UNIVERSITÉ DE MONTRÉAL

APPLICATIONS JOINTES DE L'OPTIMISATION COMBINATOIRE ET GLOBALE

SYLVAIN PERRON
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR (Ph.D.)
(MATHÉMATIQUES DE L'INGÉNIEUR)
AOÛT 2004

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

APPLICATIONS JOINTES DE L'OPTIMISATION
COMBINATOIRE ET GLOBALE

présentée par : PERRON Sylvain
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

M. SAVARD Gilles, Ph.D., président
M. AUDET Charles, Ph.D., membre et directeur de recherche
M. HANSEN Pierre, D. Agr., membre et codirecteur de recherche
M. CAPOROSSI Gilles, Ph.D., membre
M. PARDALOS Panos M., Ph.D., membre externe

REMERCIEMENTS

Je désire tout d'abord remercier tendrement Mylène qui m'a grandement aidé à mener à terme ce projet. Elle n'a jamais cessé de m'encourager tout au long de ce projet et m'a prodigué de précieux conseils tant au niveau de la rédaction que de la présentation orale de cette thèse. Je tiens aussi à dire merci à Médérik qui, sans le savoir, a été dans les deux dernières années une source de motivation continuelle à terminer mon doctorat.

Je voudrais témoigner mon extrême gratitude envers mes parents, Claude et Micheline, pour m'avoir continuellement encouragé dans mes études ainsi que dans la réalisation de mes divers projets. Merci aussi aux autres membres de ma famille et à mes amis pour leur support moral. Je pourrai enfin vous dire "oui" lorsque vous me demanderez si j'ai terminé mon doctorat !

Je ne peux pas passer sous silence l'aide informatique fournie par certains collègues. Merci à Sébastien pour les différentes classes C++ et les conseils en programmation. Merci à Anderson pour les trucs de programmation et l'utilisation de son code de l'algorithme AD-PSAT. Merci à Christophe pour l'utilisation de son code de l'algorithme exact de résolution de problèmes quadratiques 0-1 sans contraintes.

Je voudrais remercier sincèrement Charles et Pierre pour leur aide financière, pour leurs précieux conseils, pour leur grande disponibilité ainsi que pour leurs nombreux encouragements. Un merci particulier à Charles qui a accepté de prendre la relève de la direction de cette thèse.

Je voudrais également remercier Gilles Savard, Gilles Caporossi et Panos M. Pardalos pour avoir accepté d'examiner cette thèse et pour m'avoir fourni de judicieux commentaires.

Je suis très reconnaissant envers le GERAD qui m'a permis d'utiliser ses locaux et son équipement dans la réalisation de ce travail de recherche. Je désire remercier les membres du personnel de ce centre de recherche pour leur soutien technique et, plus particulièrement, Francine et Nicole pour leurs précieux trucs en LaTeX.

Je tiens aussi à remercier le CRSNG, le FCAR, la fondation de l'École Polytechnique de Montréal et la fondation de l'Université de Montréal pour leur soutien financier.

Enfin, pour être certain de ne pas oublier personne, je tiens à dire merci à tous ceux et celles qui m'ont aidé de près ou de loin dans la réalisation de cette thèse.

RÉSUMÉ

L'optimisation, en termes mathématiques, peut se définir comme l'étude de problèmes où l'on cherche à trouver les valeurs d'un ensemble de variables de manière à maximiser ou minimiser une fonction appelée *fonction-objectif* tout en respectant un ensemble de contraintes. Un tel problème est appelé *programme mathématique*.

Il existe plusieurs classes de programme mathématique. La classe et la méthode de résolution d'une programme mathématique particulier dépendent de la nature de sa fonction-objectif et de ses contraintes. Dans cette thèse, nous utilisons et développons, entre autres, certaines méthodes de résolution spécifiques aux classes suivantes de la programmation mathématique : la programmation linéaire en variables continues, la programmation linéaire en variables entières, la programmation non-linéaire et la programmation quadratique. Pour identifier l'optimum global de chaque problème d'optimisation étudié dans ce texte, nous combinons entre elles certaines de ces méthodes tout en les jumelant parfois à des approches provenant de d'autres domaines.

En premier lieu, nous combinons une méthode de la programmation linéaire à plusieurs méthodes de la programmation quadratique en variables 0–1 sans contraintes pour vérifier si une distance à valeurs réelles $d = (d_{ij})_{1 \leq i < j \leq n}$ est isométriquement plongeable dans l'espace-ℓ_1. Ce problème est équivalent à vérifier si la distance d appartient au cône des coupes sur les n points. Nous étudions aussi le problème qui consiste à trouver un plongement optimal en fonction d'un certain critère ainsi que des méthodes d'approximation visant à approcher une distance qui n'est pas plongeable dans un espace-ℓ_1 par une autre qui l'est. Nous montrons que chacun de ces problèmes se formule comme un programme linéaire contenant un nombre exponentiel de colonnes. Ce programme linéaire est alors résolu exactement par la technique de génération de colonnes tout en utilisant certaines stratégies pour accélérer sa convergence. Le problème auxiliaire est un programme quadratique en variables 0–1 sans

contraintes résolu par des heuristiques de type recherche avec tabous et recherche à voisinage variable ainsi que par un algorithme énumératif exact. Les résultats numériques présentés montrent qu'il est possible de résoudre les différentes extensions du problème de plongement-ℓ_1 en un temps raisonnable pour des instances de petite à moyenne taille selon le problème (i.e., jusqu'à 25 points pour le problème le plus difficile et jusqu'à 85 points pour le plus facile).

En second lieu, nous proposons une nouvelle méthode pour résoudre exactement le problème de la satisfiabilité probabiliste. Ce problème consiste à vérifier la cohérence d'un ensemble de probabilités attribuées à des propositions logiques et à trouver les meilleures bornes possibles sur la probabilité d'une proposition additionnelle. Pour résoudre ce problème, deux approches classiques sont proposées dans la littérature : l'approche locale ou l'approche globale. La première approche consiste à appliquer des règles pour resserrer des intervalles de probabilité tandis que la seconde consiste à utiliser la programmation linéaire. Nous proposons ici une nouvelle approche combinant les deux méthodes classiques. Cette méthode est la fusion d'une méthode locale et d'un algorithme de génération de colonnes stabilisé utilisant les techniques de la programmation non-linéaire en variables 0–1 sans contraintes pour résoudre le problème auxiliaire. Nous montrons que la fusion des deux approches classiques est bénéfique pour chacune d'elles : les solutions locales peuvent être utilisées pour accélérer l'obtention de solutions globales à l'aide de la méthode de génération de colonnes stabilisée ; les solutions globales confirment ou réfutent l'optimalité des solutions locales trouvées. En conséquence, les meilleures bornes sont toujours obtenues et, dans la plupart des cas, leur justification est disponible.

En troisième lieu, nous présentons une version améliorée d'un algorithme d'optimisation globale pour la programmation quadratique à contraintes quadratiques. Il s'agit d'un algorithme d'énumération implicite avec ajout de coupes (*branch and cut*) qui trouve en un temps fini une solution optimale globale à l'intérieur d'un certain niveau de tolérance (tant au niveau de l'optimalité que de la faisabilité). Les principales améliorations informatiques et algorithmiques de la nouvelle version sont :

- la possibilité de résoudre des problèmes avec un nombre élevé de variables linéaires et un nombre modéré de termes quadratiques ;
- un contrôle plus complet des paramètres de l'algorithme ;
- la possibilité d'effectuer des raffinements de bornes à différents noeuds de l'arbre d'énumération afin de réduire le nombre de noeuds explorés.

Nous utilisons ensuite cet algorithme pour résoudre deux applications de la programmation quadratique.

Nous répondons à une question énoncée par S. Vincze en 1950 : quel est l'octogone convexe avec des côtés de longueur unitaire ayant un diamètre minimum ? Pour répondre à cette question, nous combinons une analyse géométrique combinatoire à l'utilisation des outils de la programmation quadratique. En fait, nous formulons ce problème sous la forme d'un programme quadratique à contraintes quadratiques après avoir fait une analyse géométrique menant à l'énoncé de conditions nécessaires d'optimalité. Ces conditions permettent la résolution du problème en diminuant la taille du domaine réalisable par une réduction considérable de l'aspect combinatoire du problème. En résolvant le programme quadratique, avec une précision de sept décimales, on prouve que la valeur du diamètre de l'octogone optimal est de $2.5843054\ldots$. Nous montrons aussi que cette solution est ϵ-unique, i.e., qu'il n'y a aucun autre octogone optimal à l'extérieur d'une boule de rayon égal à $\epsilon = 0.0123$.

Finalement, nous considérons un problème fondamental en classification automatique des données qui consiste à séparer deux ensembles de points dans un espace réel à n dimensions avec un hyperplan qui minimise la somme des distances, en norme-ℓ_p, à l'hyperplan des points qui se retrouvent du mauvais côté. Malgré de récents progrès, des techniques pratiques pour la résolution exacte des cas autres que la norme-ℓ_1 n'étaient pas encore disponibles. Nous proposons une méthode qui rend possible la résolution exacte de problèmes de taille modérée pour le cas de la norme-ℓ_2. Nous résolvons en des temps de calcul raisonnables des problèmes générés aléatoirement contenant jusqu'à 20000 points pour des problèmes à 6 dimensions et jusqu'à 13 dimensions pour des problèmes contenant 2000 points. Nous avons également résolu quelques problèmes réels provenant d'une base de données publique. Finalement, des résultats numériques illustrent que, pour des problèmes dont la dimension est assez grande, les temps de calcul sont réduits considérablement grâce à l'utilisation d'une borne fournie par une heuristique.

ABSTRACT

In mathematical terms, optimization may be defined as the study of problems where we try to find the values of a set of variables in order to minimize or maximize a function called *objective function* while considering a set of constraints. Such problem is called *mathematical program*.

There are various classes of mathematical program. The class and the solution method of a particular mathematical program depend on the nature of its objective function and constraints. In this thesis, we use and develop some solution methods of the following classes of mathematical programming: linear programming with continuous variables, linear programming with integer variables, nonlinear programming and quadratic programming. To solve each problem studied in this thesis, we combine some of these methods and sometimes merge them with approaches from other fields.

First of all, we combine a linear programming method to various methods of unconstrained quadratic programming in 0-1 variables in order to verify if a real-valued distance $d = (d_{ij})_{1 \leq i < j \leq n}$ is isometrically embeddable in ℓ_1-space. This problem is equivalent to verify if the distance d belongs to the cut cone on n points. We also study the problem of finding an optimal embedding as well as several ways to approximate a distance which is not ℓ_1-embeddable by another one which is. Each of these problems is expressed as a linear program with an exponential number of columns and solved by a constructive column generation algorithm with some accelerating strategies. The subproblem is an unconstrained 0-1 quadratic program, solved by tabu search and variable neighborhood search heuristics as well as by an exact enumerative algorithm. Computational results show that solution of these problems may be performed in reasonable computing time for instances of small to moderate size (i.e., up to $n = 25$ for the most difficult problem and up to $n = 85$ for the easiest one).

Second of all, we propose a new solution method for probabilistic satisfiability. Probabilistic satisfiability (also known as probabilistic logic and entailment) aims at checking the consistency of a set of probability values for logical propositions and at finding best bounds on the probability of an additional proposition. The local approach to these problems is to apply rules to tighten probability intervals. The global approach uses linear programming to find best bounds. We propose here a new approach which is a merge of these two classical approaches. This solution method is a combination of a local method and a stabilized column generation algorithm using nonlinear programming techniques in 0-1 variables to solve the subproblem. We show that merging these approaches is profitable to both: local solutions can be used to accelerate finding global ones through stabilized column generation; global solutions confirm or refute the optimality of the local solutions found. As a result, best bounds are found and, in most cases, their justification is available.

Third of all, we present an improved version of a global optimization algorithm for quadratic programming with quadratic constraints. It is a branch-and-cut algorithm which provides in finite time a globally optimal solution (within given feasibility and optimality tolerances). The main computational and algorithmic improvements of this new version are:

- the possibility to solve problems having a large number of linear variables with few quadratic terms;
- an improved control on the algorithm's parameters;
- the possibility to refine bounds on variables anywhere in the enumeration tree in order to reduce the number of nodes explored.

We use this new implementation to solve some instances of two applications of quadratic programming.

We answers a query stated by S. Vincze in 1950 : find the convex octagon with unit-length sides and minimum diameter. To solve this problem, we use a combinatorial geometric analysis of this problem together with techniques of quadratic programming. We formulate the problem as a quadratic program with quadratic constraints after having found some necessary optimality conditions by a geometric analysis. These conditions lead to a reduction of the domain size by decreasing considerably the combinatorial aspect of the problem. This reduction allows solution of the resulting quadratic program. The optimal solution obtained, with a precision of seven digits, correspond to an octagon having a diameter of $2.5843054\ldots$. We also show that the solution is ϵ-unique, i.e., that there is no other optimal octagon outside a ball of radius $\epsilon = 0.0123$.

Finally, we consider a basic problem in automatic classification which is the problem of separating two sets of points in an n-dimensional real space with a hyperplane that minimizes the sum of L_p-norm distances to the plane of points lying on the wrong side. Despite recent progress, practical techniques for the exact solution of cases other than the L_1-norm have remained unavailable. We propose and implement a new approach that make possible the exact solution of fairly large problems for the L_2-norm. We solve in reasonable computing times random problems of up to 20000 points for problems in 6 dimensions and up to 13 dimensions for problems with 2000 points. We also solve several real-life instances from a publicly available data basis. We show that, for sufficiently large problems, computation times can be dramatically reduced by using heuristic bounds.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ALGORITHMES

LISTE DES ANNEXES

INTRODUCTION

L'optimisation, en termes mathématiques, peut se définir comme l'étude de problèmes où l'on cherche à trouver les valeurs d'un ensemble de variables de manière à maximiser ou minimiser une fonction appelée *fonction-objectif* tout en respectant un ensemble de contraintes. Un tel problème est appelé *programme mathématique*. Il existe des applications de la programmation mathématiques dans des domaines très variés tels que la logistique, les télécommunications, l'économie, la finance, l'analyse des données, l'intelligence artificielle, la gestion de la production et la géométrie.

La classe d'appartenance d'un programme mathématique dépend de la nature de sa fonction-objectif et de ses contraintes. Parmi les classes principales de programmes mathématiques, notons les programmes linéaires en variables continues, les programmes linéaires en variables entières, les programmes non-linéaires, les programmes quadratiques, les programmes stochastiques et les programmes dynamiques. Chacune de ces classes a mené à l'élaboration de méthodes de résolution spécifiques. Selon la difficulté du problème, la méthode de résolution préconisée pourrait être un algorithme exact qui procure une solution globalement optimale ou un algorithme heuristique qui procure généralement une bonne solution sans toutefois pouvoir garantir son optimalité.

Pour certaines applications de la programmation mathématique, il est possible de résoudre le programme mathématique en utilisant directement les méthodes développées expressément pour la classe d'appartenance de ce programme mathématique. Cependant, dans d'autres situations, il est indispensable de combiner certaines techniques classiques spécifiques à plusieurs classes de programmes mathématiques pour en arriver à résoudre efficacement un problème. Il arrive même qu'il faille combiner aussi une ou plusieurs approches classiques de la programmation mathématique à d'autres approches propres au domaine d'étude du problème considéré.

Dans cette thèse, nous résolvons plusieurs programmes mathématiques issus de quatre applications distinctes. La résolution se fait grâce à l'utilisation conjointe de certaines approches classiques de la programmation mathématique telles que la technique de génération de colonnes de la programmation linéaire, les méta-heuristiques, les méthodes de linéarisation et la méthode d'énumération implicite avec ajout de coupes. Dans certains cas, nous jumelons aussi ces méthodes à d'autres approches issues du domaine du problème étudié telles que l'approche de règles de la satisfiabilité probabiliste et l'analyse géométrique combinatoire.

Dans le premier chapitre, nous combinons une méthode de la programmation linéaire à plusieurs méthodes de la programmation quadratique en variables 0–1 sans contraintes pour vérifier si une distance à valeurs réelles $d = (d_{ij})_{1 \leq i < j \leq n}$ est isométriquement plongeable dans l'espace ℓ_1. Ce problème est équivalent à vérifier si la distance d appartient au cône des coupes sur les n points. Nous montrons que ce problème se formule comme un programme linéaire contenant un nombre exponentiel de colonnes. Nous résolvons ce problème par la technique de génération de colonnes tout en utilisant certaines stratégies pour accélérer sa convergence. Le problème auxiliaire est un programme quadratique en variables 0–1 sans contraintes résolu par des heuristiques de type recherche avec tabous et recherche à voisinage variable ainsi que par un algorithme énumératif exact.

Le deuxième chapitre présente une nouvelle méthode pour résoudre le problème de la satisfiabilité probabiliste. Ce problème consiste à vérifier la cohérence d'un ensemble de probabilités attribuées à des propositions logiques et à trouver les meilleures bornes possibles sur la probabilité d'une proposition additionnelle. La nouvelle approche combine les deux approches classiques pour résoudre ce problème : l'approche locale qui consiste à appliquer des règles pour resserrer des intervalles de probabilité et l'approche globale qui consiste à utiliser la programmation linéaire. En fait, la méthode proposée est la combinaison d'une méthode locale et d'un algorithme de génération de colonnes stabilisé utilisant les techniques de la programmation nonlinéaire en variables 0–1 sans contraintes pour résoudre le problème auxiliaire. Nous

montrons que la fusion des approches locale et globale est bénéfique pour chacune d'elles.

Dans le troisième chapitre, nous développons et utilisons une nouvelle version d'un algorithme d'optimisation globale pour résoudre deux applications de la programmation quadratique à contraintes quadratiques. Il s'agit d'un algorithme d'énumération implicite avec ajout de coupes (communément appelé *branch and cut*) qui trouve en un temps fini une solution optimale globale à l'intérieur d'un certain niveau de tolérance (tant au niveau de l'optimalité que de la faisabilité). La nouvelle version de l'algorithme inclut des améliorations tant au niveau de l'implémentation informatique qu'au niveau l'algorithmique.

Le premier programme quadratique résolu par cet algorithme provient du domaine de la géométrie. Nous répondons à une question ouverte datant de 1950 : quel est l'octogone convexe avec des côtés de longueur unitaire ayant un diamètre minimum ? Pour répondre à cette question, nous combinons une analyse géométrique combinatoire du problème à l'utilisation des outils de la programmation quadratique. En fait, nous formulons ce problème sous forme d'un programme quadratique à contraintes quadratiques après avoir fait une analyse géométrique menant à l'énoncé de conditions nécessaires d'optimalité. Ces conditions permettent la résolution exacte du problème en diminuant la taille du domaine réalisable par une réduction considérable de l'aspect combinatoire du problème. Le programme quadratique est alors résolu exactement par la version améliorée de l'algorithme d'énumération.

Le deuxième programme quadratique résolu est la formulation mathématique d'un cas particulier d'un problème fondamental en classification automatique des données, i.e., le problème de séparer deux ensembles de points dans un espace réel à n dimensions avec un hyperplan qui minimise la somme des distances, en norme-ℓ_2, à l'hyperplan des points qui se retrouvent du mauvais côté. Nous proposons une approche qui permet la résolution exacte d'instances assez grandes de ce problème. Nous illustrons par des résultats numériques que, pour des problèmes dont la dimension est

assez grande, les temps de calcul peuvent être réduits considérablement en jumelant l'utilisation d'une méthode heuristique à l'algorithme exact d'énumération.

CHAPITRE 1 : PLONGEMENT EN NORME-ℓ_1

Dans ce chapitre, nous étudions différents problèmes associés au plongement en norme-ℓ_1. La résolution des programmes mathématiques associés à ces problèmes se fait en utilisant la technique de génération de colonnes de la programmation linéaire conjointement avec la programmation quadratique 0-1 sans contraintes.

Le problème de plongement isométrique qui consiste à vérifier si un espace métrique (X, d) peut être plongé dans un espace prescrit, i.e., vérifier s'il existe une correspondance du premier espace au second qui préserve les distances. Ce problème a été fortement étudié depuis le milieu du XIXe siècle (voir [31] pour des références). Dans ce chapitre, nous nous concentrons sur les aspects algorithmiques du problème de plongement en norme-ℓ_1, i.e., le plongement isométrique d'un espace métrique fini (X, d) dans un espace-ℓ_1, ou espace équipé de la norme-ℓ_1.

Des liens importants ont été établis entre le plongement-ℓ_1 et certains problèmes fondamentaux de l'optimisation combinatoire et de la théorie des probabilités. Assouad [5] a montré qu'une distance à valeurs réelles $d = (d_{ij})_{1 \leq i < j \leq n}$ est isométriquement plongeable dans l'espace-ℓ_1 si et seulement si elle appartient au cône des coupes sur n points. Déterminer si cette condition est satisfaite est un problème NP-complet [12]. Caractériser des espaces métriques ℓ_1-plongeable (X, d) revient à trouver toutes les facettes du cône des coupes. Alors que beaucoup de familles de telles facettes sont connues (voir encore [31]) et qu'elles le sont toutes pour $n \leq 8$ [89], il est très peu probable que toutes ces facettes puissent être trouvées pour un n quelconque. En effet, une conséquence d'un résultat de Karp et Papadimitriou [64] est qu'il n'existe aucune méthode polynomiale simple permettant de décrire une liste d'inégalités suffisantes pour décrire le cône des coupes sauf si NP=C_o-NP.

Avis [11] a démontré que d appartient au cône des coupes si et seulement s'il existe un espace mesuré (X, A, μ) et des entités $A_1, A_2, \ldots, A_n \in A$ telles que $d_{ij} = \mu(A_i \cap A_j)$ pour tout $1 \leq i < j \leq n$.

Plusieurs résultats équivalents sur le plongement-ℓ_1 et ses applications ont été obtenus dans différents domaines. Une synthèse détaillée de ces résultats, et de plusieurs autres, est donnée dans le volume *Geometry of Cuts and Metrics* de Deza et Laurent [31]. Parmi les applications du plongement-ℓ_1, mentionnons le cas quadratrique [40, 65] du problème de la satisfiabilité probabiliste de Boole [16, 17, 18, 51] et l'analyse en composantes principales en norme-ℓ_1 [14].

La condition d'Assouad sur le plongement-ℓ_1 se ramène à un programme linéaire avec un nombre exponentiel de colonnes ; ces colonnes correspondent à toutes les coupes. Afin de résoudre ce problème, nous appliquons la technique de *génération de colonnes* de la programmation linéaire. Ainsi, la grande majorité des colonnes sont considérées de manière implicite. La colonne entrante est déterminée par la résolution d'un problème d'optimisation spécifique qui, dans ce cas, est un programme quadratique en variables 0–1 sans contraintes. Ce problème est résolu de manière heuristique par la *recherche avec tabous* ou la *recherche à voisinage variable* et exactement par un algorithme énumératif exact.

Si la distance d n'est pas plongeable en norme-ℓ_1, nous pouvons naturellement nous interroger sur la façon d'approcher cette distance par une autre qui le soit. Nous considérons différentes méthodes d'approximation, i.e., le *problème de la constante additive* dans lequel chaque distance est augmentée de la même constante [24, 34, 35], les approximations *inférieure* et *supérieure* ainsi que l'approximation *inférieure-supérieure* [35].

La section 1.1 est dédiée à l'étude du cadre théorique général du problème de plongement en norme-ℓ_1 : quelques définitions et résultats préliminaires sur les espaces métriques, le plongement-ℓ_1 et le cône des coupes sont donnés. Dans la section 1.2, nous présentons les programmes mathématiques associés à différents problèmes sur le plongement en norme-ℓ_1. L'algorithme utilisé pour résoudre ces programmes mathématiques est présenté à la section 1.3. Des résultats numériques détaillés sont présentés à la section 1.4. Le chapitre se termine par une brève discussion.

1.1 Cadre théorique général

Dans cette section, nous présentons certaines notions tirées des chapitres 3 et 4 de [31] et suivons de très près la notation qui y est présente.

1.1.1 Espaces métriques

Considérons un ensemble X. Une *distance* sur X est une fonction $d : X \times X \to \mathbb{R}$ qui est *non-négative*, i.e., $d(i,j) \geq 0$ pour tout $i,j \in X$, *symétrique*, i.e., $d(i,j) = d(j,i)$ pour tout $i,j \in X$ et telle que $d(i,i) = 0$ pour tout $i \in X$. Le couple (X,d) formé de l'ensemble X et de la distance d est appelé *espace métrique*. Une *semi-métrique* est une distance d qui satisfait l'inégalité triangulaire

$$d(i,j) \leq d(i,k) + d(k,j)$$

pour tout $i,j,k \in X$. Une *métrique* est une semi-métrique telle que $d(i,j) = 0 \Rightarrow i = j$ pour tout $i,j \in X$.

Posons $V_n = \{1,2,\ldots,n\}$ et $E_n = \{ij \mid i,j \in V_n, i \neq j\}$ où $ij = ji$ dénote la paire non-ordonnée formée des entiers i et j. Une distance d sur V_n peut être représentée par un vecteur $(d_{ij})_{1 \leq i < j \leq n} \in \mathbb{R}^{|E_n|}$. Une distance peut également être représentée par

une *matrice de distances*, notée par $D = (d_{ij})$, qui est symétrique et qui ne possède que des éléments nuls sur sa diagonale principale. En fait, tout vecteur non-négatif $d \in \mathbb{R}^{|E_n|}$ se ramène à une distance pouvant être représentée par la matrice D où $d_{ij} = d_{ji}$ pour tout $ij \in E_n$ et $d_{ii} = 0$ pour tout $i \in V_n$.

Rappelons que la *norme* sur un espace vectoriel E est une fonction $x \in E \mapsto \| x \| \in \mathbb{R}_+$ telle que

(i) $\| x = 0 \|$ si et seulement si $x = 0$;

(ii) $\| \lambda x \| = |\lambda| \, \| x \|$ pour tout $\lambda \in \mathbb{R}, x \in E$;

(iii) $\| x + y \| \leq \| x \| + \| y \|$ pour tout $x, y \in E$.

Étant donné un espace normé $(E, \| \, . \, \|)$, un espace métrique normé ou *métrique de Minkowsky*, est obtenu en fixant

$$d_{\|.\|}(x, y) := \| x - y \| \, .$$

Pour tout $p \geq 1$ la métrique-ℓ_p, d_{ℓ_p}, est obtenue en dotant $\mathbb{R}_m$ de la norme-ℓ_p

$$\| x \|_p = \left(\sum_{1 \leq k \leq m} |x_k|^p \right)^{1/p}$$

pour $x \in \mathbb{R}^m$.

La métrique-ℓ_1, d_{ℓ_1}, est le cas particulier où $p = 1$, i.e.,

$$\| x - y \|_1 = \sum_{1 \leq k \leq m} |x_k - y_k|$$

pour $x, y \in \mathbb{R}^m$. Notons que la métrique-ℓ_1 est aussi appelée *métrique rectilinéaire* ou *distance de Manhattan*.

1.1.2 Coupes et cône des coupes

Étant donné un sous-ensemble S de V_n, supposons que $\delta(S)$ dénote un vecteur dans $\mathbb{R}^{|E_n|}$ défini par

$$\delta(S)_{ij} = \begin{cases} 1 & \text{si } \mid S \cap \{i,j\} \mid = 1 \\ \\ 0 & \text{sinon} \end{cases}$$

pour $1 \leq i < j \leq n$. Il est facile de constater que $\delta(S)$ définit une distance sur V_n qui correspond à une semi-métrique ; pour cette raison $\delta(S)$ est appelée *coupe semi-métrique*.

Le *cône des coupes*, noté par CUT_n, est le cône dans $\mathbb{R}^{|E_n|}$ généré par les coupes semi-métriques $\delta(S)$ pour $S \subseteq V_n$

$$\text{CUT}_n = \left\{ \sum_{S \subseteq V_n} \lambda_S \delta(S) \mid \lambda_S \in \mathbb{R}_+ \text{ pour tout } S \subseteq V_n \right\}.$$

1.1.3 Plongements isométriques

Considérons deux espaces métriques (X, d) et (X', d'). (X, d) est *isométriquement plongeable* dans (X', d') s'il existe une correspondance Φ, appelée *plongement isométrique* de X vers X', telle que

$$d(x, y) = d'(\Phi(x), \Phi(y))$$

pour tout $x, y \in X$, i.e., telle que les distances entre toutes les paires de points soient conservées. (X, d) est alors nommé *sous-espace isométrique de* (X', d').

Le résultat d'Assouad [5], mentionné dans la section précédente, peut être exprimé de la façon suivante :

Proposition 1.1 *Soit $d \in \mathbb{R}^{|E_n|}$ et (V_n, d) son espace métrique associé, les énoncés suivants sont équivalents.*

(i) $d \in \text{CUT}_n$.

(ii) (V_n, d) *est ℓ_1-plongeable, i.e., il existe n vecteurs $w_1, w_2, \ldots, w_n \in \mathbb{R}^m$, pour un m donné, tels que $d_{ij} = \| w_i - w_j \|_1$, pour tout $1 \leq i \leq j \leq n$.*

Soit d une distance sur V_n, toute décomposition

$$d = \sum_{S \subseteq V_n} \lambda_S \delta(S)$$

où $\lambda_S \in \mathbb{R}_+$ pour tout S est appelée une *réalisation-$\mathbb{R}_+$* de d. Ainsi, pour une distance d qui est ℓ_1-plongeable, nous pouvons donc parler alternativement de plongement-ℓ_1 de d ou d'une *réalisation-$\mathbb{R}_+$* de d. À l'exemple 1.1 de la prochaine section, nous illustrons comment obtenir un plongement à partir d'une *réalisation-$\mathbb{R}_+$*.

Supposons que d soit une distance sur V_n. Si $\sum_{\emptyset \neq S \subset V_n} \lambda_S \delta(S)$ est une décomposition de d comme une combinaison linéaire de coupes semi-métriques non-nulles, alors la quantité $\sum_{\emptyset \neq S \subset V_n} \lambda_S$ est appelée la *taille* de d et la quantité

$$\min_{\lambda} \left(\sum_{\emptyset \neq S \subset V_n} \lambda_S \mid d = \sum_{\emptyset \neq S \subset V_n} \lambda_S \delta(S) \text{ avec } \lambda_S \in \mathbb{R}_+ \text{ pour tout } S \subset V_n \right) \qquad (1.1)$$

est appelée la *taille-ℓ_1 minimum* de d.

1.2 Problèmes

Dans cette section, nous rappelons les formulations mathématiques de plusieurs problèmes reliés au plongement en norme-ℓ_1 que l'on retrouve dans [31, 34, 35].

1.2.1 Problème de faisabilité

Le *problème de faisabilité* consiste à vérifier si (V_n, d) est ℓ_1-*plongeable*. Tel que mentionné par Fichet [34, 35], démontrer que (V_n, d) est ℓ_1-*plongeable* consiste à vérifier que le système linéaire suivant admet au moins une solution réalisable

$$\begin{aligned} \sum_{S^t \in H} \lambda_{S^t} \delta(S^t)_{ij} &= d_{ij} \qquad \forall\ 1 \leq i < j \leq n \\ \lambda_{S^t} &\geq 0 \qquad \forall\ S^t \in H \end{aligned} \tag{1.2}$$

où $H = \left\{S^1, S^2, \ldots, S^{2^{n-1}-1}\right\}$ est l'ensemble de tous les sous-ensembles non-vides de V_n contenant au plus $\lfloor \frac{n}{2} \rfloor$ points. On ne considère pas l'ensemble vide (ou son complément) car la coupe associée est vide. De plus, on ne considère que les sous-ensembles S^t de V_n contenant au plus $\lfloor \frac{n}{2} \rfloor$ points car les compléments de ces sous-ensembles engendrent les mêmes coupes, i.e., $\delta(S^t) = \delta(V_n \setminus S^t)$ pour tout $S^t \subset V_n$.

Le problème (1.2) peut être résolu en appliquant la phase 1 de la version de base ou révisée de l'algorithme du simplexe (voir, par exemple, Chvátal [22] pour une description de cette méthode), i.e., en résolvant le programme linéaire suivant équivalent à (1.2) obtenu en ajoutant les variables artificielles α_{ij}

$$\begin{aligned} \min_{\alpha,\lambda} \quad & \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \alpha_{ij} \\ \text{s.c.} \quad & \\ & \sum_{S^t \in H} \lambda_{S^t} \delta(S^t)_{ij} + \alpha_{ij} = d_{ij} \qquad \forall\ 1 \leq i < j \leq n \\ & \lambda_{S^t} \geq 0 \qquad \forall\ S^t \in H \\ & \alpha_{ij} \geq 0 \qquad \forall\ 1 \leq i < j \leq n. \end{aligned} \tag{1.3}$$

Le système linéaire (1.2) admet au moins une solution si et seulement si la valeur minimum de (1.3) est égale à 0.

