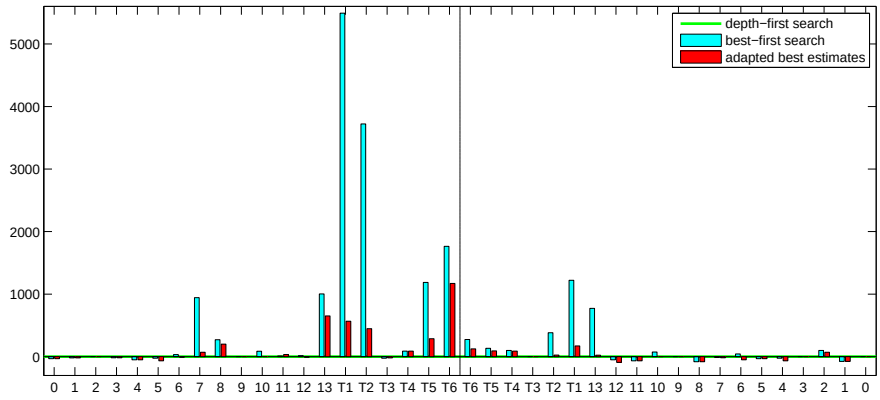


Figure 7.4: Comparison of the maximum number of nodes stored in the stack for a best-first search or for the adaptation of best estimates instead of for a depth-first search.

Finally, to limit the cost of this adaptation of best estimates, we employ in (7.4) the value of pseudocosts *available at each node*. The quantity (7.4) is evaluated only once for each open node and stored to be used each time a node must be selected. However, we have highlighted in Section 6.4 that the value of pseudocosts is not always relevant (in particular at the top of the tree when the pseudocosts have not been enough initialized). A possible way to improve this node selection rule would be to compute (7.4) for each open node with the *current pseudocosts*, Ψ_i^+ and Ψ_i^-, each time a node must be selected. That is, (7.4) would become:

$$f_L^t + \sum_{i \in N_{nl}} \min\left((u_i^t - x_i^t)\Psi_i^-, (x_i^t - l_i^t)\Psi_i^+\right) + \sum_{i \in N_{int}} \min\left((u_i^t - \lfloor x_i^t \rfloor)\Psi_i^-, (\lceil x_i^t \rceil - l_i^t)\Psi_i^+\right), \tag{7.5}$$

But this would imply to evaluate (7.5) for each open node each time a node must be selected, which would increase the cost of the node selection rule. A compromise could also be considered (using (7.5) for the first iterations and then, using (7.4), for example). Another way to possibly improve the adaptation of best estimates would be to take the quality of the approximation into account in the node selection rule, as it has been done in the branching rule.

7.4 Conclusion

This chapter has underlined the advantages and drawbacks of depth and best-first searches. The improvement produced on our test problems by the best-first search compared to the depth-first one has not been as large as expected because the best-first search does not only change the way of selecting the node to examine but also modifies the order in which the pseudocosts are updated and averaged. On the observed numerical results, it appears that an update of pseudocosts in a depth-first manner must be preferred in order to update pseudocosts on a more local basis. However, for problems which do not rapidly detect the optimum solution, the impact of a best-first search is beneficial and exceeds the drawback of updating pseudocosts in a best-first way. Nevertheless, on larger problems than the ones treated here, the best-first search is expected to build a large stack of open nodes.

To remedy this problem, another way to select the node based on best estimates has been considered. Best estimates developed in the mixed integer case predict the value of the objective function at the best feasible solution reachable from a node. While it is obvious to determine the moving in the range of an integer variable to obtain a feasible integer value, it is not the case for the quantity of moving necessary to produce a value for the outer approximation coinciding with the correct value in the original problem. This is only guaranteed if the approximation interval is reduced to one point but this observation leads to a definition of the increase in the value of the objective function which is overestimating and does not produce good results. Therefore, another definition for this increase has been given. Our numerical experiments have shown that the estimation of the best optimum value reachable from a node obtained by combining the predicted increases in the value of the objective function to satisfy integer restrictions and to produce exact outer approximations leads to good results in practice. Moreover, the developed node selection rule allows us to limit the size of the stack of open nodes compared to the best-first search. However, it is more expensive than the depth and best-first searches since it needs to evaluate and to store the smallest estimated feasible value for the objective function of the nonlinear problem reachable from each open node. Finally, some ideas have also been given to improve this adaptation of best estimates.

Chapter 8

Final comparisons of the results

We conclude this part of the thesis about the numerical results with *global comparisons* and comments. In the three previous chapters, the improvements or deteriorations have been measured locally, in the sense that the method has evolved with the experiments and we have mainly compared methods obtained with or without a proposed modification. However, we have never compared the basic method of Section 5.1.1 with the final method combining (i) the presolve and range reduction of Chapter 5, (ii) the branching rule based on a combination of pseudocosts and strong branching described in Chapter 6 and (iii) the node selection strategy given by the adaptation of best estimates developed in Chapter 7. It is one of the purposes of the present chapter to do so. Moreover, the comparisons presented so far have been performed on the basis of the number of linear problems solved instead of on the basis of the *CPU time* since our software is not optimized with regard to this aspect. We also show in this chapter that a comparison based on the CPU time leads to essentially similar results. Finally, we compare our method with competitive softwares available on the NEOS server (see Chapter 1).

8.1 Basic versus final method

We thus compare the basic method with the method finally obtained after all our numerical experiments. In order to also highlight the progresses obtained by handling the presolve and range reduction as well as the branching rule in an appropriate way, we also consider the results obtained with the methods developed at the ends of Chapters 5 and 6. Accordingly, we compare four methods:

1. the *basic method* defined in Section 5.1.1 and using the specific ranking of the variables detailed in Section 5.4 in order to have a more consistent basis of comparison,

2. the method with presolve and range reduction, simply referred thereafter to as the *presolve method*, obtained in Section 5.6 thanks to presolve and range reduction phases,

3. the *variable method*, that is, the method derived from the presolve method by adding the branching rule of Section 6.5,

4. the *final method* which corresponds to the variable method modified by employing the node strategy based on best estimates (see Section 7.3). This is also the best method that we have tested.

We recall that the numerical results obtained with these four methods are respectively given in Tables 8.8, 8.10, 8.17 and 8.19.

Figure 8.1 graphically represents the performance profiles related to the number of linear problems solved by the four methods cited above. This information is summarized in Table 8.1. In this table, stars in a field mean that the method needs more than 500.000 linear problems to solve the problem. On Figure 8.1, the curves associated to the variable and to the final methods nearly merge with the axis $\sigma = 1$. They are thus a lot better than the other ones, as one could expect.

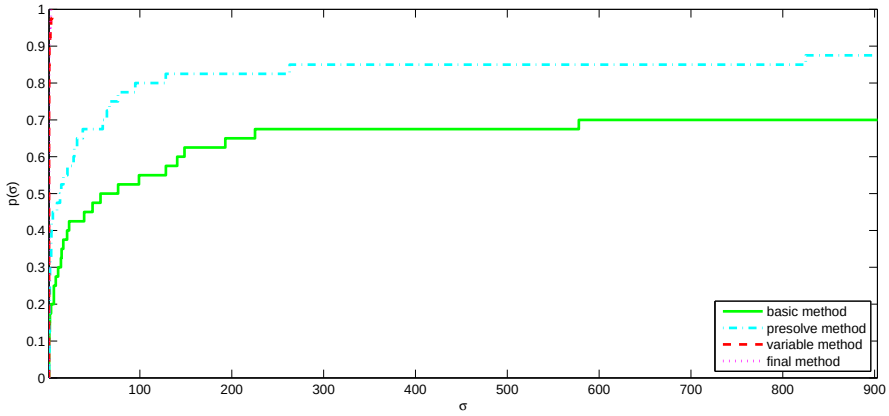


Figure 8.1: Performance profiles related to the comparison of the number of linear problems solved with the basic, presolve, variable and final methods.

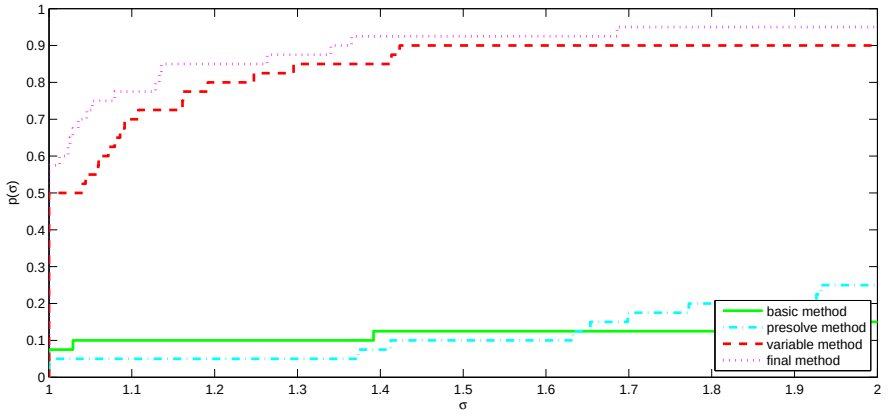


Figure 8.2: Performance profiles around $\sigma = 1$ related to the comparison of the number of linear problems solved with the basic, presolve, variable and final methods.

In order to compare the two best methods, Figure 8.2 focuses on the performance profiles around $\sigma = 1$. By observing these figures and the performance profiles of Figure 7.2 which compare the final method with the variable method, that is, the two best methods among the

	basic method	presolve method	variable method	final method
pb0	45	112	112	112
pb1	107	104	147	142
pb2	2131	5636	44	44
pb3	159	262	78	79
pb4	9303	9710	177	163
pb5	3405	396	205	259
pb6	3533	5449	598	558
pb7	16245	19859	5054	5182
pb8	111721	37107	579	976
pb9	20302	3813	266	266
pb10	20685	19928	527	551
pb11	7241	7996	1326	1142
pb12	11131	2181	1336	1367
pb13	467225	*****	20362	22983
TVC1	*****	*****	82847	87182
TVC2	*****	*****	61924	58457
TVC3	140865	1339	948	948
TVC4	*****	113739	1701	1750
TVC5	*****	*****	18379	20873
TVC6	*****	*****	32852	30994
pb0	63	63	68	68
pb1	133	131	79	68
pb2	2115	3237	194	260
pb3	135	197	121	97
pb4	15389	11388	120	120
pb5	3009	257	145	145
pb6	65800	6145	348	292
pb7	377	1353	1235	1121
pb8	*****	198817	263	241
pb9	62149	33668	442	442
pb10	113846	51816	205	197
pb11	3806	7349	558	258
pb12	*****	33407	1503	1056
pb13	*****	8093	17388	3885
TVC1	108861	40446	7756	8031
TVC2	*****	72270	5792	5547
TVC3	62045	861	627	627
TVC4	*****	38792	1396	1582
TVC5	*****	7369	5619	4338
TVC6	*****	12131	6096	5503

Table 8.1: Number of linear problems needed by the final, presolve, variable and final methods to solve our test problems.

four compared, we can conclude that the final method is the best one. Note moreover that the best progresses in the results are obtained by handling the branching rule in a convenient way. The branching variable strategy is thus the issue which has the biggest impact.

Due to the fact that our software is not optimized with regard to the CPU time, we have not based our comparison on this measure so far. We show here that using the CPU time as basis of comparison leads to similar conclusions, as shown in Figure 8.3 that represents the performance profiles associated to the CPU times observed for the four compared methods. These CPU times are given in Table 8.2. Again, stars are used in this table to represent the fact that a method cannot solve the problem considered before the maximum number of linear problems allowed to solve is reached. The comments about Figure 8.1 remain valid for Figure 8.3. Again, we zoom around $\sigma = 1$ to observe the behaviour of the curves related to the two best methods.

