Optimisation globale avec LocalSolver

Simon Boulmier

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : Mathématiques Appliquées

Arrêté ministériel : 25 mai 2016

Présentée par

Simon BOULMIER

Thèse dirigée par **Jerome MALICK**
et codirigée par **Julien DARLAY**, LocalSolver

préparée au sein du **Laboratoire Jean Kuntzmann** dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Optimisation globale avec LocalSolver

Global optimization with LocalSolver

Thèse soutenue publiquement le **18 septembre 2020**,
devant le jury composé de :

Monsieur JEROME MALICK
DIRECTEUR DE RECHERCHE, CNRS DELEGATION ALPES, Directeur de thèse
Monsieur LEO LIBERTI
DIRECTEUR DE RECHERCHE, CNRS ILE-DE-FRANCE GIF-SUR-YVETTE, Rapporteur
Monsieur SERGE GRATTON
PROFESSEUR DES UNIVERSITES, INP-ENSIACET - TOULOUSE, Rapporteur
Madame SONIA CAFIERI
PROFESSEUR DES UNIVERSITES, ENAC - TOULOUSE, Examinatrice
Madame NADIA BRAUNER
PROFESSEUR DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES, Présidente
Monsieur JULIEN DARLAY
DOCTEURI-NGENIEUR, SOCIETE LOCALSOLVER - PARIS, Examinateur

Résumé

LocalSolver est un logiciel de programmation mathématique. Originellement pensé pour traiter les grands problèmes d'optimisation combinatoire rencontrés dans l'industrie, son fonctionnement repose sur des heuristiques de recherche locale. Cette approche de résolution pragmatique, couplée à des structures de modélisation expressives, non linéaires et ensemblistes, lui ont permis de s'imposer dans le catalogue des solveurs commerciaux.

L'objet de cette thèse est le développement d'une approche duale, complémentaire à la recherche locale, qui fournira des bornes aux problèmes traités. L'intérêt principal est de qualifier la qualité des solutions retournées, voire de prouver leur optimalité, permettant ainsi d'interrompre plus rapidement la résolution. Ce n'est cependant pas le seul, puisque les techniques nécessaires au calcul de bornes permettent par exemple de prouver l'inconsistance d'un problème. Cette fonctionnalité est utile en phase de développement, où des erreurs de modélisation ou de données sont fréquentes. Trois difficultés principales se présentent alors. D'abord, les problèmes traités sont génériques, et peuvent être combinatoires, non linéaires ou encore non différentiables. Ensuite, l'intégration à un logiciel industriel impose un haut niveau de fiabilité et de qualité logicielle, ainsi que la capacité à passer à l'échelle en temps et en mémoire. Enfin, tous les besoins de reformulation doivent être pris en compte en interne, afin de permettre aux utilisateurs de LocalSolver de modéliser leurs problèmes le plus naturellement possible.

Ainsi, le module dual implémenté au sein de LocalSolver commence par transformer le problème d'optimisation fourni en un programme non linéaire en variables mixtes (MINLP). Ce programme est représenté sous une forme standard facilitant l'implémentation de divers outils utiles au calcul de bornes : génération de relaxations convexes, techniques de réduction de bornes ou encore actions de *presolve*. Ces outils sont ensuite intégrés dans une recherche arborescente de type *branch-and-reduce*, qui interagit avec les autres modules de LocalSolver grâce à des techniques de programmation concurrente.

Si l'approche décrite ci-dessus est classique, plusieurs spécificités et choix d'implémentation se différentient de l'état de l'art. En effet, les opérateurs mathématiques supportés et la technique de reformulation utilisée permettent de calculer des bornes sur plus de problèmes que les solveurs d'optimisation globale de référence. Ensuite, ces solveurs exploitent principalement des relaxations linéaires, alors que l'un de nos objectifs est de montrer que des relaxations non linéaires peuvent être compétitives. Dans cette optique, nous avons implémenté un solveur non linéaire sur-mesure, dédié au calcul de bornes inférieures d'un problème convexe, et adapté aux relaxations non linéaires utilisées. Enfin, un résultat de dualité sous contraintes de bornes est obtenu. Celui-ci permet d'améliorer la performance du solveur non linéaire et d'y inclure une méthode robuste de détection de l'inconsistance, mais aussi de garantir la fiabilité des bornes inférieures calculées par LocalSolver.

Abstract

LocalSolver is a mathematical programming solver. Originally designed to solve large scale combinatorial optimization problems such as those found in the industry, it mainly relies on local search heuristics. This pragmatic solution approach, coupled with expressive nonlinear and set-based modeling techniques, has allowed LocalSolver to establish itself as a successful commercial solver.

The purpose of this thesis is to implement a complementary dual approach for the computation of lower bounds within LocalSolver. The main stake is to qualify the solutions found by the solver and to potentially prove their optimality, thus allowing an early stop of the search. Furthermore, lower bounds have many other applications, such as the detection of inconsistent problems. This is useful in the development phase where modeling errors are frequent. We face three major challenges. First, the problems we address are generic and can be combinatorial, nonlinear or even nonsmooth. Then, the integration to an industrial software requires to produce reliable and high-quality code that can scale in time and memory. Finally, any reformulation need must be managed in-house, to allow LocalSolver's users to model their problems in the easiest way possible.

The dual module provided to LocalSolver starts by reformulating the given optimization problem into a mixed-integer nonlinear program (MINLP). This program is stored under a standard form that facilitates the implementation of various techniques aiming at computing lower bounds. Examples of such techniques are the generation of convex relaxations, bound tightening techniques and presolve actions. These building blocks are then integrated into a partitioning scheme called the branch-and-reduce algorithm, and interact with the primal modules thanks to concurrent computing techniques.

While this approach remains traditional, several choices and implementation features vary from the state of the art. The operators we support and the reformulation technique we use allow us to compute lower bounds on more problems than most global optimization solvers. These solvers also mainly use linear relaxations, whereas our goal is to show that nonlinear relaxations can be competitive. For this purpose, we implement a nonlinear solver dedicated to the computation of lower bounds to our convex relaxations. At last, we establish a duality result under bound constraints that allow us to improve the performance of our custom nonlinear solver. It is also exploited to certify the validity of the lower bounds computed by LocalSolver and to obtain robust inconsistency certificates.

Table des matières

Chapitre 1

Introduction

Les travaux de recherche menés lors de cette thèse s'articulent autour des deux dimensions distinctes du sujet. La première propose d'exploiter des relaxations non linéaires pour obtenir des bornes inférieures à des problèmes d'optimisation. Il s'agit d'un axe purement technique, qui invite à utiliser un outil particulier plutôt qu'un autre, et qui est à rapprocher d'un travail de recherche académique. La seconde, qui découle de la nature CIFRE de la thèse, demande de calculer des bornes inférieures aux problèmes traités par LocalSolver. Répondre à cette problématique industrielle suppose alors une connaissance des besoins de l'entreprise, et nécessite un travail d'implémentation adapté à un logiciel vendu à de nombreux clients à travers le monde.

Pour cette raison, ce premier chapitre constitue une introduction conséquente au contexte scientifique et industriel dans lequel ces travaux ont été effectués. En particulier, on commence par décrire la dualité théorie-pratique, qui a conduit à l'industrialisation de la recherche locale et à la genèse de LocalSolver, et qui est au cœur de la problématique de cette thèse. Après une description rapide du logiciel, on montre que des techniques de calcul de bornes inférieures - et plus généralement d'optimisation globale - sont complémentaires à la recherche locale, notamment par leurs forces et faiblesses respectives. Enfin, une présentation détaillée du contenu de la thèse est donnée, suivie d'une description de la philosophie soutenant le choix et l'implémentation des outils mathématiques utilisés.

Plan du chapitre

1.1 Contexte industriel et scientifique

1.1.1 Programmation mathématique

La programmation mathématique est un domaine riche, dans la théorie comme dans la pratique. Nous commençons par préciser quelques notions importantes pour situer le contexte scientifique et industriel de cette thèse.

Recherche opérationnelle

La recherche opérationnelle, souvent dénommée recherche opérationnelle et aide à la décision [13], est la discipline qui s'intéresse à la modélisation et à la résolution des problèmes rencontrés dans diverses branches de l'industrie. Un ensemble de décisions à prendre est représenté par des variables, dont les domaines de définition correspondent à la nature de la décision : $x \in \mathbb{R}$ pour une quantité, $x \in \mathbb{N}$ pour un nombre, ou encore $x \in \{0, 1\}$ pour un choix binaire. On cherche ensuite à assigner des valeurs aux décisions, afin de maximiser ou minimiser certains critères, tout en respectant un ensemble de contraintes.

La programmation mathématique est une discipline qui s'intéresse à la traduction de ces problèmes d'optimisation dans le langage des mathématiques. Les objectifs et contraintes sont mis sous forme d'équations, dont les variables sont les décisions à déterminer. Le problème d'optimisation ainsi écrit devient alors un programme mathématique. Le contexte métier, ainsi que l'interprétation des variables et contraintes est perdue, et seule la géométrie induite par les équations et objectifs reste, *via* leur forme algébrique.

Les outils permettant résoudre les problèmes d'optimisation sont donc des logiciels qui possèdent deux composantes principales. Premièrement, un formalisme de modélisation, souvent composé d'un langage informatique et d'un ensemble de règles mathématiques, permet de traduire le problème d'optimisation en un programme mathématique. Ensuite, le solveur d'optimisation est composé d'algorithmes mathématiques permettant de trouver des solutions à ces programmes. Généralement, un solveur se limite à une classe de programmes mathématiques donnée, définie par

LP	Programme linéaire (*linear program*) : variables dans $\mathbb{R}^n$, objectif et contraintes linéaires
QP	Programme quadratique (*quadratic program*) : variables dans $\mathbb{R}^n$, objectif quadratique et contraintes linéaires
QCQP	Programme quadratique quadratiquement contraint (*quadratically constrained quadratic program*) : variables dans $\mathbb{R}^n$, objectif et contraintes quadratiques
SOCP	Programme conique du second ordre (*second ordre cone program*) : variables dans $\mathbb{R}^n$, objectif et contraintes linéaires, contraintes coniques du second ordre
SDP	Programme semidéfini (*semi-definite program*) : variables dans $\mathbb{R}^n$, objectif et contraintes linéaires, contrainte de positivité semi-définie $X \succcurlyeq 0$
NLP	Programme non linéaire (*nonlinear program*) : variables dans $\mathbb{R}^n$, objectifs et contraintes non linéaires, contraintes définies à l'aide des comparaisons usuelles $=$ et $\leq$
MI-X	Programme de la classe X comportant de plus des contraintes d'intégrité (*mixed integer X*)

Tableau 1.1 – *Quelques exemples de classes de programmes mathématiques*

l'expressivité de son langage de modélisation et, surtout, par les méthodes de résolution sous-jacentes. Lorsqu'un programme mathématique n'est pas pris en compte par ce solveur, la modélisation du problème doit être revue ou approximée, afin de rentrer dans son paradigme d'optimisation.

Classes de problèmes

Les classes de problème de la programmation mathématique, telles qu'elles sont définies dans la littérature, sont nombreuses et variées. Quelques-unes de ces classes sont listées dans le tableau 1.1. Elles permettent d'étudier de façon spécifique des problèmes qui n'ont pas la même nature géométrique. En effet, selon le type des variables, la forme des contraintes et les propriétés mathématiques des fonctions présentes dans le modèle, les difficultés et méthodes de résolution du problème sont différentes.

De nombreux problèmes sont aujourd'hui traitées de façon satisfaisante, à la fois du point de vue théorique et du point de vue numérique. Les problèmes industriels modernes sont cependant souvent complexes et ne rentrent dans une des classes bien traitées que par des astuces de modélisation. Une conséquence est que l'utilisation de certains solveurs impose un exercice de modélisation difficile, parfois numériquement instable, souvent artificiel. Approximer un problème complexe dans un formalisme plus simple est cependant généralement possible, au prix d'un accès à une plus grande puissance de calcul que pour une approche dédiée. Ainsi, la résolution de problèmes industriels est conditionnée par l'existence de solveurs suffisamment expressifs, ou par la dextérité des ingénieurs à jongler entre les paradigmes de modélisation.

Théorie, pratique et recherche locale

Si un grand nombre de problèmes d'optimisation, y compris combinatoires, possèdent des méthodes de résolution dédiées et efficaces, ces problèmes ont le plus souvent une structure spécifique. On peut citer par exemple les problèmes de flot, de *matching* ou encore les programmes linéaires en nombres entiers ayant une matrice totalement unimodulaire. Dans le cas général, quand la structure du problème est hybride, voire inconnue, la situation est différente. Le consensus actuel sur les problèmes que l'on devrait pouvoir convenablement traiter porte sur la convexité. La révolution apportée par les méthodes de points intérieurs a montré que même dans des cas fortement non linéaires, les problèmes convexes peuvent être résolus. À l'inverse, les problèmes non convexes, en particulier les problèmes combinatoires, ne sont résolus que par l'utilisation d'une forme de recherche exhaustive dans un certain espace. Ces approches peuvent être efficaces, en particulier quand les propriétés des problèmes résolus sont favorables. Leur généricité doit cependant beaucoup aux avancées matérielles et à plusieurs décennies d'ingénierie logicielle, l'exemple le plus criant étant la programmation linéaire en nombres entiers.

La plupart des méthodes génériques de résolution des problèmes d'optimisation sont développées à partir de résultats théoriques, géométriques ou algébriques, sur les objets mathématiques présents dans les programmes correspondants. On peut citer par exemple l'étude des polyèdres pour l'algorithme du simplexe ou encore l'analyse des barrières auto-concordantes pour les méthodes de points intérieurs. D'autres méthodes, comme celles de recherche directe ou les méta-heuristiques de type évolutionnaire, exploitent plutôt des propriétés statistiques et probabilistes, faisant bon usage de la parallélisation massive des cœurs de calcul. Enfin, une dernière famille est composée des approches de type programmation par contraintes. Celles-ci attaquent frontalement la recherche exhaustive et accélèrent l'énumération en exploitant la structure des contraintes du problème ainsi que des outils algorithmiques avancés.

L'approche traditionnelle, suivie par les grands éditeurs de solveurs, tels que IBM (CPLEX) ou FICO (Xpress), est de développer et d'utiliser des méthodes traitant des classes de problèmes de plus en plus complexes (typiquement en suivant le chemin LP, QP convexe, QCQP convexe, SOCP, ainsi que leurs versions mixtes). Ces méthodes ont en commun d'être complètes, c'est-à-dire que l'on a la garantie de trouver la meilleure solution en temps fini. Lorsqu'un problème d'optimisation ne peut être directement résolu par l'une d'entre elles, les ingénieurs peuvent les utiliser en tant que briques de base pour construire des approches plus spécifiques, comme par exemple dans des approches par décomposition ou par génération de colonnes. Ceci apporte cependant une technicité supplémentaire dans les outils développés qui, une fois en production, sont plus difficiles à maintenir et à faire évoluer. Couplé à la fin de la loi de Moore et au fait que les problèmes industriels deviennent de plus en plus complexes et de grande échelle, ceci fait que les heuristiques et les méthodes de résolution approchées sont de plus en plus présentes dans le monde de l'optimisation.

La recherche locale est une approche radicalement différente, fondée sur l'idée que les problèmes d'optimisation se posant dans l'industrie ont souvent des points communs structurels. Généralement, un tel problème correspond à un processus opé-

rationnel actif, comme par exemple le fonctionnement d'une chaîne logistique. Dans ce cas, des solutions au problème sont déjà connues et utilisées, et la question est de savoir si l'on peut faire mieux. En particulier, ceci implique que ces problèmes sont la plupart du temps consistants (ils admettent des solutions) et bien conditionnés (puisque pour des questions de robustesse et de stabilité des processus opérationnels, une petite variation des décisions a souvent un petit impact sur le coût). L'autre remarque importante est qu'étant donnée une solution de mauvaise qualité, un individu ayant une connaissance parfaite du problème métier et de ses subtilités sera souvent capable de proposer des améliorations à y apporter. Ceci vient du fait que les problèmes industriels combinatoires ont souvent de nombreuses solutions. La recherche locale se construit alors sur le principe qu'une solution est soit satisfaisante pour le problème à résoudre, soit elle peut être localement améliorée, c'est-à-dire que l'on doit pouvoir l'améliorer en modifiant peu de décisions. L'idée est alors d'essayer de nombreuses modifications locales de la solution, aléatoires ou basées sur la structure du problème, et de ne conserver que celles qui améliorent l'objectif.

1.1.2 Introduction à LocalSolver

Pour trouver une meilleure solution au problème donné, la recherche locale fonctionne comme un humain : elle essaie de modifier légèrement la solution courante, et ne garde le changement que si celui-ci améliore le critère optimisé. L'efficacité de l'approche réside donc dans la façon dont la solution est modifiée. Une telle modification est ce que l'on appelle un mouvement, qui sera ensuite conservé ou annulé. Dans l'approche traditionnelle de recherche locale, c'est-à-dire dans une recherche locale dédiée à un problème particulier [97, 74], les mouvements sont basés sur la structure métier du problème. Par exemple un mouvement peut consister à interchanger deux robots sur une chaîne de production, ou à assigner une certaine tâche à un autre agent.

Le tour de force qui a fait la réussite de LocalSolver est l'industrialisation du principe de recherche locale et des mouvements associés. On parle alors de recherche locale générique. Les mouvements génériques sont uniquement basés sur la structure mathématique du problème, ce qui permet de s'extraire du contexte métier et donc de ne pas avoir à utiliser de mouvements spécifiques, différents pour chaque problème d'optimisation.

Nous continuons cette section en présentant les trois outils algorithmiques et mathématiques principaux qui composent LocalSolver, et qui permettent d'effectuer une recherche locale sur un problème quelconque. Pour plus d'informations sur l'histoire de LocalSolver, et sur l'ingénierie logicielle impliquée dans son développement, voir [79, 39, 41].

Graphe d'évaluation

Quand les objectifs et les contraintes du problème à résoudre ne s'expriment pas sous une forme simple (typiquement, linéaire ou quadratique), les équations qui les définissent sont le plus souvent représentées par un graphe d'évaluation. Il s'agit d'un graphe acyclique orienté (*directed acyclic graph*, abrévié par DAG dans la suite), dont

un exemple est donné dans la figure 1.1. Le DAG est composé de plusieurs feuilles, qui représentent les variables et constantes, et de plusieurs racines, qui représentent la fonction objectif et les contraintes. Un parcours en largeur des feuilles vers les racines permet d'évaluer l'objectif et les contraintes.

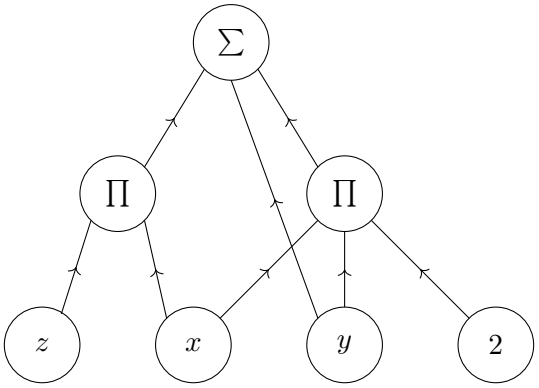

FIGURE 1.1 – *Un DAG d'évaluation de* $(x, y, z) \mapsto 2xy + xz + y$.