Exemple 1.1 Considérons la distance d sur un ensemble $V_4 = \{1, 2, 3, 4\}$ obtenue avec l'indice de dissimilarités de Rao [36, 60] exprimée par la matrice de distances

$$D = \begin{pmatrix} 0 & 0.5 & 0.5 & 1.0 \\ 0.5 & 0 & 1.0 & 1.0 \\ 0.5 & 1.0 & 0 & 1.0 \\ 1.0 & 1.0 & 1.0 & 0 \end{pmatrix}.$$

L'ensemble H, tel que défini dans l'équation (1.2), est le suivant :

$$H = \{ \quad S^1 = \{1\}, S^2 = \{2\}, S^3 = \{3\}, S^4 = \{4\}, S^5 = \{1, 2\}, S^6 = \{1, 3\}, \\ S^7 = \{1, 4\} \ \}.$$

Les coupes $\delta(S^t)$ sur ces sous-ensembles correspondent aux vecteurs suivants :

i,j	$\delta(S^1)$	$\delta(S^2)$	$\delta(S^3)$	$\delta(S^4)$	$\delta(S^5)$	$\delta(S^6)$	$\delta(S^7)$
$1, 2$	1	1	0	0	0	1	1
$1, 3$	1	0	1	0	1	0	1
$1, 4$	1	0	0	1	1	1	0
$2, 3$	0	1	1	0	1	1	0
$2, 4$	0	1	0	1	1	0	1
$3, 4$	0	0	1	1	0	1	1

Démontrer si $d \in \text{CUT}_4$ revient à vérifier si le système linéaire suivant admet au moins une solution :

$$\begin{array}{ccccccccccccccccc}
\lambda_{S^1} & + & \lambda_{S^2} & & & & & & & & & + & \lambda_{S^6} & + & \lambda_{S^7} & = & 0.5 \\
\lambda_{S^1} & & & + & \lambda_{S^3} & & & + & \lambda_{S^5} & & & & & + & \lambda_{S^7} & = & 0.5 \\
\lambda_{S^1} & & & & & + & \lambda_{S^4} & + & \lambda_{S^5} & + & \lambda_{S^6} & & & & = & 1.0 \\
& & \lambda_{S^2} & + & \lambda_{S^3} & & & + & \lambda_{S^5} & + & \lambda_{S^6} & & & & = & 1.0 \\
& & \lambda_{S^2} & & & + & \lambda_{S^4} & + & \lambda_{S^5} & & & + & \lambda_{S^7} & = & 1.0 \\
& & & & \lambda_{S^3} & + & \lambda_{S^4} & & & + & \lambda_{S^6} & + & \lambda_{S^7} & = & 1.0 \\
\lambda_{S^1} & , & \lambda_{S^2} & , & \lambda_{S^3} & , & \lambda_{S^4} & , & \lambda_{S^5} & , & \lambda_{S^6} & , & \lambda_{S^7} & \geq & 0.
\end{array}$$

Une solution de ce système linéaire (ou est *réalisation*-$\mathbb{R}_+$) est

$$\lambda_{S^4} = 0.5, \lambda_{S^2} = \lambda_{S^3} = \lambda_{S^5} = \lambda_{S^6} = 0.25, \lambda_{S^1} = \lambda_{S^7} = 0.$$

À partir de cette *réalisation*-$\mathbb{R}_+$, il est possible d'obtenir un plongement-ℓ_1. Pour ce faire, il suffit de définir, pour chaque élément de V_4, un vecteur w_i'' où

$$(w_i'')_t = \begin{cases} \lambda_{S^t} & \text{si } i \in S^t \\ 0 & \text{sinon.} \end{cases}$$

Nous obtenons alors un plongement-ℓ_1 dans un espace à sept dimensions :

$$w_1'' = (0,0,0,0,0.25,0.25,0), w_2'' = (0,0.25,0,0,0.25,0,0),$$
$$w_3'' = (0,0,0.25,0,0,0.25,0), w_4'' = (0,0,0,0.5,0,0,0).$$

En éliminant chaque dimension t pour laquelle $\lambda_{S^t} = 0$, nous obtenons un plongement dans un espace à cinq dimensions :

$$w_1' = (0,0,0,0.25,0.25), w_2' = (0.25,0,0,0.25,0),$$
$$w_3' = (0,0.25,0,0,0.25), w_4' = (0,0,0.5,0,0).$$

Finalement, en appliquant le lemme 11.1.3 de [31], nous trouvons un plongement dans un espace à deux dimensions :

$$w_1 = (0.25,0.25), w_2 = (0,0), w_3 = (0.5,0.5), w_4 = (0.25,0).$$

Ainsi, chaque élément d_{ij} de la matrice D représente la distance en norme-ℓ_1 entre les points w_i et w_j. Par exemple, on peut vérifier que

$$d_{12} = \| w_1 - w_2 \|_1 = |0.25 - 0| + |0.25 - 0| = 0.5.$$

1.2.2 Problèmes d'optimisation

Lorsque le système linéaire (1.2) possède une solution admissible, i.e., lorsque la distance d est ℓ_1-plongeable, on peut vouloir identifier une solution optimale en

fonction d'un critère spécifique ou, en d'autres termes, choisir le meilleur plongement-ℓ_1 selon un certain objectif. Une fonction-objectif est alors ajoutée à (1.2).

Sous forme de programme linéaire, le problème d'optimisation est :

$$\begin{aligned} \min_{\lambda} \quad & \sum_{S^t \in H} \lambda_{S^t} c_t \\ \text{s.c.} \quad & \\ & \sum_{S^t \in H} \lambda_{S^t} \delta(S^t)_{ij} = d_{ij} \qquad \forall\ 1 \leq i < j \leq n \\ & \lambda_{S^t} \geq 0 \qquad \forall\ S^t \in H \end{aligned} \tag{1.4}$$

où c_t représente la contribution de la coupe $\delta(S^t)$ à l'objectif considéré. Lorsque $c_t = 1$, pour tout t, le problème (1.4) correspond au problème (1.1), ainsi sa solution optimale donne la *taille-ℓ_1 minimum* de d. Benayade et Fichet [14] proposent un critère utile pour l'*analyse en composantes principales en norme-ℓ_1* qui consiste à poser

$$c_t = \min(|S^t|, |V_n \setminus S^t|) \text{ pour tout } t$$

dans le but de trouver, parmi tous les plongements-ℓ_1 de (V_n, d), celui qui minimise le critère de la médiane-ℓ_1.

On dit que (V_n, d) est ℓ_1-*rigide* si d admet une unique *réalisation-*$\mathbb{R}_+$. Ainsi, (V_n, d) est ℓ_1-*rigide* si le système linéaire (1.2) possède une solution unique. Cette propriété peut être vérifiée comme suit : résoudre (1.2) et poser $(\lambda^*_{S^t})$ la solution obtenue. Ensuite, définir

$$T^+ = \{t \mid \lambda^*_{S^t} > 0\}.$$

Si la solution n'est pas unique, il existe alors une solution avec au moins une variable λ_{S^t} strictement positive pour laquelle $t \notin T^+$. De plus, puisque tous les coefficients dans les contraintes de (1.2) sont positifs, alors au moins une variable $\lambda_{S^t} < \lambda^*_{S^t}$ est telle que $t \in T^+$. Ainsi, on peut considérer successivement un total de $|T^+|$ programmes linéaires ayant comme fonction-objectif λ_{S^t} pour tout $t \in T^+$ jusqu'à ce qu'une solution différente de $\lambda^*_{S^t}$ soit obtenue. Si aucune solution différente n'est trouvée, alors (V_n, d) est nécessairement ℓ_1-*rigide*.

1.2.3 Problèmes d'approximation

Lorsque le système linéaire (1.2) ne possède aucune solution admissible, il peut être intéressant de trouver comment modifier d de manière à obtenir d' de sorte que (V_n, d') soit ℓ_1-*plongeable*. Pour ce faire, nous considérons ici quatre différentes méthodes d'approximation.

Un premier type d'approximation correspond au *problème de la constante additive* dans lequel on cherche à augmenter chaque distance de la même valeur γ. Ce problème peut s'exprimer par le programme linéaire suivant :

$$\begin{aligned} \min_{\gamma,\lambda} \quad & \gamma \\ \text{s.c.} \quad & \\ & \sum_{S^t \in H} \lambda_{S^t} \delta(S^t)_{ij} - \gamma = d_{ij} \qquad \forall\ 1 \leq i < j \leq n \\ & \lambda_{S^t} \geq 0 \qquad \forall\ S^t \in H \\ & \gamma \geq 0. \end{aligned} \tag{1.5}$$

Tel qu'observé par Critchley [24], le programme linéaire (1.5) possède toujours une solution admissible.

Une deuxième approche nous donne une *approximation supérieure en norme-ℓ_1*. Elle est obtenue en résolvant le programme linéaire suivant :

$$\begin{aligned} \min_{\beta,\lambda} \quad & \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \beta_{ij} \\ \text{s.c.} \quad & \\ & \sum_{S^t \in H} \lambda_{S^t} \delta(S^t)_{ij} - \beta_{ij} = d_{ij} \qquad \forall\ 1 \leq i < j \leq n \\ & \lambda_{S^t} \geq 0 \qquad \forall\ S^t \in H \\ & \beta_{ij} \geq 0 \qquad \forall\ 1 \leq i < j \leq n. \end{aligned} \tag{1.6}$$

De façon similaire, on peut obtenir une *approximation inférieure en norme-ℓ_1*. Ce problème revient à résoudre le programme linéaire (1.3).

Finalement, une généralisation de ces deux dernières approximations nous donne une *approximation inférieure-supérieure en norme-ℓ_1*. Elle est obtenue en résolvant le programme linéaire :

$$\begin{aligned}
\min_{\beta,\alpha,\lambda} \quad & \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (\beta_{ij} + \alpha_{ij}) \\
\text{s.c.} \quad & \\
& \sum_{S^t \in H} \lambda_{S^t} \delta(S^t)_{ij} - \beta_{ij} + \alpha_{ij} = d_{ij} \qquad \forall\ 1 \leq i < j \leq n \\
& \lambda_{S^t} \geq 0 \qquad \forall\ S^t \in H \\
& \beta_{ij}, \alpha_{ij} \geq 0 \qquad \forall\ 1 \leq i < j \leq n.
\end{aligned} \tag{1.7}$$

1.3 Algorithme

Résoudre un des programmes linéaires (1.3) à (1.7) par l'algorithme du simplexe présente deux difficultés majeures : (i) le nombre exponentiel de colonnes ; (ii) le fait qu'à chaque itération, décider si l'algorithme doit arrêter ou continuer est un problème NP-difficile (voir section 1.3.3 plus bas). Chacun de ces programmes linéaires contient $2^{n-1} - 1$ colonnes qui, à moins que n soit petit, représente une quantité trop grande pour simplement écrire chacune de ces colonnes explicitement. Il est cependant possible de considérer implicitement toutes ces colonnes en utilisant la technique de génération de colonnes de la programmation linéaire.

1.3.1 Génération de colonnes : description générale

La méthode de génération de colonnes (voir [69] pour une revue bibliographique sur le sujet) est une méthode exacte basée sur le prolongement de la méthode révisée du simplexe dans laquelle un petit nombre de colonnes sont présentes de manière explicite et où l'on détermine la colonne entrante par la résolution d'un problème d'optimisation spécifique. Ainsi, deux programmes sont associés au programme linéaire original : d'un côté, le *problème maître* qui est identique au programme original mais contenant explicitement seulement un nombre restreint de colonnes et d'un

autre côté, le *problème auxiliaire*, ou *sous-problème*, dont le rôle consiste à déterminer la colonne entrante comme dans la version de base ou révisée de l'algorithme du simplexe. Lorsque la colonne entrante est choisie, on l'ajoute au problème maître courant et ce dernier est résolu jusqu'à son optimalité.

Nous allons maintenant expliquer la méthode de génération de colonnes en décrivant le lien qui l'unit à la méthode du simplexe. Considérons le programme linéaire

$$\begin{array}{ll} \min\limits_{x} & c^T x \\ \text{s.c.} & \\ & Ax = b \\ & x \geq 0 \end{array} \tag{1.8}$$

et sa résolution par l'algorithme du simplexe (voir, par exemple, Dantzig [25]).

Lors d'une itération donnée (après un possible ré-indexage des variables), soit $A = (B, N)$ où B et N dénotent respectivement les sous-matrices des colonnes en base et hors-base. Le problème (1.8) peut être exprimé de la façon suivante :

$$\begin{array}{ll} \min\limits_{x_B, x_N} & c_B B^{-1} b + (c_N - c_B B^{-1} N) x_N \\ \text{s.c.} & \\ & x_B + B^{-1} N x_N = B^{-1} b \\ & x_B, x_N \geq 0 \end{array} \tag{1.9}$$

où x_B, x_N sont les vecteurs des variables en bases et hors-base et c_B, c_N les vecteurs des coefficients dans la fonction-objectif correspondants.

Dans la méthode révisée du simplexe, il suffit de mémoriser, en plus des données initiales, la matrice B^{-1} (sous forme compacte), la solution de base courante $B^{-1}b$ et la valeur courante de la fonction-objectif $c_B B^{-1} b$. La variable entrante est déterminée en calculant le coût réduit le plus petit, en utilisant les données initiales, i.e.,

$$c_k - c_B B^{-1} A^k = \min_{j \in N} c_j - c_B B^{-1} A^j = c_j - u^T A^j \tag{1.10}$$

où $u = c_B B^{-1}$ est le vecteur-colonne courant des variables duales. Ce calcul n'est pas trop long lorsque la matrice A est creuse et que les colonnes ne sont pas trop nombreuses.

Ensuite, la colonne entrante est déterminée par $B^{-1}A^k$ et une itération du simplexe se produit de façon usuelle : vérification des conditions d'optimalité, choix de la variable sortante (si le problème est borné), mise à jour de la solution et mise à jour de l'inverse de la matrice de base.

Si le nombre de colonnes croît exponentiellement (comme pour le problème étudié dans ce chapitre et le suivant) avec la taille des paramètres donnés en entrée, il faut résoudre le problème

$$\min_{j \in N} c_j - u^T A^j \tag{1.11}$$

sans considérer explicitement et individuellement chacune des colonnes hors-base. Ceci peut être fait en résolvant le problème auxiliaire, dans lequel les coefficients dans les colonnes A^j sont les variables et où les valeurs des variables duales u_i sont celles associées à la solution optimale du problème maître courant. En résolvant exactement le problème (1.11), nous pouvons trouver la variable hors-base ayant le plus petit coût réduit sans avoir à considérer explicitement chacune des variables hors-base. Si la valeur optimale du problème (1.11) est non-négative, cela implique que la solution de base courante du problème maître est optimale pour le programme linéaire complet (1.8). Sinon, la colonne associée à la variable hors-base de la solution optimale de (1.11) est ajoutée au problème maître, que l'on doit à nouveau optimiser.

1.3.2 Formulation du problème auxiliaire

Afin d'illustrer la façon de formuler le problème auxiliaire pour le problème de plongement en norme-ℓ_1, considérons le problème auxiliaire du modèle linéaire (1.4) avec le critère de la médiane-ℓ_1 minimum, i.e., avec $c_t = \min(|S^t|, |V_n \setminus S^t|)$. Ce problème auxiliaire consiste à trouver le coût réduit minimum, i.e., résoudre

$$\min_{t \in N} c_t - \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} u_{ij} \delta(S^t)_{ij} \tag{1.12}$$

où N est l'ensemble des variables hors-base et u_{ij} est la variable duale de la contrainte associée à d_{ij}.

Posons $x_i = 1$ si $i \in S^t$ et 0 sinon. Alors, le problème (1.12) peut être exprimé comme suit :

$$\begin{aligned} \min_x \sum_{i=1}^{n} x_i - \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} u_{ij}[x_i(1-x_j) + x_j(1-x_i)] \\ x_i \in \{0,1\} \qquad i = 1, 2, \ldots, n \end{aligned} \tag{1.13}$$

ou, après quelques simplifications algébriques de base, par :

$$\begin{aligned} \min_x f(X) = \sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2u_{ij} x_i x_j \\ x_i \in \{0,1\} \qquad i = 1, 2, \ldots, n \end{aligned} \tag{1.14}$$

où $\alpha_i = 1 - \sum_{j=1}^{i-1} u_{ji} - \sum_{j=i+1}^{n} u_{ij}$. Ce problème est un programme quadratique en variables 0–1 sans contraintes.

Le problème (1.14) est équivalent au problème (1.12) car :

(a) $\delta(S^t)_{ij} = 1$ si et seulement si x_i et x_j prennent des valeurs complémentaires, i.e., $x_i(1-x_j) + x_j(1-x_i) = 1$ et ceci est symétrique par rapport à i et j ;

(b) comme le premier terme dans la fonction-objectif est $\sum_{i=1}^{n} x_i$, la solution correspondant à une coupe donnée va toujours être celle avec le plus petit nombre de $x_i = 1$, et jamais la coupe complémentaire à celle-ci.

Notons aussi que le problème (1.12) avec $c_t = 0$ consiste à résoudre un problème de coupe maximale dans un graphe $G(V_n, E_n)$ pour lequel, à chaque arête ij de E_n, est associé un poids u_{ij} de valeur réelle.

1.3.3 Résolution du problème auxiliaire

Le problème (1.14) doit être résolu à chaque itération de la méthode de génération de colonnes. Cette résolution peut augmenter considérablement le temps de calcul. En effet, la classe de programmes quadratiques 0–1 sans contraintes est NP-difficile

puisque le problème de coupe maximale, qui est NP-difficile [39], peut facilement être exprimé sous cette forme. Cependant, afin de garantir la convergence de l'algorithme, il n'est pas impératif de résoudre (1.14) exactement à chaque itération. Une méthode heuristique peut être utilisée tant et aussi longtemps qu'elle trouve une solution de coût réduit négatif. En effet, une itération de l'algorithme du simplexe peut être effectuée en introduisant dans la base n'importe quelle colonne de coût réduit négatif (pas nécessairement la colonne ayant le coût réduit minimum). L'utilisation de telle méthode heuristique est importante dans la résolution de grandes instances puisque, lorsque le nombre de variables 0–1 est grand, utiliser une méthode exacte peut être très coûteux en termes de temps de calcul. Pour le problème (1.4) et pour des instances non-réalisables du problème (1.2), un algorithme exact doit être utilisé pour résoudre le problème auxiliaire au moins une fois afin de prouver qu'il n'existe aucune autre colonne de coût réduit du signe désiré lorsque la méthode heuristique n'en trouve pas, i.e., prouver que les conditions d'optimalité sont satisfaites.

Méthode heuristique

Plusieurs méthodes heuristiques peuvent être utilisées pour résoudre le problème auxiliaire, i.e., pour minimiser une fonction quadratique en variables 0–1 (mentionnons, par exemple, les récentes méthodes heuristiques présentées dans [2, 3, 42, 57, 68, 79, 88]. Deux méta-heuristiques ont été implantées à l'intérieur de notre programme pour résoudre (1.14) : la *recherche avec tabous* [43] et la *recherche à voisinage variable* (RVV) [56, 75].

La *recherche avec tabous* exploite entièrement l'information fournie par le gradient tout en procurant une façon visant à sortir des minima locaux. Lorsqu'on minimise une fonction, à partir d'une solution initiale, une méthode de descente directe est appliquée jusqu'à ce qu'un minimum local soit atteint. Afin d'espérer sortir de ce minimum local, la méthode permet d'accroître la valeur de la fonction en appliquant un déplacement de plus faible montée. Aussi, dans le but de prévenir le cyclage,

lorsqu'un déplacement de montée est appliquée, le retour arrière est déclaré tabou, i.e., interdit pour quelques itérations. L'algorithme 1.1 présente les étapes de notre implantation de la recherche avec tabous. Elle diffère de l'implantation habituelle par le fait qu'elle mémorise plusieurs solutions avec une bonne valeur et pas seulement la meilleure solution trouvée. L'ensemble des solutions voisines $N(X_0)$ d'une solution X_0 correspond à l'ensemble des solutions obtenues en appliquant un déplacement élémentaire qui n'est actuellement pas interdit. Un déplacement élémentaire correspond à complémenter une variable dans le vecteur 0-1 définissant la solution courante du problème (1.14). L'étape B(2)(a) est effectuée en trouvant le déplacement élémentaire non-interdit avec la dérivée partielle

$$\Delta_i = \alpha_i + \sum_{j=1, j\neq i}^{n} 2u_{ij}x_j$$

de valeur minimum. Chaque variable t_i indique le nombre restant d'itérations pour lesquelles la variable x_i demeure tabou. X_k dénote le vecteur obtenu à partir de X_0 en complémentant x_k.

La seconde méthode heuristique que nous avons implantée pour résoudre (1.14) est la version de base de RVV présentée dans [56, 75]. RVV est une méta-heuristique basée sur l'idée de changements systématiques dans le voisinage durant la recherche. RVV explore, de façon probabiliste, les voisinages immédiats et ensuite les voisinages de plus en plus lointains de la meilleure solution trouvée. On commence par analyser des solutions voisines qui sont proches de la meilleure solution trouvée en supposant qu'elles sont probablement prometteuses. RVV applique successivement une routine de recherche locale pour passer de ces solutions voisines à des optima locaux. L'algorithme 1.2 présente les étapes de notre implantation de RVV. L'ensemble des solutions dans le k^{e} voisinage d'une solution est obtenu en appliquant k complémentations sur le vecteur 0-1 décrivant une solution du problème (1.14). Le déplacement utilisé dans la phase de recherche locale (étape B(2)(b)) est la complémentation de la variable ayant la dérivée partielle Δ_i la plus négative.

Algorithme 1.1 Étapes de la recherche avec tabous pour le problème auxiliaire

A Initialisation :
  (1) Soit $L = \emptyset$ l'ensemble des solutions dont la valeur est négative ;
  (2) Posons $t_i = 0$ pour $i = 1, \ldots, n$;
  (3) Choisir la longueur γ de la liste de tabous ;
  (4) Choisir une solution initiale X_0 ;
    (a) $f_{opt} = f(X_0)$; $X_{opt} = X_0$;
    (b) Si $f(X_0) < 0$ alors ajouter X_0 à L ;
    (c) Initialiser les dérivées partielles ;

B Répéter les étapes suivantes jusqu'à ce que $f'_{opt} = f_{opt}$:
  (1) $f'_{opt} = f_{opt}$;
  (2) Répéter les étapes suivantes jusqu'à ce que la condition d'arrêt soit satisfaite :
    (a) Choisir $X_k \in N(X_0)$ tel que $\Delta_k = f(X_k) - f(X_0) = \min_{i|t_i=0} \Delta_i$;
    (b) $X_0 = X_k$; mettre à jour les dérivées partielles ;
    (c) Si $f(X_k) < f_{opt}$ alors $f_{opt} = f(X_0); x_{opt} = X_0$; fin si ;
    (d) Si $f(X_0) < 0$ alors ajouter X_0 à L ;
    (e) Si $\Delta_k > 0$ alors $t_k = \gamma$;
    (f) Fixer $t_i = t_i - 1$ pour $t_i > 0$, $i = 1, 2, \ldots, n$.

Nos implantations de la recherche avec tabou et de la recherche à voisinage variable exploitent le fait que, dans notre algorithme de génération de colonnes, nous désirons insérer, à chaque itération, plusieurs colonnes ayant un coût réduit négatif. Ainsi, plusieurs vecteurs X avec $f(X) < 0$ sont mémorisés. Durant nos expériences avec RVV, nous avons réalisé que le fait de mémoriser seulement les optima locaux de (1.14) de valeur négative a donné de meilleurs résultats que de mémoriser toutes les solutions ayant une valeur négative. Ceci est probablement dû au fait que les colonnes obtenues avec la première méthode sont plus différentes que celles obtenues avec la deuxième méthode. La liste L obtenue à la fin de l'algorithme donne les colonnes à adjoindre au problème maître.

Deux conditions d'arrêt sont utilisées simultanément dans chaque heuristique : des limites sont imposées sur le nombre maximum d'itérations et le cardinalité maximale de L. L'heuristique s'arrête lorsqu'une de ces conditions est atteinte.

Une adaptation des formules présentées dans [55] a été implantée afin de mettre à jour efficacement les dérivées partielles Δ_i durant l'étape B de chaque heuristique.

Algorithme 1.2 Étapes de la RVV pour le problème auxiliaire

A Initialisation :
  (1) Soit $L = \emptyset$ l'ensemble des solutions de valeur négative ;
  (2) Trouver une solution initiale X ; Si $f(X) < 0$, alors ajouter X à L ;
  (3) Choisir un critère d'arrêt ;

B Répéter la séquence suivante jusqu'à l'atteinte de la condition d'arrêt :
  (1) Poser $k \leftarrow 1$;
  (2) Tant que $k < k_{max}$, répéter les étapes suivantes :
    (a) *Perturbation*. Générer un vecteur X' aléatoirement en complémentant k variables de X ;
    (b) *Recherche locale*. Appliquer une méthode de descente directe avec X' comme solution initiale et noter par X'' l'optimum local obtenu ; si $f(X'') < 0$ et $X'' \notin L$ alors ajouter X'' à L ;
    (c) *Se déplacer ou changer le voisinage*. Si $f(X'') < f(X)$, alors se déplacer vers ce nouveau point ($X \leftarrow X''$) et continuer la recherche avec $k = 1$; sinon, explorer un voisinage plus éloigné (poser $k \leftarrow k + 1$).

Mettre à jour chaque dérivée partielle peut être fait en temps constant. Ainsi, chaque itération de la recherche locale prend un temps de l'ordre de $O(n)$.

Le tableau 1.1 présente une comparaison de résultats obtenus en résolvant des problèmes de faisabilité utilisant la recherche avec tabous ou la recherche à voisinage variable pour la résolution heuristique du problème auxiliaire. Pour chaque taille de problème, 20 instances sont résolues. Les instances sont générées en utilisant la méthode 1 décrite à la section 1.4. On donne les statistiques (moyenne et parfois écart-type) du temps CPU total pris par l'algorithme, du nombre de colonnes générées par l'algorithme (Nb cols) ainsi que du temps CPU utilisé par l'heuristique. Notons aussi que les temps sont exprimés en secondes. L'analyse des résultats de ce tableau permet de constater que l'algorithme semble mieux performer lorsque le problème auxiliaire est résolu par RVV. Par conséquent, nous allons nous limiter à utiliser RVV dans les autres expériences présentées dans ce chapitre.

Tableau 1.1 – Résultats en utilisant la recherche avec tabous ou la recherche à voisinage variable pour la résolution heuristique du problème auxiliaire

Taille du pb.	Recherche avec tabous				Recherche à voisinage variable			
	Temps total		Nb cols	Temps heur.	Temps total		Nb cols	Temps heur.
n	μ	σ	μ	μ	μ	σ	μ	μ
10	0.021	0.007	5.8	0.011	0.017	0.005	5.6	0.006
15	0.128	0.029	31.9	0.058	0.089	0.021	28.5	0.022
20	0.542	0.122	69.5	0.156	0.415	0.085	58.3	0.048
25	2.263	0.433	159.8	0.332	1.849	0.339	124.4	0.085
30	7.341	1.463	245.9	0.452	5.868	0.752	199.0	0.137
35	18.296	3.283	307.2	0.316	15.973	2.855	275.4	0.189
40	38.120	6.510	335.0	0.228	36.385	6.566	323.5	0.221
45	78.027	11.814	395.0	0.231	75.181	10.578	374.9	0.239
50	152.930	23.261	420.0	0.281	143.217	19.150	417.8	0.268
55	252.193	39.151	455.0	0.308	262.894	44.552	460.0	0.282
60	499.365	84.642	500.0	0.379	430.769	43.496	500.0	0.309
65	849.762	123.311	560.0	0.483	715.600	84.283	530.0	0.331

Méthode exacte

Pour résoudre le problème auxiliaire de manière exacte, nous utilisons un algorithme récent [52] pour la programmation quadratique sans contraintes qui exploite systématiquement les dérivées du premier ordre à l'intérieur d'un algorithme de séparation et évaluation progressive. Afin d'accroître sa performance, la fonction-objectif est d'abord exprimée comme une *posiforme quadratique*, i.e., une fonction des x_j et leur complément $\overline{x}_j$ avec des coefficients positifs et un terme fixe le plus large possible [47].

1.3.4 Résolution du problème maître

La résolution des problèmes de programmation linéaire par notre algorithme utilise CPLEX 7.0 et exploite sa capacité d'ajouter aisément des colonnes.

1.3.5 Améliorations

Une première amélioration à l'algorithme général de génération de colonnes est de permettre l'introduction de plusieurs colonnes à chaque itération au lieu d'une seule. Nos expériences ont démontré que lorsqu'une seule colonne est ajoutée à chaque itération, la fonction-objectif décroît très lentement. Le fait d'ajouter plusieurs colonnes par itération permet généralement de réduire le nombre d'itérations et ainsi de réduire le temps total de résolution car l'algorithme fait alors moins d'appels à CPLEX. Tel que démontré dans la prochaine section, le temps utilisé par CPLEX constitue habituellement la plus grande part du temps total utilisé par l'algorithme. Le tableau 1.2 présente des résultats sur 20 problèmes d'optimisation du critère de la médiane-ℓ_1 minimum avec 15 points. En comparant les deux première lignes du tableau, on peut constater que l'ajout de 50 colonnes par itération au lieu d'une seule a permis de réduire le temps moyen de calcul par un facteur de 4.5.

Une deuxième amélioration consiste à insérer des colonnes prometteuses dans le problème maître initial (technique appelée *warm start* [4, 69]). En analysant les colonnes dans la base optimale de plusieurs problèmes (1.4) avec $c_t = \min(|S^t|, |V_n \setminus S^t|)$, nous avons constaté que la plupart de ces colonnes avaient la même structure : elles sont associées à des 1-dichotomies et 2-dichotomies, i.e., des coupes dont $\min(|S^t|, |V_n \setminus S^t|) = 1$ ou 2. Ainsi, ces colonnes sont au nombre de $n + n(n-1)/2$, ce qui n'est pas très élevé même lorsque n est grand. Introduire toutes ces colonnes dans le problème maître initial réduit substantiellement le nombre d'itérations et ce faisant le temps de calcul. La comparaison entre la première et la troisième ligne du tableau 1.2 illustre clairement l'impact de cette méthode : initialiser le problème maître avec des colonnes prometteuses a réduit le temps moyen de calcul par une facteur de 18 sur ces problèmes. Pour le *problème de la taille-ℓ_1 minimum*, les résultats empiriques montrent que plusieurs colonnes sont associées à des $\frac{n}{2}$-dichotomies dans la base optimale. Puisque le nombre de ce type de colonnes est très élevé, on ne peut pas toutes les introduire. Ainsi, seulement un sous-ensemble de celles-ci, choisies aléatoirement, sont introduites dans le problème maître initial.

La dernière ligne du tableau 1.2 présente les résultats obtenus en appliquant simultanément les deux améliorations. Pour les 20 problèmes avec 15 points, le temps moyen de calcul est réduit par un facteur de 96.

Tableau 1.2 – Illustration de l'effet des améliorations

Méthode de résolution	Temps total		Nb cols
	μ	σ	μ
Algo. de base (1 col. par it.)	21.285	1.019	457.4
Algo. de base + max. de 50 col. par it.	4.741	0.249	560.2
Algo. de base + *warm start*	1.178	0.523	36.4
Algo. amélioré	0.220	0.058	89.5

Suite à ces résultats préliminaires, nous présentons à la prochaine section des résultats numériques détaillés sur l'ensemble des problèmes en utilisant la version améliorée de l'algorithme.

1.4 Résultats numériques

Il existe plusieurs techniques pour générer une distance d sur un ensemble de points. Les résultats numériques présentés dans ce texte ont été obtenus sur des problèmes générés par les trois méthodes suivantes :

- Méthode 1 : cette méthode trouve d telle que (V_n, d) est toujours ℓ_1-*plongeable*. La distance est obtenue en générant un ensemble de sous-ensembles S^t de V_n, ou de façon équivalente, un ensemble de vecteurs booléens $X^t = \{x_1^t, x_2^t, \ldots, x_n^t\}$ où $x_i^t = 1$ si $i \in S^t$ et 0 sinon. Une valeur réelle, choisie aléatoirement, est associée à chaque vecteur X^t et est ajoutée à d_{ij}, (qui est initialement égale à 0) si $\mid S^t \cap \{i, j\} \mid = 1$.
- Méthode 2 : cette méthode trouve aussi d telle que (V_n, d) est toujours ℓ_1-*plongeable*. La distance est obtenue en générant un ensemble de n vecteurs

$w_1, w_2, \ldots, w_n \in \mathbb{R}^m$ (pour un m donné) correspondant aux coordonnées de n points dans un espace à m dimensions. La distance en norme-ℓ_1 entre chaque paire de points est ensuite calculée, i.e., $d_{ij} = \| w_i - w_j \|_1$ pour tout $1 \leq i \leq j \leq n$.