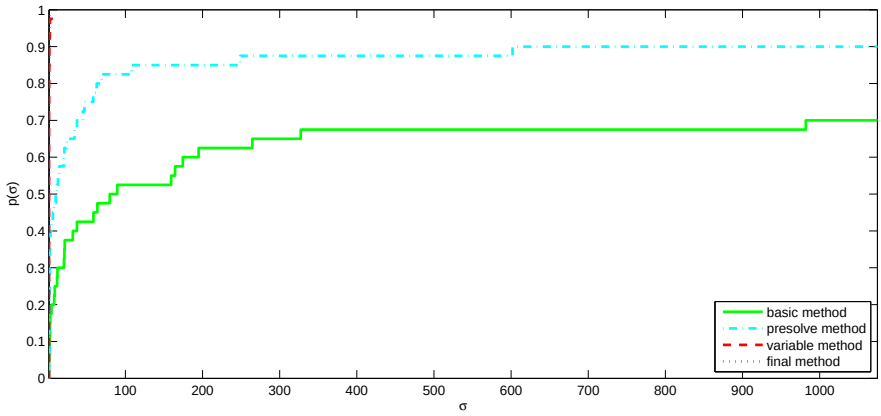


Figure 8.3: Performance profiles related to the comparison of the CPU times observed with the basic, presolve, variable and final methods.

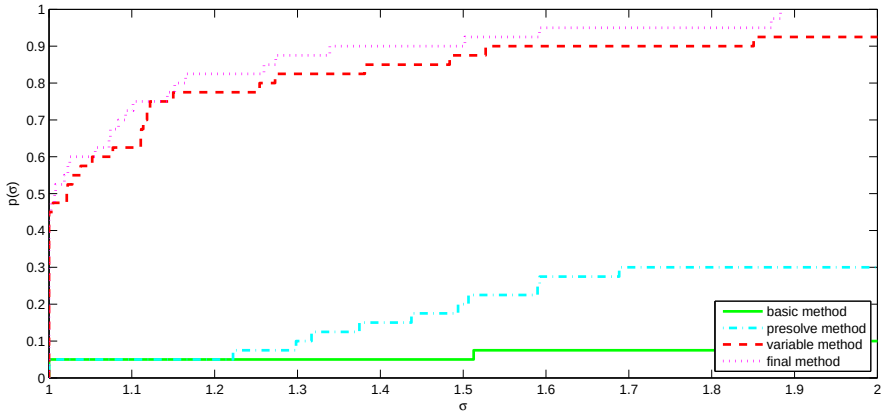


Figure 8.4: Performance profiles around $\sigma = 1$ related to the comparison of the CPU times observed with the basic, presolve, variable and final methods.

	basic method	presolve method	variable method	final method
pb0	0.154	0.260	0.285	0.290
pb1	0.419	0.277	0.423	0.441
pb2	7.590	12.872	0.119	0.129
pb3	0.562	0.662	0.224	0.256
pb4	62.050	35.024	0.778	0.850
pb5	21.470	1.593	1.002	1.262
pb6	32.553	31.865	4.674	4.208
pb7	164.018	109.898	38.225	38.317
pb8	858.543	189.060	3.244	4.873
pb9	144.914	18.081	1.620	1.633
pb10	151.166	87.069	2.577	2.625
pb11	55.544	39.612	7.486	6.509
pb12	227.770	27.416	19.072	19.506
pb13	10463.566	*****	281.064	323.717
TVC1	*****	*****	1264.972	1335.563
TVC2	*****	*****	906.001	861.052
TVC3	5969.975	42.102	30.625	31.395
TVC4	*****	3445.432	54.868	58.937
TVC5	*****	*****	980.362	1079.785
TVC6	*****	*****	1716.357	1653.214
pb0	0.215	0.118	0.148	0.158
pb1	0.492	0.336	0.235	0.211
pb2	7.583	8.336	0.654	0.833
pb3	0.501	0.502	0.365	0.246
pb4	98.798	43.030	0.634	0.620
pb5	19.741	0.931	0.621	0.618
pb6	673.784	29.253	2.309	2.058
pb7	3.810	8.180	7.728	7.132
pb8	*****	813.735	1.512	1.352
pb9	446.907	120.343	2.616	2.561
pb10	802.281	203.367	0.880	0.817
pb11	31.330	36.846	3.374	1.541
pb12	*****	423.871	29.412	21.295
pb13	*****	79.982	266.369	60.734
TVC1	2429.940	479.115	115.79	124.184
TVC2	*****	770.639	81.639	79.411
TVC3	3199.969	23.784	19.461	19.594
TVC4	*****	1040.867	49.541	57.712
TVC5	*****	317.225	311.049	244.350
TVC6	*****	495.180	368.399	331.486

Table 8.2: CPU times needed by the final, presolve, variable and final methods to solve our test problems.

Essentially, we obtain the same conclusions as for the comparison based on the number of linear problems. Nevertheless, two differences can be underlined. The first one is concerned with the efficiency of the basic method with regard to the presolve method. The most efficient method among both varies according to the basis of comparison. For the number of LP solved, the basic method is the best while both methods are equivalent with regard to the efficiency for the CPU time. This can be explained by the fact that we do not take the *number of NLP* solved into account when the measure of the quality of the method depends on the number of LP solved. As explained in Chapter 5, no NLP is associated to a LP solved during the presolve or a range reduction phase. Therefore, the proportion of the number of NLP with regard to that of LP is smaller with the presolve method than with the basic method. So, if the number of LP solved by the presolve method is a little bit larger than with the basic method, the CPU time can even though be smaller since the presolve method probably solves a smaller number of NLP. This explains the difference on the performance profiles of Figures 8.2 and 8.4 around $\sigma = 1$.

The second variation which can be observed with regard to Figure 8.2 is related to the quantity of improvement produced by the final method compared to the variable method. The improvement is larger when the comparison is done on the number of LP solved instead of on the CPU time. The reason of this is simple. Both methods only differ by the node strategy. The final method uses an adaptation of best estimates while a depth-first search is employed for the other one. As pointed out in Section 7.3, the depth-first search is less expensive to apply than the adaptation of best estimates. This is why the improvement is reduced when the CPU time is considered.

8.2 Comparison with other softwares

After having compared our own methods, we compare our final method with competitive softwares available on NEOS. As no global optimization solver appropriate for our problems is accessible on this server, we limit ourselves to test methods conceived to solve nonlinear and convex problems. For the continuous case, we are interested in *KNITRO*, *IPOPT* and *FilterSQP*, and for the discrete one, in *Bonmin* and *MINLP_BB* (see Chapter 1 for more details and references about the methods implemented in these solvers). Since these methods are not global, they are less expensive than ours. However, they cannot guarantee the convergence to a global optimum contrary to ours, even if in some cases, they find such an optimum. The following tables show the optimum values for each of our test problems obtained with the different solvers cited above. The first table is related to the continuous case and the second one to the discrete case. The second column of these tables gives the global optimum value which is found by our method within an accuracy of 10^{-3}. The two last lines of the tables show respectively the number of problems for which a feasible solution has been found and that for which this optimum is global within an accuracy of 10^{-3}.

By analyzing these results, we observe that these solvers conceived to solve nonlinear convex problems can be efficient to find a global solution for the tested nonconvex problems. However, some of our problems cannot be solved. Indeed, for the continous case, *IPOPT* which gives the best results among the three solvers compared to ours, fails to converge on three problems. It finds the global optimum within an accuracy of 10^{-3} for only fourteen problems on twenty. *FilterSQP* does not find the global optimum for the half of the problems and does not converge to a feasible solution for eight problems. If we consider the discrete case, the results

obtained with our method are still better since here, *Bonmin* fails to converge on five problems and finds the global optimum for only eleven problems on twenty, while *MINLP_BB* detects the global optimum for nine problems and does not converge for nine problems. This shows that our method, which can find the global optimum for the twenty problems can be helpful.

	Optimum value	IPOPT	KNITRO	FilterSQP
pb0	-3.00701	-3.00701	/	/
pb1	-1.88748	-1.88749	-1.88748	-1.88749
pb2	0.00000	0.00000	0.00000	0.81000
pb3	0.25000	1.16682	1.16683	3.68993
pb4	0.02482	0.02482	0.02482	/
pb5	11.60727	11.60727	11.60729	11.60727
pb6	0.00811	0.00810	0.00811	/
pb7	0.43370	/	0.43370	14.62515
pb8	0.03664	0.40037	0.04755	/
pb9	7.80941	8.18574	8.18741	8.18740
pb10	0.04230	0.04230	0.04230	/
pb11	7.83560	/	/	/
pb12	1.76118	/	/	/
pb13	0.51628	0.51628	0.53683	/
TVC1	5.65141	5.65142	5.65142	5.65142
TVC2	2.37956	2.37956	2.37956	2.37956
TVC3	5.31869	5.31869	5.31869	5.31869
TVC4	1.01230	1.01228	1.01228	1.01228
TVC5	1.16538	1.16538	1.16538	1.16538
TVC6	0.09104	0.09104	0.09104	0.09104
# solved	20	17	17	12
# global solution	20	14	13	8

Table 8.3: Optimum values obtained for our test problems (within an accuracy of 10^{-3}) with our outer approximation method (2nd column) and different solvers for convex NLPs.

	Optimum value	Bonmin	MINLP_BB
pb0	-2.91437	/	/
pb1	-1.81859	-1.81859	-1.81859
pb2	0.00000	0.00000	0.81000
pb3	0.25000	1.16683	3.68993
pb4	0.03416	/	/
pb5	11.65284	11.65284	11.65284
pb6	0.04000	0.05448	/
pb7	0.43701	0.43701	/
pb8	0.09000	0.42529	/
pb9	8.29000	8.29000	8.29000
pb10	0.09000	0.74000	/
pb11	7.94848	/	/
pb12	2.08547	/	/
pb13	0.54834	/	/
TVC1	5.66061	5.66061	5.66061
TVC2	2.38403	2.38403	2.38403
TVC3	5.33013	5.33013	5.33013
TVC4	1.02951	1.02949	1.02949
TVC5	1.18652	1.18652	1.18652
TVC6	0.09932	0.09835	0.09835
# solved	20	15	11
# global solution	20	11	9

Table 8.4: Optimum values obtained for our test problems (within an accuracy of 10^{-3}) with our outer approximation method (2nd column) and with different solvers for convex MINLPs.

Conclusions and perspectives

In our research work, we have developed a new *global optimization* method conceived to solve specific nonlinear and nonconvex problems (continuous and mixed integer). Our approach has been motivated by a real-world problem arising in the management of electrical networks. Due to the cost of solving the nonlinear continuous relaxation of this problem, an approach based on the solution of *linear* subproblems has been preferred. We have first considered and tested an *approximation method* using special ordered sets (SOS), which has already been used to solve real problems. However, because of the *nonconvex* behaviour of the problem that we aim at solving, no feasible solution could be found by this approximation method. To get round this difficulty, we have chosen to turn to global optimization methods.

From the approximation problem using SOS, an *outer approximation* problem has been built. To this goal, we have *decomposed* each function of the problem under study in components involving one or two variables. For the three kinds of components appearing in this problem, we have established the expression of the *maximum approximation errors* (underestimation and overestimation) produced by their SOS approximation. Using this information, we have constructed an *outer approximation problem* having the advantage of handling all functions in the same way, provided the expression of the associated approximation errors produced by the SOS approximation is available. The same general framework can thus be employed to treat a large class of problems. In order to refine the outer approximation problem, we have used a *branch-and-bound* strategy that allows us to refine as much as necessary the outer approximation problem. This is the key for the convergence of the method.

In Chapter 4, we have shown that, for the considered problem, the proposed outer approximations for the nonlinear components of one or two variables involved in this problem are theoretically competitive with the outer approximations commonly used. This is due to the fact that our formulation can exploit the possible *links* between the functions of the problem, but at the price of the introduction of new variables. We have also formally proved for square and bilinear functions that it is better to branch on original variables than on variables of SOS type, if one aims at discarding the largest part of the domain of possible values for the outer approximation.