La représentation d'un problème sous la forme d'un DAG est souvent utilisée en programmation mathématique, car elle possède de nombreux avantages. On peut par exemple déterminer la régularité, la différentiabilité ou encore la séparabilité du problème en analysant la structure du graphe. Ce dernier permet aussi de calculer des gradients, jacobiens et hessiennes par différentiation automatique, avec une performance parfois supérieure à celle d'une différentiation manuelle effectuée par un expert. Pour plus d'informations sur l'utilisation des DAG en programmation mathématique, voir par exemple [80].

Un point clé de la recherche locale est le fait que les modifications effectuées sur une solution, les mouvements, concernent peu de variables (c'est de là que vient le qualificatif local). Ceci a motivé l'utilisation d'une structure d'évaluation incrémentale de l'objectif et des contraintes, *via* l'exploitation du DAG. L'intérêt d'une telle structure est par exemple mis en évidence lorsque l'on veut évaluer la fonction quadratique $f : \mathbb{R}^n \to \mathbb{R}$, définie par

$$f(x_1, ..., x_n) = \sum_{i,j} q_{ij} x_i x_j.$$

Dans le pire cas, soit quand la matrice (q_{ij}) est dense, la complexité d'évaluation de f est $O(n^2)$, mais si une seule des variables x_i change, on peut mettre à jour la valeur de f en au plus $O(n)$ opérations, en ne visitant que les nœuds du graphe impactés par le changement. On peut de plus interrompre la propagation si un nœud ne varie pas, par exemple en présence d'opérateurs n-aires de type min ou max, ou encore savoir si de nouvelles contraintes seront violées avant la fin de la propagation. Toute recherche locale performante exploite une mise à jour incrémentale des objectifs et contraintes quand seules quelques variables changent, et ce principe est ici systématisé et généralisé, ce qui évite de développer un évaluateur incrémental pour chaque problème d'optimisation.

Mouvements

À chaque type de variable (entiers, booléens, réels, etc.) est associé un ensemble de mouvements, qui peuvent modifier les variables du type en question. Les mouvements de LocalSolver ont été développés pour les problèmes combinatoires binaires, puis généralisés pour prendre en compte d'autres types de variables.

Initialement, outre les déplacements aléatoires, la plupart des mouvements sont basés sur des chaînes d'éjection dans l'hypergraphe des contraintes [79, 39]. Cette approche se généralise cependant mal au cas des variables continues, et seuls certains mouvements combinatoires ont été adaptés pour les prendre en compte. Une étape de pré-traitement permet de détecter des structures particulières dans le DAG, telles que les contraintes de couverture, de partitionnement ou de sac-à-dos. Des mouvements se basant sur ces structures sont aussi présents.

On peut distinguer deux types de mouvements opérant sur les variables continues : les mouvements libres, qui sélectionnent un sous-ensemble des variables et qui les modifient de façon aléatoire, et les mouvements basés sur une contrainte. Ces derniers ciblent une contrainte et effectuent un déplacement aléatoire de la solution, qui conserve ou restore la faisabilité de cette contrainte.

Une autre spécificité de LocalSolver est la stratégie d'exploration des voisinages, dite de recherche randomisée. Contrairement à l'approche traditionnelle, ceux-ci ne sont pas explorés de façon exhaustive : seul quelques éléments, choisis de façon aléatoire ou par ciblage, sont évalués. Les fonctionnalités de ciblage en question utilisent la structure du DAG, les sens de variation des nœuds ou encore les contraintes violées pour augmenter la probabilité que les voisins sélectionnés améliorent la solution.

Heuristiques

Dans une recherche locale à voisinages variables, on trouve deux heuristiques principales. La première s'appelle l'heuristique de sélection, et détermine, à chaque itération, le voisinage à explorer. Dans LocalSolver, cette heuristique utilise des méthodes d'apprentissage statistique. L'historique des résultats des mouvements lors des itérations précédentes permet de sélectionner le candidat le plus prometteur. Cette heuristique assure aussi la diversité des mouvements appliqués, en s'assurant que chacun d'entre eux est régulièrement choisi.

L'autre heuristique importante est celle qui, étant donné un mouvement qui vient d'être effectué sur la solution courante, décide de le conserver ou de l'annuler. On l'appelle heuristique d'acceptation. L'amélioration ou la détérioration de la solution dépend de deux critères : premièrement d'une mesure de faisabilité (nulle sur le domaine admissible et qui augmente avec les violations des contraintes), et ensuite de la valeur de l'objectif.

Le processus de recherche locale est un algorithme itératif, cherchant à améliorer la solution courante par une petite modification. Une itération de la recherche locale utilise les trois concepts que nous venons de présenter, et est schématisée dans la figure 1.2.

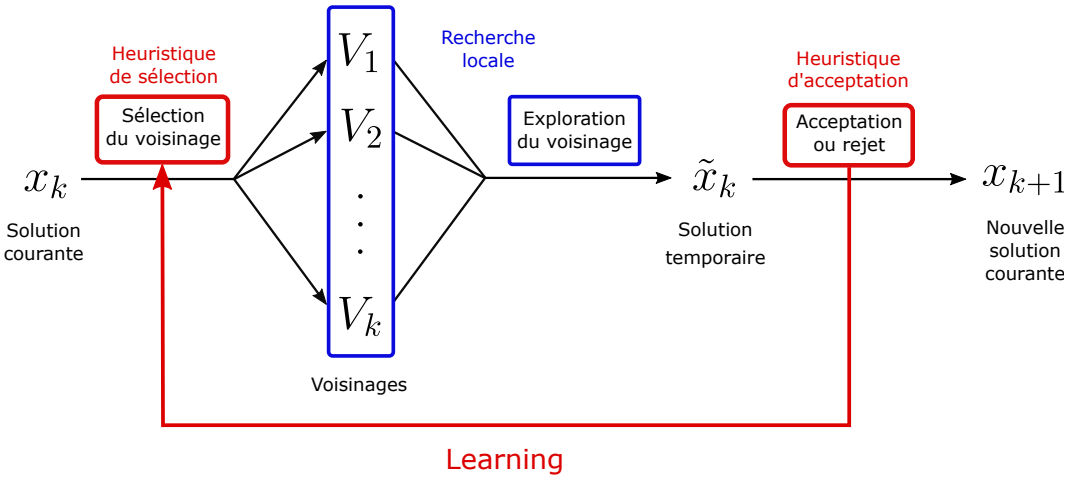

FIGURE 1.2 – *Fonctionnement schématique d'une itération de recherche locale dans LocalSolver.*

Ici, on a séparé les étapes de sélection et d'exploration des voisinages, pour plus de clarté. Dans LocalSolver, ces deux notions sont contenues dans ce que l'on appelle un mouvement, soit un opérateur qui a une solution courante associe une solution temporaire.

Forces et limites de l'approche

Plusieurs éléments ont participé à la réussite de la recherche locale sur le marché de l'optimisation. Premièrement, la recherche par voisinages variables, couplée à l'évaluation incrémentale que permet le DAG, est moins sensible à la taille du problème que beaucoup d'autres méthodes de la programmation mathématique. Ceci prend encore plus d'importance quand on sait que certaines méthodes ne trouvent les premières solutions admissibles qu'à la fin de la recherche, comme par exemple l'algorithme du simplexe dual. Ensuite, la généricité de la recherche locale permet à LocalSolver de proposer un paradigme de modélisation particulièrement expressif. En particulier, celui-ci fournit des variables représentant des collections de valeurs (ensembles, listes, tableaux), et non pas uniquement des valeurs simples (entier, flottant, booléen). Ces variables, ainsi que les opérateurs ensemblistes associés, permettent de modéliser de façon concise des objets combinatoires de haut niveau. De plus, la structure des problèmes d'optimisation ainsi écrits est mieux conservée lors de la modélisation et de l'écriture du modèle sous forme de graphe. Les efforts et les délais nécessaires au développement et à la maintenance d'une application industrielle basée sur LocalSolver s'en trouvent considérablement réduits. Enfin, le caractère heuristique et l'absence de garantie d'optimalité de la recherche locale ne sont généralement pas un problème dans un contexte industriel. Si d'autres méthodes de résolution possèdent des garanties de convergence plus fortes, les conditions de réalisation de ces garanties (en temps et en mémoire) n'ont pas toujours d'intérêt pratique. De plus, une caractéristique de nombreux problèmes industriels est que la

résolution à l'optimum n'est pas un prérequis. Cette remarque, en apparence contre-intuitive, prend tout son sens en s'intéressant à quelques retours d'expérience de projets industriels d'optimisation. D'abord, le temps de résolution d'un problème est alloué par les utilisateurs finaux, et presque toujours limité à quelques minutes, quelques heures. Ensuite, un modèle mathématique n'est qu'une approximation du problème réel, et les données fournies à celui-ci sont parfois erronées - voire inexactes - et souvent incertaines. Enfin, les solutions produites par un solveur d'optimisation sont régulièrement retouchées manuellement, par exemple pour prendre en compte certains critères subjectifs non formalisables dans le modèle.

La recherche locale a cependant des faiblesses. L'approche par mouvements, qui fonctionne bien dans le cas combinatoire, peut être inefficace pour certaines classes de problèmes, en particulier ceux ayant des contraintes concernant des variables continues. Si l'ensemble admissible est trop contraint, des outils mathématiques différents sont alors nécessaire pour capter la géométrie du problème. Par exemple, la recherche locale sera inefficace pour résoudre des programmes linéaires ayant de nombreuses égalités. L'autre défaut principal des heuristiques par rapport aux approches exactes traditionnelles est qu'aucune garantie sur la qualité de la solution n'est fournie. Si l'optimalité exacte importe peu, de telles garanties sont cependant souhaitables. En effet, sur un problème à fort enjeu économique, l'ordre de grandeur de l'écart à l'optimum peut orienter la stratégie d'une entreprise : se satisfaire des solutions actuelles si elles sont optimales à quelques pourcents près, ou investir davantage de temps et de ressources en recherche et développement si l'écart est plus important. Ainsi, l'utilisation de la recherche locale sur des problèmes où des garanties d'optimalité sont nécessaires impose de calculer en parallèle des bornes inférieures, généralement avec des approches dédiées au problème, par exemple comme dans [40].

1.2 Optimisation globale

Les limitations de la recherche locale, que nous venons de décrire, constituent des axes de recherche et de développement stratégiques pour LocalSolver. C'est dans cette optique que la thèse a été initiée, et ce sont ces limitations que nous avons cherché à pallier. On parle d'optimisation locale lorsque l'on cherche une solution localement optimale, c'est-à-dire qui ne peut être améliorée sans modification importantes. L'optimisation globale s'intéresse quant à elle à la meilleure solution possible. Il s'agit d'une approche complémentaire à la recherche locale, en particulier par sa capacité à calculer des bornes génériques. Dans la suite de ce manuscrit, on se concentre sur le cas d'un problème de minimisation, et on parle donc de bornes inférieures.

1.2.1 Calcul de bornes inférieures

On s'intéresse à l'optimisation globale déterministe, dotée d'un résultat de convergence en temps fini, par opposition à l'optimisation globale non déterministe, par exemple les algorithmes de recuit simulé, dont la convergence est probabiliste. Plus que le résultat de convergence, c'est la capacité à trouver et à améliorer des bornes

inférieures tout au long de la recherche qui nous intéresse, puisqu'il est de toute façon optimiste d'espérer atteindre l'optimum en des temps raisonnables, même pour des problèmes de taille moyenne. En effet, l'optimisation globale déterministe s'attaque aux MINLP non convexes, donc à des problèmes NP-difficiles.

Motivations

Une borne inférieure d'un problème d'optimisation est une valeur que l'on pourra peut-être atteindre, mais jamais dépasser. Elle permet ainsi de calculer un écart d'optimalité, soit une mesure du manque à gagner de la solution courante dans le cas le plus pessimiste, apportant de l'information sur la qualité des solutions renvoyées. Cette information est ce qui a motivé l'implémentation de méthodes de calcul de bornes, puisqu'elle peut avoir une forte valeur ajoutée, en particulier pour des applications industrielles. Un écart d'optimalité de quelques pourcents pourra ainsi rassurer les utilisateurs sur la qualité des solutions fournies, et leur éviter d'engager plus d'efforts pour la recherche de meilleures solutions. À l'inverse, un écart plus grand peut motiver l'investissement de plus d'efforts en recherche et développement.

Si l'écart d'optimalité passe en dessous d'un seuil prédéfini, la solution peut être déclarée globalement optimale. Comme expliqué précédemment, cette optimalité globale des solutions n'est pas primordiale dans un contexte industriel, et le temps d'obtention des bornes est un critère important, au même titre que leur qualité. De façon générale, il n'est pas nécessaire de prouver un écart d'optimalité plus petit que l'incertitude sur les données (par exemple : les problèmes où interviennent des temps de trajet, souvent approximatifs). Trouver et prouver l'optimum global sur des problèmes simples ou de petite taille reste cependant une conséquence positive du calcul de bornes par des méthodes d'optimisation globale.

Enfin, outre le rayonnement scientifique du logiciel, le calcul de bornes a aussi des intérêts d'un point de vue commercial et *marketing*. Qualifier avec précision la qualité des solutions sur les problèmes jouet, ainsi que lors des phases de prototypage et de preuve de concept, a un impact positif sur la première impression renvoyée. La prédominance de la programmation linéaire en nombres entiers sur le marché de l'optimisation fait aussi que les clients s'attendent à ce qu'un logiciel d'optimisation soit capable de trouver des bornes, et le calcul de ces dernières permet finalement d'asseoir la crédibilité d'une approche concurrente. Une dernière application concerne le développement et le support, et facilite le travail des ingénieurs en optimisation : à l'instar de la détection de l'inconsistance, le calcul de bornes permet de relever des erreurs dans les modèles d'optimisation.

Recherche primale et recherche duale

L'optimisation globale comprend la recherche de solutions (recherche primale), ainsi que celle des bornes (recherche duale) qui permettront d'en prouver l'optimalité. S'il suffit d'exhiber une solution pour prouver la borne primale associée, le calcul des bornes duales est par nature plus complexe. En effet, une borne se compare à la valeur de toute solution admissible, et donc à l'ensemble admissible du problème. Contrairement à une solution primale, qui se suffit à elle-même, une borne duale est

une démonstration, c'est-à-dire un ensemble de certificats permettant d'assurer d'en assurer la validité.

La nature des bornes primales et duales fait que leur recherche est différente. L'inspection d'une solution permet de donner une borne primale valide, et une suite de solutions, qui n'ont rien à voir les unes avec les autres, permet de l'améliorer. L'élément unitaire à considérer dans la recherche primale est donc la solution. Dans le cas de la recherche duale, c'est la notion de partition de l'espace de recherche qui est importante. Si X est l'espace auquel appartiennent les décisions, une partition $X = \bigcup_i X_i$ de cet espace permet le calcul d'une borne globale $L = \min_i L_i$, où L_i est une borne inférieure sur le sous-ensemble X_i. Dans le cas de la minimisation, l'optimisation globale peut ainsi se réduire à deux processus : la recherche d'une suite de solutions admissibles de valeurs décroissantes, et la recherche d'une partition (X_i) maximisant $\min_i L_i$.

Les recherches primales et duales interagissent cependant tout au long de la résolution. Les bornes primales sont utilisées dans la recherche duale pour resserrer les bornes des variables, et éliminer des sous-ensembles ne contenant que des solutions sous-optimales. La recherche duale permet quant à elle de fournir des solutions fractionnaires à des heuristiques primales, guidant ainsi la recherche de nouvelles solutions. Enfin, les bornes duales permettent d'interrompre la recherche, lorsque l'optimalité est prouvée.

La nature des interactions primales-duales est importante dans le design d'un solveur d'optimisation globale, tout comme elle l'a été dans les solveurs de PLNE, en particulier dans celui de la recherche arborescente. Historiquement, les solutions fractionnaires issues de la recherche duale étaient l'unique source de solutions primales pendant la recherche. L'introduction des heuristiques primales [42] a changé la donne, et des efforts importants sont aujourd'hui dédiés à l'amélioration la solution courante tout au long de la recherche arborescente. Les règles de branchement sont aussi devenues hybrides, cherchant à améliorer les bornes duales ainsi que les solutions primales. Dans notre cas, nous avons choisi le parti pris de découpler les deux recherches. LocalSolver est en effet un bloc à part entière, et n'a pas besoin de solutions fractionnaires pour fonctionner. Notre recherche duale peut donc exclusivement se concentrer sur le calcul de bornes, même si les solutions fractionnaires et admissibles rencontrées peuvent toujours être exploitées.

Cadre mathématique

On veut pouvoir fournir des bornes aux problèmes du type

$$\text{(MINLP)} \min_{x \in \mathbb{R}^n} f(x) \quad \text{s.c.} \begin{cases} \ell \leq x \leq u, \\ g(x) = 0, \\ h(x) \leq 0, \\ \forall i \in I, x_i \in \mathbb{Z}. \end{cases} \tag{1.1}$$

où $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^p$, $h : \mathbb{R}^n \to \mathbb{R}^q$ et I est un sous-ensemble des variables. Ce problème sera représenté par un DAG, qui constitue les données d'entrée, et sur

lequel il faut faire quelques hypothèses.

Le langage de modélisation de LocalSolver est plus expressif que la classe des problèmes de type MINLP, en particulier grâce aux opérateurs ensemblistes, qui permettent de traiter des objets complexes tels que des partitions, cycles ou encore tournées de véhicules. Les variables ensemblistes de LocalSolver ont été introduites pendant cette thèse et nécessitent des approches spécifiques pour le calcul de bornes. On suppose donc premièrement que les variables présentes dans le graphe sont continues, entières ou booléennes, et que leurs bornes, $\ell \leq x \leq u$, sont à valeurs finies. Cette dernière propriété est toujours vérifiée pour un problème industriel, et LocalSolver l'impose de toute façon à ses utilisateurs. Ensuite, certaines fonctions mathématiques (par exemple $\arccos, \arcsin, \arctan, \text{erf}$) ne seront pas prises en compte, car trop rares dans les problèmes d'optimisation. Dans ce cas, on se limite à un sous-ensemble d'opérateurs que l'on s'autorise dans le DAG pour le calcul de bornes. Ce choix se base sur les opérateurs mathématiques les plus courants, mais aussi sur les pratiques de modélisation de LocalSolver. Nous nous sommes ainsi restreints aux opérateurs résumés dans le tableau 1.2.

Arithmétiques	Non linéaires	Non différentiables	Logiques	Comparaisons
$+$	exp	min	and	$>$
$\times$	log	max	or	$\geq$
-	pow	abs	not	$<$
$\div$	sin	floor	xor	$\leq$
	cos	ceil	if-then-else	$=$
	tan	round		$\neq$

Tableau 1.2 – *Liste des opérateurs pris en compte pour le calcul de bornes inférieures.*

Dans le cas où le modèle fourni à LocalSolver n'entre pas dans la classe de problème que nous venons de décrire, une première reformulation sera appliquée au DAG. Celle-ci aura pour but de fournir au module dual un graphe vérifiant les hypothèses, et dont le MINLP correspondant constituera une relaxation du problème. Dans ce cas, il est probable que la borne inférieure ne converge pas vers la meilleure solution, mais il est aussi possible que le saut de dualité associé soit faible. La méthodologie de cette première relaxation est donnée dans le chapitre 2.