– Méthode 3 : cette méthode trouve une distance d où chaque d_{ij} est un nombre positif réel généré aléatoirement. Ainsi, la distance d obtenue est telle que (V_n, d) est rarement ℓ_1-*plongeable*

Chaque instance que nous avons générée par la méthode 3 est telle que (V_n, d) n'est pas ℓ_1-*plongeable*. Ainsi, pour chacune de ces instances, le système linéaire (1.2) n'admet aucune solution solution réalisable.

Tous les résultats présentés dans cette section ont été obtenus en utilisant la version améliorée de l'algorithme de génération de colonnes décrit à la section précédente. L'algorithme est implanté en C et utilise CPLEX 7.0 pour résoudre les programmes linéaires. Les résultats ont été obtenus sur un ordinateur PIII avec 750 Mhz et 768 M de RAM.

Nous donnons maintenant quelques détails au niveau des paramètres algorithmiques utilisés. Le nombre maximum de colonnes ajoutées à chaque itération est égal à 100. La méthode RVV est utilisée pour résoudre le problème auxiliaire. La résolution heuristique de chaque problème auxiliaire est arrêtée lorsque 100 colonnes ayant un coût réduit strictement négatif ont été trouvées. Sauf pour le *problème de la taille-ℓ_1 minimum*, le nombre de colonnes insérées initialement, en plus des variables d'écart, d'excédent ou artificielles, est égal à $n + n(n-1)/2$, i.e., le nombre de 1-dichotomies et 2-dichotomies. Pour le *problème de la taille-ℓ_1 minimum*, seules $n(n-1)/2$ colonnes associées à des $\frac{n}{2}$-dichotomies sont considérées au départ.

Pour chaque taille de problème, vingt instances sont résolues et les statistiques (moyenne et parfois écart-type) sur ces 20 instances sont présentées. Ces résultats sont résumés dans les tableaux 1.3 à 1.12. Les statistiques sur le temps concernent le temps CPU et sont exprimées en secondes. *Nb cols* désigne le nombre total de

colonnes générées par l'algorithme, excluant celles insérées au départ. *App.* dénote le nombre d'appels à la méthode durant le déroulement de l'algorithme. Notons que, dans chaque tableau, nous présentons uniquement les résultats pour les tailles de problèmes dont la moyenne du temps de résolution ne dépasse pas 5000 secondes.

1.4.1 Problème de faisabilité

Les tableaux 1.3, 1.4 et 1.5 présentent les résultats pour des problèmes (1.2) où d est générée par les méthodes 1, 2 et 3 respectivement. En observant le tableau 1.3, on peut constater que :

(i) RVV performe bien, car aucun appel à l'algorithme exact pour le problème auxiliaire est nécessaire ;

(ii) la proportion du temps de calcul utilisé par CPLEX 7.0 augmente avec n ; elle atteint plus de 99 % du temps CPU pour les plus gros problèmes résolus, i.e., pour $n = 85$;

(iii) des problèmes de taille assez grande peuvent être résolus en un temps raisonnable.

L'analyse du tableau 1.4 nous permet de constater que : les conclusions (i) et (ii) obtenues pour le tableau 1.3 sont encore valides ; la conclusion (iii) n'est plus vraie : la taille maximum des problèmes résolus dans 1.6 est inférieure à la moitié de ceux résolus dans le tableau 1.3 ; (iv) pour un nombre n donné de points, le temps CPU et le nombre d'itérations décroissent avec la dimension m de l'espace. Notons que les problèmes générés par la méthode 1 sont probablement plongeables dans un espace plus large que ceux générés par la méthode 2.

Tableau 1.3 – Résultats sur des problèmes de faisabilité où d est générée par la méthode 1

Taille du pb.	Algorithme complet			Heuristique		CPLEX 7.0	
	Temps		Nb cols	Temps	App.	Temps	App.
n	μ	σ	μ	μ	μ	μ	μ
10	0.017	0.005	5.6	0.006	1.9	0.008	3.0
15	0.089	0.021	28.5	0.022	2.7	0.064	3.7
20	0.415	0.085	58.3	0.048	2.7	0.361	3.7
25	1.849	0.339	124.4	0.085	2.9	1.750	3.9
30	5.868	0.752	199.0	0.137	3.0	5.700	4.0
35	15.973	2.855	275.4	0.189	3.1	15.737	4.2
40	36.385	6.566	323.5	0.221	3.3	36.090	4.3
45	75.181	10.578	374.9	0.239	3.8	74.828	4.8
50	143.217	19.150	417.8	0.268	4.2	142.793	5.2
55	262.894	44.552	460.0	0.282	4.6	262.403	5.6
60	430.769	43.496	500.0	0.309	5.0	430.175	6.0
65	715.600	84.283	530.0	0.331	5.3	714.915	6.3
70	1214.527	127.901	610.0	0.376	6.1	1213.682	7.1
75	1826.263	234.541	630.0	0.424	6.3	1825.263	7.3
80	2678.345	286.468	660.0	0.482	6.6	2677.169	7.6
85	3995.200	502.558	715.0	0.564	7.2	3993.786	8.2

En analysant le tableau 1.5, on peut remarquer que :

(i) RVV performe toujours bien puisque, sauf pour les plus gros problèmes, un seul appel à l'algorithme exact est fait pour résoudre le problème auxiliaire ;

(ii) la proportion du temps de calcul pris par CPLEX 7.0 augmente encore une fois proportionnellement à n mais elle n'est pas aussi dominante : lorsque n est grand (i.e., $n \geq 40$), CPLEX 7.0 prend la plus large proportion du temps CPU mais le temps pris par l'algorithme exact augmente rapidement : il correspond à environ la moitié du temps utilisé par CPLEX 7.0 pour $n = 70$ et son accroissement rapide empêche la résolution d'instances de taille plus grande ;

(iii) pour les instances pouvant être résolues en un temps raisonnable, le temps CPU est similaire à celui nécessaire pour résoudre des problèmes générés par la méthode 1.

Tableau 1.4 – Résultats sur des problèmes de faisabilité où d est générée par la méthode 2

Taille du pb.		Algorithme complet			Heuristique		CPLEX 7.0	
		Temps		Nb cols	Temps	App.	Temps	App.
n	m	μ	σ	μ	μ	μ	μ	μ
10	5	0.077	0.014	34.9	0.052	9.4	0.019	10.4
10	10	0.046	0.012	19.4	0.028	5.0	0.016	6.0
10	20	0.028	0.007	9.2	0.015	3.0	0.011	4.0
15	5	0.959	0.175	257.4	0.287	14.9	0.664	15.9
15	10	0.578	0.086	126.0	0.212	11.2	0.355	12.2
15	20	0.279	0.053	55.4	0.124	6.3	0.151	7.3
20	5	14.213	2.523	1040.3	0.851	18.6	13.327	19.6
20	10	8.131	1.474	518.9	0.634	12.8	7.479	13.8
20	20	2.872	0.401	186.3	0.445	8.8	2.414	9.8
25	5	218.893	55.524	2692.8	1.358	30.7	217.413	31.7
25	10	79.565	8.597	1268.0	0.991	16.4	78.515	17.4
25	20	25.164	4.167	562.9	0.966	9.7	24.165	10.7
30	5	2768.003	924.399	5164.6	1.758	53.0	2765.926	54.0
30	10	541.226	64.234	2483.1	1.282	26.3	539.773	27.3
30	20	157.675	31.135	1055.8	1.323	12.2	156.272	13.2
35	10	3125.744	292.425	4596.6	1.815	46.7	3123.486	47.7
35	20	865.202	106.542	1816.8	1.455	18.8	863.547	19.8
40	20	4065.629	453.256	2984.0	1.626	30.0	4063.623	31.0

1.4.2 Problèmes d'optimisation

Puisque toutes les instances que nous avons générées avec la méthode 3 sont non-réalisables, nous avons résolu les problèmes d'optimisation que pour les instances générées par les méthodes 1 et 2. Pour ces instances, on peut noter que les problèmes d'optimisation, qui impliquent l'utilisation des deux phases de l'algorithme du simplexe, sont plus difficiles à résoudre que les problèmes de faisabilité de même taille.

Les résultats obtenus sur les problèmes d'optimisation où l'objectif correspond au critère de la médiane-ℓ_1 minimum sont présentés dans les tableaux 1.6 et 1.7. Nous

Tableau 1.5 – Résultats sur des problèmes de faisabilité où d est générée par la méthode 3

Taille du pb.	Algorithme complet			Heuristique		Mét. exacte		CPLEX 7.0	
	Temps		Nb cols	Temps	App.	Temps	App.	Temps	App.
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	0.067	0.020	15.4	0.051	8.4	0.000	1.0	0.009	8.4
15	0.344	0.084	50.1	0.300	12.5	0.003	1.0	0.032	12.5
20	1.200	0.212	129.2	0.981	15.2	0.003	1.0	0.201	15.2
25	3.051	0.555	265.6	2.267	16.2	0.005	1.0	0.755	16.2
30	6.619	1.074	509.4	4.220	16.1	0.019	1.0	2.328	16.1
35	12.694	1.719	797.7	6.397	17.1	0.067	1.0	6.143	17.1
40	25.551	4.736	1145.2	8.853	18.9	0.211	1.0	16.338	18.9
45	56.133	12.299	1589.2	12.940	23.1	0.991	1.0	41.933	23.1
50	93.913	21.656	1903.0	19.431	26.6	2.881	1.0	71.233	26.6
55	192.461	70.907	2375.9	23.944	31.1	9.043	1.1	158.906	31.1
60	393.606	176.618	2870.1	29.126	35.7	44.006	1.1	319.687	35.7
65	737.001	254.704	3524.2	36.294	42.0	107.839	1.2	591.771	42.0
70	1488.785	725.919	4085.6	53.888	49.6	462.872	1.4	970.532	49.6

constatons encore une fois que RVV performe bien et que pour les instances les plus grandes, la majeure partie du temps de calcul est prise par CPLEX 7.0.

Les résultats pour les problèmes d'optimisation dont l'objectif correspond au critère de la taille-ℓ_1 minimum sont présentés dans les tableaux 1.8 et 1.9. Ces problèmes sont plus difficiles que les précédents. Ce phénomène est probablement dû au fait qu'il peut y avoir plus de solutions optimales, i.e., une dégénérescence primale considérable qui est néfaste pour la méthode de génération de colonnes. En effet, le nombre d'appels à l'heuristique ainsi que le nombre de colonnes générées, pour les plus grandes instances résolues, sont environ quatre fois plus élevés pour les problèmes générés par la méthode 1 et deux fois plus élevés pour ceux générés par la méthode 2.

Tableau 1.6 – Résultats sur des problèmes d'optimisation du critère de la médiane-ℓ_1 minimum où d est générée par la méthode 1

Taille du pb.	Algorithme complet			Heuristique		Mét. exacte		CPLEX 7.0	
	Temps		Nb cols	Temps	App.	Temps	App.	Temps	App.
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	0.033	0.016	10.2	0.015	4.3	0.000	1.0	0.012	5.3
15	0.220	0.058	89.5	0.059	6.8	0.000	1.0	0.148	7.8
20	1.522	0.525	271.6	0.138	7.2	0.002	1.0	1.364	8.2
25	9.352	2.322	575.5	0.259	8.6	0.003	1.0	9.055	9.6
30	38.393	10.292	819.0	0.360	10.2	0.010	1.0	37.955	11.2
35	122.610	23.674	1075.4	0.486	12.2	0.039	1.0	121.947	13.2
40	347.069	66.292	1413.5	0.664	15.2	0.212	1.0	345.968	16.2
45	829.228	111.177	1714.8	0.878	18.1	0.965	1.0	827.041	19.1
50	1970.159	288.347	2117.8	1.169	22.2	4.544	1.0	1963.917	23.2
55	4678.657	823.377	2710.0	1.506	28.1	15.283	1.0	4661.076	29.1

Tableau 1.7 – Résultats sur des problèmes d'optimisation du critère de la médiane-ℓ_1 minimum où d est générée par la méthode 2

Taille du pb.		Algorithme complet			Heuristique		Mét. exacte		CPLEX 7.0	
		Temps		Nb cols	Temps	App.	Temps	App.	Temps	App.
n	m	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	5	0.138	0.024	52.2	0.097	15.1	0.000	1.0	0.036	16.6
10	10	0.091	0.021	36.4	0.052	9.4	0.000	1.0	0.034	10.4
10	20	0.059	0.020	17.9	0.036	6.1	0.000	1.0	0.019	7.1
15	5	1.991	0.252	391.9	0.560	23.5	0.001	1.0	1.411	25.5
15	10	1.640	0.199	326.2	0.458	18.4	0.001	1.0	1.164	19.4
15	20	0.793	0.178	192.2	0.294	11.9	0.000	1.0	0.486	12.9
20	5	45.651	12.794	1456.8	1.816	30.1	0.007	1.0	43.785	32.1
20	10	25.710	3.713	1201.2	1.265	21.1	0.002	1.0	24.402	22.1
20	20	13.861	1.272	737.8	1.124	17.1	0.000	1.0	12.709	18.1
25	5	721.345	233.499	3322.3	3.177	43.4	0.092	1.1	717.888	45.2
25	10	290.113	36.203	3008.6	1.778	35.2	0.018	1.0	288.171	36.2
25	20	167.695	24.195	1882.9	1.750	23.9	0.010	1.0	165.835	24.9
30	10	1953.958	264.493	5879.4	2.996	62.5	0.199	1.1	1950.329	63.6
30	20	1300.140	161.446	3645.8	2.487	39.1	0.077	1.0	1297.311	40.1

Tableau 1.8 – Résultats sur des problèmes d'optimisation du critère de la taille-ℓ_1 minimum où d est générée par la méthode 1

Taille du pb.	Algorithme complet			Heuristique		Mét. exacte		CPLEX 7.0	
	Temps		Nb cols	Temps	App.	Temps	App.	Temps	App.
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	1.544	0.228	89.4	1.480	13.7	0.002	1.0	0.057	14.7
15	6.709	0.827	421.8	5.135	14.1	0.013	1.0	1.549	15.1
20	38.124	11.757	1014.0	12.174	17.4	0.374	1.0	25.534	18.4
25	289.590	256.024	2168.4	39.084	34.1	16.371	1.2	234.011	35.1

Tableau 1.9 – Résultats sur des problèmes d'optimisation du critère de la taille-ℓ_1 minimum où d est générée par la méthode 2

Taille du pb.		Algorithme complet			Heuristique		Mét. exacte		CPLEX 7.0	
		Temps		Nb cols	Temps	App.	Temps	App.	Temps	App.
n	m	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	5	0.163	0.029	71.5	0.096	16.4	0.000	1.0	0.061	18.0
10	10	0.153	0.026	73.8	0.089	14.4	0.000	1.0	0.058	15.4
10	20	0.153	0.020	82.2	0.094	14.4	0.001	1.0	0.051	15.4
15	5	2.347	0.366	424.9	0.470	20.6	0.002	1.0	1.854	22.6
15	10	2.238	0.469	397.4	0.393	16.9	0.002	1.0	1.827	17.9
15	20	2.183	0.508	412.4	0.364	16.2	0.007	1.0	1.793	17.2
20	5	53.515	18.185	1580.2	1.477	26.9	0.013	1.0	51.969	28.9
20	10	32.550	6.350	1186.8	0.998	20.2	0.038	1.0	31.468	21.2
20	20	42.213	11.400	1459.0	0.942	23.4	0.155	1.0	41.052	24.4
25	5	792.231	201.917	3730.6	3.109	46.0	0.179	1.0	788.726	47.7
25	10	409.865	122.989	3466.5	1.927	39.6	3.086	1.0	404.648	40.6
25	20	492.669	163.904	4018.7	2.476	46.5	3.712	1.0	486.269	47.5
30	10	4692.654	4936.892	7515.4	10.061	116.2	1324.048	11.7	3357.885	117.3

1.4.3 Problèmes d'approximation

Tel que mentionné à la section 1.2, le problème d'approximation inférieure est équivalent au problème de faisabilité. Ainsi, les résultats du tableau 1.5 correspondent aussi à ceux de l'approximation inférieure. Les résultats pour le problème d'approximation supérieure (1.6) et celui de l'approximation inférieure-supérieure (1.7) sont présentés dans les tableaux 1.10 et 1.11. Finalement, les résultats pour le problème de la constante additive (1.5) sont exposés dans le tableau 1.12.

Les problèmes d'approximation inférieure sont les plus faciles à résoudre ; les problèmes d'approximation inférieure-supérieure sont légèrement plus faciles que ceux de l'approximation supérieure. Ce phénomène peut être dû au fait que la région admissible est moins restreinte. Encore une fois, RVV performe bien puisque le nombre moyen d'appels à l'algorithme exact est relativement faible. On constate aussi que le temps pris par l'algorithme exact augmente beaucoup plus rapidement que pour les autres problèmes. Il s'agit encore du facteur limitant l'accroissement de la taille des problèmes pouvant être résolus en temps raisonnable.

Tableau 1.10 – Résultats sur des problèmes d'approximation supérieure en norme-ℓ_1 où d est générée par la méthode 3

Taille du pb.	Algorithme complet			Heuristique		Mét. exacte		CPLEX 7.0	
	Temps		Nb cols	Temps	App.	Temps	App.	Temps	App.
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	0.113	0.018	39.3	0.089	14.9	0.000	1.0	0.022	16.9
15	0.652	0.139	218.8	0.386	16.9	0.006	1.0	0.250	18.9
20	4.552	0.303	762.4	0.994	17.4	0.059	1.0	3.471	19.4
25	28.174	3.682	1436.5	1.724	21.9	0.935	1.1	25.444	23.9
30	136.572	20.996	2282.7	3.562	32.1	16.520	1.4	116.341	34.1
35	788.164	192.934	3412.6	7.713	45.7	359.708	2.1	420.423	47.7

Tableau 1.11 – Résultats sur des problèmes d'approximation inférieure-supérieure en norme-ℓ_1 où d est générée par la méthode 3

Taille du pb.	Algorithme complet			Heuristique		Mét. exacte		CPLEX 7.0	
	Temps		Nb cols	Temps	App.	Temps	App.	Temps	App.
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	0.087	0.023	23.1	0.065	11.4	0.001	1.0	0.017	12.4
15	0.417	0.067	113.3	0.274	11.9	0.004	1.0	0.135	12.9
20	1.830	0.247	333.1	0.687	10.8	0.051	1.0	1.077	11.8
25	8.423	1.336	635.8	1.170	11.8	0.777	1.0	6.436	12.8
30	36.812	5.761	949.6	2.186	14.8	10.145	1.0	24.408	15.8
35	236.762	44.145	1323.2	3.721	18.4	155.496	1.1	77.415	19.4
40	2751.747	984.009	1749.3	6.660	23.1	2550.537	1.2	194.331	24.1

Finalement, le problème de la constante additive est plus difficile que les autres problèmes d'approximation et tout aussi difficile que celui de la taille-ℓ_1 minimum.

En observant les résultats du tableau 1.12, nous observons que :

(i) le temps augmente très rapidement en fonction de n (par exemple, pour n passant de 15 à 30, le temps est multiplié par environ 900) ;

(ii) RVV performe bien puisque le nombre d'appels à l'algorithme exact pour le problème auxiliaire est presque minimum ;

(iii) pour les plus grandes instances résolues, CPLEX 7.0 accapare plus de 96 % du temps de calcul.

Tableau 1.12 – Résultats sur des problèmes de la constante additive où d est générée par la méthode 3

Taille du pb.	Algorithme complet			Heuristique		Mét. exacte		CPLEX 7.0	
	Temps		Nb cols	Temps	App.	Temps	App.	Temps	App.
n	μ	σ	μ	μ	μ	μ	μ	μ	μ
10	0.550	0.115	37.5	0.498	6.6	0.000	1.0	0.047	8.6
15	3.654	0.631	348.2	2.181	9.1	0.003	1.0	1.462	11.1
20	36.388	8.598	1133.2	4.713	15.1	0.081	1.0	31.549	17.1
25	372.783	109.836	3029.7	8.310	32.2	3.147	1.0	361.178	34.2
30	3301.986	865.874	8019.2	33.405	85.0	81.375	1.1	3186.662	87.0

1.5 Discussion

Nous avons proposé et développé une méthode de résolution pour le problème de plongement-ℓ_1 d'une distance à valeurs réelles ainsi que les problèmes connexes d'optimisation. À notre connaissance, il s'agit de la première méthode exacte, autre que l'utilisation directe de l'algorithme du simplexe, proposée pour résoudre les différentes extensions du problème de plongement-ℓ_1. Cette méthode consiste à utiliser la génération de colonnes avec quelques stratégies d'accélération conjointement avec la programmation quadratique 0-1 sans contraintes pour résoudre le problème auxiliaire. Nous avons résolu en un temps raisonnable des instances de petite à moyenne taille selon le problème (i.e., jusqu'à 25 points pour le problème le plus difficile et jusqu'à 85 points pour le plus facile).

Nos résultats démontrent que notre algorithme considère explicitement un très faible pourcentage du nombre total de colonnes (qui est égal à $2^{n-1} - 1$). Ainsi, on surmonte, dans la plupart des cas, une difficulté signalée par Fichet [34, 35] ainsi que par Benayade et Fichet [14], i.e., l'accroissement exponentiel du nombre de colonnes qui augmente dramatiquement la taille des programmes linéaires rendant impossible la résolution de problèmes où le nombre de points est relativement élevé.

CHAPITRE 2 : FUSION DES APPROCHES LOCALE ET GLOBALE AU PROBLÈME DE LA SATISFIABILITÉ PROBABILISTE

Le problème de la *satisfiabilité probabiliste* (PSAT, aussi connu sous le nom de *logique probabiliste* et *implication probabiliste* [77]) est un problème central dans le raisonnement sous incertitude. Ce problème, basé sur la logique et sur la théorie des probabilités, peut être exprimé comme suit. Considérons un ensemble de m propositions logiques $S_1, S_2, \ldots, S_m$ définies sur n variables logiques $x_1, x_2, \ldots, x_n$ avec les opérateurs booléens usuels $\vee$ (somme logique), $\wedge$ (produit logique) et $^-$ (négation). Considérons également les probabilités $\pi_1, \pi_2, \ldots, \pi_m$ associées à la véracité des m propositions logiques. PSAT, sous forme de problème de décision, consiste à vérifier si ces probabilités sont cohérentes ou non. Sous forme de problème d'optimisation, PSAT consiste à déterminer les meilleures bornes inférieure et supérieure possibles sur la probabilité d'être vraie d'une proposition logique S_{m+1} supplémentaire.

L'étude de PSAT, sous les deux formes ci-haut mentionnées, est apparue pour la première fois dans les travaux de Boole. Ce dernier a exprimé ces deux problèmes, dans son célèbre volume datant de 1854, *An Investigation of the Laws of Thought* [16], sous forme de programmes linéaires (ce qui constitue la plus remarquable préfiguration de la programmation linéaire jusqu'aux travaux de Kantorovich [63] en 1939). Considérant $\pi_1, \pi_2, \ldots, \pi_m$ comme des paramètres, Boole caractérise les conditions de conservation de cohérence de ces probabilités par un ensemble d'équations linéaires qu'il appelle les *conditions d'expérience possible*. De plus, il montre comment exprimer les bornes inférieure et supérieure les meilleures possibles sur π_{m+1} par des inégalités impliquant les probabilités $\pi_1, \pi_2, \ldots, \pi_m$, problème qu'il appelle *le problème général de la théorie des probabilités*.

Le modèle PSAT étudié par Boole constitue, en dépit d'une absence de plus d'un siècle dans les pays anglophones (voir Hailperin [45, 46]), un problème fortement étudié. PSAT a été généralisé pour prendre en compte des intervalles sur les probabilités, des probabilités conditionnelles (tant dans l'objectif que dans les contraintes), des opérations sur des événements conditionnels et des probabilités qualitatives (voir [51] pour plus de détails sur les extensions du modèle PSAT de base et pour des références sur le sujet). De plus, PSAT joue un rôle prépondérant dans la théorie des probabilités subjectives de de Finetti (voir, par exemple, Coletti et Scozzafava [23]) ainsi que dans la théorie des probabilités imprécises de Walley [86].

Il existe deux approches classiques de résolution des problèmes PSAT :

(i) l'*approche globale*, qui s'inspire de l'approche de programmation linéaire de Boole. Hailperin [44], le pionnier au niveau de cette approche, a montré formellement comment écrire PSAT sous forme de programme linéaire. Cette approche a par la suite été étudiée par différents auteurs (voir [51] pour des références) dont l'article de Nilsson [77] qui constitue un des travaux les plus importants et les plus influents sur le sujet.

La principale force de cette approche s'appuie sur sa capacité de prouver rigoureusement la cohérence, ou l'incohérence, des probabilités $\pi_1, \pi_2, \ldots, \pi_m$ et, dans le cas où la cohérence est présente, de trouver avec exactitude les meilleures bornes possibles sur π_{m+1}.

Puisque le nombre de variables dans le programme linéaire augmente exponentiellement avec le nombre de variables logiques $x_1, x_2, \ldots, x_n$, il a été souvent présumé que cette approche était limitée aux instances de petite taille. Cependant, l'utilisation de la technique de génération de colonnes, conjointement avec la programmation non-linéaire en variables 0–1 pour déterminer la variable entrante, a permis de résoudre des problèmes PSAT contenant plusieurs centaines de variables et/ou de propositions logiques (voir [32, 41, 58, 66] et les résultats présentés dans ce chapitre). Néanmoins, les grandes instances nécessitent tout de même généralement un temps de calculs important.

Une autre force de l'approche globale repose sur sa capacité de résoudre analytiquement de petites instances du problème PSAT par l'intermédiaire de l'énumération des points extrêmes et des rayons extrêmes du polyèdre du programme linéaire dual [44, 53]. Ainsi, les meilleures règles possibles pour resserrer les intervalles de probabilité peuvent être générées de façon automatique.

Le point faible de l'approche globale réside dans le fait qu'elle ne fournit pas de justification détaillée des bornes obtenues qui pourrait être analysée étape par étape par l'utilisateur.

(ii) l'*approche locale*, ou *rule-based approach* ou *anytime-deduction approach*, qui exploite les règles pour resserrer les intervalles de probabilité par propagation. De telles règles exploitent les probabilités de véracité sur des prémisses pour borner les probabilités sur la véracité d'une conclusion. Les bornes ainsi trouvées sont localement optimales, i.e., les meilleures possibles pour le petit système considéré. Frisch et Haddawy [37] ont regroupé plusieurs règles de ce type et les ont appliquées à plusieurs exemples. Ils proposent une méthode de déduction instantanée (*anytime-deduction*), dont la résolution peut être interrompue par l'utilisateur à tout moment et qui offre alors un intervalle valide sur π_{m+1}. Ceci est en effet le cas mais le résultat est significatif uniquement si les probabilités $\pi_1, \pi_2, \ldots, \pi_m$ sont cohérentes (sinon n'importe quel intervalle peut en principe être obtenu). Vérifier si les probabilités sont cohérentes est difficile puisque qu'il s'agit d'un problème PSAT en soi. Récemment, d'autres algorithmes [59, 70], qui génèrent des règles et les appliquent de façon systématique, ont été proposés.

Les points forts de l'approche locale sont sa rapidité à déterminer des bornes (même lorsqu'on applique plusieurs règles à toutes les propositions et pas seulement, comme il a été réalisé souvent, à celles impliquant les mêmes variables que la fonction-objectif), ainsi que le fait qu'elle offre une justification facile à interpréter des bornes obtenues.

Son principal point faible réside, tel que mentionné plus haut, dans son incapacité à prouver la cohérence de l'ensemble des probabilités et à garantir l'optimalité des bornes obtenues sur π_{m+1}. En d'autres termes, les méthodes

basées sur cette approche sont généralement incomplètes. De plus, ces défauts sont difficiles à corriger tout en demeurant à l'intérieur des principes de l'approche locale. Pour illustrer ces défauts, Lukasiewicz [70] donne des exemples de problèmes PSAT avec probabilités conditionnelles où les règles convergent à des bornes sur une probabilité qui ne sont pas optimales, et où trouver les bornes optimales requiert un nombre de règles qui croît exponentiellement avec le nombre de variables.

L'objectif de ce chapitre est de démontrer que les défauts de chacune des deux approches peuvent être largement réduits, ou pour certains d'entre eux, complètement éliminés par la fusion des deux approches.

La suite de ce chapitre se présente comme suit : une description détaillée de l'approche globale est donnée à la section 2.1. L'approche locale est décrite à la section 2.2. Dans la section 2.3, nous présentons l'approche fusionnée ainsi que des résultats numériques illustrant à quel point l'approche locale peut accélérer l'approche globale grâce à la technique de stabilisation de la génération de colonnes. Nous ajoutons aussi à la section 2.3, un petit exemple illustrant comment l'approche globale peut être utilisée pour prouver que les règles proposées par l'approche locale sont insuffisantes pour obtenir les bornes optimales. Le chapitre se termine par une brève discussion à la section 2.4.

2.1 Satisfiabilité probabiliste et programmation linéaire : l'approche globale

Dans cette section, nous exprimons PSAT sous forme de programme linéaire et discutons de sa résolution par la méthode de génération de colonnes de la programmation linéaire. Nous présentons également la technique de stabilisation de la génération de colonnes que nous utilisons dans l'approche fusionnée pour accélérer l'algorithme.

2.1.1 Formulation mathématique des problèmes

Afin de formuler PSAT comme un programme linéaire, associons respectivement les valeurs "vrai" et "faux" de x_k et S_i, aux valeurs 1 et 0. Il existe 2^n produits complets w_j, pour $j = 1, 2, \ldots, 2^n$, des variables $x_1, x_2, \ldots, x_n$ sous forme directe ou complémentée (x_i ou $\overline{x}_i$). Ces produits sont appelés, selon Leibniz, *mondes possibles*. Notons que Nilsson [77] utilise le terme "mondes possibles" pour décrire les colonnes distinctes apparaissant dans (2.2) et le terme "mondes impossibles" pour les colonnes n'y apparaissant pas. Pour chaque monde possible w_j toute proposition S_i est vraie ou fausse.

Le problème de la satisfiabilité probabiliste peut être reformulé ainsi : existe-t-il une distribution de probabilité $p_1, p_2, \ldots, p_{2^n}$ sur l'ensemble des mondes possibles telle que la somme des probabilités des mondes possibles dans lesquels la proposition S_i est vraie est égale à la probabilité π_i que S_i soit vraie, pour $i = 1, 2, \ldots, m$?

En définissant la matrice $A = (a_{ij})$ de dimension $m \times 2^n$ par

$$a_{ij} = \begin{cases} 1 & \text{si } S_i \text{ est vraie dans le monde possible } w_j \\ \\ 0 & \text{sinon} \end{cases} \tag{2.1}$$

le problème de la satisfiabilité probabiliste sous sa forme décisionnelle peut s'écrire :

$$\begin{array}{rcl} \mathbb{1}p & = & 1 \\ Ap & = & \pi \\ p & \geq & 0 \end{array} \tag{2.2}$$

où $\mathbb{1}$ est un vecteur-colonne unitaire de dimension 2^n, p et π sont les vecteurs-colonne $(p_1, p_2, \ldots, p_{2^n})^T$ et $(\pi_1, \pi_2, \ldots, \pi_m)^T$ respectivement. Le problème est cohérent s'il existe un vecteur p respectant (2.2) et incohérent sinon.

La considération d'une phrase logique S_{m+1} supplémentaire, avec une probabilité π_{m+1} inconnue, conduit au problème PSAT sous la forme de problème d'optimisation. Généralement, les contraintes (2.2) n'impliquent pas qu'une valeur unique à la

probabilité π_{m+1}. Comme il a démontré par de Finetti [26, 27, 28], ceci est le cas si et seulement le vecteur-ligne $A_{m+1} = (a_{m+1,j})$ où $a_{m+1,j}$ est défini comme dans l'équation (2.1), est une combinaison linéaire des lignes de la matrice A. Sinon, les contraintes (2.2) impliquent plusieurs valeurs distinctes pour la probabilité π_{m+1}. Le problème PSAT sous sa forme d'optimisation consiste à trouver les meilleures bornes sur π_{m+1}.

Sous forme de programme linéaire, il s'écrit de la façon suivante :

$$\begin{array}{lll} \min\limits_{p} \ / \ \max\limits_{p} & A_{m+1}p & \\ \text{s.c.} & & \\ & \mathbb{1}\, p = 1 & \\ & A\, p = \pi & \\ & p \geq 0. & \end{array} \tag{2.3}$$

Nilsson [77] nomme (2.2) et (2.3) *logique probabiliste* et *implication probabiliste* respectivement. Cependant, même si (2.2) et (2.3) sont des outils utiles d'inférence, ils ne constituent pas en réalité une logique, i.e., un ensemble d'axiomes et de règles d'inférence. Le nom de satisfiabilité probabiliste, proposé par Georgakopoulos, Kavvadias et Papadimitriou [41], apparaît donc plus approprié puisqu'il fait ressortir le lien étroit entre le problème (2.2) et le problème de satisfiabilité. En fait, ce dernier est un cas particulier de (2.2) où chaque $\pi = \mathbb{1}$ et où une solution ayant un seul p_j positif est requise (une telle solution peut facilement être déduite de toute solution de (2.3) si certaines colonnes se répètent). Ainsi, le problème PSAT est NP-complet (voir [41] pour une preuve détaillée).