We have implemented the developed method and tested several alternatives for the choices related to a branch-and-bound process, concerning notably, the *range reduction* of the variables, the *branching rule* and the *node selection* strategy. It has been highlighted that a suitable choice to handle these features allows us to strongly improve the speed of convergence of the method. Some other questions which would be worth to be treated have not been studied thoroughly in this thesis and would consist of a natural extension of this work, like the *place of the breakpoints* in the modelling or the *place at which branching* in the range of the continuous variables. The *frequency of solving the nonlinear problem* during the branch-and-bound process is also an

important question. In the developed method, a nonlinear problem is solved at each node to divide. Spacing the call to the nonlinear solver could also improve the speed of convergence of the method. However, the criterion employed to decide when a nonlinear problem has to be solved must be chosen with care in order to avoid to solve too many nonlinear problems, but even though, sufficiently to obtain a good upper bound on the optimum value of the original problem to fathom nodes. Adding *cuts* to the method is another way to improve it in order to discard some parts of the domain of possible values.

Coming back to the issues that we have investigated, the adaptable presolve presented in this thesis has allowed us to decrease significantly the number of subproblems to solve for the most difficult problems tested. The bound tightening performed at some fixed levels of the branch-and-bound tree has also produced the same effect. However, we think that these range reduction phases could still be improved. Indeed, the bound tightening that we have used is expensive since it needs to solve optimization problems. Nevertheless, for not too small problems, the improvement that it produces widely exceeds its cost. To try to reduce the cost of the range reduction phases, the number of variables candidate for bound tightening could be reduced to a small *subset of variables* (possibly chosen randomly or for which the previous range reduction phases have been successful, for example). Another possibility would be to strengthen the bounds on the variables by using *optimality-based range reduction* (see Section 1.2.4). This would avoid to solve optimization problems for the active bounds. Finally, since for the studied test problems, the efficiency of the bound tightening is correlated with the size of the problems, we could try to link the frequency of using bound tightening with the size of the treated problems. For each of the ideas proposed above, the ideal trade-off should be found between the decrease in the cost of the range reduction phases and the lost of information generated.

With regard to the branching rule, we have proposed, tested and discussed several variants. We have first pointed out that it is interesting to *"branch again"* on variables that have allowed us to discard nodes for the last branching phases. When branching rules based on approximation errors are used, *weighting* these errors by the coefficients appearing in the nonlinear problem has been shown to be useful. But the branching rules based on approximation errors are not as good as the branching rules based on the increase in the value of the objective function, like strong branching and pseudocosts. However, to be efficient on the developed outer approximation problems, the classical rules have been adapted in order to take the *quality of the outer approximations* into account. In some cases, we branch to increase the value of the objective function, in other cases, to improve the quality of the outer approximations. An alternative would be to use a *merit function* which would weight these two goals, instead of improving one goal without considering the other one. We have also adapted the *definition of pseudocosts* to the case of continuous variables and have highlighted that in our method, the pseudocosts are not always relevant with regard to the values that they attempt to predict. To remedy this, we have proposed to integrate *strong branching iterations in the pseudocost technique*, which has given the best results among the branching rules tested.

Our study of choices related to branch-and-bound has been concluded with the question of node selection. Three strategies have been tested and the best of them was our *adaptation of the best-estimate criterion* to the continuous case and according to our definition of pseudocosts. This rule could possibly be improved by taking into account the fact that the pseudocosts are not always relevant with regard to the values that they predict, as detailed in Section 7.3. Compared to the range reduction and to the branching rule, the use of a more suitable node selection strategy has not improved the results a lot. This could be explained by the choice of our test

problems for which the global optimum is rapidly found. On problems of larger size, the differences should be more pronounced. Therefore, it would be desirable to test our method on a larger number of test problems, and of larger size. To this aim, the method should be generalized to handle a larger class of functions than the three kinds of nonlinear components involved in the problem under study. This implies to establish the expression of the approximation errors produced by the SOS approximation of each new function to treat. Functions involving more than two variables could be also considered (trilinear functions, for example). For the sake of generalization, techniques allowing us to derive the *computational graph* of the constraints could also be integrated in our method in order to reduce the modelling work of the user.

We think that we have developed a method which is promising, although it cannot solve real-world applications yet, due to its high *cost*. In order to be competitive, its cost should be imperatively reduced. Some ideas given above could already help to reach this goal. However, the first way to reduce the cost of the method would be to optimize our software, for instance by exploiting as much as possible the information available from an iteration to another for the solution of a linear problem. We could also improve the outer approximation problem itself by reducing its size. Indeed, in this research work, we have used a standardized framework in which all functions are relaxed by their outer approximation based on SOS. In a lot of cases, when the link with other functions can be exploited, this produces outer approximations tighter than the usual ones. However, when such links cannot be used, *other relaxations* ((4.12) or (4.18) for instance), possibly equivalent but involving less variables, should be employed.

In order for the method to converge faster, the outer approximation problem could also be made closer to the original problem. The use of cuts (see Nowak [91], for instance) is one way to reach this goal. Employing a convex combination of the errors instead of the maximum error to define the domain of possible values as in (3.24) is another way. An alternative would be to consider general convex outer approximation problems instead of linear ones. But as general convex problems are more expensive to solve than linear ones, it would perhaps be judicious to first solve linear outer approximation problems, and when the outer approximation problem is considered sufficiently refined, to switch to general convex outer approximation problems. A last perspective to reduce the cost of the method would be to exploit in heuristics the ideas developed in this thesis, but at the price to lose the guarantee of convergence.

To conclude, the research work presented therein has set the bases of a novel global optimization method which needs, however, to be improved in order to be efficient for solving the real-world problems for which it has been designed. Several perspectives have been proposed to this aim. Finally, we hope that this work will be useful, in some way, for further research.

Summary of contributions

Our main contributions are the *design* and the *implementation* of a new method of *global optimization* based on special ordered sets, for solving specific *nonlinear and nonconvex continuous* problems as well as *mixed integer nonlinear and nonconvex* problems coming from electrical design. Within the method which has been implemented, the following contributions are new:

- the extension of an approximation method using special ordered sets to an *outer* approximation method guaranteeing the convergence to a *global optimum* (Sections 3.2 and 3.3);
- the highlighting that for the application under study, the outer approximations used in our outer approximation problem are theoretically competitive with regard to the outer approximations commonly used. In particular, it has been shown that in some cases, the proposed outer approximation for a bilinear product can outperform McCormick's inequalities (Section 4.1);
- the theoretical proof that under appropriate assumptions, branching on original variables is better than branching on variables of SOS type, in order to reduce as much as possible the domain of possible values for the outer approximation. The quantification of this improvement has also been established for square and bilinear functions (Section 4.2);
- the development of a presolve technique depending of its efficiency (Section 5.5);
- the improvement of existing branching rules by "branching again" on variables allowing us to discard nodes (Section 6.1.2), by weighting the approximation errors (Section 6.2), by including a criterion depending on the quality of the outer approximation in strong branching and pseudocost strategies (Sections 6.3 and 6.4), by modifying the definition of pseudocosts to adapt it to the branching on continuous variables (Section 6.4) and by combining strong branching and pseudocosts suitably for the developed outer approximation problem (Section 6.5);
- the introduction of a new definition for the best-estimate criterion adapted to global optimization (Section 7.3).

A paper presenting all these ideas is currently in preparation.

Bibliography

[1] http://www.electricityforum.com/products/trans-s.htm.

[2] Website dedicated to global optimization: http://www.mat.univie.ac.at/ neum/glopt.html.

[3] LaGO - a (heuristic) branch and cut algorithm for nonconvex MINLPs, 2006. submitted, available for download at http://www.mathematik.hu-berlin.de/publ/pre/2006/P-06-24.ps.

[4] ILOG CPLEX 7.1. *User's manual*, 2001.

[5] K. Abhishek, S. Leyffer, and J.T. Linderoth. FILMINT: An outer-approximation-based solver for nonlinear mixed integer programs. Technical Report ANL/MCS-P1374-0906, Mathematics and Computer Science Division, Argonne National Laboratory, 2006.

[6] T. Achterberg, T. Koch, and A. Martin. Talk at workshop on mixed integer programming: Branching rules revisited. School of Business, Miami, Coral Gables, Florida, June 2006.

[7] C.S. Adjiman, I.P. Androulakis, and C.A. Floudas. Global optimization of MINLP problems in process synthesis and design. *Computers and Chemical Engineering*, 21:S445–S450, 1997.

[8] 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 and Chemical Engineering*, 22:1159–1179, 1998.

[9] 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 and Chemical Engineering*, 22:1137–1158, 1998.

[10] C.S. Adjiman, C.A. Schweiger, and C.A. Floudas. Mixed-integer nonlinear optimization in process synthesis. *Handbook of Combinatorial Optimization*, 1, 1999.

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

[12] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Finding cuts in the TSP. Technical Report 95-05, DIMACS, 1995.

[13] T. Bach. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolution Programming, Genetic Algorithms*. Oxford University Press, 1996.

[14] R. Bacher. The Optimal Power Flow (OPF) and its solution by the interior point approach. EES-UETP Madrid, Course, 10-12 December 1997.

[15] E.M.L. Beale. Numerical methods. In J. Abadie, editor, *Nonlinear programming*, pages 135–205. North Holland, Amsterdam, The Netherlands, 1967.

[16] E.M.L. Beale and J.A. Tomlin. Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables. In J. Lawrence, editor, *Proceedings of Fifth International Conference on Operation Research*, pages 447–454, London, 1970. Tavistock Publications.

[17] J.F. Benders. Partitioning procedures for solving mixed-variable programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[18] J.R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Verlag, Heidelberg, Berlin, New York, 1997.

[19] P. Bonami, L.T. Biegler, A.R. Conn, G. Cornuejols, I.E. Grossmann, C.D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter. An algorithmic framework for convex mixed integer nonlinear programs. Technical Report RC23771, IBM Research Report, 2005.

[20] S. Boyd, L. El Ghoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. SIAM Publications, Philadelphia, USA, 1994.

[21] R. Breu and C.A. Burdet. Branch-and-bound experiments in zero-one programming. *Mathematical Programming*, 2:1–50, 1974.

[22] M. Bussieck. Private communication, November 2004.

[23] M. Bénichou, J.M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1:76–94, 1971.

[24] S. Caratzoulas and C.A. Floudas. A trigonometric convex underestimator for the base functions in Fourier space. *Journal of Optimization Theory and Applications*, 124(2):339–362, 2004.

[25] V. Cerny. Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[26] A.R. Conn, N.I.M. Gould, and Ph.L. Toint. *Trust-Region Methods*. Number 01 in MPS-SIAM Series on Optimization. SIAM, Philadelphia, USA, 2000.

[27] J. Czyzyk, M. Mesnier, and J. Moré. The NEOS server. *IEEE Journal on Computational Science and Engineering*, 5:68–75, 1998.

[28] R.J. Dakin. A tree search algorithm for mixed programming problems. *Computer Journal*, 8(3):250–255, 1965.

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

[30] S. Dempe. *Foundations of Bilevel Programming*. volume 61 of *Nonconvex optimization and its application*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[31] J.E. Dennis and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1983. Reprinted as *Classics in Applied Mathematics 16*, SIAM, Philadelphia, USA, 1996.

[32] P. Deuflhard and A. Hohmann. *Numerical Analysis in Modern Scientific Computing - An Introduction*. Springer, second edition, 2003.

[33] E.D. Dolan. The NEOS Server 4.0 Administrative Guide (Technical Memorandum). Technical Report ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory, 2001.

[34] E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

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

[36] J. Eckstein. Parallel branch-and-bound algorithms for general mixed integer programming on the CM-5. *SIAM Journal on Optimization*, 4(4):794–814, 1994.

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

[38] M.C. Ferris and J.S. Pang. Engineering and economic applications of complementarity problems. *SIAM Review*, 39(4):669–713, 1997.