Optimisation exacte et inexacte

Une dichotomie importante en optimisation globale concerne le caractère exact ou approché des résultats obtenus. Sur un ordinateur standard, un nombre réel est représenté par une suite de 64 bits, selon la norme IEEE 754 [99]. L'ensemble des réels représentables est ainsi fini, et l'arithmétique transcendante (fonctions exp, log, cos, ...) y est calculée de façon approchée. Les résultats de tous les calculs sont arrondis et tronqués pour tenir dans les 64 bits prévus. Ce caractère inexact peut être ignoré pour certains calculs et applications, mais devient limitant pour d'autres. Par exemple, dans le cas de la propagation et de l'inférence de bornes (chapitre 4), ou encore des algorithmes de recherche linéaire dans un solveur non

linéaire (chapitre 3), les erreurs numériques d'arrondi et de troncature représentent une difficulté majeure, qui doit être prise en compte explicitement.

Les erreurs numériques sont omniprésentes dès que l'on cherche à implémenter des techniques mathématiques utilisant des nombres réels. Ces erreurs peuvent être contrôlées par des implémentations précautionneuses, mais pas éliminées. Si des résultats exacts sont nécessaires, par exemple dans le cadre d'une démonstration mathématique, comme celui du calcul des nombres caressants en géométrie [111], d'autres techniques doivent être utilisées. Les outils développés par la communauté de l'optimisation globale peuvent ainsi être partitionnées en deux, selon qu'ils soient entièrement robustes aux erreurs numériques ou non.

L'utilisation d'une approche plutôt que l'autre dépend généralement de l'application souhaitée : si une preuve mathématique dépend du résultat ou si une précision élevée est nécessaire, il faut utiliser une approche exacte. Ceci a cependant un coût, puisque les méthodes exactes sont moins performantes (d'un point de vue du temps de calcul) que les approches inexactes. L'utilisation du simplexe exact sur les rationnels est par exemple plusieurs ordres de grandeur en dessous des performances des meilleurs solveurs linéaires généraux. Étant donné notre contexte industriel, nous nous intéressons plutôt à l'approche inexacte, et nous serons attentifs au contrôle des erreurs numériques. On peut aussi remarquer qu'en cas de besoin spécifique, l'utilisation de flottants 128 bits réduit considérablement les erreurs numériques, au prix d'une baisse de performance.

1.2.2 État de l'art

On s'intéresse dans cette partie aux méthodes de résolution du problème (1.1), réécrit ici sous la forme suivante :

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{s.c.} \begin{cases} \ell \leq x \leq u, \\ g(x) \leq 0, \\ \forall i \in I, x_i \in \mathbb{Z}. \end{cases} \tag{1.2}$$

En particulier, on se concentre sur les approches de résolution globales déterministes, qui permettent de calculer des bornes inférieures. Les techniques de résolution dépendent des opérateurs présents dans le modèle et de la convexité de la relaxation continue. Des revues bibliographiques sur le sujet sont par exemple [78, 157, 34] dans le cas général, [53, 105] dans le cas où la relaxation continue est convexe, ou encore [59] sur les logiciels correspondants.

MINLP convexe

Lorsque f et les g_i sont convexes, la relaxation continue du problème l'est aussi. On peut alors résoudre cette dernière à l'optimum global avec un solveur local, obtenant ainsi une borne inférieure. Dans ce cas, la difficulté se trouve dans la gestion des contraintes d'intégrité. Il existe dans ce cas deux familles d'approches pour résoudre le problème (1.2).

Premièrement, les approches de type *branch-and-bound* résolvent des relaxations continues et énumèrent les solutions entières par branchement, jusqu'à atteindre la faisabilité. Les relaxations continues peuvent être résolues avec un solveur non linéaire local, ou avec un solveur linéaire auquel on ajoute des coupes. De nombreux solveurs, en particulier non linéaires, implémentent cette stratégie, comme SBB [58], BONMIN [54], MINOTAUR [120], KNITRO [7], MOSEK [30], FILMINT [14].

L'autre approche principale pour attaquer le problème dans le cas convexe est celle des méthodes par décomposition, alternant la résolution d'un problème maître et d'un problème esclave. Les méthodes les plus utilisées sont l'algorithme d'approximation extérieure et la méthode des plans sécants généralisée. Par exemple dans l'algorithme d'approximation extérieure, un programme linéaire en nombres entiers est initialisé en relâchant toutes les contraintes non linéaires. La résolution de ce problème maître fournit une borne inférieure, mais aussi une solution vérifiant les contraintes d'intégrité. On définit ensuite un problème esclave, dans lequel les variables entières sont fixées à leurs valeurs optimales dans la dernière résolution du problème maître. C'est un programme non linéaire convexe, dont un certificat de KKT - d'inconsistance ou d'optimalité - permet la séparation de la solution optimale du problème maître, et dont une solution admissible fournit une borne primale. Si la solution initiale du problème esclave est optimale pour ce dernier, alors cette solution est celle recherchée et l'algorithme est interrompu. En itérant ce processus, on peut montrer la terminaison finie de l'algorithme et sa convergence vers la solution optimale. Ces techniques sont utilisées par exemple dans les solveurs BONMIN [54], ALPHAECP [163], DICOPT [89], COIN-SHOT [106], sous diverses variantes.

On peut aussi noter que tout algorithme développé pour les MINLP convexes peut être utilisé en tant qu'heuristique primale dans le cas non convexe. On ne génère cependant plus de bornes inférieures et on perd la terminaison en temps fini ainsi que la convergence.

NLP non convexe

Lorsque le problème (1.2) ne contient pas de contraintes d'intégrité, mais que ni l'objectif, ni les contraintes ne sont supposées convexes, nous sommes en présence d'un NLP non convexe. L'algorithme αBB [21, 20, 28], un outil précurseur en optimisation globale, permet d'attaquer le problème à l'aide de relaxations convexes non linéaires et d'un *branch-and-bound* spatial. Pour chaque fonction h (objectif ou contrainte), une hessienne par intervalle $H = ([a_{ij}, b_{ij}])$ de h est calculée sur le domaine $\ell \leq x \leq u$, à partir du graphe d'évaluation de h. On a alors, pour tout point du domaine,

$$\forall i, j, \quad \frac{\partial^2 h(x)}{\partial x_i \partial x_j} \in [a_{ij}, b_{ij}].$$

Cette matrice par intervalle permet de déterminer une matrice diagonale $D = \operatorname{diag} \alpha$, telle que $\alpha_i \geq 0$ et $H + D \succcurlyeq 0$ (au sens des matrices par intervalles, c'est-à-dire que pour tout x vérifiant les contraintes de bornes, $\nabla^2 h(x) + D \succcurlyeq 0$). Enfin, une relaxation convexe de h est obtenue en y ajoutant la fonction

$$\phi_{\alpha,\ell,u}(x) = \sum_i \alpha_i (x_i - \ell_i)(x_i - u_i),$$

négative sur le domaine de définition de x et telle que $h + \phi_{\alpha,\ell,u}$ est convexe. Cette méthode a l'avantage de n'ajouter aucune variable ni contrainte au problème initial, contrairement à l'approche de reformulation-convexification introduite plus bas. L'erreur de convexité commise dans le pire cas s'obtient en minimisant $\phi_{\alpha,\ell,u}$ sur le domaine des variables, et vaut

$$\min_{\ell \leq x \leq u} \phi_{\alpha,\ell,u}(x) = -\frac{1}{4} \sum_i \alpha_i (u_i - \ell_i)^2.$$

Celle-ci converge vers 0 si les bornes de x convergent les unes vers les autres, comme c'est le cas dans un *branch-and-bound* spatial.

Un développement récent [48] exploite l'algorithme αBB pour proposer une nouvelle approche de résolution de (1.2). En écrivant cette fois les contraintes sous forme d'égalités (*via* l'introduction de variables d'écart), le lagrangien augmenté du problème s'écrit

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^T g(x) + \frac{\rho}{2} \|g(x)\|^2, \tag{1.3}$$

où $\rho \geq 0$ est un facteur de pénalisation. La minimisation globale de (1.3) sous les contraintes de bornes constitue une relaxation du problème. En y appliquant l'algorithme αBB, on obtient donc une borne inférieure valide à (1.2). Contrairement à l'application directe de l'algorithme αBB sur (1.2), il n'y a qu'une seule fonction (le lagrangien augmenté) à convexifier. Dans ce cas, la convexité individuelle de chaque contrainte et de l'objectif n'a pas d'importance, c'est la convexité de la somme qui compte. Certaines expressions strictement convexes peuvent alors compenser des non convexités présentes dans d'autres termes. En contrepartie, on a autant de problèmes d'optimisation globale à résoudre que d'itérations dans le lagrangien augmenté (en général quelques dizaines d'itérations au maximum).

L'algorithme est finalement le suivant : pour des multiplicateurs et facteurs de pénalisation initiaux, le sous-problème de minimisation du lagrangien augmenté est résolu à l'optimum global avec l'algorithme αBB. Les multiplicateurs et facteurs de pénalisation sont alors mis à jour avec les règles usuelles du lagrangien augmenté, et l'algorithme est itéré jusqu'à l'atteinte de la faisabilité. Si l'approche est élégante et possède des propriétés de convergence globale, elle souffre aussi de quelques défauts. L'hypothèse de double différentiabilité est forte, et ne permet pas de traiter les problèmes mixtes, comprenant par exemple des opérateurs logiques. Cette difficulté est rédhibitoire dans notre cas, puisque les utilisateurs de LocalSolver ont justement souvent des problèmes combinatoires. De plus, l'erreur de convexité peut devenir très grande si certaines variables ont un domaine de définition large (pour une variable $0 \leq x \leq 10^6$, un terme de convexité peut soustraire 10^{12} à la borne inférieure par rapport à la valeur de l'objectif).

MINLP non convexe

Nous en venons finalement aux techniques pour résoudre (1.2) dans le cas le plus général. Il n'y a ici plus qu'une seule d'approche, appelée approche par reformulation-convexification. Une première étape, appelée factorisation du modèle, consiste à

reformuler le problème sous la forme standard dite de Smith [153] :

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} c^T x \\ &\text{s.c.} \begin{cases} \ell \leq x \leq u, \\ x \in \mathcal{P} \cap \mathcal{R} \cap \mathcal{B}, \\ x_i \in \mathbb{Z}, \forall i \in I. \end{cases} \end{aligned} \tag{1.4}$$

L'ensemble $\mathcal{P}$ est un polyèdre et rassemble les contraintes linéaires du problème et de la reformulation :

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax + b = 0,\ Cx + d \leq 0\}. \tag{1.5}$$

On n'y inclut pas les contraintes de bornes, car celles-ci jouent un rôle à part. Les contraintes non linéaires sont reformulées pour que leur arité soit la plus faible possible. Selon cette arité (1 ou 2), elles sont ensuite réparties dans $\mathcal{R}$ et $\mathcal{B}$, rassemblant respectivement les couplages non linéaires unaires :

$$\mathcal{R} = \{(i, j, f_{ij}) \mid x_j = f_{ij}(x_i)\}, \tag{1.6}$$

où les f_{ij} sont du type exp, log, x^a, ..., et les relations bilinéaires :

$$\mathcal{B} = \{(i, j, k) \mid x_k = x_i x_j\}. \tag{1.7}$$

Ces ensembles permettent de modéliser toutes les expressions non linéaires, *via* l'ajout de nouvelles variables et contraintes linéaires. On peut généraliser ce formalisme, en ajoutant d'autres types de contraintes, comme par exemple les relations trilinéaires, de la forme $t = xyz$. Ceci n'est cependant pas obligatoire, puisque toutes les fonctions non linéaires peuvent se décomposer en relations unaires et bilinéaires.

Cette reformulation permet d'obtenir une relaxation de (1.2) à partir d'une relaxation de (1.6) et de (1.7). Le problème de la convexification des fonctions non linéaires est ainsi ramené au cas des fonctions à une ou deux variables uniquement, nettement plus simple. La résolution de la relaxation convexe ainsi obtenue fournit une borne inférieure, qui est ensuite améliorée par l'application d'un *branch-and-bound* spatial.

Cette approche est celle utilisée par les solveurs globaux les plus connus : BARON [144, 155], SCIP [17, 159, 160], ANTIGONE [125, 126], COUENNE [36, 32] et LINDO GLOBAL [115]. La forme factorisée du modèle possède de nombreux avantages, comme nous le verrons tout au long du développement, mais a aussi quelques défauts. Par exemple, le nombre de variables et de contraintes augmente fortement, et un problème initialement convexe ne l'est généralement plus.

1.3 Plan et déroulement de la thèse

1.3.1 Approche retenue pour le calcul de bornes

L'approche retenue dans cette thèse est celle de reformulation-convexification. En effet, on veut une approche générique, qui ne suppose pas que certaines fonctions sont convexes. De plus, le cœur de métier de LocalSolver étant l'optimisation combinatoire, les opérateurs logiques et conditionnels sont souvent utilisés, tout comme certains opérateurs non différentiables, tels que min et max. Pour prendre en compte tous les opérateurs donnés dans le tableau 1.2, l'approche par factorisation du modèle est la plus adaptée. La forme standard (1.4) des MINLP est donc notre point de départ, et nous en augmenterons l'expressivité. La volonté d'exploiter des relaxations non linéaires nous pousse aussi à s'inspirer de l'approche αBB. Plus précisément, nous allons utiliser des techniques issues de la méthode αBB au sein de l'approche par reformulation-convexification.

Extensions de la forme standard

On commence par faire la remarque suivante : (1.4) est un programme linéaire, auquel on ajoute un ensemble de contraintes non convexes liées aux intégrités et non linéarités du problème initial. On dit dans ce cas que (1.4) est basé sur un programme linéaire sous-jacent. On peut aussi utiliser la même approche, mais avec un autre type de problème sous-jacent, par exemple un problème non linéaire. L'avantage est qu'il est alors possible d'appliquer les techniques de αBB pour obtenir des relaxations convexes non linéaires du problème sous-jacent, sans ajouter de variables ni de contraintes. La hiérarchie des problèmes non linéaires convexes - QP, QCQP, SOCP et SDP - nous propose quelques candidats pour généraliser le modèle factorisé linéaire. Les programmes semi-définis forment une alternative intéressante, par la qualité des bornes fournies. Cependant, la maturité des solveurs SDP génériques est encore insuffisante pour attaquer des problèmes de taille industrielle. Nous avons donc choisi d'étendre le modèle factorisé en utilisant un QCQP sous-jacent :

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} \; a + c^T x + x^T Q x \\ &\text{s.c.} \begin{cases} \ell \leq x \leq u, \\ x \in \mathcal{P}, \\ x^T Q_k x + c_k^T x + a_k \leqq 0, k = 1..q. \end{cases} \end{aligned} \tag{1.8}$$

Cette formulation limite l'ajout de variables et de contraintes, notamment en réduisant la taille de $\mathcal{B}$, et offre la possibilité d'utiliser des relaxations convexes non linéaires. La connaissance analytique des sources de non-linéarités dans (1.8) est une propriété appréciable, en particulier pour la gestion des problèmes numériques au sein d'un solveur non linéaire.

Les contraintes coniques du second ordre sont plus expressives que les contraintes quadratiques. Elles pourront être ajoutées à notre forme standard sans modifications majeures, pourvu que l'on ait accès à des solveurs qui les prennent en compte. Enfin, la gestion d'un objectif et de contraintes non linéaires convexes quelconques pose

quelques difficultés supplémentaires. Il faut conserver une partie du modèle factorisé sous forme de DAG, utiliser des outils de différentiation automatique, et la gestion de la stabilité numérique s'en voit complexifiée. Puisque nous partons de zéro pour le calcul de bornes, nous nous limitons au cas des contraintes quadratiques pour le moment, mais notre structure permettra d'y ajouter d'autres types de contraintes, une fois le solveur mûr.

Un exemple

Pour illustrer l'approche retenue pour le calcul de bornes inférieures, nous donnons maintenant un exemple concret sur un petit problème non linéaire. Celui-ci consiste à trouver la géométrie optimale d'un sceau, dans l'objectif de maximiser son contenu sous la contrainte que son aire totale ne dépasse pas celle d'un disque de rayon 1. Le problème est illustré dans la figure 1.3.

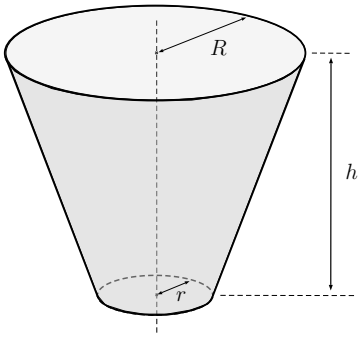

FIGURE 1.3 – *Le problème du sceau optimal. Ici, R désigne le rayon de la face ouverte, r celui de la face fermée, et h la hauteur. Le volume s'exprime alors avec $\frac{\pi}{3}h\left(r^2+rR+R^2\right)$, et l'aire totale avec $\pi r^2+\pi(R+r)\sqrt{(R-r)^2+h^2}$.*

La modélisation naturelle du problème est donnée par le programme non linéaire suivant :

$$\min_{x\in\mathbb{R}^3} \frac{\pi}{3}x_3\left(x_1^2+x_1x_2+x_2^2\right)$$
$$\text{s.c.} \begin{cases} 0\leq x_1\leq 1, \\ 0\leq x_2\leq 1, \\ 0\leq x_3\leq 1, \\ x_1^2+(x_1+x_2)\sqrt{(x_1-x_2)^2+x_3^2}\leq 1, \end{cases}$$

où l'on a noté $(x_1,x_2,x_3)=(r,R,h)$. Pour calculer des bornes inférieures à ce problème, on commence par le reformuler sous une forme standard. Si l'on souhaite utiliser des relaxations linéaires, on utilisera la forme standard linéaire. Celle-ci, ainsi que les couplages non linéaires associés, s'exprime comme suit :

$$\min_{x\in\mathbb{R}^{15}} \frac{\pi}{3}x_{15}$$
$$\text{s.c.} \begin{cases} 0 \le x_1 \le 1, \\ 0 \le x_2 \le 1, \\ 0 \le x_3 \le 1, \\ x_{11} + x_{13} \le 1, \\ x_8 = x_1 - x_2, \\ x_{10} = x_6 + x_9, \\ x_{12} = x_1 + x_2, \\ x_{14} = x_4 + x_5 + x_7, \\ x \in X. \end{cases} \qquad X := \begin{cases} x_4 = x_1^2, \\ x_5 = x_2^2, \\ x_6 = x_3^2, \\ x_7 = x_1 x_2, \\ x_9 = x_8^2, \\ x_{11} = \sqrt{x_{10}}, \\ x_{13} = x_{11} x_{12}, \\ x_{15} = x_3 x_{14}. \end{cases}$$

À l'inverse, si l'on souhaite utiliser des relaxations non linéaires, nous utiliserons plutôt la forme standard quadratique. Dans cet exemple, celle-ci est la suivante :

$$\min_{x\in\mathbb{R}^{6}} \frac{\pi}{3}x_3 x_6$$
$$\text{s.c.} \begin{cases} 0 \le x_1 \le 1, \\ 0 \le x_2 \le 1, \\ 0 \le x_3 \le 1, \\ x_1^2 + x_1 x_5 + x_2 x_5 \le 1, \\ x_4 = x_1^2 + x_2^2 + x_3^2 - 2x_1 x_2, \\ x_6 = x_1^2 + x_2^2 + x_1 x_2, \\ x \in X. \end{cases} \qquad X := \Big\{ x_5 = \sqrt{x_4}$$

L'approche par relaxations linéaires est la plus classique, même si les solveurs d'optimisation globale de l'état de l'art étendent peu à peu leurs capacités, en particulier pour traiter plus efficacement les contraintes quadratiques.