Tel que relaté par Kane [61, 62], deux colonnes de (2.3) peuvent différer seulement par rapport à leur valeur dans A_{m+1}. Ainsi, contrairement à ce que Nilsson [77] suggère, on ne peut les combiner et supposer que leurs probabilités sont égales tout en assurant l'optimalité des bornes sur π_{m+1} que l'on obtient.

Illustrons maintenant la formulation mathématique de PSAT à l'aide d'un exemple.

Exemple 2.1 (Boole's challenge problem [15])
Trouver les meilleures bornes possibles sur la probabilité de $S_6 \equiv x_3$ sous les conditions suivantes :

$$\begin{aligned}
&\text{prob}(S_1 \equiv x_1) = \pi_1 \\
&\text{prob}(S_2 \equiv x_2) = \pi_2 \\
&\text{prob}(S_3 \equiv x_1 \wedge x_3) = \pi_3 \\
&\text{prob}(S_4 \equiv x_2 \wedge x_3) = \pi_4 \\
&\text{prob}(S_5 \equiv \overline{x}_1 \wedge \overline{x}_2 \wedge x_3) = 0.
\end{aligned}$$

En considérant les 8 mondes possibles suivants :

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8
x_1	1	1	1	1	0	0	0	0
x_2	1	1	0	0	1	1	0	0
x_3	1	0	1	0	1	0	1	0

PSAT sous sa forme d'optimisation devient :

$$\begin{array}{llcccccccccccccccc}
\min\limits_{p}/\max\limits_{p} & p_1 & & & + & p_3 & & & + & p_5 & & & + & p_7 & & & & \\
\text{s.c.} & & & & & & & & & & & & & & & & & \\
& p_1 & + & p_2 & + & p_3 & + & p_4 & + & p_5 & + & p_6 & + & p_7 & + & p_8 & = & 1 \\
& p_1 & + & p_2 & + & p_3 & + & p_4 & & & & & & & & & = & \pi_1 \\
& p_1 & + & p_2 & & & & & + & p_5 & + & p_6 & & & & & = & \pi_2 \\
& p_1 & & & + & p_3 & & & & & & & & & & & = & \pi_3 \\
& p_1 & & & & & & & + & p_5 & & & & & & & = & \pi_3 \\
& & & & & & & & & & & & & p_7 & & & = & 0 \\
& p_1 & , & p_2 & , & p_3 & , & p_4 & , & p_5 & , & p_6 & , & p_7 & , & p_8 & \geq & 0
\end{array}$$

2.1.2 Génération de colonnes

Les programmes linéaires (2.2) et (2.3) ont un nombre de colonnes qui croît exponentiellement avec le minimum entre le nombre m de propositions et le nombre n de variables logiques. Étant donné la taille énorme des ces programmes (environ 10^9 colonnes pour $\min(m, n) = 30$, 10^{18} colonnes pour $\min(m, n) = 60$, etc.), il a été mentionné à plusieurs reprises dans la littérature, qu'ils sont impossibles à résoudre en pratique. Par exemple, Nilsson [78], dans une revue de son précédent article de 1986 *Probabilistic Logic* [77], discute de l'impossibilité de résoudre des instances de grande taille et suggère de regarder vers les heuristiques plutôt que vers les méthodes exactes. Nous montrons ici que de telles perspectives sont beaucoup trop pessimistes : bien que pour des instances de grande taille, il soit impossible d'écrire explicitement toutes les colonnes p_j, les problèmes PSAT peuvent être résolus efficacement en considérant ces colonnes implicitement, i.e., en utilisant la technique de génération de colonnes décrite plus amplement à la section 1.3.1 du chapitre 1. Rappelons que deux programmes sont alors associés au programme linéaire original : d'un côté, le *problème maître* qui est identique au programme original mais contenant seulement un nombre restreint de colonnes considérées explicitement, et d'un autre côté, le *problème auxiliaire*, dont le rôle consiste à déterminer la colonne entrante. L'utilisation de la technique de génération de colonnes pour des applications de PSAT à la fiabilité des réseaux a été proposée par Zemel [87]. Plus tard, Georgakopoulos *et al.* [41] ont montré comment se servir de cette technique pour le cas général de PSAT. Ces travaux ont été ensuite prolongés par Jaumard, Hansen et Poggi de Aragão [58].

Formulation du problème auxiliaire

Puisque l'analyse du problème auxiliaire est similaire selon que l'on considère l'objectif de minimisation ou maximisation, nous allons, pour la suite de cette section, analyser uniquement la formulation et la résolution du problème auxiliaire associé

au problème (2.3) avec l'objectif de minimisation. Pour cet objectif, le problème auxiliaire du problème (2.3) est

$$\min_{j \in N} c_j - u_0 - u^T A^j = a_{m+1} - u_0 - \sum_{i=1}^{m} u_i a_{ij} \tag{2.4}$$

où u est le vecteur-colonne $(u_1, \ldots, u_m)^T$ et u_i, pour $i = 0, 1, \ldots, m$ est la variable duale associée à la contrainte i du programme linéaire (2.3). Si pour chaque proposition logique, on attribue les valeurs 1 et 0 selon qu'elle soit vraie ou fausse, on peut réécrire (2.4) ainsi :

$$\min_{j \in N} c_j - u_0 - u^T A^j = S_{m+1} - u_0 - \sum_{i=1}^{m} u_i S_i \tag{2.5}$$

Cette équation peut être reformulée en une expression arithmétique faisant intervenir les variables logiques $x_1, x_2 \ldots, x_n$ apparaissant dans les propositions logiques S_i, pour $i = 1, 2, \ldots, m$. On associe également les valeurs 1 et 0 aux variables logiques selon qu'elle soit vraie ou fausse. La reformulation est réalisée à l'aide des opérateurs usuels de l'algèbre booléenne ($\vee, \wedge$ et $^-$) en utilisant les formules suivantes :

$$\begin{array}{rcl} x_i \vee x_j & \equiv & x_i + x_j - x_i \times x_j \\ x_i \wedge x_j & \equiv & x_i \times x_j \\ \overline{x}_i & \equiv & 1 - x_i. \end{array} \tag{2.6}$$

Suite à ces modifications, le problème auxiliaire devient un problème de minimisation d'une fonction non-linéaire en variables 0–1 ou fonction pseudo-booléenne (Hammer et Rudeanu [48]).

Exemple 2.1 (suite) Pour cet exemple, le problème (2.5) est le suivant :

$$\begin{array}{rcl} \min\limits_{x_1, x_2, x_3 \in \{0,1\}} & & S_6 - u_0 - u_1 S_1 - u_2 S_2 - u_3 S_3 - u_4 S_4 - u_5 S_5 \\ & = & x_3 - u_0 - u_1 x_1 - u_2 x_2 - u_3 x_1 x_3 - u_4 x_2 x_3 - u_5 \overline{x}_1 \overline{x}_2 x_3 \\ & = & -u_0 - u_1 x_1 - u_2 x_2 + (1 - u_5) x_3 + (u_5 - u_3) x_1 x_3 + \\ & & (u_5 - u_4) x_2 x_3 - u_5 x_1 x_2 x_3 \end{array}$$

Résolution du problème auxiliaire

Résoudre le problème (2.5) exactement peut être très coûteux en temps de calcul puisque le problème doit être résolu à chaque itération de l'algorithme de génération de colonnes et qu'il s'agit d'un problème NP-difficile. En effet, le problème de minimisation d'une fonction non-linéaire en variables 0–1 est un problème NP-difficile puisque plusieurs problèmes NP-difficiles [39], par exemple le problème de la coupe maximale, peuvent être facilement formulés sous cette forme. Comme pour le problème traité au chapitre 1, il n'est pas impératif de résoudre le problème auxiliaire exactement à chaque itération pour garantir la convergence de l'algorithme. Une méthode heuristique peut être utilisée tant et aussi longtemps qu'elle trouve une solution de coût réduit négatif. Pour résoudre (2.5), nous avons implanté des versions presque identiques des deux heuristiques utilisées pour résoudre le problème auxiliaire du problème de plongement en norme-ℓ_1 (voir la section 1.3.3 du chapitre 1 pour une description détaillée de ces méthodes). Puisque les résultats préliminaires étaient meilleurs avec la *recherche à voisinage variable*, nous avons décidé de nous limiter à cette méthode pour effectuer toutes les expériences numériques dont les résultats sont présentés dans ce chapitre. Pour le problème (2.3) et pour des instances non-réalisables du problème (2.2), un algorithme exact doit être utilisé pour résoudre le problème auxiliaire au moins une fois afin de prouver qu'il n'existe aucune autre colonne de coût réduit négatif lorsque la méthode heuristique n'en trouve pas, i.e., prouver que les conditions d'optimalité sont satisfaites. La méthode exacte implantée ici est basée sur la technique de linéarisation des termes non-linéaires en variables 0–1 (voir [51] pour une description complète de cette technique).

Insertion de plusieurs colonnes

Comme pour le problème de plongement en norme-ℓ_1 (voir la section 1.3.5 du chapitre 1), nous avons également implanté la possibilité d'introduire plusieurs colonnes à chaque itération de l'algorithme de génération de colonnes dans le but d'en accélérer la convergence.

Dans le tableau 2.1, des résultats obtenus par la résolution, avec insertion d'une seule colonne ou de 50 colonnes par itération, de problèmes PSAT sous la forme d'optimisation sont présentés. Notons que la version actuelle de l'algorithme lorsqu'on impose l'insertion d'une seule colonne par itération est presque identique à celle utilisée dans Jaumard *et al.* [58]. La différence majeure est l'utilisation de la *recherche à voisinage variable* au lieu de la *recherche avec tabous* pour la résolution heuristique du problème auxiliaire. Nous avons noté, sur des résultats empiriques, que l'utilisation de l'une ou l'autre de ces méthodes n'influence pas beaucoup les résultats lorsqu'on insère une seule colonne par itération. Nous pouvons donc considérer l'algorithme utilisé pour obtenir les résultats de la colonne "une seule col. par itération" du tableau 2.1 comme point de référence de l'algorithme présenté dans Jaumard *et al.* [58].

Les problèmes, avec des intervalles sur la probabilité de chaque S_i, ont été générés aléatoirement de manière similaire à ce qui est présenté dans Jaumard *et al.* [58]. Les propositions logiques correspondent à des clauses (disjonction de variables logiques) avec au plus trois variables logiques. Les résultats sur le temps CPU et sur le nombre de colonnes générées sont présentés pour 15 tailles différentes de problème. Pour chaque taille, 30 instances ont été résolues et la moyenne ainsi que l'écart-type sont présentés. Ces résultats ont été obtenus sur un ordinateur *Entreprise 10000* avec 400Mhz et 64 G de RAM. L'algorithme est programmé en C et utilise CPLEX 8.1 pour résoudre les programmes linéaires.

Le tableau 2.1 illustre le fait que la réduction de temps générée par l'insertion de plusieurs colonnes par itération augmente avec la taille du problème. Pour les plus grandes instances, le facteur de réduction est d'environ 2.2.

Tableau 2.1 – Résultats sur des problèmes PSAT obtenus avec insertion d'une seule ou de plusieurs colonnes par itération

Taille du pb.		Une seule col. par itération				Max. de 50 col. par itération			
		Temps		Nb. cols		Temps		Nb. cols	
n	m	μ	σ	μ	σ	μ	σ	μ	σ
50	50	0.499	0.078	73.9	8.101	0.506	0.065	313.7	51.511
50	100	3.383	0.484	124.8	10.852	2.864	0.388	651.0	73.701
50	200	26.257	4.163	204.5	16.788	20.887	3.163	1082.2	127.982
50	400	262.884	46.727	350.2	28.690	196.167	31.487	1621.3	163.107
50	800	2834.596	490.534	564.9	38.926	2005.112	296.974	2183.1	198.275
100	50	0.621	0.084	83.6	8.134	0.628	0.083	319.8	43.162
100	100	4.324	0.552	142.0	10.275	3.736	0.489	752.3	94.842
100	200	35.737	4.938	227.8	15.339	26.882	3.139	1498.0	130.407
100	400	373.715	40.415	387.0	21.771	247.900	26.521	2770.1	204.900
100	800	4577.427	573.359	639.7	37.816	2683.776	298.046	4063.2	324.646
200	50	0.739	0.094	90.0	7.073	0.771	0.099	327.0	39.995
200	100	4.961	0.595	150.3	10.796	4.310	0.523	741.5	72.499
200	200	43.972	6.588	251.5	21.587	30.982	3.811	1723.4	175.824
200	400	452.074	47.317	425.4	26.297	269.289	25.093	3871.9	284.206
200	800	5757.800	541.702	676.6	30.476	2628.950	246.916	6960.8	467.485

2.1.3 Stabilisation

Dans le but d'accélérer la convergence de l'algorithme de génération de colonnes, il est possible d'utiliser la technique de stabilisation proposée par Du Merle *et al.* [33]. Cette technique permet généralement de stabiliser la valeur des variables duales durant le déroulement de l'algorithme. Pour ce faire, il suffit de perturber le programme linéaire en ajoutant des variables d'écart et d'excédent dans le programme primal et de pénaliser ces variables dans la fonction-objectif. Ces modifications dans le primal correspondent à l'introduction de bornes sur les variables duales dans le programme dual de ce programme linéaire et à la pénalisation de ces variables lorsqu'elles prennent une valeur hors d'un intervalle donné. Une estimation des valeurs optimales des variables duales est utilisée, si disponible, pour déterminer les intervalles initiaux. La stabilisation conduit généralement à une réduction substantielle du nombre d'itérations nécessaires pour obtenir une solution optimale du programme linéaire original puisque les variables duales prennent alors moins d'itérations pour

se stabiliser à leur valeur optimale.

Nous allons maintenant analyser en détail les modifications apportées aux programmes linéaires (primal et dual) associés à PSAT suite à l'ajout de la stabilisation. Pour ce faire, considérons le programme primal du problème PSAT (2.3) sous la forme de minimisation :

$$\begin{array}{ll} \min\limits_{p} & A_{m+1}p \\ \text{s.c.} & \\ & \mathbb{1}\, p = 1 \\ & A\, p = \pi \\ & p \geq 0 \end{array} \tag{2.7}$$

ainsi que son programme dual :

$$\begin{array}{ll} \underset{u_o,u}{\text{Max}} & u_0 + \pi^T u \\ \text{s.c.} & \\ & \mathbb{1} u_0 + A^T u \leq A_{m+1}^T \end{array} \tag{2.8}$$

où u est le vecteur-colonne $(u_1, u_2, \ldots, u_m)^T$ et u_i, pour $i = 0, 1, \ldots, m$, est la variable duale associée à la contrainte i du programme primal.

En appliquant la stabilisation, nous obtenons le programme primal :

$$\begin{array}{ll} \min\limits_{p,y_0^-,y_0^+,y^-,y^+} & A_{m+1}p \\ & -\delta_0^- y_0^- + \delta_0^+ y_0^+ \\ & -\delta^{-T} y^- + \delta^{+T} y^+ \\ \text{s.c.} & \\ & \mathbb{1}\, p - y_0^- + y_0^+ = 1 \\ & A\, p - y^- + y^+ = \pi \\ & y_0^- \leq \epsilon_0^-, y_0^+ \leq \epsilon_0^+ \\ & y^- \leq \epsilon^-, y^+ \leq \epsilon^+ \\ & p, y_0^-, y_0^+, y^-, y^+ \geq 0 \end{array} \tag{2.9}$$

ainsi que son programme dual :

$$\begin{aligned}
\underset{u_0,u,w_0^-,w_0^+,w^-,w^+}{\text{Max}} \quad & u_0 + \pi^T u \\
& -\epsilon_0^- w_0^- - \epsilon_0^+ w_0^+ \\
& -\epsilon^{-T} w^- - \epsilon^{+T} w^+ \\
\text{s.c.} \quad & \\
& \mathbb{1} u_0 + A^T u \leq A_{m+1}^T \\
& -u_0 - w_0^- \leq -\delta_0^- \\
& u_0 - w_0^+ \leq \delta_0^+ \\
& -u - w^- \leq -\delta^- \\
& u - w^+ \leq \delta^+ \\
& w_0^-, w_0^+, w^-, w^+ \geq 0
\end{aligned} \tag{2.10}$$

où y^-, y^+, w^- et w^+ sont les vecteurs-colonne $(y_1^-, y_2^-, \ldots, y_m^-)^T$, $(y_1^+, y_2^+, \ldots, y_m^+)^T$, $(w_1^-, w_2^-, \ldots, w_m^-)^T$ et $(w_1^+, w_2^+, \ldots, w_m^+)^T$ respectivement. Dans le programme (2.9), les variables y_i^- et y_i^+ représentent respectivement les variables d'écart et d'excédent de la contrainte i, pour $i = 0, 1, \ldots, m$. Elles sont bornées par ϵ_i^- et ϵ_i^+ et elles sont pénalisées dans la fonction-objectif par δ_i^- et δ_i^+. Dans le programme (2.10), les variables w_i^- et w_i^+ correspondent à la valeur de violation de la borne inférieure δ_i^- ou borne supérieure δ_i^+ sur la variable duale u_i, pour $i = 0, 1, \ldots, m$. Ces variables sont pénalisées dans la fonction-objectif par δ_i^- et δ_i^+ respectivement.

En somme, il y a deux groupes de paramètres : (i) les pénalités δ_i^- et δ_i^+, pour $i = 0, 1, \ldots, m$, qui correspondent implicitement aux bornes sur les variables duales ; (ii) les bornes ϵ_i^- et ϵ_i^+, pour $i = 0, 1, \ldots, m$, sur les variables d'écart et d'excédent qui correspondent implicitement à des pénalités sur la violation des bornes δ_i^- et δ_i^+ sur les variables duales. À chaque itération, on fixe tous les paramètres et on résout (2.9) (ou (2.10)). Ensuite, on vérifie si les variables d'écart et d'excédent sont toutes à 0. Si c'est le cas, on arrête car on a trouvé la solution optimale du programme linéaire (2.7). Sinon, les paramètres sont mis à jour (voir la section 2.3 pour une description du type de mise à jour effectuée à l'intérieur de notre algorithme stabilisé) et une nouvelle itération prend place.

2.2 Approche locale

L'approche locale est une méthode heuristique qui résout des problèmes PSAT sous les formes de décision et d'optimisation en considérant un ensemble de règles au lieu d'utiliser les outils de la programmation linéaire. Ce type d'approche a été étudié, entre autres, par Frisch et Haddawy [37], Lukasiewicz [70] et Jaumard *et al.* [59] [80]. Nous nous concentrons ici sur l'algorithme AD-PSAT [80] qui est une version améliorée de l'algorithme TURBOSAT présenté dans [59].

AD-PSAT considère une série de règles (voir la figure 2.1 pour une illustration de ce type de règle et [80] pour la liste complète) comprenant chacune un système composé d'un ensemble de propositions logiques avec leur probabilité respective. Il utilise ces règles de façon à raffiner progressivement l'intervalle de probabilité sur chaque variable pour en arriver à prouver l'incohérence du problème PSAT analysé ou, s'il y a cohérence, pour trouver une approximation des meilleures bornes possibles sur π_{m+1}. Cet algorithme utilise des règles qui correspondent aux solutions analytiques d'un ensemble de petits problèmes PSAT. Ces solutions sont obtenues en utilisant l'énumération des points extrêmes et des rayons extrêmes du polyèdre du programme linéaire dual (2.8) de ces petits problèmes PSAT (voir Hansen *et al.* [53] pour une description détaillée de cette approche). AD-PSAT est une méthode séquentielle qui, à chaque itération, essaie de raffiner l'intervalle de probabilité sur une variable donnée. Ce raffinement se fait en utilisant la règle associée à un problème PSAT réduit. Ce problème est constitué d'un petit système logique contenant un sous-ensemble donné de l'ensemble complet des m propositions logiques S_i conjointement avec les intervalles de probabilité trouvés jusqu'à présent sur les variables contenues dans le système. La phrase logique S_{m+1} est, quant à elle, constituée uniquement de la variable dont on veut raffiner les bornes. Illustrons maintenant le fonctionnement d'AD-PSAT sur un petit exemple.

Propositions logiques	Probabilités	Cond. d'exp. possible	$\underline{\pi}$ =? maximum de :	$\overline{\pi}$ =? minimum de :
x_1	$[\underline{\pi}_1, \overline{\pi}_1]$	$\pi_i \leq 1 \quad i = 1, 2$	0	1
$\overline{x}_1 \vee x_2$	$[\underline{\pi}_2, \overline{\pi}_2]$	$\pi_i \geq 0 \quad i = 1, 2$	$\underline{\pi}_1 + \underline{\pi}_2 - 1$	$\overline{\pi}_2$
x_2	$\pi\,?$	$\underline{\pi}_i \leq \overline{\pi}_i \quad i = 1, 2$		
		$\overline{\pi}_1 + \overline{\pi}_2 \geq 1$		

Figure 2.1 – Illustration d'une règle utilisée par AD-PSAT

Exemple 2.2

Trouver les meilleures bornes possibles sur la probabilité de $S_4 \equiv x_3$ sous les conditions suivantes :

$$\text{prob}(S_1 \equiv x_1) \in [0.5, 0.9] \tag{2.11}$$

$$\text{prob}(S_2 \equiv \overline{x}_1 \vee x_2) \in [0.8, 0.9] \tag{2.12}$$

$$\text{prob}(S_3 \equiv \overline{x}_2 \vee x_3) \in [0.75, 0.85]. \tag{2.13}$$

AD-PSAT va, en premier lieu, tenter de raffiner les bornes sur $\text{prob}(x_2)$ tout en considérant le sous-système logique composé de (2.11) et (2.12). En appliquant la règle de la figure 2.1, l'algorithme conclut que le sous-système est cohérent puisque les conditions d'expérience possible sont respectées et il obtient

$$\text{prob}(x_2) \in [0.3, 0.9]. \tag{2.14}$$

Ensuite, en appliquant la même règle au sous-système logique composé de (2.14) et (2.13), AD-PSAT conclut que le sous-système est cohérent et il met à jour l'intervalle de probabilité sur S_4 à $[0.05, 0.85]$. Le déroulement de l'algorithme s'arrête alors en n'ayant détecté aucune incohérence et trouve, pour cet exemple, que la solution optimale, i.e., l'intervalle optimal sur π_4, est $[0.05, 0.85]$.

Pour les problèmes que nous avons analysés, et ce même pour les instances de grande taille, peu de propositions logiques de l'ensemble initial sont habituellement utilisées par AD-PSAT pour trouver les meilleures bornes sur π_{m+1} ou une bonne approximation de celles-ci. AD-PSAT résout des instances de grande taille très rapidement (moins de deux secondes sur des problèmes contenant jusqu'à 1000 variables logiques et 3000 propositions logiques).

2.3 Approche fusionnée

Dans cette section, nous présentons une façon de fusionner les approches locale et globale en une nouvelle approche qui réduit considérablement les défauts des deux approches classiques. L'approche fusionnée utilise l'approche locale pour obtenir rapidement une bonne estimation de la solution optimale du problème (2.3). En analysant le déroulement de la résolution faite par l'approche locale, on peut identifier les propositions S_i qui ont eu un effet direct ou indirect sur l'estimation des bornes de π_{m+1}. Cette information peut ensuite être utilisée pour accélérer la résolution du problème par l'approche globale.

2.3.1 Description détaillée de l'approche fusionnée

L'algorithme 2.1 présente les étapes détaillées de l'approche fusionnée. L'étape (a) consiste en l'utilisation d'une méthode locale pour trouver, si elle ne détecte pas d'incohérence dans le problème, une évaluation des bornes de π_{m+1} ainsi que la liste des propositions logiques qui ont influencées directement ou indirectement les bornes trouvées. Si la solution heuristique correspond à la solution exacte, la liste de propositions identifiées correspond au sous-ensemble de propositions logiques S_i qui sont déterminantes, i.e., qui sont les seules à influencer les bornes sur π_{m+1}. L'étape (b) permet d'obtenir une première évaluation $\hat{u}_i$ de chaque variable duale u_i du problème (2.8) : $\hat{u}_i$ est égale à la valeur optimale de la variable duale correspondante du programme linéaire compact considéré à l'étape (b) et égale à 0 sinon. Les approximations $\hat{u}_i$ sont utilisées pour initialiser les pénalités δ_i^- et δ_i^+ du problème (2.9). La méthode de génération de colonnes stabilisée (telle qu'expliquée à la section 2.1.3) est utilisée à l'étape (c) pour obtenir la solution exacte du problème original (2.3).

Algorithme 2.1 Étapes de l'approche fusionnée

(a) Utiliser l'approche locale (par exemple, AD-PSAT) pour trouver des conclusions provisoires pour le problème (2.3). Si l'approche locale détecte une incohérence, arrêter l'algorithme puisque le problème (2.3) est incohérent. Sinon, identifier les propositions S_i utilisées par l'approche locale pour trouver les valeurs finales de $\underline{\pi}_{m+1}$ et $\overline{\pi}_{m+1}$ et aller à l'étape (b).

(b) Utiliser l'approche globale (algorithme de génération de colonnes) pour résoudre un programme linéaire compact comprenant uniquement les contraintes associées aux S_i identifiées à l'étape (a). Si le programme linéaire compact est non-réalisable, arrêter l'algorithme puisque le problème complet (2.3) est incohérent. Sinon, utiliser les valeurs optimales des variables duales de ce programme linéaire compact comme estimations de celles du programme linéaire complet et aller à l'étape (c).

(c) Résoudre le programme linéaire complet (2.3) par l'algorithme de génération de colonnes stabilisé en utilisant les estimations des variables duales afin de confirmer les résultats obtenus par AD-PSAT ou de trouver la solution optimale exacte.

Pour l'étape (a), il faut trouver une méthode pour identifier les propositions logiques qui ont influencé directement ou indirectement l'évaluation heuristique des bornes sur π_{m+1}. Pour notre algorithme spécifique, après avoir résolu le problème avec l'algorithme AD-PSAT, on retrouve les propositions logiques déterminantes pour AD-PSAT en commençant par identifier les propositions qui ont permis de faire la dernière mise à jour des bornes sur π_{m+1}. Pour chacune de ces propositions, on identifie les variables incluses dans cette proposition dont les bornes ont été modifiées par AD-PSAT. Ensuite, pour chacune de ces variables, on détermine quelles sont les propositions qui ont permis de faire la dernière mise à jour des bornes sur cette variable et on répète la même procédure tout en prenant bien soin, pour ne pas cycler, de ne pas analyser deux fois la même variable. Il s'agit en somme d'une procédure récursive qui parcourt à rebours une arborescence associée à la mise à jour des bornes sur π_{m+1}. Il a donc fallu modifier un peu le code de l'algorithme AD-PSAT de manière à ce que, à la fin de l'algorithme, on connaisse pour chaque variable, les propositions qui ont permis de faire la dernière mise à jour des bornes sur sa probabilité.

Pour l'étape (c), il faut trouver une façon efficace d'initialiser et de mettre à jour les paramètres de l'algorithme stabilisé de génération de colonnes. Dans notre version de l'algorithme stabilisé, la gestion des paramètres prend en considération les

estimations données par l'approche locale, la solution courante ainsi que la structure des problèmes PSAT. Les mises à jour sont uniquement réalisées lors de l'obtention de la solution optimale du programme linéaire courant (2.9), i.e., le programme linéaire pour une valeur donnée des paramètres. Tel que mentionné à la section 2.3.1, les estimations $\hat{u}_i$ sont utilisées pour initialiser les pénalités δ_i^- et δ_i^+. Pour les résultats présentés à la prochaine section, l'intervalle $[\delta_i^-, \delta_i^+]$ sur la variable duale u_i est initialisée à $[\hat{u}_i - 10^{-3}, \hat{u}_i + 10^{-3}]$. Lors de la mise à jour des paramètres, si la valeur courante d'une variable duale se trouve à l'extérieur de son intervalle $[\delta_i^-, \delta_i^+]$, cet intervalle est mis à jour de manière à ce que son centre corresponde à la valeur courante de la variable duale et la longueur de l'intervalle est doublée. Les mises à jour des paramètres exploitent aussi un résultat empirique que nous avons noté : la plupart des variables duales ont une valeur optimale qui est égale à -1, 0 ou 1. Après avoir mis à jour le même intervalle à plusieurs reprises, si la variable duale semble progresser vers l'une des trois valeurs attendues, l'algorithme centre alors le nouvel intervalle autour de cette valeur. En ce qui concerne les bornes ϵ_i^- et ϵ_i^+, pour $i = 0, 1, \ldots, m$, elles sont initialisées à 10^{-1} et, durant la mise à jour des paramètres, elles sont réduites progressivement à 0.

2.3.2 Contribution de l'approche locale

Dans cette section, nous présentons des résultats numériques illustrant à quel point l'utilisation de la solution obtenue par l'approche locale peut accélérer la résolution par l'approche globale.

Les résultats présentés dans le tableau 2.2 ont été obtenus sur le même jeu de données et avec le même ordinateur que les résultats présentés dans le tableau 2.1. Pour les deux approches comparées dans le tableau 2.2, on insère jusqu'à un maximum de 50 colonnes par itération de l'algorithme de génération de colonnes. Notons que les résultats de la colonne "approche globale" du tableau 2.2 sont les mêmes que ceux de la colonne "Max. de 50 col. par itération" du tableau 2.1.

Tableau 2.2 – Résultats sur des problèmes PSAT obtenus avec l'approche globale de base ainsi qu'avec l'approche fusionnée

Taille du pb.		Approche globale				Approche fusionnée			
		Temps		Nb. cols.		Temps		Nb. cols.	
n	m	μ	σ	μ	σ	μ	σ	μ	σ
50	50	0.506	0.065	313.7	51.511	0.372	0.043	150.9	22.758
50	100	2.864	0.388	651.0	73.701	2.132	0.244	338.7	47.132
50	200	20.887	3.163	1082.2	127.982	14.861	1.549	598.8	51.205
50	400	196.167	31.487	1621.3	163.107	136.608	21.307	969.2	86.758
50	800	2005.112	296.974	2183.1	198.275	1957.698	557.632	1744.8	378.213
100	50	0.628	0.083	319.8	43.162	0.445	0.053	153.6	19.399
100	100	3.736	0.489	752.3	94.842	2.329	0.235	360.0	47.034
100	200	26.882	3.139	1498.0	130.407	15.332	1.714	726.0	75.264
100	400	247.900	26.521	2770.1	204.900	126.800	15.183	1409.7	122.005
100	800	2683.776	298.046	4063.2	324.646	1214.825	144.532	2173.3	124.342
200	50	0.771	0.099	327.0	39.995	0.505	0.045	149.2	14.462
200	100	4.310	0.523	741.5	72.499	2.474	0.251	336.7	40.802
200	200	30.982	3.811	1723.4	175.824	16.237	1.880	806.4	94.164
200	400	269.289	25.093	3871.9	284.206	137.505	16.398	1879.1	189.935
200	800	2628.950	246.916	6960.8	467.485	1122.706	81.916	3460.1	177.270

Le tableau 2.2 illustre le fait que l'approche fusionnée est toujours plus rapide que l'approche globale sans stabilisation. Ceci est dû à la réduction dans le nombre d'itérations prises par l'algorithme de génération de colonnes qui est illustré dans le tableau 2.2 par la réduction dans le nombre de colonnes générées. Dans le meilleur des cas, le temps a été réduit par un facteur de 2.3 et, dans le cas moyen, ce facteur est de 1.7. Cependant, dans certains cas, la réduction de temps est assez faible. Le pire cas est obtenu pour la taille de problème où $n = 50$ et $m = 800$. La moyenne du temps de calculs est alors sensiblement la même d'une approche à l'autre. Ce résultat est dû au fait que les instances pour cette taille de problème sont très difficiles à résoudre et l'approche locale n'a obtenu la solution optimale dans aucune de ces 30 instances. Dans de tel cas, les estimations initiales des valeurs optimales des variables duales obtenues à l'étape (b) de l'algorithme 2.1 sont mauvaises et l'algorithme prend alors beaucoup plus d'itérations pour les stabiliser à leur valeur optimale.