[39] A.V. Fiacco and G.P. McCormick. Programming under nonlinear constraints by unconstrained minimization: a primal-dual method. Technical Report RAC-TP-96, Research Analysis Corporation, McLean, Virginia, USA, 1963.

[40] A.V. Fiacco and G.P. McCormick. The sequential unconstrained minimization technique for nonlinear programming: a primal-dual method. *Management Science*, 10(2):360–366, 1964.

[41] R. Fletcher. Resolving degeneracy in quadratic programming. Technical Report NA/135, University of Dundee, United Kingdom, 1991.

[42] R. Fletcher, N.I.M. Gould, S. Leyffer, Ph.L. Toint, and A. Wächter. Global convergence of trust-region SQP-filter algorithms for nonlinear programming. *SIAM Journal on Optimization*, 13:635–659, 2002.

[43] R. Fletcher and S. Leyffer. Solving Mixed Integer Nonlinear Programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994.

[44] R. Fletcher and S. Leyffer. User manual for filterSQP. Technical Report NA/181, University of Dundee, United Kingdom, 1998.

[45] C.A. Floudas. *Nonlinear and Mixed Integer Optimization*. Oxford University Press, New York, 1995.

[46] C.A. Floudas, I.G. Akrotirianakis, S. Caratzoulas, C.A. Meyer, and J. Kallrath. Global Optimization in the 21st Century: Advances and Challenges. *Computers and Chemical Engineering*, 29(6):1185–1202, 2005.

[47] A. Forsgren, P.E. Gill, and M.H. Wright. Interior methods for nonlinear optimization. *SIAM Review*, 44:525–597, 2002.

[48] J.M. Gauthier and G. Ribiere. Experiments in mixed-integer linear programming using pseudocosts. *Mathematical Programming*, 12:26–47, 1977.

[49] A.M. Geoffrion. A generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10(4):237–260, 1972.

[50] P.E. Gill, W. Murray, and M.A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12:979–1006, 2002.

[51] F. Glover and M. Laguna. Tabu search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 71–140. Blackwell Scientific Publishing, Oxford, England, 1993.

[52] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, New York, 1989.

[53] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2000.

[54] W. Gropp and J. Moré. Optimization environments and the NEOS server. *Approximation Theory and Optimization*, pages 167–182, 1997.

[55] I.E. Grossmann. Review of nonlinear mixed-integer and disjunctive techniques. *Optimization and Engineering*, 3:227–252, 2002.

[56] O.K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31:1533–1546, 1985.

[57] P. Hansen. *Global Optimization Using Interval Analysis*. Marcel Dekker, New York, 1992.

[58] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of the National Bureau of Standards*, 49:409–436, 1952.

[59] J.B. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of convex analysis*. Springer Verlag, Berlin, 2001.

[60] R. Horst and P.M. Pardalos. *Handbook of Global Optimization: Nonconvex Optimization and Its Application*. Kluwer Academic Publishers, 1994.

[61] R. Horst and H. Tuy. *Global Optimization*. Springer Verlag, Berlin, 1990.

[62] P. Kall and S.W. Wallace. *Stochastic Programming*. John Wiley & Sons, New York, USA, 1994.

[63] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[64] W. Karush. Minima of functions of several variables with inequalities as side conditions. Master's thesis, Department of Mathematics, University of Chicago, Illinois, USA, 1939.

[65] J.E. Kelley. The cutting plane method for solving convex programs. *Journal of the SIAM*, 8:703–712, 1960.

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

[67] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220,4598:671–680, 1983.

[68] G.R. Kocis and I.E. Grossmann. Relaxation strategy for the structural optimization of process flowsheets. *Industrial and Engineering Chemistry Research*, 26:1869–1880, 1987.

[69] G.R. Kocis and I.E. Grossmann. Computational experience with DICOPT solving MINLP problems in process systems engineering. *Computers and Chemical Engineering*, 13:307–315, 1989.

[70] H.W. Kuhn and A.W. Tucker. Nonlinear programming. In *Proceedings of the second Berkeley symposium on mathematical statistics and probability*, California, USA, 1951. University of Berkeley Press.

[71] A.H. Land and A.G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[72] J. Lee and D. Wilson. Polyhedral methods for piecewise-linear functions I: the lambda method. *Discrete Applied Mathematics*, 108:269–285, 2001.

[73] S. Leyffer. *Deterministic Methods for Mixed Integer Nonlinear Programming*. PhD thesis, University of Dundee, United Kingdom, 1993.

[74] S. Leyffer. User manual for MINLP_BB. Technical Report NA XXX, University of Dundee, United Kingdom, 1999.

[75] J.T. Linderoth and M.W.P. Savelsbergh. A computational study of search strategy for mixed integer programming. *INFORMS Journal on computing*, 11:173–187, 1999.

[76] J.D.C. Little, K.G. Murty, D.W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations Research*, 28:497–520, 1963.

[77] M. Locatelli. Simulated annealing algorithms for continuous global optimization: Convergence conditions. *Journal of Optimization Theory and Applications*, 10(4):121–133, 2000.

[78] D.G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, Massachussets, USA, second edition, 1984.

[79] Z.Q. Luo, J.S. Pang, and D. Ralph. *Mathematical Programs with Equilibrium Constraints*. Cambridge University Press, Cambridge, 1996.

[80] A. Martin, M. Möller, and S. Moritz. Mixed integer models for the stationary case of gas network optimization. *Mathematical Programming*, 105:563–582, 2006.

[81] G.P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I - convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.

[82] M. Metcalf and J. Reid. *Fortran 90/95 explained*. Oxford Science Publication, 1996.

[83] M. Möller. *Mixed Integer Models for the Optimisation of Gas Networks*. PhD thesis, Technische Universität Darmstadt, 2004.

[84] J.A. Momoh, R.J. Koessler, M.S. Bond, B. Stott, D. Sun, A. Papalexopoulos, and P. Ristanovic. Challenges to Optimal Power Flow. *IEEE Transaction on Power Systems*, 12:444–455, 1997.

[85] R.E. Moore. *Interval analysis*. Prentice-Hall, New Jersey, 1979.

[86] S.G. Nash and A. Sofer. *Linear and Nonlinear Programming*. McGraw-Hill International Editions, 1996.

[87] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, USA, 1988.

[88] A. Neumaier. *Interval methods for systems of equations*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1990.

[89] A. Neumaier. Complete search in continuous global optimization and constraint satisfaction. *Acta Numerica 2004*, pages 271–369, 2004.

[90] J. Nocedal and S.W. Wright. *Numerical Optimization*. Series in Operation Research. Springer Verlag, Heidelberg, Berlin, New York, 1999.

[91] I. Nowak. *Relaxation and Decomposition Methods for Mixed Integer Nonlinear Programming*, volume 152 of *International Series of Numerical Mathematics*. Birkhäuser, 2000.

[92] J. Outrata, M. Kocvara, and J. Zowe. *Nonsmooth Approach to Optimization Problems with Equilibrium Constraints*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[93] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

[94] P. Penfield, R. Spence, and S. Duinker. *Tellegen's theorem and electrical networks*. M.I.T. Press, Cambridge, London, 1970.

[95] L. Platbrood. Optimal Power Flow: Etude comparative de méthodes d'optimisation pour la résolution de problèmes classiques. Master's thesis, FUNDP, Belgium, 2002.

[96] P. Polisetty and E. Gatzke. Piecewise linear relaxation techniques for solution of non-convex nonlinear programming problems. Technical report, Department of Chemical Engineering, University of South Carolina, USA, 2005.

[97] R. Pörn and T. Westerlund. A cutting plane method for minimizing pseudo-convex functions in the mixed integer case. *Computers and Chemical Engineering*, 16:2655–2665, 2000.

[98] I. Quesada and I.E. Grossmann. An LP/NLP based branch-and-bound algorithm for convex MINLP optimization problems. *Computers and Chemical Engineering*, 16:937–947, 1992.

[99] A. Ruszczynski. *Nonlinear Optimization*. Princeton University Press, Princeton, United Kingdom, 2006.

[100] H.S. Ryoo and N.V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8:107–139, 1996.

[101] N.V. Sahinidis. *BARON: Branch And Reduce Optimization Navigator, User's Manual, Version 4.0*, 1999-2000. Available at http://archimedes.scs.uiuc.edu/baron.html.

[102] C. Sainvitu. *Filter-Trust-Region Methods for Nonlinear Optimization*. PhD thesis, FUNDP, Belgium, 2007.

[103] R.H.W. Sargent. Reduced-gradient and projection methods for nonlinear programming. In P.E. Gill and W. Murray, editors, *Numerical Methods for Constrained Optimization*, pages 149–174. Academic Press, London, United Kingdom, 1974.

[104] A. Sartenaer. Some recent developments in nonlinear optimization algorithms. In *Actes des journées MODE*, volume 13, pages 41–64, 2003.

[105] SBB. User's guide available at http://www.gams.com/dd/docs/solvers/sbb.pdf.

[106] H. Schichl. VGTL (Vienna Graph Template Library) Version 1.0, Reference Manual. Technical report, University of Vienna, Department of Mathematics, 2003.

[107] H. Schichl. Global Optimization in the COCONUT project. In *Proceedings of the Dagstuhl Seminar Numerical Software with Result Verification*, Springer Lecture Notes in Computer Science 2991, Berlin, 2004. Springer.

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

[109] M. Tawarmalani and N.V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[110] J.A. Tomlin. A suggested extension of special ordered sets to non-separable non-convex programming problems. *Annals of Discrete Mathematics*, 11:359–370, 1981.

[111] R. Vaidyanathan and M. El-Halwagi. *Global Optimization of nonconvex MINLP's by interval analysis*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[112] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38:49–95, 1996.

[113] R.J. Vanderbei. LOQO: An interior-point code for quadratic programming. Technical report, Statistics and Operations Research, Princeton University, 1994. Revised: November 30, 1998.

[114] R.J. Vanderbei. *Linear Programming - Foundations and Extensions*. Kluwer Academic Publishers, Dordrecht, The Netherlands, second edition, 2001.

[115] J. Viswanathan and I.E. Grossmann. A combined penalty function and outer approximation method for MINLP optimization. *Computers and Chemical Engineering*, 14(7):769–782, 1990.

[116] R.A. Waltz and J. Nocedal. KNITRO user's manual. Technical Report OTC 2003/05, Optimization Technology Center, Northwestern University, Evanston, Illinois, USA, 2003.

[117] R. Weismantel. Lectures on mixed integer nonlinear programming (MINLP). CORE Lecture Series, Louvain-la-Neuve, UCL.

[118] T. Westerlund and K. Lundqvist. Alpha-ECP, version 5.101, an interactive MINLP-solver based on the extended cutting plane method. Available for download at http://web.abo.fi/ twesterl/A-ECPManual.pdf.

[119] T. Westerlund and F. Petersson. An extended cutting plane method for solving convex MINLP problems. *Computers and Chemical Engineering Suppl.*, 19:131–136, 1995.

[120] T. Westerlund and R. Pörn. Solving pseudo-convex mixed integer problems by cutting plane techniques. *Optimization and Engineering*, 3:253–280, 2002.

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

[122] T. Wildi and G. Sybille. *Electrotechnique*. De Boeck, Bruxelles, 4th edition, 2005.

[123] H.P. Williams. *Model Solving in Mathematical Programming*. John Wiley and Sons, 1993.

[124] H.P. Williams. *Model Building in Mathematical Programming*. John Wiley and Sons, 4th edition, 2005.

[125] R.B. Wilson. *A simplicial algorithm for concave programming*. PhD thesis, Harvard University, Massachussetts, USA, 1963.

[126] H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors. *Handbook of Semidefinite Programming: Theory, Algorithms and Applications*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[127] S.J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, Philadelphia, USA, 1997.

[128] A. Wächter and L.T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[129] G. M. Ziegler. *Lectures on Polytopes*. Springer-Verlag, 1995.