1.3.2 Plan et philosophie de la thèse

Le chapitre 2 de cette thèse précise ce que nous venons d'exposer sur la factorisation du problème. Il explique en détail la forme standard que nous utilisons, ainsi que sa construction. Nous présentons les reformulations et techniques de *presolve* que l'on peut y appliquer. Nous rappelons les relaxations convexes utilisées dans la littérature pour les couplages non linéaires du modèle factorisé, en les généralisant aux nouveaux types de contraintes pris en compte. Enfin, nous exposons l'approche utilisée pour générer les relaxations convexes des expressions quadratiques.

Le chapitre 3 présente les méthodes de résolution des relaxations non linéaires. Nous y obtenons un résultat de dualité important pour l'ensemble du calcul de bornes, et y expliquons le choix de l'approche utilisée dans le solveur non linéaire. Celui-ci est basé sur une méthode de lagrangien augmenté, dont nous rappelons l'origine et montrons en quoi elle est pertinente pour le calcul de bornes. Nous présentons ensuite quelques spécificités d'implémentation, basées d'une part sur notre

contexte de calcul de bornes inférieures à des problèmes non linéaires convexes, et d'autre part sur la structure des relaxations utilisées.

Le chapitre 4 présente le squelette de l'algorithme de calcul de bornes, une recherche arborescente de type *branch-and-reduce*. Nous en détaillons la cinématique : reformulation et *presolve*, résolution du *root node* puis recherche arborescente. Dans ce cheminement, les techniques de réduction de bornes sont un outil central, dont nous décrivons l'implémentation avec précision, en particulier sous l'angle de la robustesse aux erreurs numériques et de la performance. Ces techniques permettent de calculer des bornes inférieures, de resserrer les relaxations convexes et d'éliminer des sous-problèmes inconsistants lors du partitionnement.

Le chapitre 5 termine cette thèse par la présentation de quelques résultats numériques résumant l'apport des bornes dans LocalSolver. Nous nous intéressons à des *benchmarks* classiques de l'optimisation globale, mais aussi issus des problèmes industriels traités par les clients de l'entreprise. Les capacités d'optimisation de LocalSolver sont évaluées, et des comparaisons sont effectuées avec quelques concurrents sur la résolution des MINLP. Enfin, nous terminons par une discussion sur les perspectives d'améliorations du code implémenté, au regard de l'expérience numérique acquise et des comparaisons effectuées. Nous concluons par une réflexion sur les problèmes du calcul de bornes et de la résolution à l'optimum global des MINLP.

Contributions

Cette thèse est effectuée sous une convention CIFRE et ses objectifs répondent à un besoin industriel. Dans ce sens, la contribution principale reste le code produit et intégré à LocalSolver, comportant à ce jour environ 45000 lignes de code, écrites dans le langage C++. L'importance relative (en termes de taille du code) des différents composants est résumée dans le tableau 1.3. Le module dual a été développé directement au sein de LocalSolver, sur une branche dédiée. Il a été activé pour la première fois dans la version commerciale du logiciel en janvier 2019, lors de la sortie de LocalSolver 8.5, après une période de cinq mois entièrement dédiée au débogage et aux réglages numériques. Une mise à jour importante a été effectuée lors de la sortie de LocalSolver 9.0, en août 2019.

Toutes les techniques utilisées ont été implémentées depuis zéro, à l'exception de l'algorithme du simplexe dual et des méthodes de résolution des systèmes linéaires, utilisées dans le solveur non linéaire pour le calcul des directions de Newton. Si certains outils, comme la bibliothèque d'arithmétique d'intervalle décrite au chapitre 4, auraient pu être intégrés en tant que librairies extérieures, des questions de licences, de maîtrise du code, ainsi que la présence de cas particuliers non pris en compte par ces librairies nous ont poussé à en implémenter notre propre version. Les contributions de ce travail, qu'elles soient théoriques ou sous la forme de recettes numériques, sont données avec plus de précision et mises en relations avec l'état de l'art au début de chaque chapitre.

L'optimisation globale est un domaine de recherche actif auquel de nombreux acteurs participent. La performance des solveurs implémentés repose à la fois sur des résultats géométriques théoriques et sur des recettes numériquement efficaces. Cette interaction entre théorie et pratique est au cœur de notre sujet, puisqu'une im-

Module	Taille du code
Reformulation	25%
Presolve	7.5%
Relaxations	12.5%
Intégration au solveur linéaire	1.5%
Solveur non linéaire	14%
Techniques de réduction de bornes	17.5%
Branch-and-reduce	8.5%
Intégration à LocalSolver	6%
Autres	7.5%

Tableau 1.3 – *Taille relative des différents modules. Celle-ci reflète généralement le temps investi pour l'implémentation, le débogage et les réglages numériques, exception faite des modules Reformulation et Solveur non linéaire, qu'il faudrait permuter.*

plémentation naïve atteint des niveaux de fiabilité, de robustesse et de performance limités. On s'attachera donc à décrire les raisons motivant les choix scientifiques effectués, ainsi que les subtilités de leur implémentation, plutôt que leur justification théorique, exception faite des innovations apportées. Dans ce sens, nous préférons donner l'intuition derrière les approches utilisées, plutôt qu'une formalisation de leur effets (sous la forme de preuves par exemple).

Une note sur le *benchmarking*

Le *benchmark* utilisé pour le développement et l'évaluation du code est composé de plusieurs librairies. Il contient les instances classiques de l'optimisation globale, la MINLPLib [11] (1500 instances). Nous y avons aussi inclus la QPLib [3] (400 instances), rassemblant des problèmes quadratiques de toutes natures, des programmes linéaires (90 instances), notamment de la Netlib [12], quelques programmes linéaires en nombres entiers (50 instances), notamment de la MIPLib [4] ainsi qu'un sous-ensemble de CUTEr [10] (300 instances), une librairie de problèmes non linéaires généralement utilisée pour évaluer la recherche primale. Lors de nos travaux sur les relaxations convexes des fonctions quadratiques, nous avons aussi ajouté une centaine d'instances quadratiques de la ORLib [1], rassemblant des problèmes classiques de la recherche opérationnelle (tels que les problèmes de la coupe maximale, du *k-cluster*, du sac-à-dos quadratique, du sac-à-dos quadratique avec contrainte de cardinalité, d'affectation quadratique ou encore du stable de taille maximum).

À ces instances de référence, nous avons ajouté l'ensemble des instances internes de LocalSolver qui entrent dans le cadre de nos travaux (c'est-à-dire ceux qui ne contiennent pas d'opérateurs ensemblistes). Ce *benchmark* industriel contient environ 900 instances, et est couplé à un second ensemble de 250 instances issues des tests unitaires de LocalSolver, de petite taille mais souvent numériquement intéressants. Pour prendre en compte certains opérateurs rarement présents dans les librairies usuelles (en particulier les opérateurs min et max), nous avons aussi ajouté une trentaine de problèmes non différentiables, issues de [118]. Enfin, un peu plus de

200 instances, une moitié de type MIQCQP, l'autre de type NLP, ont été générées aléatoirement, afin de mettre le calcul de bornes à l'épreuve en faisant varier les domaines des variables, les coefficients de l'objectif et des contraintes, ainsi que le nombre de degrés de liberté des solutions optimales.

Nous disposons finalement d'environ 3700 instances. Pour des raisons de puissance de calcul disponible, les tests et résultats numériques ne sont pas systématiquement présentés sur l'intégralité du *benchmark*, mais souvent sur un sous-ensemble. Des illustrations numériques sont données tout au long du développement, dans le but de montrer l'intérêt des fonctionnalités implémentées, ainsi que l'importance de chaque outil. Les résultats finaux sont donnés dans le dernier chapitre. Si la MINLPLib reste la référence en optimisation globale, celle-ci n'est pas notre priorité. En effet, le contexte industriel de cette thèse rend ces instances moins pertinentes, car plus éloignées du cœur de métier de LocalSolver. De plus, si les comparaisons entre solveurs sont généralement faites avec un temps limite d'une ou deux heures par instance, le cahier des charges, basé sur la philosophie de LocalSolver, nous pousse à utiliser un temps de référence de l'ordre de la minute.

Chapitre 2

Reformulations et relaxations

L'outil mathématique le plus important pour le calcul de bornes inférieures génériques est l'optimisation convexe. En effet, l'étude des conditions d'optimalité et la théorie de la dualité y sont les plus favorables. Le calcul de bornes inférieures se ramène ainsi généralement à la recherche de relaxations convexes serrées.

Dans LocalSolver, un problème d'optimisation est représenté par un graphe d'évaluation, permettant de calculer objectif et contraintes. Si ce format est adapté à la recherche locale, par sa proximité avec les outils de modélisation algébriques, sa compacité, ou encore par l'incrémentalité d'évaluation qu'il offre, il est difficile d'en extraire directement des relaxations convexes.

Dans le cadre de l'optimisation globale, cette remarque a conduit au développement d'une forme standard pour les MINLP. Celle-ci permet de reformuler un graphe d'évaluation en un programme mathématique pour lequel la génération de relaxations convexes est plus naturelle. Ce second chapitre commence ainsi par introduire cette reformulation, que nous appelons la forme factorisée du modèle, dans notre contexte. Une fois cette étape effectuée, le graphe d'évaluation n'intervient plus dans le calcul de bornes que pour vérifier l'admissibilité primale d'une solution.

Le second objectif de ce chapitre est de présenter les relaxations convexes déduites du modèle factorisé, c'est-à-dire celles qui fourniront nos bornes inférieures. Celles-ci sont pour la plupart connues, et déjà utilisées dans des solveurs d'optimisation globale de l'état de l'art. Nous nous intéressons cependant plus particulièrement au cas des relaxations convexes non linéaires des fonctions quadratiques, dans l'optique de leur intégration au sein du solveur non-linéaire décrit au chapitre 3.

Plan du chapitre

2.1 Forme standard

Un problème traité par LocalSolver est représenté par un DAG d'évaluation. Les données d'entrées du module de calcul de bornes sont donc constituées de ce DAG, ainsi que de quelques paramètres. Comme expliqué en introduction, la première étape est de construire une reformulation du modèle prenant une certaine forme standard. Nous allons donc reformuler le problème défini par le DAG en le programme suivant :

$$\min_x \; a + c^T x + x^T Q x$$
$$\text{s.c.} \begin{cases} \ell \leq x \leq u, \\ x \in \mathcal{P} \cap \mathcal{Q} \cap \mathcal{R} \cap \mathcal{B} \cap \mathcal{M} \cap \mathcal{C}, \\ \forall i \in I, \; x_i \in \mathbb{Z}. \end{cases} \tag{2.1}$$

Les contraintes sont partitionnées en plusieurs groupes selon leur type. Premièrement, $\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax + b = 0, \; Cx + d \leq 0\}$ est un polyèdre rassemblant les contraintes linaires, et $\mathcal{Q} = \{x \in \mathbb{R}^n \mid x^T Q_i x + c_i^T x + a_i \leqq 0, \; i = 1..q\}$ est un ensemble de contraintes quadratiques. Ensuite, $\mathcal{B}$ est un ensemble de couplages bilinéaires, de la forme $z = xy$, et $\mathcal{R}$ est un ensemble de couplages unaires, du type $y = f(x) \in \{\exp(x), \; \log(x), \; a/x, \; x^a, \; \sin(x), \; \cos(x), \; |x|\}$.

Jusqu'ici, l'expressivité du modèle factorisé est similaire à celle des autres solveurs d'optimisation globale, et permet de traiter la plupart des MINLP. À ce formalisme standard, nous ajoutons deux groupes de contraintes, liés aux opérateurs présents dans les modèles des clients de LocalSolver. D'abord, $\mathcal{M}$ est un ensemble de couplages utilisant les opérateurs min et max, du type $y = \min(x_1, \cdots, x_n, a)$ ou $y = \max(x_1, \cdots, x_n, a)$. Enfin, $\mathcal{C}$ représente les couplages logiques, de la forme $y = (x \gtreqless a)$, où y est un booléen. Ce dernier groupe permet d'intégrer la programmation disjonctive généralisée au modèle factorisé.

La forme factorisée du modèle n'est pas unique. En effet, on peut par exemple avoir $Q = 0$ et $\mathcal{Q} = \mathbb{R}^n$ si tous les termes quadratiques sont dans $\mathcal{R}$ et $\mathcal{B}$. Ce sera le cas si l'on utilise des relaxations convexes linéaires. De plus, les contraintes

$z = xy$, $y = a/x$ et $y = x^2$ peuvent être considérées comme des contraintes quadratiques, ou comme des couplages non linéaires. Pour des raisons d'implémentation, et afin de comparer les mérites des relaxations linéaires et non linéaires, nous utilisons deux formes du modèle factorisé. Le modèle factorisé dit linéaire est tel que $\mathcal{Q} = \varnothing$, c'est-à-dire que toutes les contraintes quadratiques sont linéarisées par l'ajout de relations bilinéaires. Le modèle factorisé dit quadratique maintient toutes les expressions quadratiques du modèle initial, sauf celles s'écrivant directement comme une relation bilinéaire. Cette distinction se règle par un paramètre et est présente dès la construction du modèle et pour le reste de la résolution.

L'expressivité de notre forme standard est l'une des plus larges des solveurs d'optimisation globale. Comme écrit dans les différentes documentations, chaque solveur supporte un certain nombre d'opérateurs uniquement. Les opérateurs min et max ne sont pas pris en compte par BARON, SCIP, Couenne et Antigone. Les fonctions trigonométriques sin et cos ne sont pas supportées par BARON, SCIP et Antigone. Seul le solveur LINGO supporte les mêmes opérateurs.

Nous terminons cette présentation par quelques hypothèses numériques effectuées sur (2.1). Sur un ordinateur 64 bits, la précision relative des nombres flottants est environ $\varepsilon = 2{,}2.10^{-16}$ et le plus grand entier standard est environ $2{,}1.10^{9}$. Afin d'éviter des problèmes numériques pénibles et peu pertinents dans un contexte industriel, nous supposons qu'une solution vérifie les propriétés suivantes. On suppose premièrement que tous les nœuds du DAG sont définis et ont une valeur finie. On interdit les divisions par zéro, y compris la forme indéterminée $0/0$. Par cohérence avec la précision ε, on suppose que la solution optimale du problème est à valeur dans $[-1/\varepsilon, 1/\varepsilon]$, soit environ $[-10^{15}, 10^{15}]$. De plus, on impose des bornes à toutes les variables, de la forme $[-10^k, 10^k]$, avec $k = 12$ pour les flottants présents dans le DAG, $k = 9$ pour les entiers, et $k = 30$ pour les autres. Enfin, on suppose qu'aucun réel ne se trouve entre 0 et ε^2, c'est-à-dire que $[0, \varepsilon^2] = \{0, \varepsilon^2\}$ et $[0, \varepsilon^2] = \{0, \varepsilon^2\}$. Cette hypothèse numériquement raisonnable est pratique pour les techniques de réduction de bornes présentées au chapitre 4.

2.1.1 Construction du modèle

L'implémentation de la transformation décrite ci-dessus est présente dans tous les solveurs d'optimisation globale. La façon dont celle-ci est effectuée, les structures de données utilisées, ainsi que les relations entre le graphe et sa forme factorisée sont cependant présentées succinctement. Certaines techniques de l'optimisation globale peuvent ainsi être effectuées sur le DAG ou sur le modèle factorisé. Pour présenter le plus clairement possible les choix qui ont été faits, ainsi que pour fournir un document qui pourra servir de référence au code produit, nous abordons tous les aspects qui entrent en jeu lors de la reformulation initiale.

Prétraitement du graphe d'évaluation

Comme expliqué précédemment, LocalSolver reçoit en entrée un DAG représentant le problème. La résolution de ce dernier commence par une étape appelée *preprocessing*, composée de deux phases. Premièrement, une série de transformations

communes à tous les modules de LocalSolver est appliquée au DAG. Ensuite, pour chaque module de LocalSolver actif sur le problème, le DAG est dupliqué et une série d'actions spécifiques au module y est appliquée. Nous ne décrivons pas en détail les actions de *preprocessing* communes, qui ont pour objectif de réduire la taille du graphe, ainsi que d'en améliorer la structure. Les actions spécifiques que nous utilisons ont quant à elles le but de simplifier l'étape de factorisation du graphe.

Premièrement, les nœuds portant des opérateurs non supportés sont remplacés par des variables, dont les domaines sont définis par les bornes propagées sur les nœuds en question. Ceci permet de générer une première relaxation du problème, dans laquelle toutes les opérations sont connues. Par exemple, les opérateurs d'arrondi ne sont pas explicitement pris en compte dans notre module dual, mais restent fréquents dans les modèles des clients. Nous les reformulons donc par :

— $\lfloor x \rfloor$ est transformé en $y \in \mathbb{Z}$ avec les contraintes $x - 1 < y \leq x$,
— $\lceil x \rceil$ est transformé en $y \in \mathbb{Z}$ avec les contraintes $x \leq y < x + 1$,
— round(x) est transformé en $y \in \mathbb{Z}$ avec les contraintes $-1/2 \leq x - y < 1/2$,

Cette reformulation fait intervenir des comparaisons strictes entre valeurs continues. Leur remplacement par des comparaisons simples constitue cependant une relaxation du problème, et reste valide pour nos objectifs.

Ensuite, les contraintes définies avec des comparaisons strictes sont reformulées à l'aide de comparaisons simples. Dans le cas de comparaison stricte entre nœuds à valeurs continues, on suppose que l'égalité stricte n'a jamais lieu. Ceci donne l'ensemble des simplifications suivantes, pour x et y des nœuds quelconques du DAG :

— $x < y$ est transformé en $x \leq y$, si $x \in \mathbb{R}$ ou $y \in \mathbb{R}$,
— $x < y$ est transformé en $x \leq y + 1$, si $x \in \mathbb{Z}$ et $y \in \mathbb{Z}$,
— $x \neq y$ est transformé en $(x \leq y + 1) \vee (x \leq y + 1)$ si $x \in \mathbb{Z}$ et $y \in \mathbb{Z}$.

De façon générale, l'utilisation de comparaisons strictes entre variables continues est à éviter, car numériquement instable.