2.3.3 Contribution de l'approche globale

Puisque l'approche locale trouve une solution heuristique, il n'y a pas de garantie quant à l'optimalité de cette solution. En appliquant l'approche globale après avoir utilisé l'approche locale, on peut : (i) confirmer l'optimalité de la solution heuristique, (ii) trouver la solution optimale lorsque l'approche locale ne la trouve pas ou (iii) prouver que le problème est incohérent. Dans le cas (i), l'approche locale donne une justification complète de la solution optimale obtenue, qui peut être facilement analysée étape par étape par l'utilisateur. Nous considérons maintenant un exemple tiré de [37] dans lequel la solution optimale n'a pas été trouvée par l'approche locale proposée dans cet article. Même dans un cas semblable, l'approche fusionnée détecte l'incohérence grâce à l'étape (c) où le problème est résolu par l'approche globale.

Exemple 2.3 (Exemple 4 de Frisch et Haddawy [37])
Trouver les meilleures bornes possibles sur la probabilité de $S_7 \equiv x_2 \wedge x_3$ sous les conditions suivantes :

$$\text{prob}(S_1 \equiv x_1) \in [0.6, 1] \tag{2.15}$$

$$\text{prob}(S_2 \equiv \overline{x}_1 \vee x_2) \in [0.8, 0.9] \tag{2.16}$$

$$\text{prob}(S_3 \equiv \overline{x}_1 \vee x_3) \in [0.9, 1] \tag{2.17}$$

$$\text{prob}(S_4 \equiv \overline{x}_2 \vee x_4) \in [0.5, 0.8] \tag{2.18}$$

$$\text{prob}(S_5 \equiv \overline{x}_3 \vee x_4) \in [0.8, 0.9] \tag{2.19}$$

$$\text{prob}(S_6 \equiv x_4) \in [0, 0.2]. \tag{2.20}$$

En utilisant les règles présentées dans [37] (notées ci-dessous en chiffres romains), il est possible de démontrer l'incohérence de ces intervalles de probabilité. En appliquant la règle (XX) à (2.15) et (2.17), nous obtenons

$$\text{prob}(x_3) \in [0.5, 1.0]. \tag{2.21}$$

En appliquant la règle (XXi) à (2.19) et (2.20), nous obtenons

$$\text{prob}(x_3) \in [0.1, 0.4]. \tag{2.22}$$

En appliquant la règle (xvii) à (2.21) et (2.22), nous obtenons

$$0.5 \leq \text{prob}(x_3) \leq 0.4,$$

ce qui prouve l'incohérence du système.

Frisch et Haddawy n'ont pas détecté cette incohérence parce qu'ils n'ont pas appliqué les règles pour raffiner les bornes sur la probabilité de x_3 dans la recherche des meilleures bornes sur la probabilité de S_7. AD-PSAT détecte cette incohérence car cet algorithme applique des règles pour raffiner des bornes sur la probabilité des variables. Pour cet exemple, il s'avère qu'AD-PSAT tente de mettre à jour les bornes sur chaque variable logique du système puisque chaque variable apparaît dans S_7 ou est reliée directement ou indirectement (par les propositions $S_1, S_2, \ldots, S_6$) aux variables dans S_7. Immédiatement à l'étape (a), l'incohérence est détectée par AD-PSAT. En fait, il aurait été possible (tel qu'illustré ci-haut) de le trouver avec les règles regroupées dans [37] si les bornes sur chaque variable de S_7 avaient été considérées.

2.4 Discussion

Une toute nouvelle approche pour résoudre le problème PSAT a été proposée et développée dans ce chapitre. Cette approche fusionne les deux approches classiques habituellement utilisées pour résoudre ce problème : l'approche locale et l'approche globale. L'approche fusionnée est une méthode exacte combinant une méthode locale et un algorithme de génération de colonnes stabilisé qui utilise les techniques de la programmation non-linéaire en variables 0–1 sans contraintes pour résoudre le problème auxiliaire.

Nous avons montré que la fusion des approches locale et globale est bénéfique pour chacune d'elles. L'approche locale accélère l'approche globale en fournissant une solution duale heuristique qui est ensuite utilisée par l'approche globale via un algorithme

de génération de colonnes stabilisé. L'approche globale, détecte une possible incohérence non-détectée par l'approche locale ou dans le cas de cohérence du système, confirme ou réfute l'optimalité de la solution locale trouvée. En conséquence, l'approche fusionnée garantit l'obtention des meilleures bornes et, si les bornes obtenues par les deux approches coïncident, fournit une justification détaillée de la solution optimale.

CHAPITRE 3 : PROGRAMMATION QUADRATIQUE

Le problème quadratique à contraintes quadratiques (QQP) est un problème d'optimisation globale général et très difficile. Sa généralité provient du fait qu'il englobe une série d'autres problèmes d'optimisation dont les programmes fractionnaires, les programmes bilinéaires et les programmes biniveaux. Ces problèmes peuvent facilement être reformulés comme des cas particuliers de QQP [6]. Cette généralité fait en sorte qu'il existe des difficultés théoriques et pratiques à la résolution de problèmes QQP. La complexité de QQP se présente à deux niveaux : son domaine réalisable et sa fonction-objectif. En effet, la simple question de trouver une solution réalisable de QQP est un problème fortement NP-complet [6]. Même si on considère un problème QQP ne contenant aucun terme quadratique dans les contraintes, la complexité demeure. En effet, optimiser une fonction quadratique est un problème fortement NP-difficile (Hansen, Jaumard et Savard [54] démontrent qu'un problème équivalent, le problème linéaire maxmin, est NP-difficile). On ne doit donc pas s'attendre à trouver d'ici peu un algorithme exact qui résolve en temps raisonnable des problèmes de grande taille avec un nombre très élevé de termes quadratiques.

Le cas général du problème QQP peut se formuler ainsi :

$$\begin{array}{lll} \min\limits_{x\in X} \left(\max\limits_{x\in X}\right) & Q^0(x) & \\ & \text{s.c.} \quad Q^k(x) \mathrel{\overline{\leq}} b_k & k = 1, 2, \ldots, \overline{k}, \end{array}$$

où $X = \{x \in \mathbb{R}^n : \ Ax \leq a\}$, et pour chaque indice k dans l'ensemble $K = \{0, 1, \ldots, \overline{k}\}$

$$\begin{array}{rcl} Q^k : \mathbb{R}^n & \rightarrow & \mathbb{R} \\ x & \mapsto & Q^k(x) = \sum\limits_{(i,j)\in M} C_{ij}^k x_i x_j + \sum\limits_{i\in N} c_i^k x_i^2 + \sum\limits_{i\in N} d_i^k x_i, \end{array}$$

sont des fonctions quadratiques où $N = \{1, 2, \ldots, n\}$ et $M = \{(i, j) \in N \times N : i > j\}$ sont des ensembles d'indices. Le symbole $\overset{=}{\leq}$ signifie que les contraintes peuvent être autant des égalités que des inégalités. Les dimensions des matrices et des vecteurs sont les suivantes :

$$x \in \mathbb{R}^n; A \in \mathbb{R}^{m \times n}; a \in \mathbb{R}^m; b \in \mathbb{R}^{\overline{k}};$$
$$C_{ij}^k, c_i^k, d_i^k \in \mathbb{R} \text{ pour tout } (i, j) \in M \text{ et } k \in K.$$

Aucune restriction n'est imposée en ce qui concerne la convexité ou la concavité de la fonction-objectif ou des contraintes. La seule hypothèse considérée dans l'établissement de l'algorithme présenté à la prochaine section est qu'il est possible de trouver des bornes inférieure et supérieure de valeur finie sur chaque variable et donc que $\forall i \in N, x_i \in [\ell_i, u_i]$.

La section 3.1 donne une description de l'algorithme utilisé pour résoudre les problèmes quadratiques présentés dans les sections subséquentes de ce chapitre. Dans la section 3.2, nous présentons l'octogone convexe avec des côtés de longueur unitaire et un diamètre minimum dont la preuve d'optimalité repose sur des aspects géométriques ainsi que sur la résolution d'un programme quadratique à contraintes quadratiques. Finalement, la section 3.3 est consacrée à la résolution d'un problème de classification automatique des données, i.e., le problème de recherche d'un hyperplan séparateur en norme-ℓ_2, qui se formule comme un cas particulier de QQP.

3.1 Description de l'algorithme

Tout problème QQP à variables bornées peut être résolu, entre autres, par l'algorithme d'*énumération implicite* avec ajout de coupes (*branch and cut*) présenté dans [6, 9], qui procure en un temps fini une solution optimale globale à l'intérieur d'un certain niveau de tolérance (tant au niveau de l'optimalité que de la faisabilité). On présente, dans cette section, une description de la version de base de cet

algorithme où le texte et les figures sont tirés en partie de [6]. Ensuite, on décrit les améliorations informatiques et algorithmiques qui ont été apportées par rapport à l'implémentation utilisée dans [6, 7, 8, 9, 10].

3.1.1 Description de la version de base de l'algorithme

Nous donnons, dans cette section, une description abrégée de l'algorithme d'énumération implicite avec ajout de coupes décrit plus amplement dans [6, 9]. L'idée principale de cet algorithme consiste à estimer tous les termes quadratiques par des linéarisations successives, ou approximations extérieures, à l'intérieur d'un arbre d'énumération en utilisant les techniques de reformulation-linéarisation (voir, par exemple, [9, 82, 83] pour plus de détails sur le sujet). L'idée essentielle de ces techniques est de remplacer chaque terme quadratique x_i^2 apparaissant dans le programme quadratique par un terme linéaire v_i et chaque terme bilinéaire $x_i x_j$ par un terme linéaire w_{ij}. Le problème d'optimisation ainsi obtenu, noté $[QQP]_\ell$ pour la suite, est en fait la relaxation linéaire du problème QQP. Ensuite, les termes linéaires sont contraints par des linéarisations de façon à ce que v_i soit une approximation de x_i^2 et w_{ij} une approximation $x_i x_j$. En fait, tout au long du déroulement de l'algorithme, on ajoute des linéarisations de manière à raffiner les approximations linéaires des termes quadratiques tout en garantissant que les solutions réalisables pour QQP demeurent réalisables dans $[QQP]_\ell$.

Classes de linéarisations

L'algorithme repose sur quatre classes de linéarisations, notées par $[\cdot]_\ell$. Les linéarisations des quatre classes engendrent une approximation extérieure du domaine réalisable. Elles sont en fait des coupes, i.e., des inégalités valides qui permettent d'éliminer la solution optimale $(\hat{x}, \hat{v}, \hat{w})$ d'un problème relaxé pour laquelle $x_i^2 \neq v_i$ ou $\hat{x}_i \hat{x}_j \neq \hat{w}_{ij}$.

Les deux premières classes d'inégalités sont des sous-estimations tangentes à des fonctions convexes. Elles sont donc valides partout sur le domaine réalisable. Au contraire, les inégalités des deux autres classes ne sont pas nécessairement valides partout. Pour remédier à ce problème, une dichotomie est utilisée de manière à raffiner les approximations des termes quadratiques dans des sous-intervalles. Des variables binaires sont alors ajoutées au problème relaxé dont le rôle est de relâcher ou non ces contraintes. En fait, lors d'un branchement, on fixe une de ces variables à 0 ou 1 selon l'intervalle considéré. Grâce à ce procédé, toutes les linéarisations créées en un noeud sont valides partout dans l'arbre et pas seulement dans le sous-arbre enraciné en ce noeud.

La première classe de linéarisation, due à Al-Khayyal et Falk [1], est une sous-estimation de la fonction carré. Pour une valeur de $\alpha_i \in [\ell_i, u_i], i \in N$ l'inégalité

$$0 \;\leq\; [(x_i - \alpha_i)^2]_\ell \;=\; v_i - 2\alpha_i x_i + \alpha_i^2$$

définit le demi-espace tangent à la fonction convexe x_i^2 au point $x_i = \alpha_i$ (voir la figure 3.1 pour une illustration). Lorsque la solution optimale $(\hat{x}, \hat{v}, \hat{w})$ du problème $[QQP]_\ell$ courant est telle que $\hat{v}_i < \hat{x}_i^2$, on doit choisir $\alpha_i \in]\hat{x}_i - \sqrt{\hat{x}_i^2 - \hat{v}_i}, \hat{x}_i + \sqrt{\hat{x}_i^2 - \hat{v}_i}[$ de manière à éliminer cette solution non-réalisable pour QQP. Parmi toutes les valeurs de α_i dans cet intervalle, on choisit celle qui minimise un certain type d'erreur.

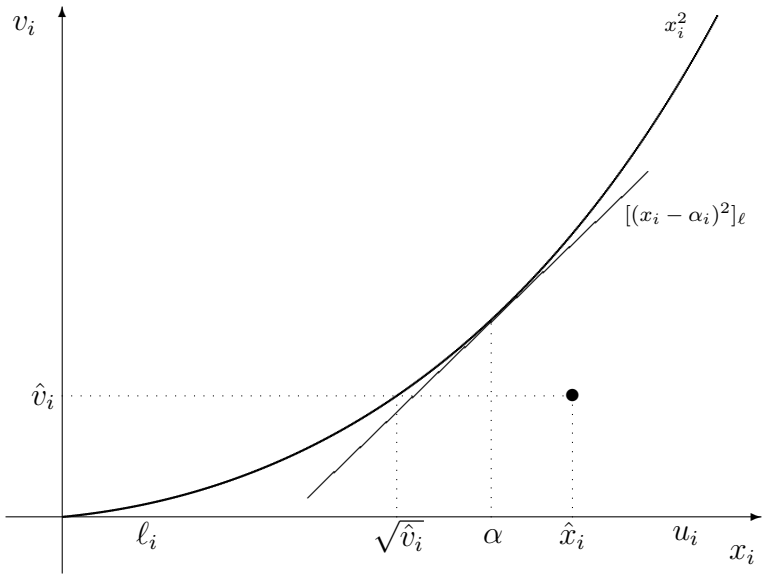

Figure 3.1 – Sous-estimation de la fonction $f(x_i) = x_i^2$ (figure 5.1 de [6])

La seconde classe de linéarisations généralise la première. La linéarisation est obtenue par une sous-estimation linéaire d'une paraboloïde. L'idée consiste à ajouter une inégalité valide qui implique deux variables x_i et x_j, les approximations v_i et v_j de leur carré ainsi que l'approximation w_{ij} de leur produit. Pour des valeurs de $\alpha_i \in [\ell_i, u_i], \alpha_j \in [\ell_j, u_j], i, j \in N, \gamma \in \mathbb{R}$ l'inégalité

$$\begin{aligned} 0 \leq [((\alpha_i - x_i) + \gamma(\alpha_j - x_j))^2]_\ell \quad &= \quad (v_j - 2\alpha_j x_j + \alpha_j^2)\gamma^2 + \\ & \quad 2(w_{ij} - \alpha_i x_j - \alpha_j x_i + \alpha_i \alpha_j)\gamma + \\ & \quad (v_i - 2\alpha_i x_i - \alpha_i^2) \end{aligned}$$

définit le demi-espace tangent à la fonction convexe $(x_i + \gamma x^j)^2$ au point $(x_i, x_j) = (\alpha_i, \alpha_j)$ (voir la figure 3.2 pour une illustration). Lorsque la solution optimale du problème $[QQP]_\ell$ courant est telle que $\hat{x}_i \hat{x}_j \neq \hat{w}_{ij}$, les valeurs des paramètres α_i, α_j et γ sont choisies de manière à minimiser, encore une fois, une certain type d'erreur.

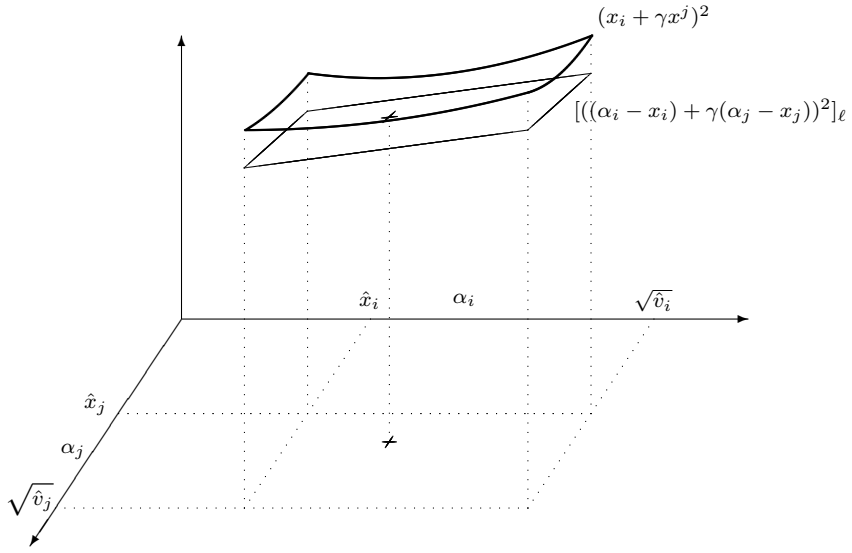

Figure 3.2 – Sous-estimation d'une paraboloïde (variante de la figure 5.4 de [6])

La troisième classe de linéarisations est une surestimation de la fonction carré. Pour une valeur de $x_i \in [\alpha_i, \beta_i], i \in N$, considérons l'inégalité

$$0 \;\leq\; [(x_i - \alpha_i)(\beta_i - x_i)]_\ell \;=\; -\alpha_i\beta_i + (\alpha_i + \beta_i)x_i - v_i.$$

Pour des valeurs données de α_i et β_i, l'inégalité définit un demi-espace obtenu à partir de la corde reliant les points $(x_i, v_i) = (\alpha_i, \alpha_i^2)$ et (β_i, β_i^2). Cette contrainte est valide seulement lorsque $x_i \in [\alpha_i, \beta_i]$. Ainsi, elle est valide partout uniquement dans le cas particulier où $\alpha_i = \ell_i$ et $\beta_i = u_i$. Pour formuler cette contrainte de manière à pouvoir la laisser de façon permanente dans $[QQP]_\ell$, on ajoute une variable binaire qui permet de relâcher la contrainte si on se trouve à l'extérieur de l'intervalle $[\alpha_i, \beta_i]$ (voir [6, 9] pour plus de détails). Lorsque la solution optimale $(\hat{x}, \hat{v}, \hat{w})$ du problème $[QQP]_\ell$ courant est telle que $\hat{v}_i > \hat{x}_i^2$, il suffit d'utiliser deux contraintes de ce type pour éliminer le point courant, i.e., utiliser deux cordes reliées en un point qui se situe à l'intérieur de l'intervalle $[\hat{x}_i, \sqrt{\hat{v}_i}]$. On peut aussi, comme pour les autres classes, choisir un point permettant de minimiser un certain niveau d'erreur.

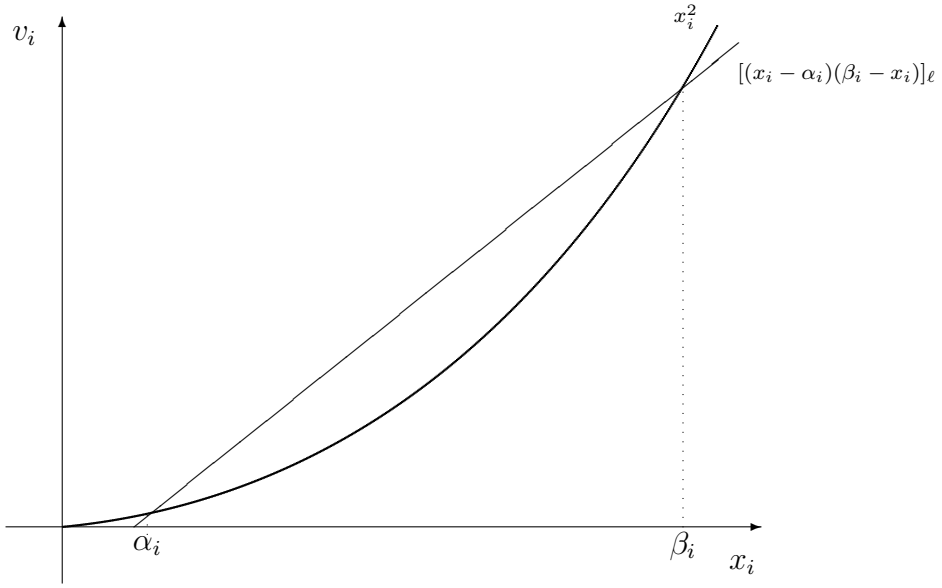

Figure 3.3 – Surestimation de la fonction $f(x_i) = x_i^2$ aux points α_i et β_i

La quatrième classe de linéarisations introduit des coupes pour l'estimation des produits de variables. Puisque que tout point de la fonction $g(x_i, x_j) = x_i x_j$ est un point de selle, le plan tangent Π en un point, donne une sous-estimation ou une sur-estimation de cette fonction selon le quadrant considéré. Formellement, considérons l'approximation tangente de la fonction g autour du point (α_i, α_j), le plan tangent Π satisfait les trois propriétés suivantes :

(i) (x_i, x_j, w_{ij}) appartient à Π si et seulement si $x_i = \alpha_i$ ou $x_j = \alpha_j$.
(ii) Π sous-estime strictement (x_i, x_j, w_{ij}) si et seulement si $x_i < \alpha_i$ et $x_j < \alpha_j$, ou bien si $x_i > \alpha_i$ et $x_j > \alpha_j$.
(iii) Π surestime strictement (x_i, x_j, w_{ij}) si et seulement si $x_i < \alpha_i$ et $x_j > \alpha_j$, ou bien si $x_i > \alpha_i$ et $x_j < \alpha_j$.

Définissons les quatre quadrants associés au point (α_i, α_j) :

$$\begin{array}{rclrcl} \Omega^I & = & \{(x_i, x_j) : x_i \geq \alpha_i, x_j \geq \alpha_j\}, & \Omega^{II} & = & \{(x_i, x_j) : x_i \leq \alpha_i, x_j \geq \alpha_j\}, \\ \Omega^{III} & = & \{(x_i, x_j) : x_i \leq \alpha_i, x_j \leq \alpha_j\}, & \Omega^{IV} & = & \{(x_i, x_j) : x_i \geq \alpha_i, x_j \leq \alpha_j\}. \end{array}$$

Nous pouvons aisément obtenir des inégalités valides pour chaque quadrant. Par exemple, pour les quadrants Ω^I et Ω^{III}, on considère l'inégalité suivante :

$$0 \leq [(x_i - \alpha_i)(x_j - \alpha_j)]_\ell = w_{ij} - \alpha_i x_j - \alpha_j x_i + \alpha_i \alpha_j.$$

Cette inégalité est valide seulement dans ces deux quadrants. En utilisant un raisonnement similaire, on peut facilement obtenir une inégalité valide pour les quadrants Ω^{II} et Ω^{IV}. Comme pour la classe précédente, on utilise des variables binaires pour formuler ces inégalités de façon à les rendre valides sur tout le domaine et on choisit le point (α_i, α_j) de façon à minimiser un certain type d'erreur.

Déroulement de l'algorithme

L'algorithme se divise en deux étapes principales : le *pré-traitement* et l'*exploration de l'arbre d'énumération*. La première étape permet d'obtenir la première approximation extérieure. La deuxième étape traite les noeuds de l'arbre récursivement en utilisant une stratégie de type *meilleur d'abord* où meilleur désigne le noeud dont la valeur optimale de la relaxation linéaire est la moins (ou la plus) élevée pour un problème de minimisation (ou de maximisation).

L'étape de pré-traitement est exécutée au noeud racine de l'arbre d'énumération. Notons qu'au tout début, on ajoute aux contraintes de $[QQP]_\ell$ les linéarisations associées aux bornes initiales connues sur chaque variable. Ensuite, on essaie de raffiner le plus possible la borne inférieure l_i et la borne supérieure u_i de chaque variable x_i apparaissant dans au moins un terme quadratique. Le raffinement de bornes consiste à simplement résoudre, pour chacune de ces variables x_i, les problèmes linéaires obtenus en remplaçant la fonction-objectif de $[QQP]_\ell$ par les quatre fonctions-objectif $\pm x_i$ et $\pm v_i$. Si ces évaluations permettent d'améliorer les bornes sur x_i, on les change et on ajoute les linéarisations correspondantes dans $[QQP]_\ell$. On répète ce procédé jusqu'à ce qu'on ne puisse plus améliorer suffisamment les bornes. Après la phase de pré-traitement, on ajoute à la relaxation linéaire $[QQP]_\ell$, les contraintes associées aux quatre classes de linéarisations qui sont valides partout en fonction des bornes trouvées dans le raffinement. L'étape de pré-traitement, lorsqu'elle améliore les bornes, permet d'obtenir une meilleure approximation extérieure et ainsi réduire considérablement le nombre de noeuds explorés lors de la seconde étape.

Durant l'étape d'exploration de l'arbre, l'algorithme branche récursivement sur les variables apparaissant dans les termes quadratiques. À chaque noeud de l'arbre d'énumération, l'algorithme commence par essayer d'éliminer certaines erreurs d'approximation en ajoutant des contraintes associées aux classes 1, 2 et 4. Il identifie ensuite, s'il y a encore des erreurs, la variable x_i pour laquelle l'erreur pondérée $\max\{\eta_i|x_i^2 - v_i|, \eta_{ij}|x_i x_j - w_{ij}| : i \neq j\}$ est la plus grande, où η_i et η_{ij} sont des pondérations non-négatives qui augmentent en fonction de la fréquence d'apparition de la variable x_i dans les termes quadratiques. Le processus de branchement crée alors deux sous-problèmes : un premier dans lequel $x_i \leq \alpha_i$ et un autre dans lequel $x_i \geq \alpha_i$ pour une valeur α_i choisie minutieusement. On ajoute alors, pour éliminer l'erreur de surestimation, des inégalités de la classe 3 (ou de la classe 4 s'il n'y a pas d'erreur de surestimation de x_i^2).

L'algorithme arrête lorsqu'une solution optimale (à l'intérieur d'un certain niveau de tolérance tant au niveau de l'optimalité que de la faisabilité) est trouvée. Une solution est dite réalisable à l'intérieur d'une tolérance de faisabilité $\epsilon_r > 0$ donnée si l'erreur d'approximation de chaque terme quadratique est inférieure à ϵ_r, i.e., si $|x_i^2 - v_i| < \epsilon_r$ et $|x_i x_j - w_{ij}| < \epsilon_r$ pour tout i et j. Une solution ϵ_r-réalisable est dite optimale à l'intérieur d'une tolérance d'optimalité $\epsilon_z > 0$ donnée, s'il n'y a pas de possibilité de trouver une autre solution ϵ_r-réalisable améliorant la valeur de l'objectif de plus de ϵ_z, i.e., si la différence entre la valeur optimale de la relaxation et la valeur de la solution trouvée est inférieure à ϵ_z.

3.1.2 Améliorations

Les résultats présentés dans ce chapitre ont été obtenus en utilisant une implémentation améliorée de l'algorithme décrit à la section 3.1. Cette nouvelle implémentation est écrite en C++ et utilise CPLEX 8.1 pour résoudre les problèmes de programmation linéaire. Les expériences de calcul sont faites sur un ordinateur Intel Xeon 3.06

GHz avec 1 Mb de mémoire cache et 2 GO de mémoire RAM. Les améliorations, en rapport à l'implémentation de [6, 9], se situent à deux niveaux : le code informatique et l'algorithme. Nous présentons ici les deux principales améliorations apportées au code informatique ainsi que la principale amélioration apportée à l'algorithme.

Au niveau du code informatique, notons tout d'abord que l'algorithme a fait l'objet d'une reprogrammation complète. Parmi les améliorations apportées au code, la plus importante concerne la possibilité de résoudre des problèmes de grande taille. Le code de [6, 9] avait été conçu pour l'optimisation quadratique de faible taille comprenant une grande proportion de termes quadratiques. Ce code a été appliqué dans [6, 7, 8, 9, 10] et tous les problèmes possédaient moins de 212 variables. Pour simplifier la gestion des indices des variables dans le code informatique, ce code considérait explicitement tous les $\frac{n(n-1)}{2} + n$ termes quadratiques en prenant bien soin, par la suite, de différencier les termes quadratiques présents dans le problème original QQP de ceux créés artificiellement. Cette façon de faire réduisait considérablement la flexibilité en rendant impossible la résolution de programmes quadratiques contenant un nombre élevé de variables et un nombre restreint de termes quadratiques. La gestion des structures de données et l'indexage des variables ont été repensés dans le but de ne pas avoir à créer inutilement tous les termes quadratiques. Le nouveau code peut donc résoudre autant les problèmes de petite taille avec une grande proportion de termes quadratiques que les problèmes de grande taille avec une faible proportion de termes quadratiques. Notons que cette amélioration était nécessaire pour la résolution de certains problèmes réels (ou réalistes), par exemple, les instances du problème étudié à la section 3.3.

Une deuxième amélioration importante apportée au code concerne la gestion des paramètres. Le programme a été recodé de manière à offrir à l'utilisateur une plus grande flexibilité sur la fixation de certains paramètres algorithmiques. En fait, l'utilisateur peut maintenant facilement contrôler plusieurs paramètres de l'algorithme. Ainsi, il peut utiliser un choix de paramètres adapté au problème qu'il veut résoudre.

La principale amélioration apportée à l'algorithme consiste à permettre d'effectuer un raffinement de bornes sur les variables x_i à différentes profondeurs de l'arbre d'énumération. Un procédé similaire à celui appliqué lors de l'étape de pré-traitement peut maintenant être effectué à différentes profondeurs de l'arbre. Le choix des profondeurs est déterminé par l'utilisateur via la fixation des paramètres algorithmiques concernés. De cette manière, à un noeud donné de l'arbre, les bornes sur les variables peuvent être raffinées en considérant les contraintes de séparation ainsi que les linéarisations ajoutées lors du traitement des noeuds précédents. Ensuite, des linéarisations associées à ces bornes sont ajoutées pour obtenir une meilleure approximation extérieure du domaine de QQP pour ce noeud et pour tous les noeuds du sous-arbre enraciné en ce noeud. Cependant, certaines de ces contraintes ne sont pas nécessairement valides à l'extérieur de ce sous-arbre. Il a donc fallu programmer adéquatement la gestion de ces contraintes de manière à considérer, à chaque moment, que les inégalités valides.

Grâce à ces raffinements de bornes répétés, l'approximation extérieure du domaine s'améliore plus rapidement et l'algorithme trouve des solutions réalisables et la solution optimale en traitant, généralement, un moins grand nombre de noeuds. Cependant, cette réduction du nombre de noeuds n'engendre pas nécessairement une réduction du temps global de résolution car le temps requis pour effectuer les multiples raffinements de bornes peut être supérieur à la diminution de temps générée par la réduction du nombre de noeuds explorés. Illustrons ce phénomène par la résolution de sept exemples de modèles quadratiques présentés dans le chapitre 5 de [6].

Pour ne pas allonger le texte, nous ne présentons pas l'ensemble des modèles de ces exemples mais nous nous référons à ces problèmes en utilisant la numérotation utilisée dans [6]. Nous présentons néanmoins les modèles des exemples 5.25, 5.26 et 5.27 qui contenaient quelques erreurs typographiques. Nous ne présentons pas non plus les solutions de ces modèles car nous avons trouvé des solutions presqu'identiques à celles données dans [6].