Appendix A

This appendix illustrates the fact that the linear outer approximation based on SOS for the bilinear product can be better than the one given by McCormick's inequalities (4.12) since it is able to exploit the multiple presence of a same variable in the problem. Let us consider the problem (NLP_1) below, of three variables x, y and z where the variable z intervenes linearly in the problem while x and y appear in linear functions, in a bilinear product and also in square functions. The solution of this problem is given by $(x^*, y^*, z^*) = (0.5,\ -1.5,\ 0.25)$ and produces an optimum value equal to 4.25.

$$(NLP_1)\begin{cases} \min & y^2 + xy + 5x + z, \\ \text{s.t.} & x^2 - z \leq 0, \\ & -x - z \leq -0.75, \\ & -x + y \leq -2, \\ & -1 \leq x \leq 2, \\ & -3 \leq y \leq 0, \\ & 0.25 \leq z \leq 4. \end{cases}$$

Note first that only the *underestimations* of the nonlinear components are needed to solve the problem because these components only appear in an inequality constraint and in the objective function of a minimization problem. In this case, the overestimations of the nonlinear components are useless. Accordingly, we talk about *linear underestimation problems* instead of linear outer approximation problems.

The linear underestimation problem based on SOS is built by applying to problem (NLP_1) the formulation $(\widetilde{OP})$ of Section 3.2.5 without overestimating the nonlinear components. Only one set λ is needed since the same set can be employed to underestimate the square and the bilinear functions, as explained in Section 3.2.3. Three equally spaced breakpoints are used in each dimension: $\{x_i\}_{i=1}^3 = \{-1,\ 0.5,\ 2\}$ for x and $\{y_j\}_{j=1}^3 = \{-3,\ -1.5,\ 0\}$ for y, which gives nine variables $\lambda_{i,j}$. Three new variables $w_{x^2},\ w_{y^2}$ and w_{xy} are also added to underestimate x^2, y^2 and xy, respectively. The linear underestimation problem built in this way is given by (SOS_1).

$$(SOS_1)\begin{cases} \min & w_{y^2} + w_{xy} + 5x + z, \\ \text{s.t.} & w_{x^2} - z \leq 0, \\ & -x - z \leq -0.75, \\ & -x + y \leq -2, \\ & w_{x^2} \geq \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{i,j}\, x_i^2 \; - \frac{9}{16}, \\ & w_{y^2} \geq \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{i,j}\, y_j^2 \; - \frac{9}{16}, \\ & w_{xy} = \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{i,j}\, x_i y_j, \\ & x = \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{i,j} x_i, \\ & y = \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{i,j} y_j, \\ & 1 = \sum_{i=1}^{3}\sum_{j=1}^{3} \lambda_{i,j}, \\ & 0 \leq \lambda_{i,j}, \quad 1 \leq i \leq 3,\ 1 \leq j \leq 3, \\ & -1 \leq x \leq 2, \\ & -3 \leq y \leq 0, \\ & 0.25 \leq z \leq 4. \end{cases}$$

At the solution of this problem, the components x, y and z are given by (x^*, y^*, z^*), that is, by $(0.5,\ -1.5,\ 0.25)$ like at the solution of (NLP_1). Because of the constraints of the problem, the variables $\lambda_{i,j}$ directly fulfill the SOS condition $(\lambda_{2,2} = 1)$. The optimum value of (SOS_1) is equal to 3.6875 which corresponds, as expected, to a lower bound for (NLP_1). The value of the underestimation of xy is given by $w^*_{xy} = -0.75$ which coincides with the exact value of x^*y^*.

Let us now examine the problem obtained by underestimating the bilinear product with McCormick's inequalities and the square functions by their tangent lines at the breakpoints. The resulting underestimation problem is given by $(McCor_1)$.

$$(McCor_1)\begin{cases} \min & w_{y^2} + w_{xy} + 5x + z, \\ \text{s.t.} & w_{x^2} - z \leq 0, \\ & -x - z \leq -0.75, \\ & -x + y \leq -2, \\ & w_{x^2} \geq 2x_i(x - x_i) + x_i^2, \quad 1 \leq i \leq 3, \\ & w_{y^2} \geq 2y_i\,(y - y_i) + y_i^2, \quad 1 \leq i \leq 3, \\ & w_{xy} \geq -y - 3x - 3, \\ & w_{xy} \geq 2y, \\ & -1 \leq x \leq 2, \\ & -3 \leq y \leq 0, \\ & 0.25 \leq z \leq 4. \end{cases}$$

The components x, y and z at the solution of problem $(McCor_1)$ are again equal to x^*, y^* and z^*, respectively, but the optimum value of the problem is different from the one of the two previous problems since it is equal to 2. This is a weaker bound than the one found with the underestimations based on SOS. The latter approach is thus better. Contrary to the underestimation based on SOS, McCormick's inequalities produce, for a same value of x and y, an underestimation of the bilinear product xy which is not exact ($w_{xy} = -3$ instead of -0.75 as mentioned earlier). In fact, the use of the same set λ for the three nonlinear functions of problem (NLP_1) prevents the solution of problem $(McCor_1)$ from being feasible for the problem (SOS_1), which allows us to have a better underestimation problem.

Appendix B

The general formulation of the discrete version of the test problems *pb* is given in this appendix. In the following formulations, the set $D_{11}(x)$ corresponds to a discrete set composed of eleven equally spaced values between the original lower and upper bounds on the variable x. The continuous version of these problems is obtained by removing the constraints implying the sets $D_{11}(x)$.

$$(pb0)\begin{cases} \min & x_1 \sin(x_4), \\ \text{s.t.} & 4x_1 - x_2^2 - 0.2x_2x_4\sin(x_3) \leq 1, \\ & x_2 - 0.5x_2x_4\cos(x_3) \leq 2, \\ & 0 \leq x_1 \leq 4, \\ & 0 \leq x_2 \leq 3, \\ & 0 \leq x_i \leq 2\pi, \quad 3 \leq i \leq 4, \\ & x_4 \in D_{11}(x_4). \end{cases}$$

$$(pb1)\begin{cases} \min & x_1 \sin(x_4), \\ \text{s.t.} & x_1 + x_2^2 - 2x_2x_4\sin(x_3) \leq -0.5, \\ & x_2 - 3x_2x_4\cos(x_3) \leq -4, \\ & 0 \leq x_1 \leq 2, \\ & 1 \leq x_2 \leq 4, \\ & 0 \leq x_3 \leq 2\pi, \\ & -2 \leq x_4 \leq 0, \\ & x_4 \in D_{11}(x_4). \end{cases}$$

$$(pb2)\begin{cases} \min & (x_2 - 1)^2, \\ \text{s.t.} & x_1 - x_4^2 - 0.5x_3x_4\cos(x_5 - x_6) = 0.2, \\ & x_2 - 0.3x_3^2 - x_3x_4\sin(x_6 - x_5) = 0.1, \\ & 0 \leq x_i \leq 2, \quad 1 \leq i \leq 6, \\ & x_4 \in D_{11}(x_4). \end{cases}$$

$$(pb3)\begin{cases} \min & (x_2 - 1)^2, \\ \text{s.t.} & x_1 - x_4^2 - 0.5x_3x_4\cos(x_5 - x_6) = 2.8, \\ & x_2 - 0.3x_3^2 - x_3x_4\sin(x_6 - x_5) = 0.2, \\ & 1.5 \leq x_i \leq 6, \quad 1 \leq i \leq 4, \\ & 0 \leq x_i \leq 2, \qquad 5 \leq i \leq 6, \\ & x_4 \in D_{11}(x_4). \end{cases}$$

$$(pb4)\begin{cases}\min & (x_2-1)^2+(x_7-0.3)^2,\\ \text{s.t.} & x_1-x_4^2-0.5x_3x_8\cos(x_5-x_6)=0.1,\\ & x_2-0.3x_7^2+x_7x_4\sin(x_{12}-x_{10})=0.2,\\ & x_1-x_8^2-0.5x_8x_9\cos(x_{12}-x_{10})=0.3,\\ & x_2-0.3x_3^2+x_4x_{11}\sin(x_5-x_6)=3,\\ & 0\le x_i\le 2,\quad 1\le i\le 12,\\ & x_i\in D_{11}(x_i),\quad i\in\{4,8\}.\end{cases}$$

$$(pb5)\begin{cases}\min & (x_2-1)^2+(x_7-0.3)^2,\\ \text{s.t.} & x_1-x_4^2-5x_3x_8\cos(x_5-x_6)=1,\\ & x_2-3x_7^2+x_7x_4\sin(x_{12}-x_{10})=1,\\ & x_1-x_8^2-5x_8x_9\cos(x_{12}-x_{10})=1,\\ & x_2-3x_3^2+x_4x_{11}\sin(x_5-x_6)=1,\\ & 2.5\le x_i\le 6.5,\quad i\in\{1,2,3,4,7,8,9,11\},\\ & 0\le x_i\le 2,\quad i\in\{5,6,10,12\},\\ & x_i\in D_{11}(x_i),\quad i\in\{4,8\}.\end{cases}$$

$$(pb6)\begin{cases}\min & (x_2-1.1)^2+(x_4-1.4)^2+(x_7-0.1)^2+(x_9-0.1)^2,\\ \text{s.t.} & x_1-x_4^2-0.5x_3x_8\cos(x_5-x_6)-0.2x_3^2-0.5x_3x_9\cos(x_{12}-x_{10})=0.1,\\ & x_2-0.3x_7^2-x_4x_7\sin(x_{10}-x_{12})-0.5x_3^2-x_7x_{11}\sin(x_6-x_5)=0.2,\\ & x_1-x_8^2-0.5x_8x_9\cos(x_{12}-x_{10})-x_4^2-0.5x_3x_4\cos(x_5-x_6)=0.5,\\ & x_2-0.3x_3^2-x_4x_{11}\sin(x_6-x_5)-0.3x_7^2-x_7x_8\sin(x_{10}-x_{12})=0.4,\\ & 0\le x_i\le 2,\quad 1\le i\le 12,\\ & x_i\in D_{11}(x_i),\quad i\in\{3,4,8\}.\end{cases}$$

$$(pb7)\begin{cases}\min & (x_2-1.1)^2+(x_4-1.4)^2+(x_7-0.1)^2+(x_9-0.1)^2,\\ \text{s.t.} & x_1-x_4^2-0.1x_3x_8\cos(x_5-x_6)-1.1x_3^2-0.1x_3x_9\cos(x_{12}-x_{10})=0.2,\\ & x_2-0.9x_7^2-x_4x_7\sin(x_{10}-x_{12})-0.1x_3^2-x_7x_{11}\sin(x_6-x_5)=0.2,\\ & x_1-x_8^2-0.1x_8x_9\cos(x_{12}-x_{10})-x_4^2-0.1x_3x_4\cos(x_5-x_6)=0.2,\\ & x_2-0.9x_3^2-x_4x_{11}\sin(x_6-x_5)-0.9x_7^2-x_7x_8\sin(x_{10}-x_{12})=1.5,\\ & 0.5\le x_i\le 6,\quad 1\le i\le 4 \text{ or } 7\le i\le 12,\\ & 0\le x_i\le 2,\quad 5\le i\le 6,\\ & x_i\in D_{11}(x_i),\quad i\in\{3,4,8\}.\end{cases}$$

$$(pb8)\begin{cases}\min & (x_2-1)^2+(x_7-0.3)^2,\\ \text{s.t.} & x_1-x_4^2-0.5x_3x_7x_8\cos(x_9-x_{10})=0.1,\\ & x_2-0.3x_7^2-x_2x_4x_7\sin(x_{11}-x_{12})=0.2,\\ & x_1-x_8^2-0.5x_2x_5x_8\cos(x_{12}-x_{11})=0.3,\\ & x_2-0.3x_3^2-x_4x_6x_8\sin(x_{10}-x_9)=0.4,\\ & 0\le x_i\le 2,\quad 1\le i\le 12,\\ & x_i\in D_{11}(x_i),\quad i\in\{7,8\}.\end{cases}$$