Enfin, certaines opérations logiques sont d'abord linéarisées, afin de faciliter leur intégration au modèle factorisé. Pour x et les x_i des expressions à valeurs booléennes, on applique les transformations suivantes :

— $\neg x = 1 - x$,
— $\wedge_{1..n} x_i = y$, avec $y \in \{0, 1\}$, $\forall i,\ y \leq x_i$ et $y \geq \sum_i x_i - (n - 1)$,
— $\vee_{1..n} x_i = y$, avec $y \in \{0, 1\}$, $\forall i,\ y \geq x_i$ et $y \leq \sum_i x_i$,
— $\otimes_{1..n} x_i = \sum_i x_i - ny$, avec $y \in \{0, 1\}$ (opérateur `xor`),
— If (x_1) Then (x_2) Else $(x_3) = x_1 x_2 + (1 - x_1) x_3$.

Factorisation du modèle

Pour construire le modèle factorisé, on utilise une approche symbolique. Deux structures de données représentant des expressions linéaires et quadratiques creuses sont utilisées. Une expression linéaire est composée d'une constante c et d'une liste de termes linéaires (i, c_i). Une expression quadratique est composée d'une constante c, d'une liste de termes linéaires (i, c_i) et d'une liste de termes quadratiques (i, j, q_{ij}). Lorsqu'une telle expression est construite pour être équivalente à un nœud du DAG, on dit que c'est un modèle du nœud. On implémente alors une liste d'opérations élémentaires sur ces structures de données :

— création du modèle d'une variable : $m \leftarrow x_i$,
— création du modèle d'une constante : $m \leftarrow a$,
— addition de deux modèles : $m_1 \leftarrow m_1 + m_2$,
— multiplication par un scalaire : $m \leftarrow am,\ a \in \mathbb{R}$,
— multiplication de deux modèles linéaires : $m \leftarrow m_1 m_2$,
— mise au carré d'un modèle linéaire : $m \leftarrow m_1^2$,
— reformulation du modèle : $m \leftarrow z$, où z est une nouvelle variable et où $m - z = 0$ est ajoutée au modèle factorisé.

Une fois ces opérations disponibles, le principe de la factorisation est le suivant. Le type du modèle factorisé (linéaire ou quadratique) est déterminé, puis on effectue les opérations suivantes :

— le DAG est parcouru en largeur, en partant des variables et en remontant vers les objectifs et contraintes,
— un modèle du type choisi est construit pour chaque nœud, à l'aide des opérations élémentaires,
— une fois les modèles de chaque contrainte et objectif du DAG obtenus, ces derniers sont ajoutés au modèle factorisé.

La factorisation du modèle est un principe général qui laisse une grande marge de manœuvre lors de l'implémentation. Si celle-ci ne pose pas de difficultés théoriques, elle requiert cependant un certain nombre de réglages.

Premièrement, il faut faire attention à l'évolution du nombre de non-zéros dans le modèle, par exemple lors de la multiplication ou de la mise au carré d'un modèle, mais aussi lorsqu'un modèle à de nombreux parents (auquel cas il peut être avantageux de le reformuler par une variable même si cela n'est pas symboliquement nécessaire). Ensuite, des structures de données permettant de garantir l'unicité des couplages non linéaires sont nécessaires. En effet, certaines structures ne sont pas simplifiées par le *preprocessing* de LocalSolver. Par exemple, $1/x + 2/x + 3/x$ serait naïvement reformulée en $y_1 + y_2 + y_3$, avec $xy_1 = 1$, $xy_2 = 2$ et $xy_3 = 3$, alors que $6y$ avec $xy = 1$ est la bonne représentation. Enfin, même à taille similaire, les reformulations conservant la structure du graphe sont toujours préférables. Par exemple, pour une variable x et quatre réels a, b, c et d, si $(ax+b)(cx+d)$ peut être reformulé par le modèle linéaire z et les contraintes $z = y_1 y_2$, $y_1 = ax + b$ et $y_2 = cx + d$, une meilleure reformulation est obtenue en développant le trinôme et en le reformulant sous la forme d'un carré. Une remarque similaire s'applique à $(ax + b)/(cx + d)$.

Quelques statistiques sur la reformulation

Nous terminons cette partie par quelques statistiques sur la reformulation, en analysant les DAG des 3244 instances de plus petite taille de notre *benchmark*. Une première remarque concerne les hauteurs (nombre de rang) de ces DAG. En effet, celles-ci permettent d'estimer le degré de non linéarité d'un problème, par exemple en remarquant que des compositions linéaires successives peuvent se reformuler en une unique transformation linéaire, composée d'un unique rang. La répartition des instances de notre *benchmark* par hauteur de DAG est donnée dans le tableau 2.1.

Cette analyse fait émerger une remarque sur le champ d'application de l'optimisation globale. Si des progrès théoriques et pratiques importants ont été obtenus lors des deux dernières décennies, la résolution des MINLP reste cantonnée aux pro-

Hauteur du DAG	Nombre d'instances	Exemple
$r < 3$	3	LP/MILP non contraint
$r = 3$	534	LP/MILP
$r = 4$	1299	QP/MIQP
$r = 5$	897	QCQP/MIQCQP
$5 < r \leq 10$	550	Programme polynomial de degré r
$r \geq 10$	43	Modélisation de systèmes dynamiques

Tableau 2.1 – *Statistiques sur la hauteur (rang maximal d'un nœud) du DAG pour les instances de notre* benchmark.

blèmes dont le graphe est de faible hauteur (moins de 10 rangs). Comme le tableau 2.1 le montre, certains types de problèmes sont peu fréquents dans les librairies classiques, puisqu'encore hors de portée.

Un autre aspect important de la factorisation est l'impact de celle-ci sur la taille du modèle. Toute arrête du DAG reliant un opérateur non linéaire avec son argument est par exemple remplacée par une contrainte, et induit l'ajout d'une nouvelle variable. Le modèle factorisé ne ressemble ainsi pas beaucoup au problème initial, mais possède cependant un invariant important pour le passage à l'échelle en temps et en mémoire : le nombre de non-zéros reste du même ordre de grandeur, comme le montre la figure 2.1. Cette invariance a été obtenue par de nombreux réglages de l'algorithme de factorisation, comme expliqué à la fin du paragraphe précédent. Dans LocalSolver, on observe que le nombre de non-zéros ne varie jamais de plus d'un facteur 4, et rarement (moins de 14%) de plus d'un facteur 2.

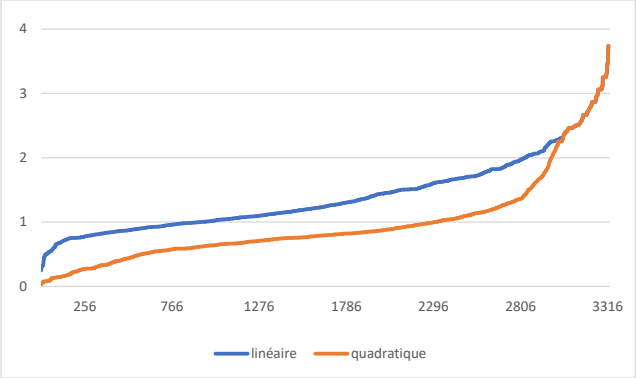

FIGURE 2.1 – *En définissant le nombre de non-zéros comme le nombre d'opérations élémentaires pour évaluer objectif et contraintes, impact de la reformulation sur la taille du modèle (rapport des nombres de non-zéros du modèle factorisé et du DAG). Pour chaque cas (linéaire et quadratique), ces rapports ont été triés par ordre croissants.*

La factorisation consiste à relâcher la définition de la valeur d'un nœud, en remplaçant celui-ci par une nouvelle variable et en transformant les arrêtes entrantes en

une contrainte de couplage entre ses enfants et la variable introduite. Ceci permet d'obtenir des relaxations convexes plus facilement, au prix d'une augmentation du nombre de variables et de contraintes. L'analyse de cette augmentation, résumée dans la figure 2.2, nous permet de faire trois remarques importantes. Premièrement, l'utilisation d'une forme standard plus riche (ici, par l'ajout de contraintes quadratiques) permet de limiter l'augmentation du nombre de variables. C'est une première motivation pour l'exploitation de relaxations non linéaires les plus riches possibles. Ensuite, quelle que soit l'expressivité du modèle factorisé, il existe des instances où le nombre de variables augmente fortement, voire explose. Ceci motive l'hybridation entre l'approche reformulation-convexification utilisée dans la plupart des solveurs globaux, et l'approche par sous-estimation convexe, utilisée par exemple dans le solveur αBB [21], cette dernière ayant l'avantage de ne pas introduire de nouvelles variables ou contraintes. Enfin, comme le montre la courbe correspondant au modèle factorisé quadratique, près de la moitié des instances sont du type MIQCQP. Dans ce cas, la reformulation laisse inchangés les nombres de variables et de contraintes. Ceci motive cette fois l'hybridation avec une approche spécifique, comme c'est par exemple le cas pour le solveur global Antigone [126], construit comme une surcouche du solveur GloMIQO [125], dédié quant à lui aux instances MIQCQP.

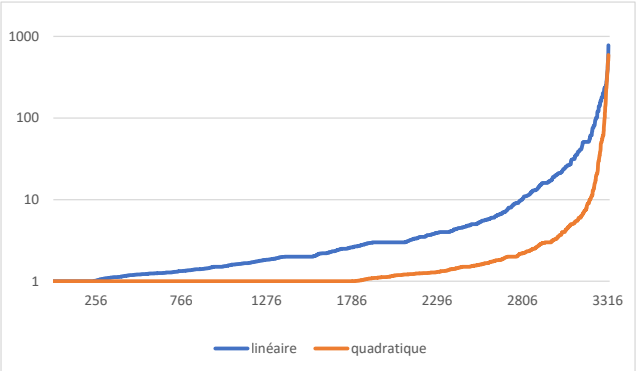

FIGURE 2.2 – *Effet de la reformulation sur le nombre de variables : rapports des nombres de variables du modèle factorisé et du DAG, triés par ordre croissant.*

Malheureusement, une telle approche hybride demande de commencer par l'implémentation de trois solveurs globaux. Si des gains majeurs en termes de performances seraient ainsi obtenus, le travail d'ingénierie nécessaire est important. Aujourd'hui, aucun solveur global n'intègre plusieurs techniques concurrentes complémentaires.

2.1.2 Actions de *presolve*

On appelle action de *presolve* une transformation du modèle qui, au sens large, rend sa formulation plus favorable aux solveurs qui seront utilisés. Le *presolve* consiste alors à exécuter ces actions au début de la résolution, souvent cycliquement jusqu'à l'obtention d'un point fixe. Au même titre que les différentes familles de coupes, les actions de *presolve* sont une source illimitée d'améliorations de per-

formance, et méritent que l'on y consacre des efforts importants. Par exemple, en programmation linéaire en nombres entiers, le *presolve* est considéré comme l'un des modules les plus importants. Dans [51], le *presolve* est le deuxième module ayant le plus d'impact sur la performance de CPLEX 8.0 (le premier étant la génération de coupes), et la désactivation de celui-ci induit une perte de performance d'un facteur 10.8. Dans [18], un résultat similaire est observé pour le solveur GUROBI, avec la remarque supplémentaire que plus les instances sont longues à résoudre, plus l'impact du *presolve* est important.

Si le cas linéaire a été largement étudié et approfondi, il n'en est pas de même pour les autres classes de problèmes. De la littérature existe sur le sujet, mais les expériences numériques restent limitées. Ceci s'explique principalement par le fait que le *presolve* se résume généralement à la détection et à la simplification de sous-structures particulières dans le problème. Or, la géométrie de telles sous-structures devient significativement plus complexe dès que l'on sort du cadre linéaire, tout comme l'implémentation d'actions de *presolve* élémentaires. Par exemple, on peut comparer l'ensemble admissible d'une égalité linéaire à deux variable ($ax+by+c=0$) à celui d'une égalité quadratique à deux variables ($ax^2+by^2+cxy+dx+ey+f=0$), ainsi que la difficulté d'y appliquer une substitution linéaire $x \leftarrow \sum_i c_i z_i$. On peut cependant citer quelques publications qui parlent explicitement du sujet du *presolve* : [86] pour le cas de la programmation quadratique, [80] pour le cas de l'optimisation non linéaire, [35] et [159] pour l'optimisation globale, et [112] pour le cas général.

On peut distinguer trois niveaux de *presolve* dans LocalSolver. Le premier est la phase de *preprocessing* présentée précédemment, et travaille sur le DAG. Le second est l'objet de la fin de cette partie, et travaille sur la forme standard du modèle factorisé. Il est composé de neuf actions appliquées les unes à la suite des autres, jusqu'à ce qu'aucune d'entre elles ne modifie le modèle. Enfin, le troisième est constitué des techniques de réduction de bornes, qui sont l'objet du chapitre 4. L'intérêt d'implémenter une couche de *presolve* une fois le DAG reformulé est que la forme standard obtenue simplifie les raisonnements et l'implémentation des actions.

Actions implémentées

1) Suppression des variables dupliquées : certaines maladresses de modélisation, ainsi que la technique de factorisation du modèle, peuvent induire des couples de variables (x, y) liées par une égalité linéaire $ax = ay$, que nous pouvons résoudre en remplaçant chaque occurrence de y par x.

2) Suppression des variables fixées : par application des techniques de réduction de bornes, ou directement présentes dans le modèle, on rencontre parfois des variables dont le domaine de définition est un singleton $x \in [l_x, l_x]$. Dans ce cas, on peut remplacer chaque occurrence de x par sa valeur numérique.

3) Suppression des contraintes inactives : on appelle contrainte inactive une contrainte qui est toujours satisfaite dans le domaine de définition des variables. Les techniques de propagation de bornes du chapitre 4 permettent de détecter ces situations, où l'on peut alors supprimer la contrainte.

4) Pivots affines : une égalité linéaire $ax + by + c = 0$ touchant deux variables permet parfois de substituer l'une des variables présentes par l'autre. Contrairement

à des pivots linéaires quelconques, le cas à deux variables permet de ne pas augmenter le nombre de non-zéros ni le nombre d'inégalités du modèle.

5) Pivots non linéaires : pour des questions de modélisation plus naturelle, ou par conséquence d'actions antérieures, il est possible qu'une variable x ne soit présente dans le modèle que pour définir un terme non linéaire $y = f(x)$. Si toute valeur de y admet un antécédent par f, on peut éliminer x du problème, et la définir à partir de y une fois celui-ci résolu. Par exemple, dans :

$$\min_{x,y,z} x^2 + y^2 - z^2$$
$$\text{s.c.} \begin{cases} x + y \geq 1, \\ 1 \leq z \leq 2, \end{cases}$$

z n'est présent que pour définir le terme z^2. De plus, $x \mapsto x^2$ est bijective sur $[1, 2]$. On peut alors poser $Z = z^2$, éliminer z du problème

$$\min_{x,y,Z} x^2 + y^2 - Z$$
$$\text{s.c.} \begin{cases} x + y \geq 1, \\ 1 \leq Z \leq 4, \end{cases}$$

et reconstruire la solution optimale en posant $z^* = \sqrt{Z^*}$.

6) Opérateurs min, max et $|\,.\,|$: dans une majorité des cas, ces opérateurs sont utilisés par facilité de modélisation, et seul leur sens convexe est actif. Par analyse de la structure du modèle, on peut parfois remplacer ces contraintes par des inégalités linéaires correspondant au sens convexe de l'opérateur.

7) Variables monotones : si l'on analyse le problème d'optimisation où toutes les variables sont fixes à l'exception de l'une d'entre elles, on peut parfois montrer que quelle que soit la valeur des autres variables, on a toujours intérêt à pousser la variable courante à l'une de ses bornes. On dit alors que la variable est monotone, et la transformer en un singleton permet de réduire le problème. Par exemple, en analysant la monotonie de chaque contrainte et de l'objectif dans

$$\min_{x,y,z \in [0,1]^3} y + z$$
$$\text{s.c.} \begin{cases} z - y \leq 0, \\ z = x^3, \end{cases}$$

on montre que l'on a toujours intérêt à pousser x et z à leurs bornes inférieures. En revanche, on ne peut rien dire sur y, puisque l'on voudrait la diminuer pour minimiser l'objectif, mais l'augmenter pour améliorer la faisabilité de l'inégalité linéaire. Dans cet exemple, on a donc prouvé que $x^* = z^* = 0$.

8) Découpe de contraintes : comme nous le verrons dans le chapitre 3, le solveur non linéaire que nous avons implémenté souffre de la présence de contraintes globales, c'est-à-dire touchant beaucoup de variables. On implémente donc dans ce cas une action qui divise les contraintes en ajoutant d'autres variables et contraintes, jusqu'à ce qu'aucune contrainte de dépasse la taille maximale autorisée.

9) Linéarisation de produits booléens : dans le cas où x et y sont deux variables booléennes, les contraintes $xy = a$ et $xy = z$ peuvent se reformuler à l'aide de contraintes linéaires. C'est aussi le cas si seul x ou y est booléen, mais la linéarisation peut devenir moins serrée que les relaxations convexes que nous développerons plus tard, en particulier après réduction des bornes, et n'est donc pas effectuée.

L'implémentation des actions de *presolve* nécessite la prise en compte de certaines contraintes particulières qui ne rentrent pas dans le formalisme du modèle factorisé, puisqu'elles couplent une variable à elle-même, avec une contrainte de la forme $x = f(x, y)$. Dans ce cas, il faut détecter et résoudre ces situations au fur et à mesure qu'elles se présentent, à l'aide des identités suivantes :

— $x = x^{2k}$ est transformé en $x \in \{0, 1\}$,
— $x = x^{2k+1}$ est transformé en $x \in \{-1, 0, 1\}$,
— $x = x^a,\ a \notin \mathbb{Z}$ est transformé en $x \in \{0, 1\}$,
— $x = \log(x)$ et $x = \exp(x)$ sont inconsistants,
— $x = xy$ est transformé en $(y = 1) \vee (x = 0)$,
— $x = \sin(x)$ est transformé en $x = 0$,
— $x = \cos(x)$ est transformé en $x = 0.739085133215$,
— $x = |x|$ est transformé en $x \geq 0$,
— $x = \max(x, y_1, \cdots, y_k, a)$ est transformé en $x \geq a$ et $\forall i,\ x \geq y_i$,
— $x = \min(x, y_1, \cdots, y_k, a)$ est transformé en $x \leq a$ et $\forall i,\ x \leq y_i$,
— $x = (x \gtreqless a)$ est évaluée puis déclarée inconsistante ou vraie.

Effet du présolve

Comme pour l'analyse de la factorisation du modèle, on termine cette partie par une analyse de l'effet du *presolve*. Les actions implémentées étant élémentaires, on se contente d'une analyse statique, en inspectant la taille du modèle, ainsi que la fréquence d'activation de chaque action. Il est délicat de donner des statistiques moyennes sur le nombre de réductions apportées une action donnée, puisqu'il existe toujours des instances ou celle-ci a un effet majeur (par exemple l'élimination de la moitié des variables). On commence donc par analyser la fréquence d'activation de chaque action, sur les 3244 instances de plus petites tailles de notre *benchmark*. Les résultats sont résumés dans le tableau 2.2.