Pour le modèle de l'exemple 5.25, nous avons corrigé des erreurs au niveau du sens de certaines inégalités, ajouté une contrainte manquante et remplacé chaque contrainte quadratique d'égalité par deux contraintes d'inégalité en donnant un petit intervalle admissible. Le retrait des contraintes d'égalité simplifie légèrement la résolution par l'algorithme d'énumération et la formulation résultante est valide car on se rapproche alors de la formulation initiale du problème donné dans [29]. Les contraintes d'égalité avaient été introduites lors de la reformulation de ce problème par des techniques de transformations provenant de [50]. Suite à ces modifications, le modèle de l'exemple 5.25 devient :

$$
\begin{aligned}
\min_{x} \quad & z = 12.62626(x_8 + x_9 + x_{10}) - 1.231059(x_1x_8 + x_2x_9 + x_3x_{10}) \\
\text{s.c.} \quad & \\
& 12.62626x_8 - 1.231059x_1x_8 \leq 150 \\
& 12.62626x_9 - 1.231059x_2x_9 \leq 150 \\
& 12.62626x_{10} - 1.231059x_3x_{10} \leq 150 \\
& -3.475x_1 + 100x_4 + 0.0975x_1^2 - 9.75x_1x_4 \geq 0 \\
& -3.475x_1 + 100x_4 + 0.0975x_1^2 - 9.75x_1x_4 \leq 0.05 \\
& -3.475x_2 + 100x_5 + 0.0975x_2^2 - 9.75x_2x_5 \geq 0 \\
& -3.475x_3 + 100x_6 + 0.0975x_3^2 - 9.75x_3x_6 \geq 0 \\
& -x_4x_7 + x_5x_7 - x_1x_8 + x_4x_8 \geq 0 \\
& 50x_5 + 50x_6 - x_1x_8 + x_5x_8 + x_2x_9 - x_6x_9 \geq 499.95 \\
& 50x_5 + 50x_6 - x_1x_8 + x_5x_8 + x_2x_9 - x_6x_9 \leq 500 \\
& -50x_2 + 50x_6 + x_2x_9 - x_6x_9 + x_2x_{10} - x_3x_{10} \geq 0 \\
& 50x_2 - x_2x_{10} + x_3x_{10} \geq 450 \\
& 0 \leq x_7 - x_8 \leq 50 \\
& x_5 \leq x_1 \leq x_2 \\
& x_6 \leq x_2 \leq x_3 \\
& 1 \leq x_1 \leq 8.03773157 \\
& 4.5 \leq x_2 \leq 9.71853961 \\
& 9 \leq x_3 \leq 10 \\
& 0.001 \leq x_4 \leq 10 \\
& 1 \leq x_5 \leq 4.68381588 \\
& 1 \leq x_6 \leq 9 \\
& 0.01 \leq x_7 \leq 100 \\
& 0.01 \leq x_8 \leq 54.91813487 \\
& 50 \leq x_9 \leq 100 \\
& 0.01 \leq x_{10} \leq 50 \\
& 50 \leq z \leq 250.
\end{aligned}
$$

Le modèle de l'exemple 5.26, suite à la correction de quelques erreurs au niveau du sens de certaines inégalités, devient :

$$
\begin{aligned}
\min_{x} \quad & z = 12.62626(x_{12}+x_{13}+x_{14}+x_{15}+x_{16}) \\
& \quad -1.231059(x_1x_{12}+x_2x_{13}+x_3x_{14}+x_4x_{15}+x_5x_{16}) \\
\text{s.c.} \quad & \\
& -3.475x_1+100x_6+0.0975x_1^2-9.75x_1x_6 \geq 0 \\
& -3.475x_2+100x_7+0.0975x_2^2-9.75x_2x_7 \geq 0 \\
& -3.475x_3+100x_8+0.0975x_3^2-9.75x_3x_8 \geq 0 \\
& -3.475x_4+100x_9+0.0975x_4^2-9.75x_4x_9 \geq 0 \\
& -3.475x_5+100x_{10}+0.0975x_5^2-9.75x_5x_{10} \geq 0 \\
& -x_6x_{11}+x_7x_{11}-x_1x_{12}+x_6x_{12} \geq 0 \\
& 50x_7-50x_8-x_1x_{12}+x_2x_{13}+x_7x_{12}-x_8x_{13}=0 \\
& 50x_8+50x_9-x_2x_{13}+x_3x_{14}+x_8x_{13}-x_9x_{14} \leq 500 \\
& -50x_9+50x_{10}-x_3x_{14}+x_4x_{15}-x_8x_{15}+x_9x_{14} \leq 0 \\
& 50x_4-50x_{10}-x_4x_{15}-x_4x_{16}+x_5x_{16}+x_{10}x_{15} \leq 0 \\
& 50x_4-x_4x_{16}+x_5x_{16} \geq 450 \\
& -x_1+2x_7 \leq 1 \\
& x_1 \leq x_2 \leq x_3 \leq x_4 \leq x_5 \\
& x_6 \leq x_7 \\
& x_8 \leq x_9 \leq x_{10} \leq x_4 \\
& 0 \leq x_{11}-x_{12} \leq 50 \\
& 1 \leq x_1 \leq 8.03773157 \\
& 1 \leq x_2 \leq 9 \\
& 4.5 \leq x_i \leq 9 \qquad i=3,4 \\
& 9 \leq x_5 \leq 10 \\
& 0.001 \leq x_6 \leq 1 \\
& 1 \leq x_7 \leq 4.51886579 \\
& 1 \leq x_i \leq 9 \qquad i=8,9,10 \\
& 0.1 \leq x_{11} \leq 100 \\
& 10^{-7} \leq x_{12} \leq 100 \\
& 1 \leq x_{13} \leq 50 \\
& 50 \leq x_i \leq 100 \qquad i=14,15 \\
& 10^{-7} \leq x_{16} \leq 50 \\
& 50 \leq z \leq 250.
\end{aligned}
$$

Pour l'exemple 5.27, après la correction du sens de l'objectif et des erreurs de signe ainsi que l'ajout d'une borne inférieure sur z, le modèle est le suivant :

$$\begin{aligned}
\max_{x,y} \quad & z = \tfrac{1}{2}\{(x_2 + x_3 - 4x_1)y_1 + (3x_1 - 2x_3 + x_5)y_2 + (3x_1 - 2x_2 + x_4)y_3 \\
& \quad +(x_3 - 2x_1)y_4 + (x_2 - 2x_1)y_5\} + x_1 \\
\text{s.c.} \quad & \\
& (x_1 - x_2)^2 + (y_1 - y_2)^2 \leq 1 \\
& (-x_1 + x_3 - x_5)^2 + (y_1 - y_3 + y_5)^2 \leq 1 \\
& (x_1 - x_2 + x_4)^2 + (y_1 - y_2 + y_4)^2 \leq 1 \\
& (-x_1 + x_3)^2 + (y_1 - y_3)^2 \leq 1 \\
& (2x_1 - x_2 - x_3 + x_5)^2 + (y_2 - y_3 + y_5)^2 \leq 1 \\
& (2x_1 - x_2)^2 + y_2^2 \leq 1 \\
& (x_1 - x_2)^2 + (y_1 - y_2 - 1)^2 \leq 1 \\
& (2x_1 - x_2 - x_3)^2 + (-y_2 + y_3)^2 \leq 1 \\
& (x_3 - x_5)^2 + (-y_3 + y_5)^2 \leq 1 \\
& (-x_1 + x_3 - x_5)^2 + (y_1 - y_3 + y_5 - 1)^2 \leq 1 \\
& (2x_1 - x_3 + x_5)^2 + (-y_3 + y_5)^2 \leq 1 \\
& (2x_1 - x_2 - x_3 + x_4 + x_5)^2 + (-y_2 + y_3 + y_4 - y_5)^2 = 1 \\
& (-2x_1 + x_2 - x_4)^2 + (y_2 - y_4)^2 \leq 1 \\
& (x_1 - x_2 + x_4)^2 + (y_1 - y_2 + y_4 - 1)^2 \leq 1 \\
& (x_1 - x_3)^2 + (1 - y_1 + y_3)^2 \leq 1 \\
& (x_2 - x_4)^2 + (y_2 - y_4)^2 \leq 1 \\
& (2x_1 - x_3)^2 + y_3^2 \leq 1 \\
& (2x_1 - x_2 - x_3 + x_4)^2 + (-y_2 + y_3 + y_4)^2 \leq 1 \\
& x_2 - x_3 \geq 0 \\
& x_i^2 + y_i^2 = 1 \quad i = 1, 2, 3, 4, 5 \\
& 0 \leq x_1 \leq 0.5 \\
& 0 \leq x_i \leq 1 \quad i = 2, 3, 4, 5 \\
& y_i \geq 0 \quad i = 1, 2, 3, 4, 5 \\
& z \geq 0.72531999.
\end{aligned}$$

Nous donnons, dans le tableau 3.1, les principales caractéristiques concernant la taille des programmes quadratiques résolus dans cette section. Les trois premières colonnes contiennent respectivement le nombre de variables qui ne sont pas impliquées dans des termes quadratiques, le nombre de variables qui le sont, et le nombre total de variables. La colonne centrale indique le nombre de termes quadratiques. Les trois dernières contiennent respectivement le nombre de contraintes d'inégalités linéaires (incluant les bornes sur les variables), et le nombre d'inégalités et d'égalités comprenant des termes quadratiques.

Tableau 3.1 – Taille des programmes quadratiques des exemples de [6]

Ex	Variables			Termes	Contraintes		
	Lin	Quad	Tot	Quad	Lin	Quad	
					$\leq$	$\leq$	$=$
5.20	2	3	5	2	8	2	0
5.21	0	4	4	6	8	3	0
5.22	0	6	6	9	8	1	3
5.23	0	4	4	2	9	2	0
5.24	0	7	7	6	14	5	2
5.25	0	10	10	15	26	14	0
5.26	0	16	16	24	43	12	1
5.27	0	10	10	43	11	18	6

Pour chacun de ces exemples, l'algorithme a été exécuté à deux reprises. En un premier temps, nous avons résolu, avec une précision $\epsilon_r = \epsilon_z = \epsilon$, le problème en effectuant, comme dans la version de base de l'algorithme, uniquement le raffinement de bornes simple du pré-traitement. En un deuxième temps, nous avons résolu les mêmes problèmes, avec la même précision, en effectuant des raffinements de bornes supplémentaires aux noeuds de profondeurs 4, 8 et 12. Le tableau 3.2 présente ces résultats où les colonnes *Noeuds* indiquent le nombre total de noeuds de l'arbre d'énumération, les colonnes *Temps RBS* et *Temps tot* donnent les temps CPU en secondes pour faire les raffinements de bornes supplémentaires *RBS* et pour faire la résolution complète et finalement les dernières colonnes donnent le gain (en %), sur le nombre de noeuds et le temps total, réalisé par les raffinements de bornes supplémentaires.

On constate que l'utilisation de raffinements de bornes multiples a permis de réduire le nombre de noeuds explorés dans l'arbre dans tous les cas sauf pour les problème 5.20 et 5.21 où le nombre de noeuds est demeuré constant. Au niveau du temps total de calcul, on note qu'il y a un gain dans quatre cas sur sept. En somme, le tableau 3.2 illustre le fait que le gain de temps est fortement relié au gain obtenu dans le nombre de noeuds et que le gain global de temps est intéressant uniquement pour les problèmes difficiles, i.e., pour les problèmes où le nombre de noeuds et le temps, pour la résolution sans raffinement de bornes, sont assez élevés.

Tableau 3.2 – Effet des raffinements de bornes multiples

		Raf. bornes simple		Raf. bornes multiples			Gain	
Ex	ϵ	Temps tot	Noeuds	Temps tot	Temps RBS	Noeuds	Noeuds	temps
5.20	1×10^{-6}	0.02	7	0.03	0	7	0.0%	-50.0%
5.21	1×10^{-6}	0.04	9	0.08	0.04	9	0.0%	-100.0%
5.22	1×10^{-6}	36.96	2211	3.4	2.52	123	94.4%	90.8%
5.23	1×10^{-6}	2.33	643	3.31	1.03	513	20.2%	-42.1%
5.24	1×10^{-6}	16.92	1049	12.38	9.58	273	74.0%	26.8%
5.25	1×10^{-6}	504.06	10417	41.67	25.45	817	92.2%	91.7%
5.26	1×10^{-6}	183.66	1407	84.99	73.65	339	75.9%	53.7%
5.27	5×10^{-4}	86314.54	13947	3906.83	3309.48	1493	89.3%	95.5%

Finalement, notons que l'exemple 5.27 est un problème très difficile contenant un très grand nombre de termes quadratiques, voilà pourquoi il n'a pas été résolu avec la même précision. Il s'agit du programme quadratique associé à un problème de géométrie où l'on cherche l'octogone de diamètre unitaire ayant l'aire maximale. L'octogone optimal a été trouvé dans [10] avec un précision de 1×10^{-5}. La preuve d'optimalité repose sur des arguments géométriques ainsi que sur la programmation quadratique. En utilisant une implémentation de la version de base de l'algorithme décrit à la section 3.1, Audet [6] résout ce programme quadratique en plusieurs étapes où, à chaque fois, il raffine les bornes sur z. En fait, cette technique n'est pas automatisée et nécessite une analyse poussée par l'utilisateur à chaque étape. Il obtient tout de même la solution après avoir exploré un total de 35139 noeuds en six étapes. En effectuant du raffinement de bornes aux profondeurs 4, 8 12, nous avons résolu ce problème, avec la même précision, en explorant seulement 26561 noeuds. Notons que cette résolution s'est faite en une seule étape et en un seul appel au programme.

3.2 Octogone de diamètre minimum

Dans cette section, nous présentons un problème de géométrie qui est résolu en combinant des arguments géométriques et des outils de la programmation quadratique à contraintes quadratiques. En fait, nous répondons à une question de Vincze [85] : trouver l'octogone convexe avec des côtés de longueur unitaire et un diamètre minimum.

Notons tout d'abord qu'un polygone avec n côtés de longueur unitaire est appelé *n-gone* et noté par U_n. Le *diamètre* d'un polygone est la plus grande distance euclidienne entre n'importe quelle paire de ses sommets ou, en d'autres termes, la longueur de la plus grande diagonale (ou segment de droite reliant deux sommets) de ce polygone. Pour un nombre entier n donné, notons par Δ_n le diamètre minimum parmi tous les *n-gones* U_n. Le diamètre d'un polygone U_n représente aussi la diagonale de U_n dont la longueur est égale au diamètre de U_n.

En 1950, Vincze [85] a étudié le problème de trouver ou borner Δ_n pour tout n (voir aussi Reinhardt [81] ainsi que Larman et Tamvakis [67] pour des résultats similaires à ceux obtenus par Vincze ou d'autres problèmes connexes). Il est facile de montrer que pour $n = 3$, 4 et 5, Δ_n est égal à la longueur de la plus grande diagonale du *n-gone* régulier. Cependant, Bateman et Erdős [13] ont observé que cette propriété n'est plus vraie pour $n = 6$, puisque l'hexagone de diamètre minimum possède alternativement des angles de $\frac{\pi}{2}$ et $\frac{5\pi}{6}$. Le résultat principal de Vincze est que Δ_n est borné inférieurement par la moitié du diamètre du *2n-gone* régulier avec des côtés de longueur unitaire. De plus, cette borne est atteinte pour tout n possédant au moins un facteur premier impair. Ceci conduit au théorème suivant.

Théorème 3.1 *(Théorèmes 1 et 2 de [85].)*

$$\Delta_n \geq \frac{1}{2\sin(\frac{\pi}{2n})} \tag{3.1}$$

et est satisfaite à égalité si $n = (2k+1)2^s$ *pour certains entiers non-négatifs* $k \geq 1$ *et* s.

Une borne supérieure évidente s'obtient par le fait que Δ_n ne peut pas excéder le diamètre du *n-gone* régulier, i.e.,

$$\Delta_n \leq \frac{1}{\sin(\frac{\pi}{n})}. \tag{3.2}$$

Ainsi, les seuls cas qui ne sont pas encore résolus sont ceux pour lesquels $n = 2^s$, avec $s \geq 3$. Pour le premier cas de ce type, i.e., $n = 8$, Vincze présente un octogone (pour lequel il donne le crédit à sa femme sans plus d'explications) ayant un diamètre de $2.588\ldots$ qui se trouve entre les deux bornes (3.1) et (3.2) (voir la figure 3.11 pour une représentation de cet octogone). Il propose ensuite une conjecture stipulant que certains *n-gones* irréguliers satisfont strictement l'inégalité (3.2) pour des valeurs plus grandes de s.

Nous montrons que $\Delta_8 = 2.5843054\ldots$. Ce résultat prouve que l'octogone proposé par la femme de Vincze n'est pas optimal. L'octogone optimal, illustré à la figure 3.9, est obtenu en résolvant une équation non-linéaire à une variable. La preuve d'optimalité repose sur un résultat de Vincze, i.e., que tout sommet d'un *n-gone* optimal doit être l'extrémité d'au moins un diamètre. Dans la section 3.2.1, nous utilisons ce résultat ainsi que certaines autres considérations géométriques pour éliminer plusieurs autres configurations de diamètres qui ne sont pas optimales. Ceci conduit à l'observation que la configuration est telle que toutes les quatre paires de sommets opposés sont reliés par des diamètres. Dans la section 3.2.2, nous présentons l'octogone optimal ainsi que sa preuve d'optimalité en formulant le problème de trouver Δ_8 comme un programme quadratique à contraintes quadratiques. Ce programme est résolu en utilisant l'algorithme présenté à la section 3.1. Une précision de sept décimales sur la valeur de Δ_8 est garantie par l'algorithme.

3.2.1 Conditions nécessaires d'optimalité

Nous étudions d'abord quelques propriétés d'un octogone U_8^* optimal, i.e., un octogone avec un diamètre minimal U_8^*. Nous énonçons aussi quelques propriété de l'ensemble D_8 des diamètres de U_8^*. Toutes ces propriétés sont utiles dans l'élaboration des conditions d'optimalité de U_8^*. Notons aussi que, pour la suite de cette section, nous supposons que les sommets de l'octogone $v_1, v_2, v_3, v_4, v_5, v_6, v_7$ et v_8 sont numérotés en sens horaire.

Théorème 3.2 (Théorème 5 de [85].) *Une condition nécessaire pour qu'un polygone U_n ait un diamètre minimum est que chaque sommet ait un autre sommet qui se trouve à une distance égale au diamètre Δ_n.*

En combinant ce résultat avec la borne inférieure $\Delta_8 > 2.56$ obtenue à partir de (3.1), on déduit que le diamètre de U_8^* ne peut pas joindre deux sommets qui sont à une distance de 1 ou 2 (en parcourant les arêtes formant le périmètre de l'octogone). On obtient donc la propriété suivante :

Propriété 3.1 *Tout sommet v_k, $k \in \{1, 2, \ldots, 8\}$, de U_8^* est l'extrémité d'au moins un et d'au plus trois diamètres dont l'autre extrémité correspond à un des sommets v_{k+3}, v_{k+4} ou v_{k+5}, en prenant, pour chaque somme, le modulo de 8.*

Notons que si deux diamètres de D_8 sont disjoints (supposons, sans perte de généralité, que v_1v_4 et v_5v_8 sont disjoints), alors au moins une diagonale du quadrilatère $(v_1v_4v_5v_8)$ a une longueur plus grande que Δ_8, ce qui engendre une contradiction. Ceci nous donne la propriété suivante :

Propriété 3.2 *Il n'existe pas deux diamètres de U_8^* qui sont disjoints ou, en d'autres termes, chaque diamètre doit posséder un point en commun (correspondant ou non à un sommet de l'octogone) avec au moins un autre diamètre.*

Nous donnons maintenant l'énoncé et la preuve d'une proposition qui est utile pour éliminer plusieurs configurations non-optimales de diamètres.

Proposition 3.1 *Quelque soit $k \in \{1, 2, \ldots, 8\}$, D_8 peut contenir au plus deux des trois diagonales $v_k v_{k+3}$, $v_k v_{k+4}$ et $v_{k+2} v_{k+7}$, en prenant, pour chaque somme, le modulo de 8.*

Preuve. Supposons sans perte de généralité que $k = 1$. Représentons $v_1 v_4$, $v_1 v_3$ et $v_3 v_8$ dans le plan cartésien, avec v_1 à l'origine, v_3 en (x_3, y_3), v_4 en (x_4, y_4), v_5 en $(\Delta_8, 0)$ et v_8 en $(x_8, -y_8)$ (voir la figure 3.4 où, comme pour toutes les autres figures de cette section, les lignes pleines représentent $v_i v_j \in D_8$ et les lignes pointillées les paires de sommets $v_i v_j$ telles que $\text{dist}(v_i, v_j) = 1$).

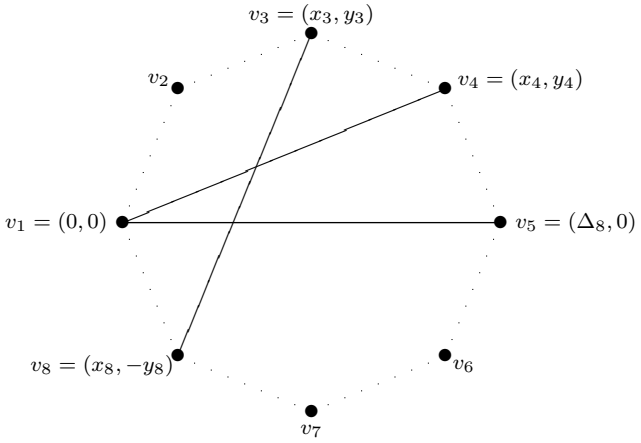

Figure 3.4 – Une configuration de diamètres interdite

Puisque v_4 et v_5 sont à une distance de 1, après quelques calculs élémentaires, nous obtenons que $x_4 = \Delta_8 - \frac{1}{2\Delta_8}$ et $y_4 = \sqrt{1 - \frac{1}{4\Delta_8^2}}$; de plus, comme $2.56 < \Delta_8 < 2.62$ (à partir de (3.1) et (3.2)), on obtient que $x_4 > 2.36$ et $y_4 > 0.98$.

Ainsi, étant donné que $0 \leq x_8 \leq 1 < 2.36 < x_4$ et que la longueur du segment $v_4 v_8$ ne peut pas excéder Δ_8, il s'ensuit que

$$2.62^2 > \Delta_8^2 \geq (x_4 - x_8)^2 + (y_4 + y_8)^2 > (2.36 - x_8)^2 + \left(0.98 + \sqrt{1 - x_8^2}\right)^2 .$$

Notons que si $x_8 \in [0, 0.45]$ alors

$$(2.36 - x_8)^2 + \left(0.98 + \sqrt{1 - x_8^2}\right)^2 \geq (2.36 - 0.45)^2 + \left(0.98 + \sqrt{1 - 0.45^2}\right)^2 > 2.67^2$$

ce qui contredit le fait que $\Delta_8^2 < 2.62^2$. Ainsi, $x_8 > 0.45$ et $y_8 < \sqrt{1 - 0.45^2} < 0.9$.

En combinant les inégalités $x_3^2 + y_3^2 \leq 2^2$, $y_3 \leq \sqrt{4 - x_3^2}$ avec le fait que $0.45 < x_8 \leq 1 < 1.36 < x_3$, tout en prenant comme hypothèse que v_3v_8 est un diamètre, on obtient

$$2.56^2 < \Delta_8^2 = (x_3 - x_8)^2 + (y_3 + y_8)^2 < (x_3 - 0.45)^2 + \left(\sqrt{4 - x_3^2} + 0.9\right)^2.$$

La simplification de cette dernière inégalité nous donne

$$45x_3^2 + 30.822x_3 - 117.611231 < 0$$

ce qui implique que $x_3 < 1.3101$, une contradiction avec le fait que $x_3 > 1.36$. □

La série de lemmes qui suit donne une information croissante sur l'ensemble optimal de diamètres D_8, et conduit vers le théorème 3.3.

Lemme 3.1 *Au moins une des quatre diagonales v_1v_5, v_2v_6, v_3v_7 ou v_4v_8 est un diamètre de l'octogone optimal.*

Preuve. Supposons que cette affirmation soit fausse. Alors, à partir de la propriété 3.1, D_8 doit contenir v_1v_4 ou v_1v_6. Supposons, sans perte de généralité, que $v_1v_4 \in D_8$. Ainsi, la propriété 3.2 implique que $v_5v_8 \notin D_8$ et alors, par la propriété 3.1, il s'ensuit que $v_2v_5 \in D_8$ et $v_3v_8 \in D_8$. Encore une fois, la propriété 3.2 assure que $v_4v_7 \notin D_8$ et $v_1v_6 \notin D_8$. Mais, par la propriété 3.1, il s'ensuit que $v_2v_7 \in D_8$ et $v_3v_6 \in D_8$, ce qui contredit la propriété 3.2 (voir la figure 3.5 pour une illustration des étapes de la preuve où, comme pour les autres figures de cette section, les lignes en trait discontinu représentent $v_iv_j \notin D_8$).

□

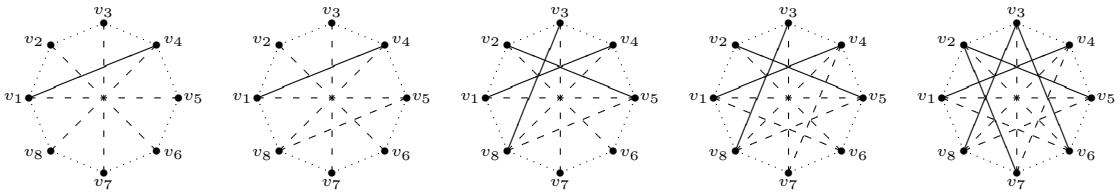

Figure 3.5 – Illustration de la preuve du lemme 3.1

Lemme 3.2 *Au moins deux des quatre diagonales v_1v_5, v_2v_6, v_3v_7 et v_4v_8 sont des diamètres de l'octogone optimal.*

Preuve. À partir du lemme 3.1, nous savons que D_8 contient au moins une des diagonales v_1v_5, v_2v_6, v_3v_7 et v_4v_8. Supposons, sans perte de généralité, que D_8 contient v_1v_5 et aucune autre de ces quatre diagonales. Alors, par la propriété 3.1, nous savons que $v_3v_6 \in D_8$ et $v_3v_8 \in D_8$. Supposons, sans perte de généralité, que $v_3v_8 \in D_8$. Par la propriété 3.2, $v_4v_7 \notin D_8$ mais alors, la propriété 3.1 assure que $v_1v_4 \in D_8$ et $v_2v_7 \in D_8$. Ceci contredit la proposition 3.1 avec $k = 1$ (voir la figure 3.6).

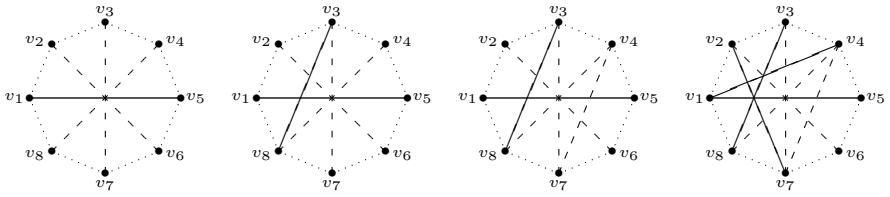

Figure 3.6 – Illustration de la preuve du lemme 3.2

□

Lemme 3.3 *Au moins trois des quatre diagonales v_1v_5, v_2v_6, v_3v_7 et v_4v_8 sont des diamètres de l'octogone optimal.*

Preuve. À partir du lemme 3.2, nous savons que D_8 contient au moins deux des diagonales v_1v_5, v_2v_6, v_3v_7 et v_4v_8. Supposons qu'il en possède seulement deux. Considérons d'abord le cas où les sommets aux extrémités des deux diamètres sont adjacents, i.e.,

sans perte de généralité, $v_1v_5 \in D_8$ et $v_2v_6 \in D_8$. Ainsi, par la propriété 3.1, il faut que $v_1v_4 \in D_8$ ou bien $v_4v_7 \in D_8$. Si $v_1v_4 \in D_8$, par la propriété 3.2, $v_5v_8 \notin D_8$ mais alors la propriété 3.1 assure que $v_3v_8 \in D_8$. Ceci contredit la proposition 3.1 avec $k = 1$ (voir la partie I de la figure 3.7). Si $v_4v_7 \in D_8$, alors la propriété 3.2 assure que $v_3v_8 \notin D_8$. Alors, par la propriété 3.1, $v_5v_8 \in D_8$, ce qui contredit la proposition 3.1 avec $k = 5$ (voir la partie II de la figure 3.7).

Considérons maintenant le cas où les sommets aux extrémités des deux diamètres ne sont pas adjacents, i.e., sans perte de généralité, $v_1v_5 \in D_8$ et $v_3v_7 \in D_8$. Alors, la propriété 3.1 implique que $v_1v_4 \in D_8$ ou bien $v_4v_7 \in D_8$. Supposons, sans perte de généralité, que $v_1v_4 \in D_8$. Par la propriété 3.2, $v_5v_8 \notin D_8$ mais alors la propriété 3.1 assure que $v_3v_8 \in D_8$. Ceci contredit la proposition 3.1 avec $k = 1$ (voir la partie III de la figure 3.7).

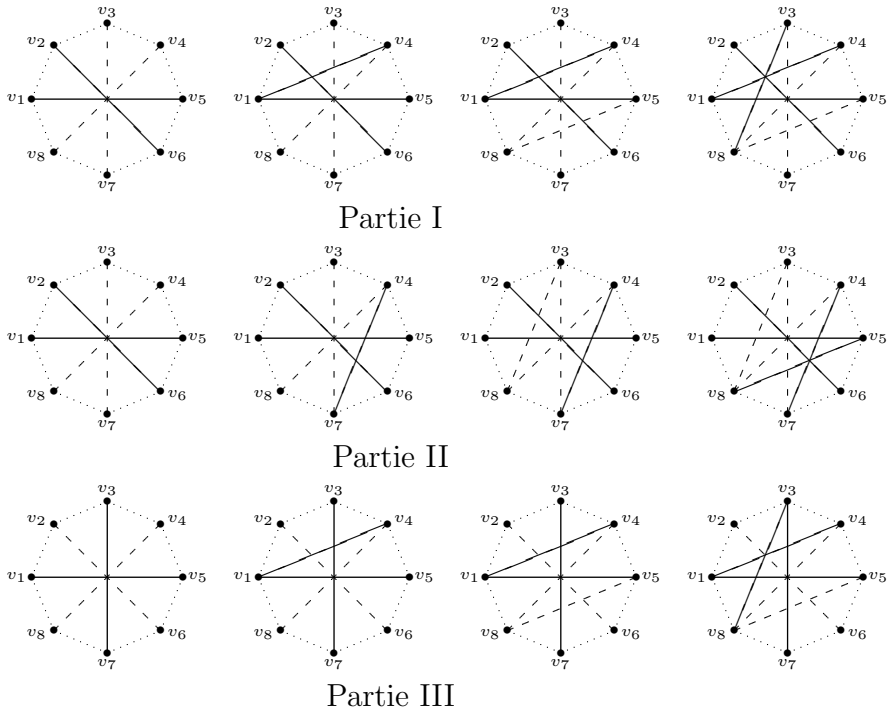

Figure 3.7 – Illustration de la preuve du lemme 3.3

□

Théorème 3.3 *Toutes les quatre diagonales v_1v_5, v_2v_6, v_3v_7 et v_4v_8 sont des diamètres de l'octogone optimal.*

Preuve. D'après le lemme 3.3, D_8 contient au moins trois des diagonales v_1v_5, v_2v_6, v_3v_7 et v_4v_8. Supposons que D_8 possède seulement trois de ces diagonales et supposons, sans perte de généralité, que $v_4v_8 \notin D_8$. Par la propriété 3.1, $v_3v_8 \in D_8$ ou $v_5v_8 \in D_8$. Assumons, sans perte de généralité, que $v_3v_8 \in D_8$. Alors, d'après la propriété 3.2, $v_4v_7 \notin D_8$. Ainsi, la propriété 3.1 assure que $v_1v_4 \in D_8$. Ce qui contredit la proposition 3.1 avec $k = 1$ (voir la figure 3.8).

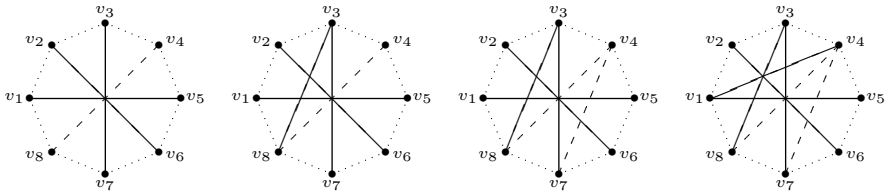

Figure 3.8 – Illustration de la preuve du théorème 3.3

□

3.2.2 Octogone optimal

Cette section contient trois principaux résultats. Premièrement, nous présentons un ensemble de diamètres qui donne un octogone ayant un diamètre inférieur à celui proposé par Vincze. Deuxièmement, nous utilisons l'algorithme de programmation quadratique décrit à la section 3.1 pour démontrer que cet octogone est optimal. Troisièmement, nous prouvons que cette solution est ϵ-unique en utilisant, encore une fois, la programmation quadratique.

Octogone de la femme de Vincze est sous-optimal

La proposition qui suit propose un octogone ayant un diamètre inférieur à celui de [85].

Proposition 3.2 *Il existe un seul octogone avec* $D_8 = \{v_1v_5, v_2v_6, v_3v_7, v_4v_8, v_1v_4, v_1v_6\}$*, et son diamètre est* $d = 2.5843054402\ldots$.

Preuve. Un octogone dont l'ensemble de diamètres correspond à celui de l'énoncé de la proposition est dessiné à la figure 3.9. Notons qu'étant donné que les côtés de l'octogone ont une longueur d'une unité, v_2 et v_8 ainsi que v_4 et v_6 sont symétriques autour de l'axe des abscisses et, par conséquent, le diamètre v_3v_7 est verticale. De plus, les coordonnées de tous les sommets dépendent uniquement du diamètre d. Ce qui permet de trouver assez facilement la solution.