$$(pb9)\begin{cases}\min & (x_2-1)^2+(x_7-1)^2,\\ \text{s.t.} & x_1-x_4^2-2x_3x_7x_8\cos(x_9-x_{10})=3,\\ & x_2-x_7^2-x_2x_4x_7\sin(x_{11}-x_{12})=2.1,\\ & x_1-x_8^2-2x_2x_5x_8\cos(x_{12}-x_{11})=3.5,\\ & x_2-x_3^2-x_4x_6x_8\sin(x_{10}-x_9)=3.2,\\ & 2\le x_i\le 6, \quad 1\le i\le 4,\\ & 1\le x_i\le 4, \quad 5\le i\le 8,\\ & 0\le x_i\le 2, \quad 9\le i\le 12,\\ & x_i\in D_{11}(x_i), \quad i\in\{7,8\}.\end{cases}$$

$$(pb10)\begin{cases}\min & (x_2-1)^2+(x_7-0.3)^2,\\ \text{s.t.} & x_1-x_4^2-0.5x_3x_5x_7\cos(0.5+x_9-x_{10})=0.1,\\ & x_2-0.3x_7^2-x_2x_4x_7\sin(0.5+x_9-x_{10})=0.2,\\ & x_1-x_8^2-0.5x_5x_7x_8\cos(0.5+x_{10}-x_9)=0.3,\\ & x_2-0.3x_3^2-x_4x_5x_6\sin(0.5+x_{10}-x_9)=0.4,\\ & 0\le x_i\le 2, \quad 1\le i\le 10,\\ & x_i\in D_{11}(x_i), \quad i\in\{6,7\}.\end{cases}$$

$$(pb11)\begin{cases}\min & (x_2-1)^2+(x_7-0.3)^2,\\ \text{s.t.} & x_1-x_4^2-0.89x_3x_5x_7\cos(0.89+x_9-x_{10})=2.1,\\ & x_2-0.3x_7^2-x_2x_4x_7\sin(0.89+x_9-x_{10})=1.4,\\ & x_1-x_8^2-0.89x_5x_7x_8\cos(0.89+x_{10}-x_9)=0.7,\\ & x_2-0.3x_3^2-x_4x_5x_6\sin(0.89+x_{10}-x_9)=0,\\ & -3\le x_i\le 0, \quad 1\le i\le 8,\\ & 0\le x_i\le 2, \quad 9\le i\le 10,\\ & x_i\in D_{11}(x_i), \quad i\in\{6,7\}.\end{cases}$$

$$(pb12)\begin{cases}\min & (x_2-1.1)^2+(x_4-1.4)^2+(x_7-0.1)^2+(x_9-0.1)^2+(x_{14}-0.2)^2\\ & +(x_{16}-0.5)^2,\\ \text{s.t.} & x_1-x_4^2-0.5x_9x_{12}\cos(x_{19}-x_{20})-0.2x_3^2-0.5x_3x_6\cos(x_{17}-x_{18})=0.1,\\ & x_2-0.3x_{16}^2+x_7x_{14}\sin(x_{19}-x_{20})-0.5x_3^2+x_7x_{12}\sin(x_{17}-x_{18})=0.2,\\ & x_1-x_8^2-0.5x_5x_{15}\cos(x_{19}-x_{20})-x_{14}^2-0.5x_4x_5\cos(x_{17}-x_{18})=0.3,\\ & x_2-0.3x_3^2+x_5x_{11}\sin(x_{19}-x_{20})-0.3x_{15}^2+x_8x_{13}\sin(x_{17}-x_{18})=0.4,\\ & x_3-x_4^2-0.5x_3x_8\cos(x_{21}-x_{22})-0.2x_3^2-0.5x_5x_9\cos(x_{23}-x_{24})=0.5,\\ & x_4-0.3x_{13}^2+x_7x_{16}\sin(x_{21}-x_{22})-0.5x_3^2+x_7x_{15}\sin(x_{23}-x_{24})=0.6,\\ & x_3-x_8^2-0.5x_{11}x_{12}\cos(x_{21}-x_{22})-x_4^2-0.5x_3x_{13}\cos(x_{23}-x_{24})=0.7,\\ & x_4-0.3x_{14}^2+x_4x_{10}\sin(x_{21}-x_{22})-0.3x_7^2+x_6x_{14}\sin(x_{23}-x_{24})=0.8,\\ & 0\le x_i\le 1, \quad i\in\{1,2,5,6,7,8,9,10,13,14,15,16\},\\ & 1\le x_i\le 10, \quad 3\le i\le 4,\\ & 1\le x_i\le 2, \quad 11\le i\le 12,\\ & 0\le x_i\le 2, \quad 17\le i\le 24,\\ & x_i\in D_{11}(x_i), \quad i\in\{5,6,7,10,12,13\}.\end{cases}$$

$$(pb13)\begin{cases}
\min & (x_2-1.1)^2+(x_4-1.4)^2+(x_7-0.1)^2+(x_9-0.1)^2+(x_{14}-0.2)^2 \\
& +(x_{16}-0.5)^2, \\
\text{s.t.} & x_1-x_4^2)-0.5x_9x_{12}\cos(x_{19}-x_{20})-0.2x_3^2-0.5x_3x_6\cos(x_{17}-x_{18}=0.1, \\
& x_2-0.3x_{16}^2+x_7x_{14}\sin(x_{19}-x_{20})-0.5x_3^2+x_7x_{12}\sin(x_{17}-x_{18})=0.2, \\
& x_1-x_8^2-0.5x_5x_{15}\cos(x_{19}-x_{20})-x_{14}^2-0.5x_4x_5\cos(x_{17}-x_{18})=0.3, \\
& x_2-0.3x_3^2+x_5x_{11}\sin(x_{19}-x_{20})-0.3x_{15}^2+x_8x_{13}\sin(x_{17}-x_{18})=0.4, \\
& x_3-x_4^2-0.5x_3x_8\cos(x_{21}-x_{22})-0.2x_3^2-0.5x_5x_9\cos(x_{23}-x_{24})=0.5, \\
& x_4-0.3x_{13}^2+x_7x_{16}\sin(x_{21}-x_{22})-0.5x_3^2+x_7x_{15}\sin(x_{23}-x_{24})=0.6, \\
& x_3-x_8^2-0.5x_{11}x_{12}\cos(x_{21}-x_{22})-x_4^2-0.5x_3x_{13}\cos(x_{23}-x_{24})=0.7, \\
& x_4-0.3x_{14}^2+x_4x_{10}\sin(x_{21}-x_{22})-0.3x_7^2+x_6x_{14}\sin(x_{23}-x_{24})=0.8, \\
& 0\le x_i\le 2, \quad i\in\{1,2,6,7,9,10,13,14,15,16,17,18,19,20,21,22,23,24\}, \\
& 1\le x_i\le 10, \quad 3\le i\le 4, \\
& 1\le x_i\le 2, \quad 11\le i\le 12, \\
& x_i\in D_{11}(x_i), \quad i\in\{3,6,7,15\}, \\
& x_5,x_8 \text{ binary}.
\end{cases}$$

Note that for this latter problem, in the continuous version, the lower and upper bounds on x_5 and x_8 are given respectively by 0 and 2.

Appendix C

We present here the numerical results obtained with the different techniques detailed in Chapters 5, 6 and 7. A table is associated to each specific method. The top part of the tables is related to the continuous problems while the bottom part is concerned with the discrete versions of these problems. For each method and test problem, some of the following information are mentioned depending on their relevance:

- # LP solved: the total number of linear problems solved (this number takes the number of linear problems solved during presolve, range reduction and strong branching phases into account),
- # NLP solved: the total number of nonlinear problems solved,
- # LP presolve: the total number of linear problems solved during presolve,
- # LP range reduction: the total number of linear problems solved during range reduction phases (this number comprises the number of linear problems solved during presolve),
- # LP strong branching: the total number of linear problems solved during strong branching phases,
- reduction percentages: the reduction percentages in the sum of the ranges of the original variables,
- % LP strong branching/LP: the ratio between the number of linear problems solved during strong branching iterations and the total number of linear problems solved,
- % NLP/LP: the ratio between the number of nonlinear problems solved and the one of linear problems solved,
- # nodes in the stack: the maximum number of nodes stored in the stack of open nodes,
- CPU times: the CPU times needed to solve the original problem.

Stars in a field mean that the method cannot solve the problem before the maximum number of linear problems solved (500.000) is reached.

	# LP solved	# NLP solved
pb0	41	20
pb1	81	42
pb2	1367	683
pb3	157	78
pb4	5063	2531
pb5	1445	722
pb6	3225	1612
pb7	15315	7657
pb8	32763	16381
pb9	18931	9465
pb10	12817	6408
pb11	4523	2261
pb12	5661	2830
pb13	373313	186656
TVC1	*****	*****
TVC2	*****	*****
TVC3	185735	92867
TVC4	*****	*****
TVC5	*****	*****
TVC6	*****	*****
pb0	35	17
pb1	198	125
pb2	1441	720
pb3	135	67
pb4	5145	2572
pb5	7307	3653
pb6	6861	3430
pb7	361	180
pb8	363169	181564
pb9	30853	15426
pb10	23329	11664
pb11	11255	5628
pb12	150373	75186
pb13	38209	19616
TVC1	194767	97383
TVC2	*****	*****
TVC3	88825	44412
TVC4	*****	*****
TVC5	*****	*****
TVC6	*****	*****

Table 8.5: Results obtained with the basic method.

	# LP solved	# LP presolve	reduction percentages	# NLP solved
pb0	62	25	21.3	18
pb1	100	25	43.4	37
pb2	1656	25	4.3	815
pb3	168	25	19.9	71
pb4	5466	53	20.5	2706
pb5	804	53	49.3	375
pb6	3288	71	5.5	1608
pb7	15096	71	24.9	7512
pb8	89338	61	1.7	44638
pb9	7752	61	16.4	3845
pb10	16874	59	23.1	8407
pb11	6876	59	22.5	3408
pb12	5226	139	57.1	2543
pb13	478860	139	25.8	239361
TVC1	*****	141	63.6	*****
TVC2	*****	145	80.9	*****
TVC3	79540	221	87.3	39659
TVC4	*****	221	89.5	*****
TVC5	*****	315	85.4	*****
TVC6	*****	329	91.4	*****
pb0	40	25	23.9	7
pb1	201	25	44.3	119
pb2	1504	25	6.0	739
pb3	148	25	20.6	61
pb4	3248	53	23.8	1957
pb5	458	53	50.4	202
pb6	4400	71	6.4	2164
pb7	783	71	27.1	357
pb8	*****	61	2.7	*****
pb9	20764	61	16.4	10351
pb10	26848	59	23.1	13394
pb11	2844	59	23.7	1392
pb12	26390	139	57.6	13125
pb13	4222	139	31.5	2220
TVC1	133028	141	63.7	66443
TVC2	456644	145	80.9	228249
TVC3	18234	221	88.9	9006
TVC4	*****	221	89.5	*****
TVC5	168664	315	86.3	84174
TVC6	57742	329	92.6	28706

Table 8.6: Results obtained with a one sweep presolve.