Action de *presolve*	Cas linéaire	Cas quadratique
Variable dupliquée	20%	15%
Variable fixée	12.9%	10.6%
Contrainte inactive	21%	18%
Pivot non linéaire	8%	3%
Pivot affine	39%	25%
Min, max et $\lvert\,.\,\rvert$	74%	79%
Variables monotones	11%	13%
Contraintes découpées	0%	2%
Produits booléens	17%	2%

Tableau 2.2 – *Fréquence d'activation de chaque action de* presolve *implémentée pour les 3244 instances de plus petite taille.*

Une première remarque concerne les opérateurs min, max et $|.|$: seules 272 des 3244 instances possèdent l'un de ces opérateurs. Dans près de trois quarts d'entre elles, des relations les faisant intervenir ont pu être reformulées par des contraintes linéaires, montrant que le sens non convexe (pour un problème de minimisation) est rarement actif. Ensuite, et malgré la simplicité de chaque action, leur fréquence d'activation est élevée, ou en tout cas plus élevée que ce à quoi nous nous attendions. En effet, seule une moitié des instances n'ont pu être simplifiées et des simplifications peuvent en débloquer d'autres, nécessitant parfois plusieurs dizaines d'exécutions cycliques des actions, comme montré dans le tableau 2.3. Si certaines actions ont pour effet d'augmenter la taille du modèle, comme par exemple la linéarisation des produits booléens, le *presolve* a tendance à la réduire. La figure 2.3 représente le facteur multiplicatif apporté par le *presolve* sur le nombre de non-zéros, pour la moitié des instances dont la taille a été réduite.

Effet du présolve sur la taille du modèle	Cas linéaire	Cas quadratique
La taille diminue	49.6%	45.8%
La taille ne change pas	30.7%	47.7%
La taille augmente	19.7%	6.6%
Le problème est résolu	1.3%	1.5%
Nombre de passes nécessaires au point fixe	Cas linéaire	Cas quadratique
1	30.5%	47.2%
2	48.1%	41.5%
3	17.8%	7.3%
4+	3.4%	2.9%

Tableau 2.3 – *Répartition des instances selon l'effet qualitatif du* presolve *et le nombre d'exécutions cycliques des actions nécessaires à l'atteinte du point fixe.*

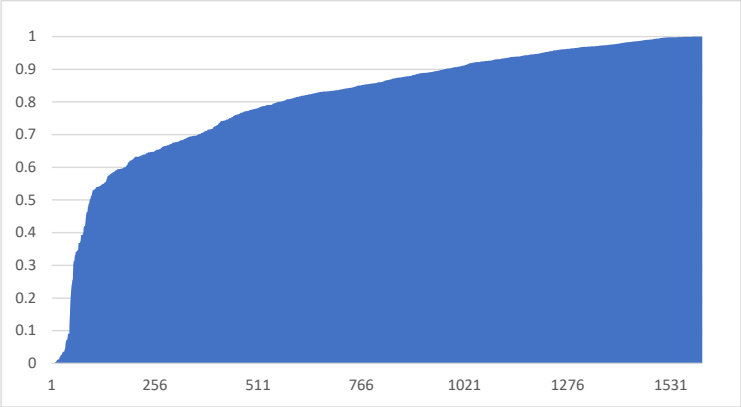

FIGURE 2.3 – *Pour le modèle factorisé linéaire, impact du* presolve *sur la taille du modèle lorsque celle-ci diminue.*

Les réductions obtenues, liées à la technique de factorisation du modèle et à la façon dont les utilisateurs l'ont écrit, illustrent l'importance du *presolve*. C'est un garde-fou contre les maladresses de modélisations, en particulier dans un contexte

industriel et pour un paradigme de modélisation plus fin que celui de la programmation linéaire en nombres entiers, approche encore dominante dans l'industrie. De plus, différents solveurs utilisant différentes techniques de résolution, les modélisations convenant aux uns ne sont pas forcément adaptées aux autres. En particulier, la MINLPLib utilise des conventions de modélisation inadaptées à LocalSolver. Par exemple, l'objectif est transformé en une variable, *via* l'ajout d'une nouvelle variable et d'une contrainte d'égalité. Si ceci permet d'unifier le calcul de bornes inférieures et la réduction de bornes sur les variables et n'est pas un problème si des relaxations linéaires sont utilisées, la recherche locale et le solveur non linéaire voient leur performance dégradée. Un pivot non linéaire permet, en théorie de résoudre le problème, mais l'ajout de bornes artificielles sur la variable représentant l'objectif empêche de le faire de façon automatique.

2.2 Relaxations convexes

La résolution d'une relaxation convexe fournit une borne inférieure au problème dont celle-ci est issue. Si les techniques de propagation de bornes présentées au chapitre 4 permettent aussi d'obtenir une borne inférieure, cette dernière est en général de moins bonne qualité. Les problèmes convexes sont en théorie résolubles en temps polynomial, mais les caractéristiques pratiques qui nous intéressent le plus dans les relaxations convexes sont différentes. En particulier, on peut les résoudre à l'optimum avec un solveur local, puisque les conditions de KKT du premier ordre sont suffisantes. De plus, la théorie de la dualité s'y applique de façon plus favorable que pour des problèmes non convexes.

La structure du modèle factorisé (2.1) simplifie la génération de relaxations convexes. Nous nous concentrons pour le moment sur les couplages non linéaires, et les relaxations convexes des expressions quadratiques seront présentées dans la prochaine section. Les enveloppes convexes des fonctions en présence dans notre forme standard ont déjà été établies dans la littérature, par exemple voir le résumé donné dans le tableau 1 de [126]. Nous les rappelons et les généralisons aux nouveaux opérateurs supportés, tout en présentant quelques précautions qu'il faut prendre pour assurer le bon comportement numérique des relaxations obtenues. La forme standard utilisée pour le modèle factorisé contient neuf types de couplages non linéaires, résumés dans le tableau 2.4.

$y = f(x)$	$y = \|x\|$	$y = (x = a)$
$z = xy$	$y = \max(x_i)$	$y = (x \leq a)$
$xy = a$	$y = \min(x_i)$	$y = (x \geq a)$

Tableau 2.4 – *Les différents couplages non linéaires du modèle factorisé*

2.2.1 Couplages unaires

Pour les couplages unaires, c'est-à-dire qui peuvent s'écrire $y = f(x)$, nous utilisons ce que l'on appelle des estimateurs convexes et concaves. Un sous-estimateur convexe de f sur un intervalle $[\ell_x, u_x]$ est une fonction $\overline{f}$ telle que $\forall x \in [\ell_x, u_x]$, $f(x) \geq \underline{f}(x)$. On définit de façon analogue un sur-estimateur concave de f. Étant donnés un sur-estimateur concave et un sous-estimateur convexe de f, on obtient une relaxation convexe de $y = f(x)$ avec $\underline{f}(x) \leq y \leq \overline{f}(x)$. Il reste donc à choisir les estimateurs les plus serrés possible, auquel cas nous auront obtenu l'enveloppe convexe de l'ensemble admissible du couplage.

Cas où f est convexe ou concave

On s'intéresse premièrement au cas où la convexité de f est constante, par exemple f est convexe sur $[\ell_x, u_x]$. Ceci comprend les couplages du type $y = \mathrm{e}^x$, $y = x^{2k}$ et $y = x^a,\ a > 1$. Dans ce cas, le sous-estimateur est f elle-même, puisqu'elle est convexe. Le meilleur sur-estimateur concave est donné par l'équation de la sécante à la courbe de f aux bornes de x. Finalement, l'enveloppe convexe, illustrée dans la figure 2.5 est donnée par :

$$f(x) \leq y \leq \mathrm{sec}_{f,\ell_x,u_x}(x) = \frac{f(u_x) - f(\ell_x)}{u_x - \ell_x} x + \frac{u_x f(u_x) - \ell_x f(\ell_x)}{u_x - \ell_x}.$$

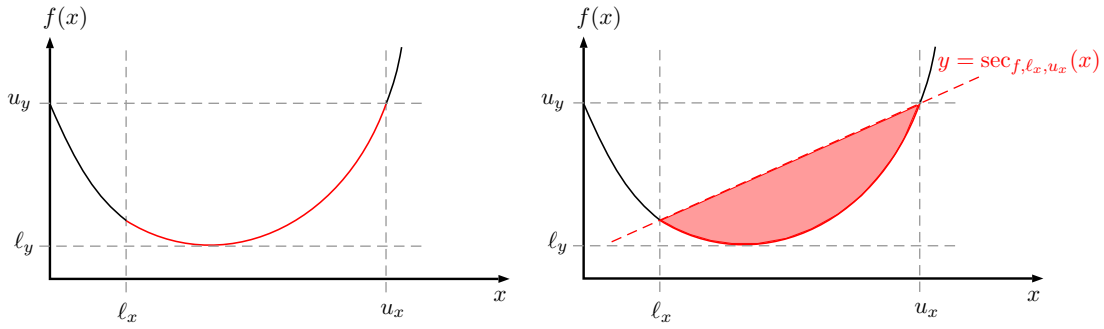

FIGURE 2.4 – *Ensemble admissible de $y = f(x)$ (à droite) et son enveloppe convexe (à gauche) pour une fonction f convexe.*

Dans ce cas où l'on souhaite utiliser des relaxations convexes linéaires, on doit utiliser d'autres sous-estimateurs. Comme f est convexe, toute tangente à sa courbe fournit un sous-estimateur valide. Dans ce cas, on a autant de sous-estimateurs que l'on souhaite, donnés par l'équation suivante :

$$\forall \alpha,\ \mathrm{tan}_{f,\alpha}(x) = f'(\alpha)x + (f(\alpha) - \alpha f'(\alpha)) \leq y.$$

Idéalement, ces inégalités valides sont ajoutées dynamiquement à la relaxation, sous forme de coupes. On utilise plutôt un paramètre variable k, définissant le nombre de contraintes utilisées dans le sous-estimateur. Puisque nous n'utilisons pas de techniques de génération de coupes, le choix a été fait d'utiliser une valeur élevée

de k pour obtenir des relaxations les plus serrées possibles. La valeur retenue est $k = 10$, au-delà de laquelle les gains en bornes inférieures sont trop faibles pour compenser l'augmentation du coût de résolution des relaxations. Une illustration de ces estimateurs linéaires est donnée dans la figure 2.5.

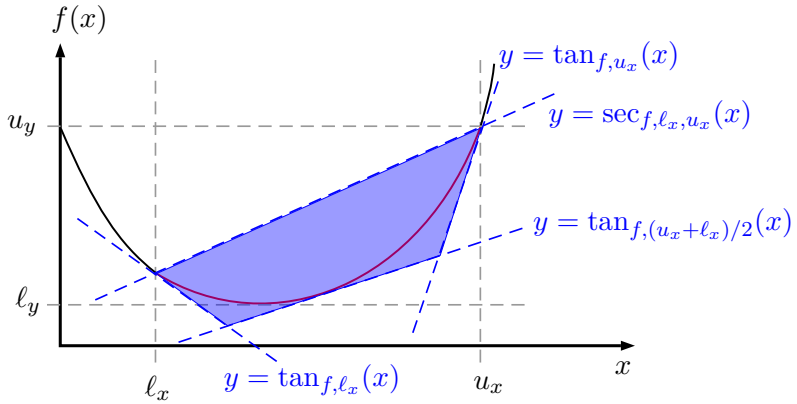

FIGURE 2.5 – *Une relaxation polyédrale de $y = f(x)$, dans le cas où f est convexe et où l'on utilise $k = 3$ tangentes.*

Puissances impaires

Pour les couplages de type puissance impaire, f est convexe sur $\mathbb{R}^+$ et concave sur $\mathbb{R}^-$. Les relaxations convexes sont alors définies à l'aide de deux points L et U, définis comme les uniques solutions de $\tan_{f,L}(x) = \sec_{f,\ell,L}(x)$ et $\tan_{f,U}(x) = \sec_{f,U,u}(x)$. Ces points sont illustrés sur la figure 2.6.

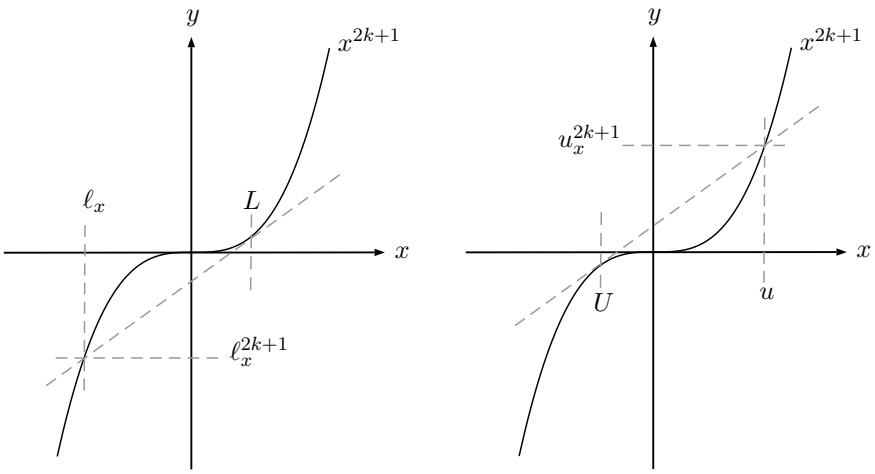

FIGURE 2.6 – *Définition de L et de U.*

Le calcul analytique de L et U est traité dans [114]. Il se ramène à celui de r_k, la plus petite racine réelle d'un polynôme de degré $2k - 1$. La relation entre L, U, ℓ_x, u_x et r_k est homogène : $L = r_k \ell_x$ et $U = r_k u_x$. On peut donc précalculer ces racines

à l'avance, pour les k dont on a besoin, avec une méthode itérative (dichotomie ou Newton). Ainsi, on pourra générer des relaxations convexes quelque soient les bornes de x. L'enveloppe convexe de l'ensemble admissible dépend alors des positions de ℓ_x, u_x, L et U.

Premier cas : $\ell_x > 0$ ou $u_x < 0$.

Dans ce premier cas, nous sommes dans la situation où f est convexe sur le domaine de x, mais pas sur $\mathbb{R}$. On utilise alors la méthode que nous avons expliqué plus haut.

Second cas : $\ell_x < U$ et $u_x > L$.

Ici, la relaxation convexe s'écrit $\underline{f}(x) \leq y \leq \overline{f}(x)$, avec

$$\underline{f}(x) = \begin{cases} \tan_L(x), & x < L, \\ x^{2k+1}, & x \geq L. \end{cases} \quad \text{et } \overline{f}(x) = \begin{cases} x^{2k+1}, & x \leq U, \\ \tan_U(x), & x > U. \end{cases}$$

Comme pour les couplages de convexité constante, les parties non linéaires de cette enveloppe convexe peuvent être remplacées par des tangentes afin d'obtenir une relaxation linéaire. Si l'on considère par exemple les tangentes en ℓ_x et en u_x, la relaxation polyédrale est donnée par les quatre contraintes $y \geq \tan_L(x)$, $y \geq \tan_{u_x}(x)$, $y \leq \tan_U(x)$ et $y \leq \tan_{\ell_x}(x)$. La situation est illustrée dans la figure 2.7

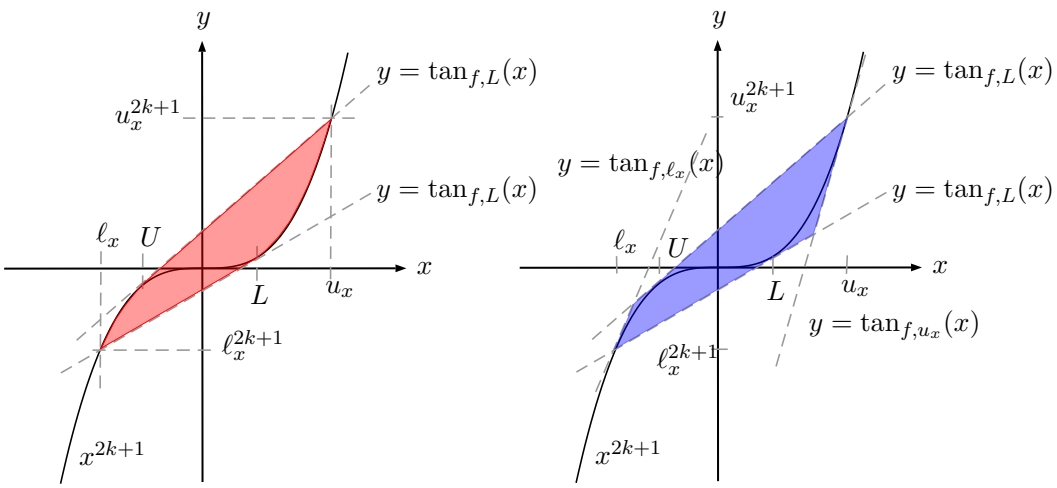

FIGURE 2.7 – *Enveloppe convexe d'un couplage par puissance impaire et approximation polyédrale de celle-ci, dans le cas $\ell_x < U$ et $u_x > L$.*

Troisième cas : $U < \ell_x$ ou $u_x < L$. Les deux cas étant symétriques et mutuellement exclusifs, on se contente de traiter ici le cas $U < \ell_x$ (< 0).

Ici, le sous-estimateur convexe est le même que précédemment. Le sur-estimateur concave peut cependant être amélioré en considérant la sécante à la courbe entre ℓ_x et u_x. Celui-ci devient alors linéaire, et est illustré dans la figure 2.8.