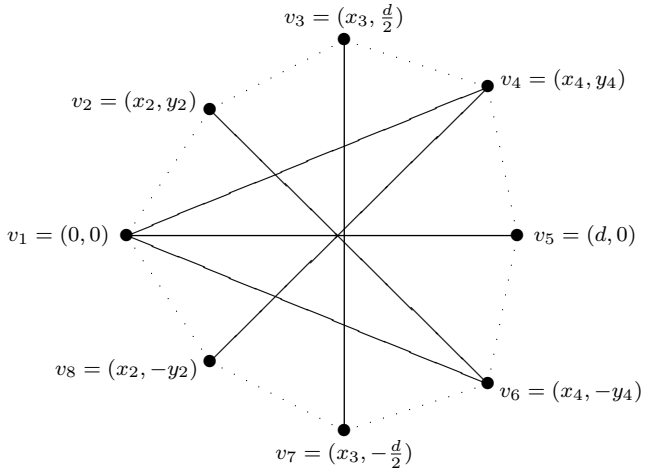

Figure 3.9 – $D_8 = \{v_1v_5, v_2v_6, v_3v_7, v_4v_8, v_1v_4, v_1v_6\}$ et les conséquences sur les coordonnées

Tel que mentionné dans la preuve de la proposition 3.1, $v_1v_4 \in D_8$ et $v_1v_5 \in D_8$ impliquent

$$x_4 = d - \frac{1}{2d} \quad \text{et} \quad y_4 = \sqrt{1 - \frac{1}{4d^2}} \tag{3.3}$$

En combinant (3.3) et $(x_3 - x_4)^2 + \left(\frac{d}{2} - y_4\right)^2 = 1$, on obtient

$$x_3 = d - \frac{1}{2d} - \sqrt{1 - \left(\frac{d}{2} - \sqrt{1 - \frac{1}{4d^2}}\right)^2}. \tag{3.4}$$

La combinaison de $(x_4 - x_2)^2 + (y_4 + y_2)^2 = d^2$ avec $x_2 = \sqrt{1 - y_2^2}$ implique

$$d^2 y_2^2 + y_4 y_2 + (\frac{1}{4} - x_4^2) = 0.$$

En résolvant par rapport à y_2 et en substituant x_4 et y_4 par (3.3), nous trouvons, après quelques simplifications,

$$y_2 = \frac{(d^2 - 1)\sqrt{4d^2 - 1}}{2d^3} \tag{3.5}$$

ainsi que

$$x_2 = \frac{1}{2d^3}\sqrt{9d^4 - 6d^2 + 1}. \tag{3.6}$$

Ainsi, comme l'arête v_2v_3 a une longueur d'une unité, nous avons

$$(x_3 - x_2)^2 + (\frac{d}{2} - y_2)^2 = 1$$

qui, après les substitutions de x_2, y_2 et x_3 par (3.7) (3.5) et (3.4), correspond à une équation non-linéaire en fonction de la variable d seulement. Pour résoudre cette équation, il suffit donc de trouver les zéros de la fonction implicite suivante :

$$f(d) = (x_3 - x_2)^2 + (\frac{d}{2} - y_2)^2 - 1. \tag{3.7}$$

La figure 3.10 illustre la fonction $f(d)$ dans l'intervalle $2.56 \leq d \leq 2.59$. L'obtention des zéros de $f(d)$, dans l'intervalle $2.56 \leq d \leq 2.59$, a été faite numériquement par une méthode de bisection en utilisant MATLAB®. Cette résolution numérique donne un unique zéro en

$$d = 2.5843054402\ldots$$

□

L'octogone de la proposition 3.2, illustré à la figure 3.9, améliore la borne supérieure $\overline{\Delta}_8$ de $2.588\ldots$ proposée par Vincze, dont la configuration de diamètres est

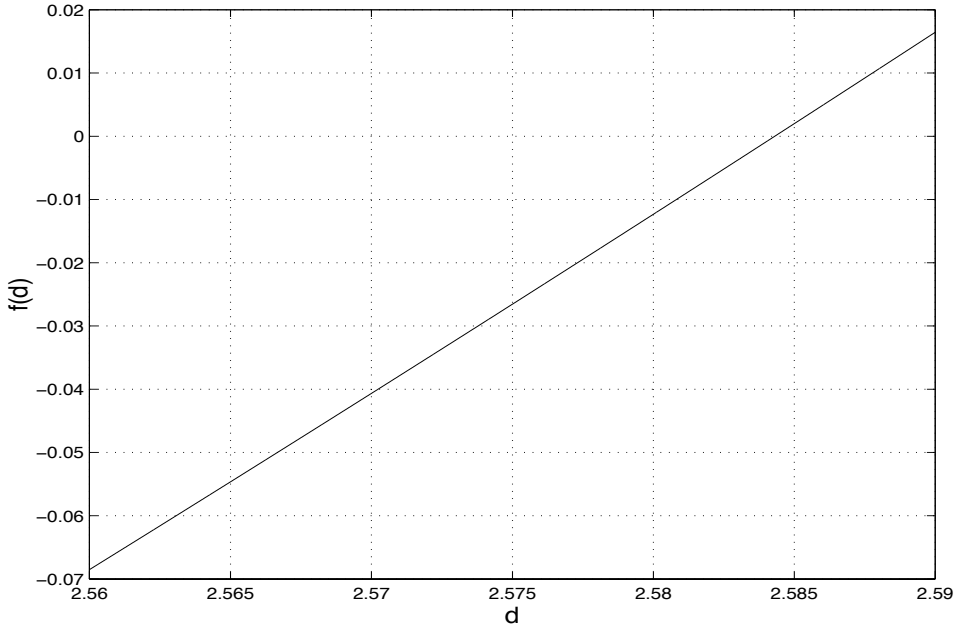

Figure 3.10 – Graphique de la fonction $f(d)$ dans l'intervalle $2.56 \leq d \leq 2.59$

représentée à la figure 3.11. Ce résultat démontre que l'octogone proposé par Vincze est sous-optimal. En observant les deux figures, il est possible de noter que les deux solutions satisfont la condition nécessaire d'optimalité du théorème 3.3 et aussi que la configuration de diamètres de chaque octogone est similaire : l'ensemble D_8 de l'octogone de Vincze illustré à la figure 3.11 est strictement inclus dans celui de la figure 3.9, mais notons que Vincze avait supposé un axe de symétrie différent.

Preuve d'optimalité

Nous allons maintenant utiliser les outils de programmation mathématique pour obtenir une borne inférieure $\underline{\Delta}_8$ et ainsi démontrer que $\Delta_8 = 2.5843054\ldots$.

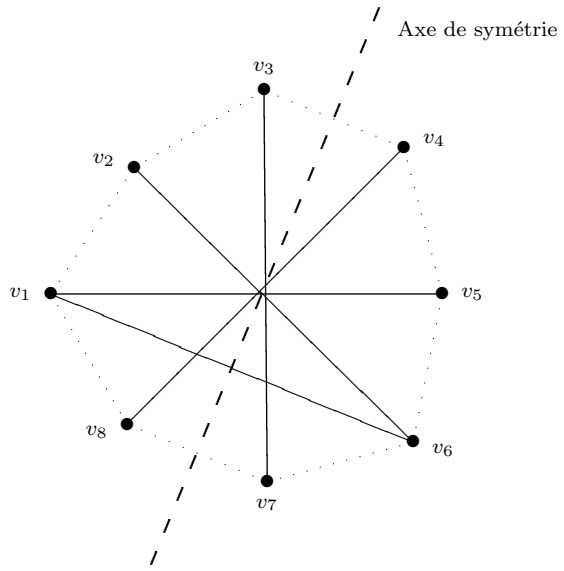

Figure 3.11 – La configuration de diamètres pour l'octogone de Vincze [85], avec $d = 2.588\ldots$

Théorème 3.4 *Le diamètre minimum d'un octogone convexe ayant des côtés de longueur unitaire est*

$$\Delta_8 = 2.5843054\ldots \tag{3.8}$$

Les coordonnées de ses sommets (à une translation et rotation près) sont données par

$$\begin{array}{ll} x_1 = 0 & y_1 = 0 \\ x_2 = 0.551457\ldots & y_2 = 0.834203\ldots \\ x_3 = 1.440436\ldots & y_3 = 1.292153\ldots \\ x_4 = 2.390830\ldots & y_4 = 0.981105\ldots \\ x_5 = 2.584305\ldots & y_5 = 0 \\ x_6 = 2.390830\ldots & y_6 = 0.981105\ldots \\ x_7 = 1.440436\ldots & y_7 = 1.292153\ldots \\ x_8 = 0.551457\ldots & y_8 = 0.834203\ldots \end{array}$$

Preuve. En utilisant le théorème 3.3 ainsi que la proposition 3.2, le problème de trouver l'octogone de côtés de longueur unitaire ayant un diamètre minimal peut être exprimé par le programme quadratrique non-convexe à contraintes quadratiques

non-convexes suivant :

$$
\begin{aligned}
\Delta_8 = \min_{x,y,d} \quad & d \\
\text{s.c.} \quad & \\
& x_2^2 + y_2^2 = 1 \\
& x_2^2 - 2x_2x_3 + x_3^2 + y_2^2 - 2y_2y_3 + y_3^2 = 1 \\
& x_3^2 - 2x_3x_4 + x_4^2 + y_3^2 - 2y_3y_4 + y_4^2 = 1 \\
& d^2 - 2x_4d + x_4^2 + y_4^2 = 1 \\
& d^2 - 2x_6d + x_6^2 + y_6^2 = 1 \\
& x_6^2 - 2x_6x_7 + x_7^2 + y_6^2 - 2y_6y_7 + y_7^2 = 1 \\
& x_7^2 - 2x_7x_8 + x_8^2 + y_7^2 - 2y_7y_8 + y_8^2 = 1 \\
& x_8^2 + y_8^2 = 1 \\
& x_2^2 - 2x_2x_6 + x_6^2 + y_2^2 + 2y_2y_6 + y_6^2 - d^2 = 0 \\
& x_3^2 - 2x_3x_7 + x_7^2 + y_3^2 + 2y_3y_7 + y_7^2 - d^2 = 0 \\
& x_4^2 - 2x_4x_8 + x_8^2 + y_4^2 + 2y_4y_8 + y_8^2 - d^2 = 0 \\
& x_4^2 + y_4^2 - d^2 \leq 0 \\
& x_6^2 + y_6^2 - d^2 \leq 0 \\
& -2x_2d + x_2^2 + y_2^2 \leq 0 \\
& x_2^2 - 2x_2x_7 + x_7^2 + y_2^2 + 2y_2y_7 + y_7^2 - d^2 \leq 0 \\
& x_3^2 - 2x_3x_6 + x_6^2 + y_3^2 + 2y_3y_6 + y_6^2 - d^2 \leq 0 \\
& x_3^2 - 2x_3x_8 + x_8^2 + y_3^2 + 2y_3y_8 + y_8^2 - d^2 \leq 0 \\
& x_4^2 - 2x_4x_7 + x_7^2 + y_4^2 + 2y_4y_7 + y_7^2 - d^2 \leq 0 \\
& -2x_8d + x_8^2 + y_8^2 \leq 0 \\
& 2.5629154 \leq d \leq 2.5843055 \\
& x_2, x_3, x_4, x_6, x_7, x_8, y_2, y_3, y_4, y_6, y_7, y_8 \geq 0.
\end{aligned}
\tag{3.9}
$$

Les huit premières contraintes correspondent aux côtés de longueur unitaire, en sens horaire. Les trois contraintes suivantes correspondent aux diamètres v_2v_6, v_3v_7 et v_4v_8 du théorème 3.3 ; le quatrième diamètre du théorème 3.3 est exprimé par le fait que $v_1 = (x_1, y_1)$ se situe à l'origine et que $v_5 = (d, 0)$ est sur l'axe des abscisses. Les huit dernières contraintes correspondent aux diagonales joignant des sommets à une distance de 3 sur le périmètre de l'octogone, et qui ne peuvent avoir une longueur supérieure à d.

L'algorithme décrit à la section 3.1 résout une relaxation de (3.9) puisqu'un certain niveau de tolérance (tant au niveau de la faisabilité que de l'optimalité) est accepté.

La solution produite par l'algorithme est telle que

$$\underline{\Delta}_8 = 2.58430541\ldots.$$

Rappelons que nous avons obtenue une solution réalisable pour (3.9) dans la preuve de la proposition 3.2 qui procurait la borne supérieure suivante :

$$\overline{\Delta}_8 = 2.5843054403.$$

Il s'ensuit que

$$2.58430541 \leq \Delta_8 \leq 2.5843054403$$

et donc que les sept premières décimales sont exactes. □

Résolution du programme quadratique

Avant de résoudre (3.9) avec l'algorithme présenté à la section 3.1, nous pouvons accélérer sa résolution en éliminant les solutions symétriquement équivalentes. Nous pouvons supposer, sans perte de généralité, un ordre donné d'apparition de la projection des sommets sur l'axe des abscisses en ajoutant les quatre contraintes suivantes :

$$\begin{aligned} x_2 - x_3 &\leq 0 \\ x_3 - x_4 &\leq 0 \\ x_8 - x_7 &\leq 0 \\ x_7 - x_6 &\leq 0. \end{aligned} \tag{3.10}$$

Nous pouvons aussi supposer, sans perte de généralité, qu'aucune diagonale parmi l'ensemble $\{v_1v_4, v_1v_6, v_2v_5, v_2v_7, v_3v_6, v_3v_8, v_4v_7, v_5v_8\}$ est plus grande que v_1v_4, et aussi que la diagonale v_4v_7 n'est pas plus grande que la diagonale v_1v_6. Ceci amène

à remplacer les huit dernières contraintes de (3.9) par les contraintes suivantes :

$$\begin{aligned}
x_4^2 + y_4^2 - d^2 &\leq 0 \\
x_6^2 + y_6^2 - x_4^2 - y_4^2 &\leq 0 \\
d^2 - 2x_2 d + x_2^2 + y_2^2 - x_4^2 - y_4^2 &\leq 0 \\
x_2^2 - 2x_2 x_7 + x_7^2 + y_2^2 + 2y_2 y_7 + y_7^2 - x_4^2 - y_4^2 &\leq 0 \\
x_3^2 - 2x_3 x_6 + x_6^2 + y_3^2 + 2y_3 y_6 + y_6^2 - x_4^2 - y_4^2 &\leq 0 \\
x_3^2 - 2x_3 x_8 + x_8^2 + y_3^2 + 2y_3 y_8 + y_8^2 - x_4^2 - y_4^2 &\leq 0 \\
x_4^2 - 2x_4 x_7 + x_7^2 + y_4^2 + 2y_4 y_7 + y_7^2 - x_6^2 - y_6^2 &\leq 0 \\
d^2 - 2x_8 d + x_8^2 + y_8^2 - x_4^2 - y_4^2 &\leq 0.
\end{aligned} \tag{3.11}$$

Le programme résultant de ces modifications a été résolu jusqu'à l'optimalité, avec des tolérances relatives ϵ_z et ϵ_r égales à 10^{-8} en utilisant des raffinements de bornes supplémentaires aux profondeurs 6 et 12. L'algorithme a trouvé la solution en explorant 197 noeuds en 6.4 secondes. Notons que les contraintes d'élimination de symétrie (3.10) et (3.11) accélèrent considérablement la résolution qui, sans ces contraintes, nécessite l'exploration de 3113 noeuds et un temps de 1072 secondes.

Comme pour les problèmes résolus dans la section 3.1.2 (voir tableau 3.1), nous donnons dans le tableau 3.3, les principales caractéristiques des trois programmes quadratiques résolus dans cette section :

- le programme quadratique (3.9) ;
- le programme quadratique (3.9) avec les contraintes d'élimination de symétrie ;
- le programme quadratique (3.9) avec les contraintes d'élimination et la contrainte d'unicité.

Tableau 3.3 – Taille des programmes quadratiques pour le problème d'octogone de diamètre minimum

Problème	Variables			Termes	Contraintes		
	Lin	Quad	Tot	Quad	Lin	Quad	
					$\leq$	$\leq$	$=$
QQP (3.9)	0	13	13	22	14	8	11
QQP (3.9) + sym.	0	13	13	22	18	8	11
QQP (3.9) + sym. + unicité	0	13	13	22	18	9	11

Preuve d'unicité

Dans cette section, nous montrons que la solution trouvée à la section précédente est ϵ-unique. Pour ce faire, définissons d'abord le concept d'unicité-proche d'une solution optimale d'un problème général d'optimisation.

Définition 3.1 *Soit $x^* \in \mathbb{R}^n$ une solution optimale d'un problème d'optimisation*

$$(P) \qquad \min_{x \in \Omega} f(x)$$

où $\Omega \subset \mathbb{R}^n$ et $f : \mathbb{R}^n \to \mathbb{R}$. Pour un $\epsilon^* \geq 0$, $x^* \in \Omega$ est une solution optimale ϵ^*-unique de (P) si

$$f(x^*) \; < \; \min_{x \in \Omega, \|x - x^*\| \geq \epsilon} f(x)$$

pour tout $\epsilon > \epsilon^*$.

Pour $\epsilon^* = 0$, cette définition revient à la définition classique de l'unicité. Nous allons maintenant démontrer qu'il n'existe aucun autre octogone optimal à une distance euclidienne supérieure à 0.0123 de l'octogone proposé.

Théorème 3.5 *L'octogone optimal du théorème 3.4 est ϵ^*-unique, avec $\epsilon^* = 0.0123$, en utilisant la norme euclidienne.*

Preuve. Considérons x_i^* et y_i^* pour $i = 1, 2, \ldots, 8$ la solution donnée dans l'énoncé du théorème 3.4. En fixant $x_1 = y_1 = y_5 = 0$ et $x_5 = d$, l'algorithme présenté à la section 3.1 démontre (par l'exploration de 15 noeuds en 6.7 secondes) que le fait d'ajouter la contrainte

$$\sum_{i=1}^{8} (x_i - x_i^*)^2 + (y_i - y_i^*)^2 \geq 0.000151$$

au problème (3.9) avec les contraintes d'élimination de symétrie (3.10), (3.11) le rend non-réalisable étant donné la borne supérieure sur la variable d. Rappelons que l'algorithme résout, à chaque noeud, la relaxation linéaire de ce problème quadratique. Dans ce cas-ci, l'ajout de coupes fait en sorte de rendre cette relaxation linéaire non-réalisable. Ce qui implique que le problème quadratique est non-réalisable. □

3.3 Hyperplan séparateur en norme-ℓ_2

Dans cette section, nous présentons une application de la programmation quadratique à un problème important en *classification automatique des données*. Nous montrons comment la résolution de ce problème peut, dans certains cas, être considérablement accélérée par l'utilisation d'une borne heuristique.

Dans le problème considéré, on cherche à séparer (ou discriminer) deux ensembles de points dans un espace réel $\mathbb{R}^n$ avec un (hyper)plan qui assigne un demi-espace à chaque ensemble de points. Si les enveloppes convexes de ces deux ensembles se chevauchent, une séparation parfaite est impossible puisqu'aucun plan ne peut être construit de manière à ce que tous les points d'un ensemble se retrouvent du même côté du plan et que tous les autres points se retrouvent du côté opposé. Nous cherchons alors un plan minimisant un certain type de mesure de l'erreur de séparation (voir [84] pour des références sur le sujet). Ce problème se formule généralement sous la forme de modèles d'optimisation complexes.

Plusieurs approches pour le problème de séparation reposent sur le critère de séparation où l'on minimise la somme des distances des points mal classés par rapport au plan. Ces approches sont généralement basées sur la programmation linéaire. Les fonctions-objectif de ces programmes linéaires sont, cependant, des approximations linéaires des vraies mesures de distances, qui essentiellement ignorent la propriété non-linéaire du problème réel.

Mangasarian [71] formule le problème de minimisation de la somme des distances en norme-ℓ_p des points mal classés par rapport au plan sous forme d'un programme mathématique. Il obtient alors un problème de minimisation d'une fonction convexe (une somme d'opérateurs $\max\{0, \cdot\}$) sur une sphère unitaire définie par la norme duale de la norme choisie pour mesurer les distances des points mal classés par rapport au plan. Mangasarian mentionne que la résolution exacte, pour le cas de la norme-ℓ_1, peut se faire en résolvant $2n$ programmes linéaires.

Nous considérons le cas de la norme-ℓ_2, i.e., le problème de séparation où l'on cherche le plan qui minimise la somme des distances euclidiennes au plan des points mal classés. Marcotte et Savard [74] considèrent un problème proche qui consiste à déterminer un hyperplan qui sépare deux ensembles de points de manière à minimiser la distance euclidienne maximale d'un point mal classé. Ils formulent ce problème comme un programme quadratique et le reformulent, de façon équivalente, comme un programme linéaire biniveau pour ensuite le résoudre par un algorithme de programmation linéaire biniveau.

À la section 3.3.1, nous montrons que le problème de déterminer le plan qui minimise la somme des distances euclidiennes au plan des points mal classés se formule comme un programme quadratique en utilisant le formulation générale de Mangasarian [71]. À la section 3.3.2, nous présentons des résultats obtenus sur des problèmes générés aléatoirement ainsi que sur des problèmes réels provenant d'une base de données publiques. Toutes ces instances sont résolues exactement avec la version améliorée de l'algorithme présenté à la section 3.1. Nous illustrons aussi, par des expériences numériques, que l'utilisation d'une solution heuristique peut, dans certains cas, accélérer considérablement la résolution exacte.

3.3.1 Formulation mathématique

Dans cette section, nous présentons le programme mathématique du problème général de recherche d'un hyperplan séparateur en norme arbitraire pour ensuite considérer le cas de la norme-ℓ_2. En utilisant la notation de [71], nous dénotons $\mathcal{A}$ et $\mathcal{B}$ les deux ensembles de points dans $\mathbb{R}^n$ et nous supposons qu'ils contiennent respectivement m et k points. Les coordonnées de tous ces points sont représentées par les lignes des matrices $A \in \mathbb{R}^{m\times n}$ et $B \in \mathbb{R}^{k\times n}$.

Un point $x \in \mathcal{A} \cup \mathcal{B}$ est considéré mal classé par rapport à un plan P donné

$$P = \left\{ x \left| w^T x = \gamma \right. \right\}, \gamma \in \mathbb{R}, w \in \mathbb{R}^n, w \neq 0 \tag{3.12}$$

lorsque

$$\begin{cases} w^T x \geq \gamma & \text{si } x \in \mathcal{A}, \\ w^T x \leq \gamma & \text{si } x \in \mathcal{B}. \end{cases} \tag{3.13}$$

Pour une valeur quelconque $p \in [0, \infty]$, la distance en norme-ℓ_p entre le point $x \in \mathcal{A} \cup \mathcal{B}$ et sa projection $\pi(x)$ sur le plan P est donnée par

$$\|x - \pi(x)\|_p = \frac{\left| w^T x - \gamma \right|}{\|w\|'_p} \tag{3.14}$$

où $\|\cdot\|'_p$ représente la norme duale de $\|\cdot\|_p$. Rappelons que pour $p = 2$, on a $\|\cdot\|'_2 = \|\cdot\|_2$.

Notons qu'il existe un degré de liberté dans la caractérisation du plan P, qui peut être utilisé pour fixer une échelle arbitraire. La contrainte de mise à l'échelle

$$\|w\|'_p = 1 \tag{3.15}$$

permet d'éliminer le dénominateur de (3.14) et d'interdire la solution dégénérée $w = 0$. Cette approche, utilisées dans différents travaux (voir, par exemple, [73] et les références qui s'y trouvent), semble être proposée pour la première fois dans [20].

Le problème d'optimisation résultant (équivalent au problème (17) de [71]) est :

$$\begin{aligned} \min_{w,\gamma} \quad & \sum_{i=1}^{m} \max\left\{-w^T A_i + \gamma, 0\right\} + \sum_{j=1}^{k} \max\left\{w^T B_j - \gamma, 0\right\} \\ \text{s.c.} \quad & \\ & \| w \|'_p = 1 \end{aligned} \tag{3.16}$$

où $w \in \mathbb{R}^n$, $\gamma \in \mathbb{R}$, $A_i \in \mathbb{R}^n$ représente la i^e ligne de A et $B_j \in \mathbb{R}^n$ représente la j^e ligne de B.

En linéarisant les opérateurs $\max\{\cdot, 0\}$ dans la fonction-objectif, on obtient :

$$\begin{array}{ll} \min\limits_{w,\gamma,y,z} & \sum\limits_{i=1}^{m} y_i + \sum\limits_{j=1}^{k} z_j \\ \text{s.c.} & \\ & y_i \geq -w^T A_i + \gamma \quad \text{pour } i = 1,..,m \\ & z_j \geq w^T B_j - \gamma \quad \text{pour } j = 1,..,k \\ & \|w\|'_p = 1 \\ & y_i \geq 0 \quad \text{pour } i = 1,..,m \\ & z_j \geq 0 \quad \text{pour } j = 1,..,k \end{array} \tag{3.17}$$

où $w \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, $z \in \mathbb{R}^k$ et $\gamma \in \mathbb{R}$.

Notons que ce problème est linéaire à l'exception de la contrainte $\|w\|'_p = 1$. Sous la norme euclidienne, cette contrainte est équivalente à

$$\|w\|'_2 = \sqrt{w^T w} = 1 \Leftrightarrow w^T w = 1. \tag{3.18}$$

Nous obtenons alors un programme quadratique ne contenant aucun terme bilinéaire et possédant une fonction-objectif linéaire. Il s'agit donc d'un cas particulier du problème quadratique général QQP énoncé en début de chapitre.

3.3.2 Résultats numériques

Les résultats présentés dans cette section ont été obtenus avec la version améliorée de l'algorithme décrit à la section 3.1. Nous avons résolu un ensemble de problèmes provenant de la base de données *UCI Repository* [21] ainsi que deux séries de problèmes générés aléatoirement avec le générateur NDC de Musicant [76] qui produit des ensembles normalement distribués. Ce générateur est disponible publiquement et a été utilisé dans d'autres études de classification (par exemple, [38, 72]). Les paramètres utilisés pour la génération des instances aléatoires ainsi que les étapes de pré-traitement effectuées sur les problèmes provenant de la base de données réelles *UCI Repository* sont détaillés à l'annexe A.

Pour la première série de problèmes aléatoires (notée $2k$ pour la suite), chaque instance contient 2000 points et nous mesurons l'effet d'augmenter la dimension de 4 à 13. Pour la deuxième série de problèmes aléatoires (notée $6d$ pour la suite), la dimension est fixée à 6 et nous explorons l'effet d'augmenter le nombre de points de 2000 à 20000 (avec un pas de 2000). Les tableaux 3.4, 3.5 et 3.6 présentent les principales caractéristiques relatives à la taille des programmes quadratiques associées à ces problèmes. Les colonnes *Dim* et *Pts* réfèrent à la dimension et au nombre de points du problème. Pour les autres colonnes, on utilise la même notation que dans le tableau 3.1. En comparant ces tableaux avec les tableaux 3.1 et 3.3, on constate que les problèmes résolus dans la présente section possèdent un nombre beaucoup plus élevé de variables et de contraintes linéaires mais une proportion beaucoup plus faible de termes quadratiques.

Tableau 3.4 – Taille des programmes quadratiques pour la série de problèmes aléatoires $2k$

Problème		Variables			Termes	Contraintes		
Dim	Pts	Lin	Quad	Tot	Quad	Lin	Quad	
						$\leq$	$\leq$	$=$
4	2000	2001	4	2005	4	4000	0	1
5	2000	2001	5	2006	5	4000	0	1
6	2000	2001	6	2007	6	4000	0	1
7	2000	2001	7	2008	7	4000	0	1
8	2000	2001	8	2009	8	4000	0	1
9	2000	2001	9	2010	9	4000	0	1
10	2000	2001	10	2011	10	4000	0	1
11	2000	2001	11	2012	11	4000	0	1
12	2000	2001	12	2013	12	4000	0	1
13	2000	2001	13	2014	13	4000	0	1

Tableau 3.5 – Taille des programmes quadratiques pour la série de problèmes aléatoires $6d$

Problème		Variables			Termes	Contraintes		
Dim	Pts	Lin	Quad	Tot	Quad	Lin	Quad	
						$\leq$	$\leq$	$=$
6	2000	2001	6	2007	6	4000	0	1
6	4000	4001	6	4007	6	8000	0	1
6	6000	6001	6	6007	6	12000	0	1
6	8000	8001	6	8007	6	16000	0	1
6	10000	10001	6	10007	6	20000	0	1
6	12000	12001	6	12007	6	24000	0	1
6	14000	14001	6	14007	6	28000	0	1
6	16000	16001	6	16007	6	32000	0	1
6	18000	18001	6	18007	6	36000	0	1
6	20000	20001	6	20007	6	40000	0	1

Tableau 3.6 – Taille des programmes quadratiques pour la série de problèmes UCI

Problème			Variables			Termes	Contraintes		
Nom	Dim	Pts	Lin	Quad	Tot	Quad	Lin	Quad	
							$\leq$	$\leq$	$=$
Cancer	9	683	684	9	693	9	1366	0	1
Diabetes	8	768	769	8	777	8	1536	0	1
Echocardiogram	7	74	75	7	82	7	148	0	1
Glass	9	214	215	9	224	9	428	0	1
Housing	13	506	507	13	520	13	1012	0	1
Hepatitis	16	150	151	16	167	16	300	0	1

Nous utilisons également une heuristique pour obtenir une première solution réalisable dans le but d'accélérer la résolution exacte. L'utilisation de ce type d'approche est de plus en plus répandue (voir, par exemple, [30, 49]). Pour chacune des instances résolues, nous utilisons d'abord une implémentation spécifique de la recherche à voisinage variable [19] pour obtenir une borne supérieure sur la valeur optimale de la fonction-objectif. Cette borne est ensuite ajoutée comme contrainte au problème (3.17). Dans le but de mesurer l'efficacité de cet ajout, nous avons également résolu (lorsque le temps de résolution était raisonnable) les mêmes instances avec la méthode exacte sans l'ajout de la borne sur la fonction-objectif.

Les tableaux 3.7, 3.8 et 3.9 présentent les résultats obtenus pour les séries $2k$, $6d$ et UCI respectivement. Pour les problèmes aléatoires, nous avons généré 10 instances pour chaque taille et les résultats rapportés dans les tableaux sont les moyennes correspondantes. Dans tous ces tableaux, la colonne *Ecart* indique à quel pourcentage de la valeur de la solution exacte se situe la la valeur de la solution heuristique. La colonne *Class* mesure le pourcentage de points bien classés par rapport à l'hyperplan associé à la solution. La valeur de la fonction-objectif est arrondie à la précision relative de la méthode, i.e., 10^{-5}. Pour la méthode heuristique, le temps de calcul est donné tandis que pour la méthode exacte, le temps et le nombre de noeuds sont donnés, à la fois pour la résolution avec ou sans l'utilisation de la borne heuristique. Le temps CPU est mesuré en secondes. La dernière colonne du tableau donne le gain de temps (en pourcentage) obtenu par l'utilisation de la borne heuristique, incluant évidemment le temps nécessaire pour obtenir la solution heuristique.

En observant les tableaux 3.7 et 3.8, on note que l'heuristique RVV performe très bien et trouve une solution équivalente (en fonction de la précision) à la solution exacte lorsque la dimension est faible. La différence entre les deux solutions s'accroît légèrement lorsque la dimension augmente. Ce phénomène est encore plus évident lorsqu'on analyse le tableau 3.9. Cette différence est probablement due en partie au fait que la méthode exacte trouve une solution approchée (tant au niveau de sa faisabilité que de son optimalité). Cependant, une bonne part de cette différence s'explique probablement aussi par le fait que l'heuristique performe moins bien pour les problèmes où la dimension est plus grande.