	# LP solved	# LP presolve	reduction percentages	# NLP solved
pb0	171	100	25.2	35
pb1	384	325	53.6	29
pb2	1681	50	4.3	815
pb3	218	75	19.9	71
pb4	5660	371	21.3	2644
pb5	1014	371	50.3	321
pb6	3656	497	6.1	1579
pb7	15383	426	25.1	7478
pb8	85229	366	1.9	42431
pb9	489	488	53.6	1
pb10	16933	118	23.1	8407
pb11	8044	1475	32.3	3284
pb12	7874	1807	57.8	3033
pb13	473179	556	25.8	236311
TVC1	*****	564	65.5	*****
TVC2	*****	725	86.2	*****
TVC3	19493	1326	96.1	9083
TVC4	*****	1105	90.6	*****
TVC5	*****	3150	86.3	*****
TVC6	*****	1316	92.0	*****
pb0	115	100	27.4	7
pb1	436	325	54.2	56
pb2	1529	50	6.0	739
pb3	173	50	20.6	61
pb4	4952	159	24.0	2396
pb5	570	371	53.3	99
pb6	4608	213	7.3	2197
pb7	1093	426	27.2	335
pb8	*****	305	3.0	*****
pb9	35752	305	53.9	17723
pb10	26907	118	23.1	13394
pb11	4590	1003	32.9	1793
pb12	23846	1807	58.2	11019
pb13	4500	417	31.5	2220
TVC1	61977	564	65.7	30706
TVC2	245615	870	86.3	122732
TVC3	4012	1105	97.5	1453
TVC4	*****	884	90.7	*****
TVC5	34786	2205	87.7	16290
TVC6	33666	987	93.2	16339

Table 8.7: Results obtained with a full presolve.

	# LP solved	# NLP solved	% NLP/LP	CPU times
pb0	45	22	48.9	0.154
pb1	107	55	51.4	0.419
pb2	2131	1065	50.0	7.590
pb3	159	79	49.7	0.562
pb4	9303	4651	50.0	62.050
pb5	3405	1702	50.0	21.470
pb6	3533	1766	50.0	32.553
pb7	16245	8122	50.0	164.018
pb8	111721	55860	50.0	858.543
pb9	20302	10152	50.0	144.914
pb10	20685	10342	50.0	151.166
pb11	7241	3620	50.0	55.544
pb12	11131	5565	50.0	227.770
pb13	467225	233612	50.0	10463.566
TVC1	*****	*****	*****	*****
TVC2	*****	*****	*****	*****
TVC3	140865	70432	50.0	5969.975
TVC4	*****	*****	*****	*****
TVC5	*****	*****	*****	*****
TVC6	*****	*****	*****	*****
pb0	63	31	49.2	0.215
pb1	133	67	50.4	0.492
pb2	2115	1057	50.0	7.583
pb3	135	67	49.6	0.501
pb4	15389	7694	50.0	98.798
pb5	3009	1504	50.0	19.741
pb6	65800	32905	50.0	673.784
pb7	377	188	50.0	3.810
pb8	*****	*****	*****	*****
pb9	62149	31071	50.0	446.907
pb10	113846	56924	50.0	802.281
pb11	3806	1908	50.1	31.330
pb12	*****	*****	*****	*****
pb13	*****	*****	*****	*****
TVC1	108861	54430	50.0	2429.940
TVC2	*****	*****	*****	*****
TVC3	62045	31022	50.0	3199.969
TVC4	*****	*****	*****	*****
TVC5	*****	*****	*****	*****
TVC6	*****	*****	*****	*****

Table 8.8: Results obtained without presolve but with a specific ranking to consider the variables.

	# LP solved	# LP presolve	reduction percentages	# NLP solved	% NLP/LP
pb0	105	34	25.0	35	33.3
pb1	100	9	9.4	45	45.0
pb2	1640	9	4.3	815	49.7
pb3	152	9	19.9	71	46.7
pb4	5430	19	13.1	2705	49.8
pb5	907	72	50.1	417	46.0
pb6	3604	19	5.2	1792	49.7
pb7	15115	90	24.9	7512	49.7
pb8	87638	19	1.1	43809	50.0
pb9	18950	19	0.0	9465	49.9
pb10	16891	76	23.1	8407	49.8
pb11	6888	17	16.4	3693	52.8
pb12	5889	178	57.8	2855	48.4
pb13	478899	178	25.8	239361	50.0
TVC1	*****	168	65.5	*****	*****
TVC2	*****	319	86.2	*****	*****
TVC3	30206	481	95.9	14862	49.2
TVC4	*****	260	90.5	*****	*****
TVC5	*****	370	86.1	*****	*****
TVC6	*****	386	92.0	*****	*****
pb0	49	34	27.3	7	14.3
pb1	143	9	9.4	77	53.8
pb2	1438	9	6.0	714	49.7
pb3	157	34	20.6	61	38.9
pb4	15100	19	17.0	7540	49.9
pb5	295	72	50.8	111	37.6
pb6	4150	19	6.4	2065	49.8
pb7	802	90	27.1	357	44.5
pb8	*****	19	1.4	*****	*****
pb9	30872	19	0	15426	50.0
pb10	26865	76	23.1	13394	49.9
pb11	6960	17	16.8	3471	49.9
pb12	22467	178	58.0	11144	49.6
pb13	4261	178	31.5	2220	52.1
TVC1	67285	168	65.4	33558	49.9
TVC2	242840	319	86.2	121260	49.9
TVC3	3644	481	97.4	1581	43.4
TVC4	*****	260	90.4	*****	*****
TVC5	148459	370	87.1	74044	49.9
TVC6	28065	386	93.2	12839	45.7

Table 8.9: Results obtained with an adaptable presolve.

	# LP solved	# LP range reduction	# NLP solved	% NLP/LP	CPU times
pb0	112	79	16	14.2	0.260
pb1	104	45	30	28.8	0.277
pb2	5636	3051	1405	24.9	12.872
pb3	262	135	65	24.8	0.662
pb4	9710	7123	1585	16.3	35.024
pb5	396	281	62	15.7	1.593
pb6	5449	3154	1174	21.5	31.865
pb7	19859	12910	3796	19.1	109.898
pb8	37107	19142	9484	25.6	189.060
pb9	3813	2162	917	24.0	18.081
pb10	19928	12501	4086	20.5	87.069
pb11	7996	4871	1747	21.8	39.612
pb12	2181	1748	236	10.8	27.416
pb13	*****	*****	*****	*****	*****
TVC1	*****	*****	*****	*****	*****
TVC2	*****	*****	*****	*****	*****
TVC3	1339	1260	49	3.7	42.102
TVC4	113739	105020	7547	6.6	3445.432
TVC5	*****	*****	*****	*****	*****
TVC6	*****	*****	*****	*****	*****
pb0	63	52	6	9.5	0.118
pb1	131	62	36	27.5	0.336
pb2	3237	1332	995	30.7	8.336
pb3	197	94	55	27.9	0.502
pb4	11388	6935	2435	21.4	43.030
pb5	257	186	38	14.8	0.931
pb6	6145	3614	1325	21.6	29.253
pb7	1353	827	299	22.1	8.180
pb8	198817	121198	43359	21.8	813.735
pb9	33668	26405	4916	14.6	120.343
pb10	51816	35617	9560	18.4	203.367
pb11	7349	4142	1825	24.8	36.846
pb12	33407	26662	3847	11.5	423.871
pb13	8093	6424	960	11.9	79.982
TVC1	40446	34389	4205	10.4	479.115
TVC2	72270	65331	5365	7.4	770.639
TVC3	861	832	20	2.3	23.784
TVC4	38792	35693	2540	6.5	1040.867
TVC5	7369	6910	302	4.1	317.225
TVC6	12131	11352	474	3.9	495.180

Table 8.10: Results obtained with range reduction and branching on the largest range.

	# LP solved	# LP range reduction	# NLP solved
pb0	112	79	16
pb1	104	45	30
pb2	3396	1719	905
pb3	262	135	65
pb4	9642	7085	1568
pb5	396	281	62
pb6	5416	3173	1149
pb7	19775	12872	3772
pb8	29731	16780	6962
pb9	10321	7614	1685
pb10	18044	12527	3216
pb11	7622	4873	1562
pb12	2181	1748	236
pb13	*****	*****	*****
TVC1	*****	*****	*****
TVC2	*****	*****	*****
TVC3	1339	1260	49
TVC4	109762	101501	7208
TVC5	*****	*****	*****
TVC6	*****	*****	*****
pb0	63	52	6
pb1	131	62	36
pb2	3171	1368	945
pb3	195	94	54
pb4	10585	6650	2168
pb5	257	186	38
pb6	5940	3653	1217
pb7	1381	872	292
pb8	151206	108299	25935
pb9	27393	21432	4023
pb10	39915	28096	7070
pb11	8875	5370	2018
pb12	15815	13250	1582
pb13	7465	5978	862
TVC1	36144	30705	3875
TVC2	59131	53480	4397
TVC3	861	832	20
TVC4	38752	35711	2509
TVC5	5712	5383	224
TVC6	12239	11466	472

Table 8.11: Results obtained by basing the branching rule on the largest range and by exploiting the best candidate for branching.

	# LP solved	# LP range reduction	# NLP solved
pb0	170	115	27
pb1	188	81	57
pb2	2339	1008	684
pb3	461	228	128
pb4	18853	12806	3407
pb5	405	300	58
pb6	3585	2140	742
pb7	24730	16343	4645
pb8	26159	17818	4920
pb9	37868	27601	6369
pb10	17551	11466	3464
pb11	6510	4359	1251
pb12	2518	2021	279
pb13	*****	*****	*****
TVC1	*****	*****	*****
TVC2	*****	*****	*****
TVC3	1438	1357	55
TVC4	25040	22949	1632
TVC5	136493	128430	6254
TVC6	*****	*****	*****
pb0	110	79	16
pb1	156	72	49
pb2	1224	423	413
pb3	307	168	76
pb4	5479	3820	929
pb5	347	262	46
pb6	4161	2506	852
pb7	2287	1482	463
pb8	177288	120151	32244
pb9	82154	63395	12413
pb10	35518	21803	7480
pb11	3386	2291	635
pb12	3105	2528	334
pb13	15087	12009	1773
TVC1	14556	12599	1426
TVC2	46056	40611	3950
TVC3	850	819	20
TVC4	30548	28201	1986
TVC5	7197	6792	317
TVC6	13035	12284	536

Table 8.12: Results obtained by basing the branching rule on the maximum theoretical approximation errors without weighting them.

	# LP solved	# LP range reduction	# NLP solved	% NLP/LP
pb0	168	115	26	15.5
pb1	188	81	57	30.3
pb2	1367	504	439	32.1
pb3	649	324	178	27.4
pb4	4295	3002	750	17.5
pb5	650	471	97	14.9
pb6	3356	2009	694	20.7
pb7	22385	15322	3897	17.4
pb8	13352	8738	2567	19.2
pb9	46161	33580	7805	16.9
pb10	8403	5012	1815	21.6
pb11	6510	4483	1207	18.5
pb12	1955	1532	233	11.9
pb13	*****	*****	*****	*****
TVC1	373873	321102	31163	8.3
TVC2	125647	111702	9209	7.3
TVC3	1281	1200	50	3.9
TVC4	3295	3024	179	5.4
TVC5	30328	28181	1316	4.3
TVC6	85049	79042	3504	4.1
pb0	101	70	15	14.9
pb1	156	72	49	31.4
pb2	1005	288	365	36.3
pb3	384	205	99	25.8
pb4	2256	1449	432	19.1
pb5	799	604	115	14.4
pb6	5554	3751	959	17.3
pb7	4613	2692	1221	26.5
pb8	113101	75126	21194	18.7
pb9	64432	46985	10851	16.8
pb10	16560	9511	3742	22.6
pb11	3070	1989	632	20.6
pb12	2702	2223	287	10.6
pb13	14489	11539	1692	11.7
TVC1	7144	6139	634	8.9
TVC2	6620	5967	482	7.3
TVC3	701	676	14	2.0
TVC4	2454	2259	128	5.2
TVC5	5380	5063	210	3.9
TVC6	8268	7739	323	3.9

Table 8.13: Results obtained by basing the branching rule on the maximum theoretical weighted approximation errors.