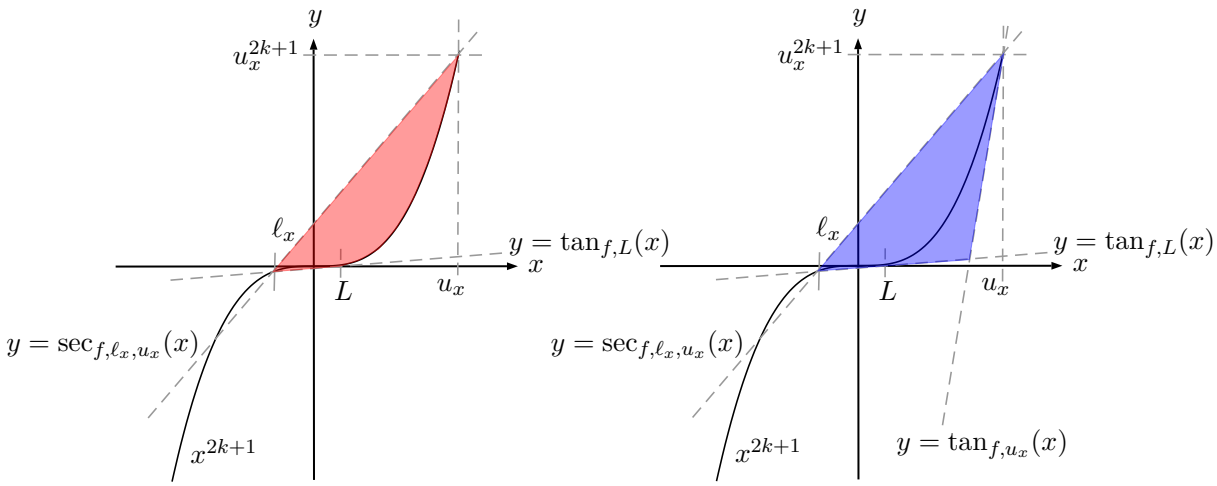

FIGURE 2.8 – *Enveloppe convexe d'un couplage par puissance impaire et approximation polyédrale de celle-ci, dans le cas $\ell_x > U$.*

Unification. Dans le cas où l'on utilise des relaxations non linéaires, les trois cas précédents peuvent être unifiés avec la méthode suivante. On définit

$$\tilde{L} = \begin{cases} L \text{ si } u_x \geq L, \\ +\infty \text{ sinon.} \end{cases} \quad \text{et } \tilde{U} = \begin{cases} U \text{ si } \ell_x \leq U, \\ -\infty \text{ sinon.} \end{cases}$$

ainsi que $\hat{L} = \max(\ell, \tilde{L})$ et $\hat{U} = \min(u, \tilde{U})$. On peut alors résumer les estimateurs convexes des cas précédents avec

$$\underline{f}(x) = \begin{cases} \sec_{f,\ell_x,\min(\tilde{L},u_x)}(x), & x < \hat{L}, \\ x^{2k+1}, & x \geq \hat{L}. \end{cases} \quad \text{et} \quad \overline{f}(x) = \begin{cases} x^{2k+1}, & x \leq \hat{U}, \\ \sec_{f,\max(\tilde{U},\ell_x),u_x}(x), & x > \hat{U}. \end{cases}$$

Couplages trigonométriques

On s'intéresse maintenant aux couplages faisant intervenir les fonctions trigonométriques sinus, cosinus et tangente. La méthode exposée pour calculer l'enveloppe convexe non linéaire ainsi que des relaxations polyédrales pour les contraintes $y = x^{2k+1}$ se généralise au cas de la fonction tangente. Si un pôle est présent dans l'intérieur du domaine de x, soit $]\ell, u[\cap \frac{\pi}{2} + \pi\mathbb{Z} \neq \emptyset$, l'enveloppe convexe est donnée par les contraintes de borne. Dans le cas contraire, les enveloppes convexes s'obtiennent comme pour les puissances impaires.

En utilisant $\cos(x) = \sin(x + \pi/2)$, on peut déduire l'enveloppe de convexe de $y = \cos(x)$ à partir de celle de $y = \sin(x)$ et on se contente de traiter le second cas. La fonction sinus a de nombreux points d'inflexion, mais sa convexité en un point donné est facile à établir : c'est l'opposé du signe de sa valeur. Si $[\ell, u]$ contient zéro ou un point d'inflexion, on peut obtenir l'enveloppe convexe comme précédemment, en utilisant des branchements avec des fonctions linéaires.

Dans le cas général, comme illustré dans la figure 2.9 ci-dessous, l'enveloppe

convexe de $y = \sin x$ sur $[\ell, u]$ peut être complexe, avec de nombreux sous-cas dans la définition du sous-estimateur convexe et du sur-estimateur concave.

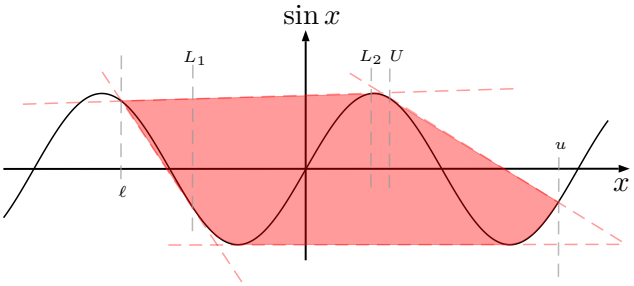

FIGURE 2.9 – *Enveloppe convexe de l'ensemble admissible* $\ell \leq x \leq u$, $y = \sin(x)$ *dans un cas complexe :* 3 *parties dans le sur-estimateur,* 4 *pour le sous-estimateur.*

Puisque les modèles comportant des fonctions trigonométriques sont plutôt rares chez les clients de LocalSolver, on ne calcule pas les enveloppes convexes. On se contente d'ajouter aux contraintes de bornes quatre inégalités linéaires valides, illustrées ci-dessous dans la figure 2.10. Ces inégalités se basent encore sur le calcul (numérique) de points où les sécantes coïncident avec les tangentes.

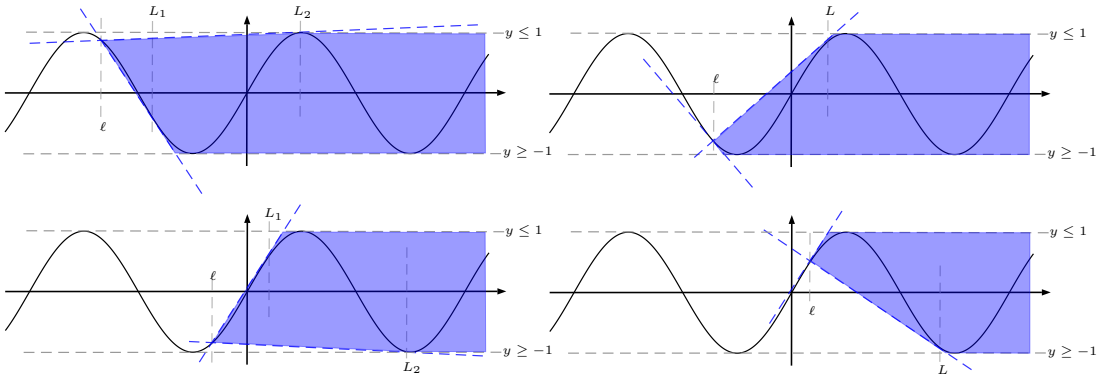

FIGURE 2.10 – *Inégalités valides pour* $y = \sin(x)$, $x \geq \ell$, *selon les* 4 *zones de convexité de la fonction sinus. Les inégalités valides pour* $x \leq u$ *sont analogues.*

2.2.2 Couplages par multiplication ou division

Nous nous intéressons désormais aux contraintes de la forme $z = xy$ et $xy = a$. Suivant les signes des variables en présence, ces multiplications peuvent se reformuler avec des divisions $y = 1/x$ et $z = x/y$, voire en être issues.

Les estimateurs convexes de xy sont connus, puisqu'il s'agit des enveloppes de Mc Cormick [23]. Elles s'obtiennent en multipliant deux à deux les contraintes de bornes sur x et y. Ainsi, l'enveloppe convexe de l'ensemble admissible de $z = xy$ sur

les bornes en présence est donnée par :

$$\begin{cases} z \geq \ell_y x + \ell_x y - \ell_x \ell_y, \\ z \geq u_y x + u_x y - u_x u_y, \\ z \leq \ell_y x + u_x y - \ell_x u_y, \\ z \leq u_y x + \ell_x y - u_x \ell_y, \end{cases}$$

Pour un couplage $xy = a$, on peut encore utiliser les relations de Mc Cormick pour obtenir une relaxation convexe :

$$\begin{cases} \ell_y x + \ell_x y \leq a + \ell_x \ell_y, \\ u_y x + u_x y \leq a + u_x u_y, \\ \ell_y x + u_x y \geq a + \ell_x u_y, \\ u_y x + \ell_x y \geq a + u_x \ell_y. \end{cases}$$

Ces estimateurs peuvent cependant ne pas être serrés, en particulier si les variables sont de signe déterminé. Pour le voir, on inspecte les trois cas possibles pour la nature de la contrainte $xy = a$. Celle-ci peut être une contrainte de complémentarité $(x = 0) \vee (y = 0)$ si $a = 0$, une contrainte unaire de division $y = a/x$ si 0 n'est pas dans le domaine de x, et un couplage bilinéaire $z = xy$ couplé à l'égalité $z = a$ sinon. L'ensemble admissible, ainsi que son enveloppe convexe, est illustré dans chacun de ces les trois dans la figure 2.11.

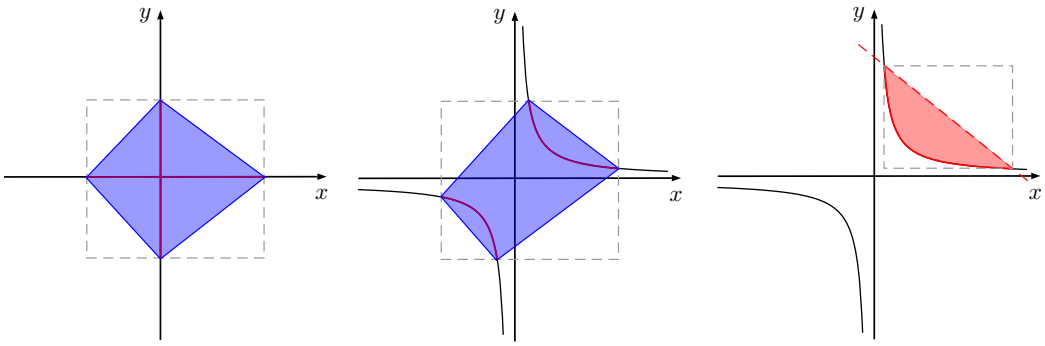

FIGURE 2.11 – *Ensemble admissible et enveloppe convexe de $xy = a$ pour $a = 0$ (à gauche), $a = 1$ et x, y de signe non déterminé (au milieu) et $a = 1$, $x > 0$ et $y > 0$ (à droite).*

Pour les cas $a = 0$ ou x de signe non déterminé, l'enveloppe convexe est linéaire. Dans les autres cas, la contrainte se réduit à un couplage unaire de convexité constante, et est donc relâchée comme telle. Par exemple, si $x > 0$ et $a > 0$, la fonction $y = a/x$ est convexe sur le domaine de x, et les meilleurs estimateurs sont donnés par :

$$\phi(x) \leq y \leq \sec_{\ell_x, u_x}(x),$$

où $\phi(x)$ est défini par un branchement $\mathcal{C}^1$:

$$\phi(x) = \begin{cases} \frac{1}{x}, & x \geq \ell_x, \\ \tan_{f,\ell_x}(x), & x < \ell_x. \end{cases}$$

Comme précédemment, on obtient une relaxation polyédrale de ce dernier cas en utilisant des tangentes à la courbe en tant que sous-estimateurs.

2.2.3 Couplages non différentiables

Les couplages non différentiables, définis par les opérateurs min, max et valeur absolue, ne sont que rarement présents dans les solveurs d'optimisation globale. À notre connaissance, seul LINDO Global [115] les supporte. Les couplages définis par comparaison sont une nouveauté, volontairement supportés car présents dans les modèles des clients de LocalSolver. Généralement, tous ces opérateurs sont linéarisables, et peuvent donc être traités par les autres solveurs par linéarisation. Cette linéarisation est cependant laissée à la charge de l'utilisateur, et a de plus pour conséquence une perte d'information sur la structure du problème.

Fonction valeur absolue

Les couplages de la forme $y = |x|$ sont un cas particulier où la convexité de f est constante. Cependant, le sous-estimateur convexe $|x| - y \leq 0$ n'est pas différentiable en $x = 0$. Ceci pose des problèmes numériques si l'on utilise un solveur non linéaire pour résoudre les relaxations. On utilise donc la forme polyédrale de la relaxation convexe :

$$\begin{cases} y \geq x, \\ y \geq -x, \\ y \leq \sec_{\ell,u}(x). \end{cases}$$

Fonctions min et max

Les couplages min, de la forme $y = \min_i(x_i)$, et max, de la forme $y = \max_i(x_i)$, étant analogues, on se contente ici de traiter le cas du max. Comme pour les valeurs absolues valeurs absolues, le sens convexe du couplage max, $y \geq \max_i(x_i)$, n'est pas différentiable. En revanche, il peut être reformulé par des inégalités linéaires, avec $\forall i,\ y \geq x_i$.

L'autre sens est plus complexe, car non convexe. L'approche habituelle est la linéarisation du couplage, par l'ajout de variables booléennes (b_i), modélisant la saturation du max par chaque argument. Dans ce cas, le sens non convexe s'écrit $y \leq x_i + (U - \ell_i)(1 - d_i)$, avec $\sum_i b_i = 1$ et $U = \max_i u_i$. Une relaxation convexe de ce dernier s'obtient alors en relâchant les contraintes d'intégrité sur les (b_i). Cette approche est néanmoins coûteuse, puisqu'elle ajoute de nombreuses variables booléennes.

Le choix que nous avons fait est de ne pas ajouter de contraintes pour le sens non convexe, contrairement au cas des valeurs absolues. Les violations du couplage par la solution optimale de la relaxation convexe seront alors traitées par des règles de branchement spécifiques, exposées dans le chapitre 5. Une exception est cependant le cas d'un seuillage, c'est-à-dire d'un couplage de la forme $y = \max(x, a)$. Dans ce cas, et contrairement à celui où plusieurs variables sont présentes dans le max, on

peut déterminer l'enveloppe convexe de l'ensemble admissible. Celle-ci s'obtient en utilisant l'identité $\max(a, b) = (|a - b| + (a + b))/2$, et est illustrée dans la figure 2.12.

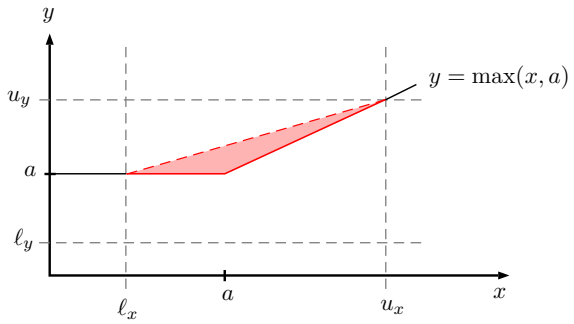

FIGURE 2.12 – *Ensemble admissible et enveloppe convexe du couplage* $y = \max(x, a)$ *sur le domaine des variables.*

Couplages logiques

Pour les couplages de type comparaison, $y = (x = a)$ ou $y = (x \leq a)$, on peut obtenir une reformulation linéaire de l'ensemble admissible. Cependant, cette reformulation dépend des bornes de x, et peut donc être resserrée lors de la recherche. En traçant les ensembles admissibles de ces couplages, on peut obtenir l'enveloppe convexe exacte, comme illustré dans la figure 2.13 ci-dessous.

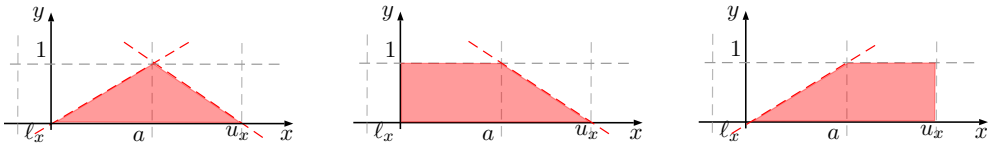

FIGURE 2.13 – *Enveloppe convexe des ensembles admissibles des couplages par comparaison.*

Une note sur l'implémentation

Les relaxations convexes présentées dans cette section sont valides en théorie, ou avec une implémentation en arithmétique exacte. L'utilisation de l'arithmétique flottante usuelle induit des erreurs d'arrondi, et peut mener à des relaxations fausses, qui coupent des points admissibles. Des techniques permettent de surestimer les relaxations afin de prendre en compte les erreurs commises, comme par exemple [55]. L'implémentation de ces techniques augmente cependant la complexité du code, et est peu pertinent dans un contexte industriel, comme expliqué en introduction.

Nous utilisons l'approche consistant à contrôler les erreurs numériques potentielles. Lorsque le coefficient directeur d'une tangente ou sécante dépasse un certain

seuil, la contrainte correspondante est désactivée. De même, la gestion des *overflow* (quand un nombre atteint $+\infty$ en arithmétique flottante) est indispensable. À la place de désactiver les contraintes, il est parfois possible de la déplacer, comme une tangente qui peut être prise en un autre point. Si les bornes des variables en présence sont trop proches, les formules de calcul de sécantes sont instables. Dans ce cas, l'enveloppe convexe est assimilée aux contraintes de bornes. À l'inverse, si des bornes sont trop grandes en valeur absolue, certaines relaxations ne peuvent être obtenues. C'est le cas des fonctions trigonométriques, pour lesquelles le calcul des relaxations passe par un algorithme itératif, de type recherche dichotomique ou méthode de Newton. Pour assurer la stabilité et la convergence de celui-ci, le calcul des inégalités valides est désactivé pour les bornes dont la valeur absolue dépasse un certain seuil.

Enfin, les calculs nécessaires à l'obtention des contraintes valides sont effectués avec une arithmétique exploitant la technique de sommation de Kahan [98], offrant une meilleure précision numérique. Le caractère relâché des inégalités générées est aussi vérifié explicitement, en évaluant la contrainte initiale et les composants de sa relaxation en différents points du domaine (typiquement les bornes et le milieu).

2.3 Traitement des fonctions quadratiques

Dans cette section, on suppose que l'on utilise le modèle sous-jacent quadratique sous contraintes quadratiques (1.8), de la forme

$$\min_{x\in\mathbb{R}^n} \; x^TQx + c^Tx + a$$
$$\text{s.c.} \begin{cases} \ell \leq x \leq u, \\ Ax + b = 0, \\ Cx + d \leq 0, \\ x^TQ_ix + c_i^Tx + a_i \leqq 0, \end{cases}$$

où Q et les Q_i sont des matrices réelles symétriques. Une méthode de relaxation de ce problème, couplée à celles données précédemment pour les couplages non linéaires, permettra d'obtenir une relaxation de (2.1). Nous commençons par un rapide état de l'art des différentes classes de relaxations, puis nous présenterons l'implémentation retenue.

2.3.1 État de l'art

Les enveloppes convexes des couplages non linéaires peuvent être calculés exactement, et c'est aussi le cas pour les expressions quadratiques. En effet, l'enveloppe convexe d'une équation $x^TQx + c^Tx + a = 0$, intersectée avec un polyèdre borné (comme c'est notre cas grâce aux contraintes de bornes), peut s'exprimer à l'aide de contraintes SOCP [148]. Malheureusement, ceci n'est pas applicable en pratique, puisque la reformulation introduit un nombre exponentiel de variables. En fait, la relaxation des expressions quadratiques est nécessairement traitée par un compromis. Les relaxations les plus complexes, celles qui fournissent les meilleures bornes,

sont numériquement coûteuses, et inversement, les méthodes les plus légères, celles qui fournissent les relaxations les plus rapide à résoudre, donnent des bornes de mauvaise qualité. Nous commençons par exposer le cas limite des relaxations les plus serrées et des relaxations les plus rapides à résoudre, pour terminer sur les relaxations intermédiaires les plus intéressantes pour notre compromis.

Relaxations semi-définies [119, 29, 57]

Les relaxations semi-définies consistent à introduire des variables correspondant aux produits entre les variables originelles. Le problème devient alors linéaire avec une unique contrainte non linéaire :

$$\min_{X \in \mathbb{R}^{(n+1)^2}} Q \bullet X$$
$$\text{s.c.} \begin{cases} \mathcal{A}_i \bullet X + \mathcal{B}_i \leq 0, \\ X = \begin{pmatrix} 1 & x^T \\ x & xx^T \end{pmatrix} = \begin{pmatrix} 1 \\ x \end{pmatrix} \begin{pmatrix} 1 \\ x \end{pmatrix}^T . \end{cases}$$

La contrainte non linéaire peut être décomposée en deux contraintes : $X \succcurlyeq 0$, qui est convexe, et $\text{rg}(X) = 1$, qui ne l'est pas. La relaxation semi-définie est alors obtenue en relâchant la contrainte de rang.