En observant le tableau 3.7, on constate que le temps CPU (et le nombre de noeuds) requis par l'algorithme exact s'accroît très rapidement avec la dimension, tandis que l'accroissement du temps de l'heuristique est modéré. L'analyse du tableau 3.8 nous permet de constater que le nombre de points affecte énormément le temps CPU mais pas le nombre de noeuds requis par l'algorithme exact. Le nombre de points n'affecte pas vraiment le nombre de noeuds car ce dernier est fortement relié au nombre de

Tableau 3.7 – Résultats obtenus sur la série de problèmes aléatoires $2k$

Taille problème		Heuristique			Algorithme exact						Gain global temps
							Sans borne heur.		Avec borne heur.		
Dim	Pts	Ecart	Class	Temps	Obj	Class	Temps	Noeuds	Temps	Noeuds	
4	2000	0.000%	94.41%	10.5	3.10957	94.41%	5.0	168	6.3	17	-238.0%
5	2000	0.000%	93.83%	11.1	3.45771	93.84%	11.4	418	17.4	42	-149.2%
6	2000	0.000%	93.21%	13.9	4.33043	93.23%	34.5	1150	38.1	266	-50.8%
7	2000	0.000%	91.73%	21.8	5.03871	91.75%	118.3	3117	95.6	1490	0.8%
8	2000	0.000%	90.13%	29.2	5.96413	90.16%	557.4	7031	223.1	3258	54.7%
9	2000	0.001%	90.13%	44.3	6.29355	90.17%	1796.3	15461	680.1	8373	59.7%
10	2000	0.002%	89.51%	56.8	6.48006	89.55%	3194.4	34760	1728.4	19334	44.1%
11	2000	0.002%	83.65%	64.0	11.27714	83.74%	38530.5	170373	17491.7	112062	54.4%
12	2000	0.008%	88.87%	78.5	7.26097	89.02%	-	-	18970.8	141496	-
13	2000	0.005%	86.02%	93.8	9.49367	86.15%	-	-	60818.1	378378	-

Tableau 3.8 – Résultats obtenus sur la série de problèmes aléatoires $6d$

Taille problème		Heuristique			Algorithme exact						Gain global temps
							Sans bornes heur.		Avec borne heur.		
Dim	Pts	Ecart	Class	Temps	Obj	Class	Temps	Noeuds	Temps	Noeuds	
6	2000	0.000%	93.14%	14.6	3.97815	93.15%	36.0	983	38.2	209	-46.6%
6	4000	0.000%	92.22%	28.3	7.64622	92.22%	212.6	1081	210.6	380	-12.4%
6	6000	0.000%	91.59%	34.2	14.21926	91.60%	635.5	1496	726.1	643	-19.6%
6	8000	0.000%	92.27%	54.9	15.94860	92.28%	959.7	1302	947.1	343	-4.4%
6	10000	0.000%	91.04%	77.7	23.78685	91.04%	1433.3	1261	1950.2	449	-41.5%
6	12000	0.000%	91.01%	69.6	27.07589	91.01%	2593.5	1270	2131.5	268	15.1%
6	14000	0.000%	89.68%	74.9	35.80013	89.69%	2858.2	1457	3877.6	596	-38.3%
6	16000	0.000%	92.25%	112.4	25.78079	92.25%	2146.0	1134	3143.3	345	-51.7%
6	18000	0.000%	92.80%	115.5	36.51885	92.80%	5915.7	1377	5700.6	562	1.7%
6	20000	0.000%	94.59%	118.5	24.93259	94.59%	5777.2	1184	3508.1	603	37.2%

Tableau 3.9 – Résultats obtenus sur la série de problèmes UCI

Problème			Heuristique			Algorithme exact						Gain sur temps
	taille							Sans borne heur.		Avec borne heur.		
nom	Dim	Pts	Ecart	Class	Temps	Obj	Class	Temps	Noeuds	Temps	Noeuds	
Cancer	9	683	0.577%	96.93%	15.8	2.06696	97.07%	156.2	18415	57.0	8385	53.38%
Diabetes	8	768	0.001%	75.39%	10.0	12.24314	75.39%	611.6	19273	211.2	13219	63.83%
Echocardiogram	7	74	0.008%	74.32%	0.73	1.20730	75.68%	4.6	2677	1.3	529	56.06%
Glass	9	214	1.172%	95.33%	6.4	0.03114	95.33%	4.5	1449	1.4	75	-72.12%
Housing	13	506	0.539%	84.39%	30.8	0.89714	84.39%	550.2	24639	30.4	871	88.89%
Hepatitis	16	150	0.511%	86.67%	10.4	0.87113	88.00%	14181.3	350767	50.7	10399	99.57%

termes quadratiques qui lui est invariable pour une dimension donnée. Cependant, le temps augmente très rapidement avec le nombre de points car la relaxation linéaire résolue à chaque noeud contient alors un plus grand nombre de contraintes qui est égal au double du nombre de points auquel on ajoute un.

L'ajout d'une borne heuristique permet de réduire considérablement le nombre de noeuds explorés par l'algorithme exact dans tous les cas. Cependant, le gain global sur le temps est parfois positif, parfois négatif. Le gain de temps est positif et très significatif pour les exemples à dimension élevée de la série $2k$ et encore plus pour ceux de la série UCI. Par exemple, les instances à 12 et 13 dimensions de la série $2k$ ne peuvent être résolues en des temps raisonnables sans la borne heuristique tandis que les instances à 13 dimensions sont résolues, avec la borne heuristique, environ 3 fois plus rapidement que les instances à 11 dimensions sans la borne. Néanmoins, le gain global est parfois négatif pour les instances à faible dimension. Ce phénomène est dû au fait que nous effectuons un raffinement à tous les noeuds de profondeur 5 dans l'arbre pour augmenter l'effet d'utiliser une solution initiale heuristique. Le temps utilisé par ces raffinements est, pour les problèmes de faible dimension, trop élevé par rapport au gain de temps généré par la réduction du nombre de noeuds. Ce phénomène a été mis en évidence dans l'analyse des améliorations apportées à l'algorithme à la section 3.1.2.

3.4 Discussion

Dans ce chapitre, nous avons discuté de la programmation quadratique à contraintes quadratiques. Nous avons d'abord décrit un algorithme d'énumération implicite avec ajout de coupes pour résoudre exactement ce type de problème ainsi que les améliorations que nous avons apportées à cet algorithme dans le cadre de cette thèse. Les améliorations les plus importantes sont :

– la possibilité de résoudre des programmes quadratiques avec un nombre élevé de variables linéaires et un nombre modéré de termes quadratiques ;

– un contrôle plus complet des paramètres de l'algorithme ;
– la possibilité d'effectuer des raffinements de bornes à différentes profondeurs de l'arbre afin de réduire le nombre de noeuds explorés.

Ensuite, nous avons montré comment utiliser cet algorithme, conjointement à l'utilisation d'une borne fournie par une heuristique ou à une analyse géométrique pour résoudre efficacement deux problèmes.

Nous avons d'abord trouvé l'octogone convexe ayant des côtés de longueur unitaire et un diamètre minimum avec une précision de sept décimales, répondant ainsi à une question ouverte datant de 1950. Nous avons aussi démontré qu'il n'y a aucun autre octogone optimal à l'extérieur d'une boule de rayon égal à 0.0123. Des résultats avec une meilleure précision numérique pourrait évidemment être obtenus en réduisant les paramètres de tolérance (ϵ_z et ϵ_r) de l'algorithme de programmation quadratique. Cependant, notre version actuelle de l'algorithme conjointement avec CPLEX 8.1 ne peut considérer une tolérance inférieure.

Enfin, nous avons étudié un problème fondamental en classification automatique des données : la séparation en norme-ℓ_p. Malgré de récents progrès, des techniques pratiques pour la résolution exacte des autres cas que la norme-ℓ_1 n'étaient pas encore disponibles. Nous avons proposé et mis en oeuvre une nouvelle approche qui rend possible la résolution exacte de problèmes de taille assez grande pour le cas de la norme-ℓ_2. Nous avons résolu en des temps de calcul raisonnables des problèmes générés aléatoirement contenant jusqu'à 20000 points pour des problèmes à 6 dimensions et jusqu'à 13 dimensions pour des problèmes avec 2000 points. Nous résolvons également quelques problèmes réels provenant d'une base de données publique. Finalement, des résultats numériques illustrent que, pour des problèmes dont la dimension est assez grande, les temps de calcul peuvent être réduits considérablement en utilisant une solution initiale fournie par une heuristique.

CONCLUSION

La programmation mathématique est une discipline des mathématiques appliquées où l'on étudie la formulation et la résolution de problèmes d'optimisation. Les techniques développées pour résoudre les différents types de programmes mathématiques sont très variées. Dans certains cas, il est possible de résoudre un programme mathématique en utilisant directement une méthode conçue spécialement pour ce type de modèle. Cependant, dans d'autres situations, il est indispensable de combiner certaines méthodes de résolution pour en arriver à résoudre efficacement un problème d'optimisation difficile. Dans cette thèse, nous avons discuté de quatre applications dont la résolution efficace passe par l'utilisation conjointe de certaines approches classiques de la programmation mathématique jumelée parfois à une autre approche issue du domaine du problème étudié.

Dans un premier temps, nous avons proposé de combiner la technique de génération de colonnes de la programmation linéaire à quelques techniques de résolution de la programmation quadratique sans contraintes en variables 0–1 pour résoudre le problème de plongement-ℓ_1 d'une distance à valeurs réelles ainsi que certains problèmes connexes d'optimisation. La méthode proposée a permis de résoudre tous ces problèmes en un temps raisonnable pour des instances de petite à moyenne taille selon le problème.

Nous avons également proposé de combiner plusieurs approches afin d'établir une nouvelle méthode de résolution pour le problème de la satisfiabilité probabiliste. La méthode proposée est l'utilisation d'une méthode locale basée sur les règles jumelée à un algorithme de génération de colonnes stabilisé utilisant les techniques de la programmation non-linéaire en variables 0–1 sans contraintes pour résoudre le problème auxiliaire. Nous montrons que cette méthode permet de réduire et parfois même d'éliminer les défauts des méthodes de résolution classiques de la satisfiabilité probabiliste.

Nous avons également développé et utilisé une nouvelle version d'un algorithme d'énumération implicite avec ajout de coupes conçu pour la résolution de programmes quadratiques à contraintes quadratiques. Cet algorithme trouve en un temps fini une solution optimale globale à l'intérieur d'un certain niveau de tolérance. La nouvelle version de l'algorithme inclut des améliorations informatiques et algorithmiques.

Nous avons résolu, en utilisant conjointement cet algorithme et une analyse géométrique combinatoire, un problème qui consiste à trouver l'octogone convexe avec des côtés de longueur unitaire possédant un diamètre minimum. En fait, nous avons montré comment formuler ce problème sous la forme d'un programme quadratique à contraintes quadratiques après avoir réduit considérablement l'aspect combinatoire du problème par l'établissement de conditions nécessaires d'optimalité.

Nous avons proposé et appliqué une méthode de résolution exacte d'un problème fondamental en classification automatique des données, i.e., le problème de séparer deux ensembles de points dans un espace réel à n dimensions avec un hyperplan qui minimise la somme des distances, en norme-ℓ_2, à l'hyperplan des points mal classés. Nous avons résolu d'assez grandes instances de ce problème et avons illustré que, pour les problèmes à dimension élevée, le fait de jumeler une méthode heuristique à la méthode exacte permet d'accélérer la résolution.

Parmi les nouvelles voies de recherche prometteuses qui s'inscrivent dans la continuité des travaux réalisés dans cette thèse, notons (i) l'identification de "bons" paramètres pour la nouvelle version de l'algorithme de résolution de programmes quadratiques, (ii) la résolution d'une extension du problème de séparation en classification automatique des données et (iii) la résolution de nouvelles applications de la programmation quadratique dont le problème de la satisfiabilité qualitative conditionnelle.

Élaborons sur chacune de ces voies de recherche.

(i) La nouvelle version de l'algorithme de résolution de programmes quadratiques décrite au chapitre 3 offre à l'utilisateur un grand contrôle sur la fixation des paramètres algorithmiques. Il serait intéressant d'utiliser cette flexibilité pour en arriver à

identifier des ajustements de paramètres potentiellement bons pour différentes classes de problèmes. Pour ce faire, une approche possible serait de résoudre plusieurs programmes quadratiques en utilisant différents ajustements de paramètres. Ensuite, l'analyse des résultats nous permettrait peut-être de discriminer les problèmes selon le type d'ajustements de paramètres qui semble le plus approprié. Un des paramètres les plus intéressants à analyser est évidemment le choix des profondeurs où l'on doit exécuter des raffinements de bornes.

(ii) Une extension possible au problème d'identification d'un hyperplan séparateur étudié au chapitre 3 consiste en l'établissement d'un modèle d'exploitation de données utilisant une somme pondérée de classificateurs associés à des plans séparateurs. Ce problème peut se résoudre par la technique de génération de colonnes où le problème auxiliaire, formulé comme un programme mixte, consiste à trouver un plan minimisant la somme des poids des variables duales avec un signe + ou - selon qu'un point est bien ou mal classé.

(iii) Parmi les applications intéressantes de la programmation quadratique que nous pourrions étudier, notons certains problèmes de gestion de chaînes d'offre (*supply chain management*), de géométrie, d'optimisation de portefeuille d'actions (*portfolio optimization*) ainsi que le problème de la satisfiabilité probabiliste qualitative conditionnelle. Élaborons davantage sur cette dernière extension puisque une méthode envisageable pour sa résolution englobe la presque totalité des techniques utilisées dans cette thèse.

Incorporer des probabilités qualitatives et des probabilités conditionnelles est parfois indispensable en logique probabiliste. En effet, dans certaines situations, un expert peut avoir de la difficulté à évaluer quantitativement la probabilité de certaines propositions logiques mais peut alors être en mesure de transmettre des jugements du type "A est plus probable que B" ou "A est au moins aussi probable que B". Ainsi, au lieu de donner une approximation de la probabilité de chaque proposition, l'expert fournit une série de relations entre les probabilités de certaines paires d'événements.

Ces relations sont appelées *probabilités qualitatives*. De plus, il arrive aussi qu'un expert soit en mesure de donner un jugement quantitatif ou qualitatif sur certaines probabilités qu'en présence d'une condition. On doit alors utiliser les probabilités conditionnelles.

Des problèmes contenant des relations entre probabilités conditionnelles sont généralement très difficiles à résoudre. En effet, lorsque les événements conditionnant sont différents, ce problème se formule comme un programme quadratique à contraintes quadratiques contenant un nombre exponentiel de variables linéaires avec un nombre relativement restreint de termes quadratiques. Pour le résoudre, une méthode envisageable consiste à combiner les approches vues aux chapitres 2 et 3 de cette thèse pour obtenir un algorithme d'énumération implicite avec coupes où la relaxation linéaire se résout par la technique de génération de colonnes. Il serait intéressant d'inclure aussi dans cet algorithme certaines des techniques d'accélération vues dans cette thèse telles que l'insertion de colonnes multiples, la stabilisation et l'utilisation d'une borne initiale heuristique.

L'implantation d'un tel algorithme nécessite tout de même un travail considérable au niveau informatique mais aussi au niveau de l'évaluation et de l'identification des bons choix algorithmiques pour s'assurer de l'efficacité de la méthode. Cependant, cette implantation s'inscrit très bien dans la suite de cette thèse puisqu'elle combine la presque totalité des techniques utilisées et élaborées. Ainsi, l'expertise acquise au cours de cette thèse permettrait fort probablement d'élaborer et d'implanter cette méthode de façon efficace.

BIBLIOGRAPHIE

[1] Al-Khayyal, F.A. et Falk, J.E. (1983). Jointly constrained biconvex programming. *Math. Oper. Res.*, 8(2) : 273–286.

[2] Alkhamis, T.M., Hasan, M. et Ahmed, M.A. (1998). Simulated annealing for the unconstrained quadratic pseudo-boolean function. *European Journal of Operational Research*, 108(3) : 641–652.

[3] Allemand, K., Liebling, T.M. et Lodi, A. (1997). A genetic algorithm for quadratic 0-1 programming. Rapport Technique OR-97-12, DEIS-Universita di Bologna.

[4] Anbil, R., Forrest, J.J. et Pulleyblank, W.R. (1998). Column Generation and the Airline Crew Pairing Problem. *Documenta Mathematica*, Extra Volume ICM 3 : 677 – 686.

[5] Assouad, P. (1979-1980). Plongements isométriques dans L^1 : aspect analytique. Dans Choquet, G. *et al.*, éditeurs, *Séminaire d'Initiation à l'analyse*, numéro 14, Université de Paris VI.

[6] Audet, C. (1997). *Optimisation globale structurée : propriétés, équivalences et résolution*. Thèse de doctorat, École Polytechnique de Montréal.

[7] Audet, C., Brimberg, J., Hansen, P., Le Digabel, S. et Mladenovic, N. (2004). Pooling Problem : Alternate Formulations and Solution Methods. *Management Science*, 50(6) : 761–776.

[8] Audet, C., Carrizosa, E. et Hansen, P. (2004). An Exact Method for Fractional Goal Programming. *Journal of Global Optimization*, 29(1) : 113–120.

[9] Audet, C., Hansen, P., Jaumard, B. et Savard, G. (2000). A branch and cut algorithm for nonconvex quadratically constrained quadratic programming. *Math. Program.*, 87(1, Ser. A) : 131–152.

[10] Audet, C., Hansen, P., Messine, F. et Xiong, J. (2002). The largest small octagon. *J. Combin. Theory Ser. A*, 98(1) : 46–59.

[11] Avis, D. (1977). *Some Polyhedral Cones Related to Metric Spaces*. Thèse de doctorat, Stanford University.

[12] Avis, D. et Deza, M. (1991). The cut cone, L^1 embeddability, complexity, and multicommodity flows. *Networks*, 21(6) : 595–617.

[13] Bateman, P. et Erdős, P. (1951). Geometrical extrema suggested by a lemma of Besicovitch. *Amer. Math. Monthly*, 58 : 306–314.

[14] Benayade, M. et Fichet, B. (1994). Algorithms for a geometrical P.C.A. with the L_1-norm. Dans Diday, E. *et al.*, éditeurs, *New Approaches in Classification and Data Analysis*, pp. 75–84. Springer-Verlag, Berlin.

[15] Boole, G. (1851). Proposed Question in the Theory of Probabilities. *The Cambridge and Dublin Mathematical Journal*, 6(186).

[16] Boole, G. (1854). *An Investigation of the Laws of Thought, on which are Founded the Mathematical Theories of Logic and Probabilities*. London : Walton and Maberley. (Reprint New York : Dover 1958).

[17] Boole, G. (1854). On the Conditions by which Solutions of Questions in the Theory of Probabilities are Limited. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, 4(8) : 91–98.

[18] Boole, G. (1854). On the General Method in the Theory of Probabilitites. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, 4(8) : 431–444.

[19] Caporossi, G., Hansen, P. et Karam, A. (2004). Arbitrary-norm plane Separation by Variable Neighborhood Search. *in preparation*.

[20] Cavalier, T.M., Ignizio, J.P. et Soyster, A.L. (1989). Discriminant analysis via mathematical programming : certain problems and their causes. *Comput. Oper. Res.*, 16(4) : 353–362.

[21] Christopher, J.M. et Murphy, P.M. (1998). UCI Repository of Machine Learning Databases, *http ://www.ics.uci.edu/∼mlearn/MLRepository.html*.

[22] Chvátal, V. (1983). *Linear programming*. A Series of Books in the Mathematical Sciences. W. H. Freeman and Company, New York.

[23] Coletti, G. et Scozzafava, R. (2002). *Probabilistic Logic in a Coherent Setting*. Kluwer Acad. Publ., Dordrecht.

[24] Critchley, F. (1980). Optimal norm characterizations of multidimensional Scaling Methods and Some Related Data Analysis Problems. Dans Diday, E. *et al.*, éditeurs, *Data Analysis and Informatics*, North Holland.

[25] Dantzig, G.B. (1963). *Linear programming and extensions*. Princeton University Press, Princeton, N.J.

[26] Finetti, B.de. (1937). La prévision : ses lois logiques, ses sources subjectives. *Annales de l'Institut Henri Poincaré*, 7 : 1–68.

[27] De Finetti, B. (1974). *Theory of Probability – A Critical Introductory Treatment*, volume 1. Wiley, New York.

[28] De Finetti, B. (1975). *Theory of Probability – A Critical Introductory Treatment*, volume 2. Wiley, New York.

[29] Dembo, R.S. (1976). A set of geometric programming test problems and their solutions. *Math. Programming*, 10(2) : 192–213.

[30] Desaulniers, G., Desrosiers, J. et Solomon, M.M. (2002). Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. Dans *Essays and surveys in metaheuristics (Angra dos Reis, 1999)*, Oper. Res./Comput. Sci. Interfaces Ser., pp. 309–324. Kluwer Acad. Publ., Boston, MA.

[31] Deza, M.M. et Laurent, M. (1997). *Geometry of cuts and metrics*. Springer-Verlag, Berlin.

[32] Douanya Nguetsé, G.B., Hansen, P. et Jaumard, B. (1995). Probabilistic satisfiability and decomposition. Dans *Symbolic and quantitative approaches to reasoning and uncertainty*, volume 946 de *Lecture Notes in Comput. Sci.*, pp. 151–161. Springer, Berlin.

[33] Merle, O.du, Villeneuve, D., Desrosiers, J. et Hansen, P. (1999). Stabilized column generation. *Discrete Math.*, 194(1-3) : 229–237.

[34] Fichet, B. (1987). The role played by L_1 in data analysis. Dans Dodge, Y., éditeur, *Statistical data analysis based on the L_1-norm and related methods (Neuchâtel, 1987)*, pp. 185–193. North-Holland, Amsterdam.

[35] Fichet, B. (1994). Dimensionality problems in L_1-norm representations. Dans Van Custem, B., éditeur, *Classification and dissimilarity analysis*, Lecture Notes in Statistics, pp. 201–224. Springer, New York.

[36] Fichet, B. et Calvé, G. Le. (1984). Structure géométrique des principaux indices de dissimilarité sur signe de présence-absence. *Statistiques et Analyse de données*, 9(3) : 11–44.

[37] Frisch, A.M. et Haddawy, P. (1994). Anytime deduction for probabilistic logic. *Artificial Intelligence*, 69(1-2) : 93–122.

[38] Fung, G. et Mangasarian, O.L. (2001). Proximal support vector machine classifiers. Dans *Knowledge Discovery and Data Mining*, pp. 77–86.

[39] Garey, M.R. et Johnson, D.S. (1979). *Computers and intractability*. W. H. Freeman and Co., San Francisco, Calif. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[40] Georgakopoulos, G., Kavvadias, D. et Papadimitriou, C.H. (1988). Probabilistic Satisfiability. *Journal of Complexity*, 4 : 1–11.

[41] Georgakopoulos, G., Kavvadias, D. et Papadimitriou, C.H. (1988). Probabilistic satisfiability. *J. Complexity*, 4(1) : 1–11.

[42] Glover, F., Kochenberger, G.A. et Alidaee, B. (1998). Adaptive memory tabu search for binary quadratic programs. *Management Science*, 44(3) : 336–345.

[43] Glover, F. et Laguna, M. (1997). *Tabu Search*. Dordrecht, Kluwer.

[44] Hailperin, T. (1965). Best possible inequalities for the probability of a logical function of events. *Amer. Math. Monthly*, 72 : 343–359.

[45] Hailperin, T. (1976). *Boole's logic and probability*. North-Holland Publishing Co., Amsterdam. A critical exposition from the standpoint of contemporary algebra, logic and probability theory, Studies in Logic and the Foundations of Mathematics, Vol. 85.

[46] Hailperin, T. (1986). *Boole's logic and probability*, volume 85 de *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam.

[47] Hammer, P. L., Hansen, P. et Simeone, B. (1984). Roof duality, complementation and persistency in quadratic 0-1 optimization. *Math. Programming*, 28(2) : 121–155.

[48] Hammer, P.L. et Rudeanu, S. (1966). Boolean Methods in Operations Research and Related Areas. Berlin : Springer.

[49] Hansen, P., Brimberg, J., Urosevic, D. et Mladenovic, N. (2003). Primal-Dual Variable Neighborhood Search for Bounded Heuristic and Exact Solution of the Simple Plant Location Problem. *Les Cahiers du GERAD*, G-2003-64.

[50] Hansen, P. et Jaumard, B. (1992). Reduction of indefinite quadratic programs to bilinear programs. *J. Global Optim.*, 2(1) : 41–60.

[51] Hansen, P. et Jaumard, B. (2000). Probabilistic satisfiability. Dans *Algorithms for uncertainty and defeasible reasoning*, volume 5 de *Handb. Defeasible Reason. Uncertain. Manag. Syst.*, pp. 321–367. Kluwer Acad. Publ., Dordrecht.

[52] Hansen, P., Jaumard, B. et Meyer, C. (2000). A simple enumerative algorithm for unconstrained 0–1 quadratic programming. *Les Cahiers du GERAD*, G-2000-59.

[53] Hansen, P., Jaumard, B. et Aragão, M.Poggi de. (1995). Boole's conditions of possible experience and reasoning under uncertainty. *Discrete Appl. Math.*, 60(1-3) : 181–193. ARIDAM VI and VII (New Brunswick, NJ, 1991/1992).

[54] Hansen, P., Jaumard, B. et Savard, G. (1992). New branch-and-bound rules for linear bilevel programming. *SIAM J. Sci. Statist. Comput.*, 13(5) : 1194–1217.

[55] Hansen, P., Jaumard, B. et Silva, E. Da. (1991). Average-linkage divisive hierarchical clustering. *Les Cahiers du GERAD*, G-91-55.

[56] Hansen, P. et Mladenović, N. (2001). Variable neighbourhood search : principles and applications. *European Journal of Operational Research*, 130 : 449–467.

[57] Hasan, M., Alkhamis, T. et Ali, J. (2000). A comparison between simulated annealing, genetic algorithm and tabu search methods for the unconstrained quadratic pseudo-boolean function. *Computers and Industrial Engineering*, 38(3) : 323–340.

[58] Jaumard, B., Hansen, H. et Aragão, M.Poggi de. (1991). Column Generation Methods for Probabilistic Logic. *ORSA Journal on Computing*, 3 : 135–148.

[59] Jaumard, B., Lemaire, S. et Parreira, A.D. (2000). A Deductive Approach for Probabilistic Logic Problems. *AMAI*.

[60] Joly, S. et Le Calvé, G. (1994). Similarity functions. Dans Van Custem, B., éditeur, *Classification and dissimilarity analysis*, Lecture Notes in Statistics, pp. 67–86. Springer, New York.

[61] Kane, T.B. (1990). Enhancing the Inference Mechanism of Nilsson's Probabilistic Logic. *International Journal of Intelligent Systems*, 5(5) : 487–504.

[62] Kane, T.B. (1992). *Reasoning with Uncertainty Using Nilsson's Probabilistic Logic and the Maximum Entropy Formalism*. Thèse de doctorat, Heriot-Watt University, Edinburgh.

[63] Kantorovich, L.V. (1939). *Mathematical methods in the organization and planing of production*. Engligh translation : *Management Science* 6 : 366-422, 1960.

[64] Karp, R.M. et Papadimitriou, C.H. (1982). On linear characterizations of combinatorial optimization problems. *SIAM J. Comput.*, 11(4) : 620–632.

[65] Kavvadias, D. et Papadimitriou, C.H. (1990). A Linear Programming Approach to Reasoning about Probabilities. *Annals of Mathematics and Artificial Intelligence*, 1 : 189–205.

[66] Kavvadias, D. et Papadimitriou, C.H. (1990). A Linear Programming Approach to Reasoning about Probabilities. *Annals of Mathematics and Artificial Intelligence*, 1 : 189–205.

[67] Larman, D. G. et Tamvakis, N. K. (1984). The decomposition of the n-sphere and the boundaries of plane convex domains. Dans *Convexity and graph theory (Jerusalem, 1981)*, volume 87 de *North-Holland Math. Stud.*, pp. 209–214. North-Holland, Amsterdam.

[68] Lodi, A., Allemand, K. et Liebling, T.M. (1999). An evolutionary heuristic for quadratic 0-1 programming. *European Journal of Operational Research*, 119(3) : 662–670.

[69] Lübbecke, M.E. et Desrosiers, J. . (2002). Selected Topics in Column Generation. *Les Cahiers du GERAD*, G-2002-64. Soumis à Operations Research.

[70] Lukasiewicz, T. (1999). Local probabilistic deduction from taxonomic and probabilistic knowledge-bases over conjunctive events. *Internat. J. Approx. Reason.*, 21(1) : 23–61.

[71] Mangasarian, O.L. (1999). Arbitrary-norm Separating Plane. *Operations Research Letters*, 24(1–2) : 15–23.

[72] Mangasarian, O.L. et Musicant, D.R. (2000). Active Support Vector Machine Classification. Dans *NIPS*, pp. 577–583.

[73] Marcotte, P., Marquis, G. et Savard, G. (1995). A New Implicit Enumeration Scheme for the Discriminant Analysis Problem. *Computers and Operations Research*, 26(6) : 625–639.

[74] Marcotte, P. et Savard, G. (1992). Novel approaches to the discrimination problem. *Z. Oper. Res.*, 36(6) : 517–545.

[75] Mladenović, N. et Hansen, P. (1997). Variable neighbourhood search. *Computers and Operations Research*, 24 : 1097–1100.

[76] Musicant, D.R. (1998). NDC : Normally Distributed Clustered Datasets.

[77] Nilsson, N.J. (1986). Probabilistic logic. *Artificial Intelligence*, 28(1) : 71–87.

[78] Nilsson, N.J. (1993). Probabilistic logic revisited. *Artificial Intelligence*, 59(1-2) : 39–42.

[79] Palubeckis, G. (1995). A heuristic-based branch and bound algorithm for unconstrained quadratic zero-one programming. *Computing*, 54(4) : 283–301.

[80] Parreira, A.D. (2002). *Raisonnement sous incertitude - logique probabiliste et variantes*. Thèse de doctorat, École Polytechnique de Montréal.

[81] Reinhardt, K. (1922). Extremale Polygone gegebenen Durchmessers. *Jahresber. Deutsch. Math. Verein*, 31 : 251–270.

[82] Sherali, H.D. et Alameddine, A. (1992). A new reformulation-linearization technique for bilinear programming problems. *J. Global Optim.*, 2(4) : 379–410.

[83] Sherali, H.D. et Tuncbilek, C.H. (1992). A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique. *J. Global Optim.*, 2(1) : 101–112.

[84] Stam, A. (1997). Nontraditional approaches to statistical classification : Some perspectives on L-p-norm methods. *Annals of Oper. Res.*, 74 : 1–36.

[85] Vincze, S. (1950). On a geometrical extremum problem. *Acta Sci. Math. Szeged*, 12(Leopoldo Fejer et Frederico Riesz LXX annos natis dedicatus, Pars A) : 136–142.

[86] Walley, P. (1991). *Statistical reasoning with imprecise probabilities*, volume 42 de *Monographs on Statistics and Applied Probability*. Chapman and Hall Ltd., London.

[87] Zemel, E. (1982). Polynomial Algorithms for Estimating Network Reliability. *Networks*, 12 : 439–452.

[88] Zhou, X., Wang, Y., Tian, X. et Guo, R. (1997). A tabu search algorithm for quadratic 0-1 programming problem. *Chinese Quarterly Journal of Mathematics*, 12(4) : 98–102.

[89] Ziegler, G. M. (2000). Lectures on 0/1-polytopes. Dans *Polytopes—combinatorics and computation (Oberwolfach, 1997)*, volume 29 de *DMV Sem.*, pp. 1–41. Birkhäuser, Basel.

ANNEXE A : DÉTAILS DES ENSEMBLE DE DONNÉES UTILISÉES POUR LE PROBLÈME D'HYPERPLAN SÉPARATEUR EN NORME-ℓ_2

Notons tout d'abord que toutes les données ont été linéairement standardisées dans l'intervalle $[0, 1]$.

Pour les problèmes provenant de la base de données *UCI Machine Learning Repository* [21], nous avons considéré les problèmes contenant seulement deux classes ou pouvant aisément être convertis en des problèmes de ce type. Nous avons alors retenu les problèmes avec aucune ou très peu de variables de catégorie. Voici quelques informations supplémentaires sur chacun de ces problèmes :

- *Cancer* se réfère à la base de données *Wisconsin Breast Cancer database*. Les lignes avec des attributs manquants ont été enlevées.
- *Diabetes* se réfère à la base de données *Pima Indians Diabetes database*.
- Pour le problème *Echocardiogram*, toutes les observations dont la classe était inconnue ont été enlevées, et les attributs manquants ont été remplacés par la moyenne de la classe correspondante.
- Pour la base de données *Glass*, les deux classes considérées sont les fenêtres et les autres types de verre.
- *Housing* se réfère la base de données *Boston Housing database*.
- Pour la base de données *Hepatitis*, toutes les observations avec plus de 6 attributs manquants ont été éliminées. Les colonnes 16 et 18 ont été enlevées en raison de l'absence trop importante d'entrées. Les observations manquantes ont été remplacées par des valeurs moyennes (dans le cas continu) ou, sinon, des valeurs modales. La colonne 3 qui sépare de façon triviale les deux ensembles, a aussi été enlevée.

Le générateur NDC de Musicant est un programme fonctionnant sous MATLAB®. Il localise aléatoirement un nombre donné de centres, il assigne ensuite ceux-ci à une des deux classes en séparant l'ensemble des centres avec un plan généré aléatoirement et finalement, il distribue des observations autour de ces centres à l'aide d'une loi normale multi-variée. Cette approche procure une généralité plus considérable que ce qui est obtenu avec les techniques habituelles (par exemple, en fixant un nombre arbitraire de centres pour ensuite générer des colonnes indépendantes). Cependant, certaines configurations (par exemple, celle où un petit groupe d'observations est telle que son centre se trouve du mauvais côté du plan) ne peuvent être obtenues par ce générateur. Nous avons néanmoins opté pour ce générateur pour sa généralité et sa disponibilité. Le fait que le programme soit disponible publiquement facilite la reproduction des mêmes ensembles de données pour des études futures. Pour les instances résolues dans ce texte, le nombre de centres est égal à la dimension et le paramètre de dispersion $nExpandFactor$ du générateur est fixé à 15.