	# LP solved	# LP range reduction	# NLP solved	% NLP/LP
pb0	157	106	25	15.9
pb1	194	81	60	30.9
pb2	2207	1008	621	28.1
pb3	203	90	57	28.1
pb4	4927	3268	921	18.7
pb5	711	528	104	14.6
pb6	3415	2062	695	20.4
pb7	23451	15350	4280	18.3
pb8	19569	13612	3443	17.6
pb9	24817	17836	4280	17.2
pb10	14026	9315	2586	18.4
pb11	4939	3260	965	19.5
pb12	1282	997	156	12.2
pb13	*****	*****	*****	*****
TVC1	374971	321768	31279	8.3
TVC2	95256	83801	7155	7.5
TVC3	1465	1378	54	3.7
TVC4	2611	2440	125	4.8
TVC5	42970	39873	1831	4.3
TVC6	82942	76359	3618	4.4
pb0	99	70	14	14.1
pb1	169	81	52	30.8
pb2	1171	468	359	30.7
pb3	194	97	48	24.7
pb4	4463	2894	894	20.0
pb5	716	509	112	15.6
pb6	7997	4866	1626	20.3
pb7	4076	2427	944	23.2
pb8	182116	129991	30787	16.9
pb9	105518	73303	18727	17.7
pb10	43359	30564	7733	17.8
pb11	1916	1283	383	20.0
pb12	4623	3731	548	11.9
pb13	19331	14941	2444	12.6
TVC1	11081	9790	930	8.4
TVC2	10536	9407	797	7.6
TVC3	828	793	21	2.5
TVC4	2176	2015	107	4.9
TVC5	5269	5004	188	3.6
TVC6	14321	13290	587	4.1

Table 8.14: Results obtained by basing the branching rule on the real approximation errors.

	# LP solved	# LP range reduction	# LP strong branching	# NLP solved	% LP strong branching/LP	% NLP/LP
pb0	116	61	46	7	39.7	6.0
pb1	121	18	88	13	72.7	10.7
pb2	44	9	28	5	63.6	11.4
pb3	78	18	48	8	61.5	10.3
pb4	152	19	122	9	80.3	6.0
pb5	201	72	118	7	58.7	3.5
pb6	522	95	392	29	75.1	5.6
pb7	3963	584	3122	213	78.8	5.4
pb8	879	152	662	46	75.3	5.2
pb9	282	19	246	14	87.2	5.0
pb10	544	127	384	27	70.6	5.0
pb11	1309	136	1072	83	81.9	6.3
pb12	1320	399	886	31	67.1	2.3
pb13	18782	3337	14856	514	79.1	2.7
TVC1	114128	41345	68356	4718	59.9	4.1
TVC2	66474	23679	40448	2178	60.8	3.3
TVC3	990	715	262	12	26.5	1.2
TVC4	1672	845	788	32	47.1	1.9
TVC5	20667	8328	11954	367	57.8	1.8
TVC6	41688	14545	26394	706	63.3	1.7
pb0	68	43	20	3	29.4	4.4
pb1	99	18	68	9	68.7	9.1
pb2	224	9	180	26	80.4	11.6
pb3	127	34	74	13	58.3	10.2
pb4	118	19	92	6	78.0	5.1
pb5	143	72	64	5	44.8	3.5
pb6	654	95	518	37	79.2	5.7
pb7	632	147	446	25	70.6	4.0
pb8	259	38	202	13	78.0	5.0
pb9	471	114	332	21	70.5	4.5
pb10	223	76	134	10	60.0	4.5
pb11	740	102	578	40	78.1	5.4
pb12	1348	295	1006	27	74.6	2.0
pb13	16242	3142	12464	509	76.7	3.1
TVC1	7284	2595	4396	251	60.4	3.4
TVC2	5370	1993	3194	169	59.5	3.1
TVC3	627	520	102	4	16.3	0.6
TVC4	1947	803	1094	41	56.2	2.1
TVC5	6486	2423	3938	104	61.7	1.6
TVC6	6243	2580	3558	92	57.0	1.5

Table 8.15: Results obtained by basing the branching rule on strong branching.

	# LP solved	# LP range reduction	# LP strong branching	# NLP solved	% NLP/LP
pb0	124	79	12	16	12.9
pb1	170	36	32	30	23.6
pb2	44	9	28	5	11.4
pb3	78	18	48	8	10.3
pb4	208	95	56	32	15.4
pb5	222	129	46	24	10.8
pb6	753	418	40	133	18.5
pb7	13795	8976	94	2613	18.9
pb8	367	206	82	45	12.3
pb9	266	19	228	14	5.3
pb10	624	307	142	95	15.2
pb11	1094	599	96	211	19.3
pb12	2498	1869	146	256	10.3
pb13	38077	29300	156	4582	12.0
TVC1	107330	92849	144	8897	8.3
TVC2	72701	63052	252	5851	8.0
TVC3	1060	949	68	29	2.7
TVC4	1738	1523	90	82	4.7
TVC5	22436	20895	172	903	4.0
TVC6	41835	38603	174	1750	4.2
pb0	79	52	12	8	10.1
pb1	186	63	30	50	26.9
pb2	102	9	64	16	15.7
pb3	148	43	74	18	12.2
pb4	120	19	92	7	5.8
pb5	156	91	38	14	9.0
pb6	254	133	50	35	13.8
pb7	1338	729	170	253	18.9
pb8	263	38	202	15	5.7
pb9	424	95	304	20	4.7
pb10	205	76	112	9	4.4
pb11	432	221	82	75	17.4
pb12	1661	1215	112	181	10.9
pb13	15639	12547	122	1969	12.6
TVC1	6161	5376	110	479	7.8
TVC2	5254	4483	100	417	7.9
TVC3	706	637	50	11	1.6
TVC4	1473	1302	86	58	3.9
TVC5	5467	4798	374	203	3.7
TVC6	11782	10717	276	465	3.9

Table 8.16: Results obtained by basing the branching rule on pseudocosts.

	# LP solved	# LP range reduction	# LP strong branching	# NLP solved	% NLP/LP	# nodes in the stack	CPU times
pb0	112	61	40	7	6.3	4	0.285
pb1	147	36	80	20	13.6	6	0.423
pb2	44	9	28	5	11.4	3	0.119
pb3	78	18	48	8	10.3	7	0.224
pb4	177	76	84	12	6.8	6	0.778
pb5	205	72	118	9	4.4	5	1.002
pb6	598	133	352	75	12.5	12	4.674
pb7	5054	1439	2656	629	12.4	14	38.225
pb8	579	152	356	46	7.9	7	3.244
pb9	266	19	228	14	5.3	5	1.620
pb10	527	182	280	42	8.0	7	2.577
pb11	1326	463	606	165	12.4	17	7.486
pb12	1336	607	600	81	6.1	12	19.072
pb13	20362	9065	9382	1260	6.2	23	281.064
TVC1	82847	55530	19354	5615	6.8	24	1264.972
TVC2	61924	40193	16468	3767	6.1	19	906.001
TVC3	948	715	212	16	1.7	5	30.625
TVC4	1701	1148	484	51	3.0	8	54.868
TVC5	18379	12432	5158	561	3.1	17	980.362
TVC6	32852	22901	8548	1003	3.1	17	1716.357
pb0	68	43	20	3	4.4	2	0.148
pb1	79	18	44	10	12.7	7	0.235
pb2	194	9	138	27	13.9	7	0.654
pb3	121	34	68	12	9.9	4	0.365
pb4	120	19	92	7	5.8	5	0.634
pb5	145	72	64	6	4.1	4	0.621
pb6	348	95	206	29	8.3	9	2.309
pb7	1235	620	394	155	12.6	26	7.728
pb8	263	38	204	15	5.7	9	1.512
pb9	442	95	322	20	4.5	5	2.616
pb10	205	76	114	9	4.4	4	0.880
pb11	558	110	364	58	10.4	10	3.374
pb12	1503	613	770	78	5.2	21	29.412
pb13	17388	7275	8438	1303	7.5	62	266.369
TVC1	7756	5103	1988	485	6.3	14	115.79
TVC2	5792	4073	1252	345	6.0	12	81.639
TVC3	627	520	102	4	0.6	3	19.461
TVC4	1396	837	496	40	2.9	9	49.541
TVC5	5619	3542	1846	150	2.7	12	311.049
TVC6	6096	4323	1572	151	2.5	12	368.399

Table 8.17: Results obtained by basing the branching rule on pseudocosts combined with strong branching iterations and using a depth-first search.

	# LP solved	# LP range reduction	# LP strong branching	# NLP solved	# nodes in the stack
pb0	112	61	40	7	3
pb1	138	27	80	19	5
pb2	44	9	28	5	3
pb3	70	9	48	12	6
pb4	140	19	102	10	4
pb5	205	72	118	9	4
pb6	583	206	296	57	16
pb7	5606	1591	2932	670	146
pb8	938	377	434	77	26
pb9	266	19	228	14	5
pb10	551	216	266	49	13
pb11	1145	408	514	147	19
pb12	1129	568	454	68	14
pb13	28307	12596	13094	1723	254
TVC1	100356	66461	24432	6857	1342
TVC2	65549	42718	17372	4041	726
TVC3	954	715	218	15	4
TVC4	1717	1098	552	47	15
TVC5	23426	15873	6518	739	219
TVC6	32697	22616	8718	984	317
pb0	68	43	20	3	2
pb1	68	15	40	9	4
pb2	286	45	170	35	14
pb3	97	34	50	10	4
pb4	120	19	92	7	4
pb5	145	72	64	6	3
pb6	476	57	340	57	13
pb7	1091	538	370	124	23
pb8	304	57	222	17	5
pb9	442	95	322	21	5
pb10	212	76	116	13	7
pb11	256	59	164	25	6
pb12	940	295	574	51	14
pb13	8791	3532	4180	1066	541
TVC1	8686	5074	2848	563	185
TVC2	5410	3787	1228	292	58
TVC3	627	520	102	4	3
TVC4	1414	842	518	43	18
TVC5	3975	2354	1492	95	28
TVC6	6046	4225	1606	160	45

Table 8.18: Results obtained by using a best-first search.

	# LP solved	# LP range reduction	# LP strong branching	# NLP solved	# nodes in the stack	CPU times
pb0	112	61	40	7	3	0.290
pb1	142	27	82	19	5	0.441
pb2	44	9	28	5	3	0.129
pb3	79	18	48	12	6	0.256
pb4	163	38	102	12	4	0.850
pb5	259	110	126	13	3	1.262
pb6	558	133	320	67	11	4.208
pb7	5182	1629	2660	575	73	38.317
pb8	976	377	462	83	21	4.873
pb9	266	19	228	14	5	1.633
pb10	551	216	270	46	7	2.625
pb11	1142	425	490	150	23	6.509
pb12	1367	646	592	83	11	19.506
pb13	22983	9002	11880	1423	173	323.717
TVC1	87182	57507	21282	5926	160	1335.563
TVC2	58457	37658	16080	3547	104	861.052
TVC3	948	715	212	16	4	31.395
TVC4	1750	1139	538	51	15	58.937
TVC5	20873	14034	5898	660	66	1079.785
TVC6	30994	20903	8734	969	216	1653.214
pb0	68	43	20	3	2	0.158
pb1	68	15	40	10	4	0.211
pb2	260	9	190	30	12	0.833
pb3	97	34	50	10	4	0.246
pb4	120	19	92	7	3	0.620
pb5	145	72	64	6	3	0.618
pb6	292	57	180	33	6	2.058
pb7	1121	546	390	128	22	7.132
pb8	241	38	176	15	5	1.352
pb9	442	95	322	21	5	2.561
pb10	197	76	104	10	4	0.817
pb11	258	59	164	25	6	1.541
pb12	1056	373	592	60	11	21.295
pb13	3885	1582	1856	350	76	60.734
TVC1	8031	5317	2038	492	38	124.184
TVC2	5547	3930	1226	302	15	79.411
TVC3	627	520	102	4	3	19.594
TVC4	1582	949	578	47	17	57.712
TVC5	4338	2719	1474	109	23	244.350
TVC6	5503	3712	1590	143	27	331.486

Table 8.19: Results obtained by using a criterion inspired by best estimates.