Cette approche a été utilisée avec succès dans de nombreuses approches dédiées, et même dans certaines approches génériques. Le solveur MIQCQP BiqCrunch[104] utilise par exemple ces relaxations pour résoudre des problèmes combinatoires en variables booléennes, en exploitant le fait que dans ce cas, $X_{ii} = x_i^2 = x_i$. Il existe aussi des résultats théoriques montrant que les bornes inférieures fournies par la relaxation semi-définie peuvent être d'excellente qualité. Par exemple pour le problème de la coupe maximum d'un graphe, cette relaxation fournit une fameuse borne à moins de 13% de l'optimum global [85].

En pratique, ces relaxations doivent souvent être renforcées à l'aide de contraintes valides pour devenir serrées, comme par exemple pour le problème d'affectation quadratique [173, 143]. Dans ce cas, le nombre de contraintes à ajouter peut exploser et nécessite un oracle de séparation peu coûteux en temps. De plus, la reformulation du problème met au carré le nombre de variables. Le passage à l'échelle des relaxations semi-définies est donc limité par la mémoire requise pour stocker le problème. Enfin, les solveurs semi-définis ne sont pas encore au niveau de robustesse et de performance suffisants pour rendre l'approche compétitive dans le cas général, et ne sont de toutes façons pas capables de traiter les contraintes non linéaires $g(x) \leq 0$ qui sont aussi présentes dans notre relaxation.

Relaxations linéaires [151, 23]

Une relaxation des expressions quadratiques peut être obtenue en ajoutant des variables correspondant aux termes bilinéaires, puis en relâchant les couplages ainsi obtenus avec les enveloppes de Mc Cormick. On peut aussi noter que dans le cas des variables booléennes, cette relaxation est en fait une reformulation. L'intérêt de cette

approche est de fournir des relaxations linéaires, bénéficiant donc de la puissance des solveurs associés. Dans ce cas, il faut aussi utiliser des relaxations linéaires pour les couplages non linéaires. Les bornes ainsi obtenues sont rapides à calculer, mais peuvent être de mauvaise qualité. Il existe alors une large gamme d'outils permettant de les améliorer.

L'approche la plus complète permettant de resserrer les bornes inférieures obtenues utilise ce que l'on appelle la hiérarchie RLT (*reformulation-linearization technique*). L'idée est de multiplier des contraintes linéaires du modèle initial par des variables, obtenant ainsi des contraintes valides portant sur les variables ajoutées. Le même principe s'applique si l'on met une contrainte au carrée ou si l'on multiplie deux contraintes entre elles. Comme pour les relaxations SDP, la taille de la relaxation peut ainsi exploser, et cette approche appelle plutôt à l'utilisation d'un algorithme de type *branch-and-cut*.

On peut aussi améliorer la formulation du modèle en appliquant des étapes de *preprocessing*. Deux exemples sont la désagrégation des produits [156], qui conduit à des relaxations plus serrées, ou encore la génération de contraintes de réduction [109], qui réduisent le nombre de termes bilinéaires.

Enfin, l'exploitation de structures particulières est une source infinie d'améliorations. Par exemple, dans GloMIQO [125], des enveloppes convexes exactes sont calculées pour une classe d'expressions quadratiques appelées *edge-concave* (expressions quadratiques dont les coefficients devant les termes carrés sont négatifs) dans le cas où celles-ci sont de petite dimension. Ceci améliore de façon significative la relaxation par rapport à la linéarisation usuelle de Mc Cormick.

Des développements plus récents [61, 129] remplacent les enveloppes de Mc Cormick par des enveloppes par morceaux, au prix de l'ajout de variables booléennes. La relaxation obtenue est alors linéaire en nombres entiers, mais fournit une meilleure borne et exploite au mieux la puissance des solveurs linéaires en nombres entiers.

Relaxations quadratiques convexes [123, 46, 45]

La méthode classique pour convexifier une fonction quadratique sans ajouter de variable ni de contraintes utilise les αBB sous-estimateurs. Cette approche a été introduite dans [91] pour le cas des variables booléennes, puisque dans ce cas, c'est une reformulation exacte du problème. L'idée est d'appliquer à chaque fonction quadratique une perturbation diagonale, négative sur le domaine des variables, qui la rend convexe. Par exemple, pour $q(x) = x^T Q x + c^T x + a$ et $\ell \leq x \leq u$, on utilise le sous-estimateur suivant, dépendant d'un paramètre $\alpha \in \mathbb{R}^n$:

$$q_\alpha(x) = q(x) + \sum_i \alpha_i (x_i - \ell_i)(x_i - u_i) \leq q(x),$$

qui revient à ajouter à Q une perturbation diagonale $\mathrm{diag}(\alpha)$. On cherche alors α tel que $Q_\alpha = Q + \mathrm{diag}(\alpha) \succcurlyeq 0$, et tel que la borne de la relaxation obtenue soit la meilleure possible.

De nombreuses méthodes sont alors proposées pour calculer un vecteur de perturbation α. L'idée la plus naturelle consiste à utiliser une perturbation uniforme,

qui revient à translater toutes les valeurs propres. Dans ce cas, α est donné par $\alpha_i = \max(0, -\lambda_{\min}(Q))$, et on a besoin de calculer la plus petite valeur propre de Q pour obtenir la relaxation convexe. Une autre approche est d'utiliser le théorème de Gershgorin [81], et consiste à calculer α tel que Q_α soit diagonale-dominante. Dans ce cas, puisque Q est symétrique, on utilise $\alpha_i = \max\left(0, -|q_{ii}| + \sum_{j\neq i} |q_{ij}|\right)$. Cette seconde perturbation est moins coûteuse à calculer, puisque que l'on peut l'obtenir en un unique parcours de Q. Ces deux méthodes ne sont en général pas comparables, et aucune ne semble dominer l'autre.

L'obtention de la perturbation optimale, dans le sens où celle-ci maximise la borne inférieure obtenue, a été étudié dans [46] : il s'agit de l'approche QCR (*quadratic convex reformulation*). Si l'on note v_α la valeur de la relaxation continue lorsque l'on utilise la perturbation α, l'objectif est de résoudre

$$\text{(DSDP)} \quad \max \; v_\alpha$$
$$\text{s.c.} \begin{cases} \alpha \geq 0, \\ Q + \operatorname{diag}(\alpha) \succcurlyeq 0. \end{cases}$$

Dans [46], ce problème est interprété comme le dual d'un programme semi-défini, qui est alors résolu pour obtenir α^*. Cette approche est ensuite améliorée en considérant des termes de convexification non diagonaux lorsque des contraintes linéaires sont en présence. Si l'on a $Ax + b = 0$ et $L \leq Cx \leq U$, la fonction de convexification suivante est considérée :

$$\phi(x) = \sum_i \alpha_i (x_i - \ell_i)(x_i - u_i) + \sum_j \beta_j \left(a_j^T x + b_j\right)^2 + \sum_k \gamma_k \left(c_k^T x - L_k\right)\left(c_k^T x - U_k\right).$$

De la même manière, la résolution d'un programme SDP permet d'obtenir α^*, β^* et γ^*. Pour les mêmes raisons qu'expliqué lors du paragraphe sur les relaxations semi-définies, cette approche se limite aux problèmes de petite et moyenne taille.

Autres relaxations [52]

Nous terminons cet état de l'art par des relaxations intermédiaires, ajoutant des variables et des contraintes, mais en restant quadratiques. Ces approches sont notamment utilisées dans le solveur commercial CPLEX [52]. L'idée est de décomposer chaque fonction en termes convexes et non convexes. La première approche est la reformulation *Q-space* : on écrit $Q = P + \tilde{Q}$, où $P \succcurlyeq 0$ et est généralement diagonale ou bloc-diagonale. $\tilde{Q}$ est ensuite linéarisée par ajout de variables et utilisation des enveloppes de Mc Cormick.

L'autre approche de ce type est appelée factorisation des vecteurs propres. Après diagonalisation de Q en $x^T Q x = y^T D y$ avec $y = Lx$ et D diagonale, le nouvel objectif est partitionné en $D = D^+ - D^-$, avec $D^+ \succcurlyeq 0$ et $D^- \succcurlyeq 0$. Les termes convexes D^+ sont conservés, et ceux de D^- sont relaxés avec la formule de la sécante.

2.3.2 Convexification

L'approche que nous avons retenue utilise les sous-estimateurs quadratiques de type αBB. La motivation principale pour ceci est le fait que nous souhaitons utiliser

un solveur non linéaire pour résoudre les relaxations. Dans ce cas, ne pas ajouter de variables ou de contraintes est un avantage important et fournit des relaxations compactes. On se concentre sur une fonction quadratique non convexe pouvant être l'objectif ou une contrainte, et notée $q(x) = \frac{1}{2}x^T Qx + c^T x + a$.

L'objectif de cette partie est de trouver $\alpha \in \mathbb{R}^+_n$ tel que le sous-estimateur $q_\alpha(x) = q(x) + \sum_i \alpha_i(x_i - \ell_i)(x_i - u_i)$ de Q soit convexe. L'erreur de convexité induite par ϕ_α est dans le pire cas $\frac{1}{4}\sum_i \alpha_i(u_i - \ell_i)^2$, que l'on peut chercher à minimiser pour guider la recherche de termes de convexification de meilleure qualité. Nous détaillons les différentes méthodes élémentaires permettant d'atteindre cet objectif, ainsi que les pistes envisagées pour des méthodes plus avancées, qui ont conduit à une méthode heuristique rapide et efficace.

Méthodes élémentaires

Les méthodes les plus simples pour obtenir un terme de convexification valide utilisent des conditions suffisantes de convexité. Le théorème de Gershgorin nous dit par exemple qu'une matrice à diagonale dominante est semi-définie positive. Ce résultat est exploité depuis longtemps en optimisation globale, puisqu'il est l'une des méthodes de base du solveur αBB [21, 22]. Son utilisation permet d'obtenir le terme de convexification suivant :

$$\alpha_i = \max\left(0, -q_{ii} + \sum_{j \neq i} |q_{ij}|\right). \tag{2.2}$$

L'intérêt principal de cette approche est qu'elle est peu coûteuse en temps de calcul, puisque le terme de convexification est obtenu en un unique parcours des non-zéros de la matrice. Une amélioration notable consiste à remarquer que des changements de variables linéaires ne modifient pas la convexité d'une fonction quadratique. Cette approche fournit ainsi toute une famille de termes de convexification valides, avec

$$\alpha_i = \max\left(0, -q_{ii} + \sum_{j \neq i} \frac{s_j}{s_i}|q_{ij}|\right), \tag{2.3}$$

où les (s_i) forment un vecteur d'éléments strictement positifs, qui jouent le rôle de mise à l'échelle des variables. Le choix typique pour s_i est d'utiliser la taille des domaines : $s_i = u_i - \ell_i$. Ce choix permet de prendre en compte le fait qu'une variable de grand domaine induira plus d'erreur de convexité qu'une de petit domaine, à termes de convexification égaux. Cette amélioration est aussi utilisée dans αBB, et autorise une mise à jour rapide du terme de convexification lorsque des variables voient leurs bornes réduites, lors du branchement par exemple.

L'autre approche élémentaire utilise la propriété de translation des valeurs propres sous une perturbation diagonale uniforme. Plus précisément, si l'on note (λ_i) les valeurs propres de Q, triées par ordre croissant, on a :

$$\lambda_i(Q + \sigma I) = \lambda_i(Q) + \sigma.$$

Une condition de convexité de $Q + \sigma I$ est alors $\sigma \geq -\lambda_0(Q)^- = -\lambda^-_{\min}(Q)$, ce qui donne un dernier terme de convexification valide avec :

$$\forall i,\ \alpha_i = \max(0, -\lambda_{\min}(Q)). \tag{2.4}$$

La difficulté de cette approche est qu'elle nécessite de calculer la plus petite valeur propre de Q. Les résultats d'algèbre linéaires que nous utilisons sont classiques, voir par exemple [145]. Pour obtenir $\lambda_{\min}$, l'approche naturelle est d'effectuer une première translation $Q_\sigma = Q + \sigma I$, avec σ choisi tel que toutes les valeurs propres de Q_σ soient de même signe, puis de s'intéresser aux valeurs propres de Q_σ, puisque l'on peut en déduire celles de Q. Si $\sigma < -\lambda_{\max}(Q)$, les valeurs propres de Q_σ sont négatives, et la plus petite (qui est alors la plus grande en valeur absolue) peut être calculée avec la méthode de la puissance. Le problème est ici que la translation par σ a pour effet de détériorer la vitesse de convergence de la méthode (linéaire, proportionnelle au rapport des deux plus grandes valeurs propres en valeur absolue). Si $\sigma > -\lambda_{\min}(Q)$, les valeurs propres de Q_σ deviennent positives, et on peut rechercher la plus petite à l'aide de la méthode de la puissance inverse (qui nécessite une unique factorisation de Q_σ), ou par la méthode du quotient de Rayleigh (qui nécessite une factorisation par itération).

On souhaite obtenir une méthode qui se passe complètement de factorisations, pour des questions de passage à l'échelle. On utilise donc les deux résultats suivants : premièrement, la plus petite valeur propre de Q vérifie

$$\lambda_{\min}(Q) = \min_{x \in \mathbb{R}^n} \frac{x^T Q x}{\|x\|^2},$$

c'est-à-dire qu'elle minimise le quotient de Rayleigh de Q. Ensuite, un algorithme de gradient appliqué à ce dernier à partir d'un point initial aléatoire converge (en valeur, mais pas nécessairement en solution) avec probabilité un [1]. Finalement, nous avons implémenté une méthode de gradient conjugué à pas optimal pour minimiser le quotient de Rayleigh. La méthode est rapide et robuste, puisque nous n'avons jamais constaté de convergence vers un minimum local (par exemple, en vérifiant les valeurs propres calculées avec des outils fiables tels que LAPACK [25]). De plus, son implémentation bénéficie de notre expérience en optimisation sans contrainte, acquise lors de l'implémentation des techniques présentées au chapitre 3, en particulier pour le réglage des conditions d'arrêt.

Méthodes avancées

Au-delà des conditions suffisantes, on peut chercher à obtenir une convexification qui donne la meilleure relaxation possible. L'approche QCR [123, 46, 45] résout ce problème, en le formulant comme un programme semi-défini. Le calcul du α donnant la meilleure borne parmi toutes les perturbations valides est cependant trop coûteuse en temps pour les instances de grande taille.

1. Le résultat de convergence d'une descente de gradient sur le quotient de Rayleigh n'est pas évident, puisque celui-ci n'est pas convexe. Pour le montrer, on utilise un premier résultat de [93]. En notant $(E_i)_{1 \leq i \leq r}$ les sous-espaces propres de Q, et pour un point initial $x_0 = \sum_{i=1}^{r} a_i y_i$, avec $y_i \in E_i$, une descente de gradient à pas optimal converge vers $\lambda = \min\{\lambda_i \mid a_i \neq 0\}$. Ainsi, un échec de la convergence vers la plus petite valeur propre signifie que le point initial avait une projection nulle sur le sous-espace propre associé, et qu'il appartenait donc au complémentaire de ce sous-espace. Ce complémentaire étant de dimension strictement plus petite que la dimension de l'espace complet, il est de mesure nulle. La probabilité qu'un point x_0, choisi de façon aléatoire (par exemple de façon uniforme dans une boule unité), donne lieu à un échec de la convergence est finalement 0.

Notre approche consiste alors à décomposer la convexification en plusieurs étapes, ou plusieurs itérations. Pour une perturbation α (pas nécessairement valide), on mesure l'erreur de convexité déjà commise et celle restant à ajouter si l'on termine par une convexification avec la plus petite valeur propre. En notant $g_i = (u_i - \ell_i)^2/4$ et $G = \sum_i g_i$, et en supposant $\lambda_{\min}(Q_\alpha) < 0$ cette erreur s'exprime avec

$$\mathcal{G}(\alpha) = \sum_i (\alpha_i - \lambda_{\min}(Q_\alpha)) g_i = \alpha^T g - \lambda_{\min}(Q_\alpha) G.$$

Une itération de convexification chercherait donc à atteindre $Q_\alpha \succcurlyeq 0$, tout en minimisant $\mathcal{G}(\alpha)$.

La première approche que nous avons considérée s'inspire d'une amélioration apportée à la méthode QCR, dans [45]. En effet, lorsque les programmes semi-définis sont de trop grande taille, ils sont résolus avec une méthode de faisceaux spectraux, exploitant les propriétés analytiques des fonctions valeurs propres. Ces propriétés permettent de montrer que $\mathcal{G}$ est sous-différentiable, et d'obtenir tous ses sous-gradients. Pour cela, on rappelle la propriété suivante : $h : S_n(\mathbb{R}) \to \mathbb{R}$, $X \mapsto \lambda_{\min}(X)$ est convexe et son sous-différentiel en X est :

$$\partial h(X) = \mathrm{Conv}\left\{ -xx^T \mid x \in E_{\lambda_{\min}}(X) \right\},$$

où $E_{\lambda_{\min}}(X)$ est le sous-espace propre de X associé à sa plus petite valeur propre. En particulier, h est différentiable en X si et seulement si $\lambda_{\min}(X)$ est simple, et tout vecteur propre associé à $\lambda_{\min}(X)$ fournit un sous-gradient de h. Les règles de calcul différentiel nous permettent alors d'obtenir que $\mathcal{G}$ est sous-différentiable et que

$$\partial \mathcal{G}(\alpha) = \left\{ g + Gv \mid v \in \mathrm{Conv}\left\{ (x_1^2, \cdots, x_n^2) \mid x \in E_{\lambda_{\min}}(Q_\alpha) \right\} \right\}.$$

En particulier, la donnée d'un vecteur propre x associé à $\lambda_{\min}(Q_\alpha)$ donne un sous-gradient de $\mathcal{G}$:

$$\frac{g}{G} - x \circ x \in \partial \mathcal{G}(\alpha),$$

où $x \circ x = (x_i^2)$ désigne le produit de Hadamard.

Cette approche nous permet d'appliquer une méthode de sous-gradient pour minimiser $\mathcal{G}$, suivie d'une convexification avec la plus petite valeur propre. Malheureusement, plusieurs problèmes numériques se posent avec cette approche. Premièrement, les méthodes de sous-gradient convergent lentement, nécessitant de nombreuses itérations (qui coûtent ici un calcul de plus petite valeur propre). De plus, des choix génériques de pas de déplacement sont rares, et demandent l'appel à une recherche linéaire lors de la première itération. Nous avons donc le choix entre des pas de déplacement généralement trop petits et l'utilisation d'une recherche linéaire, qui s'avère trop coûteuse.

Une heuristique itérative

Si une méthode de descente simple n'est pas performante, elle nous donne tout de même une information qualitative. En effet, une itération de minimisation de $\mathcal{G}$ modifie les perturbations α suivant la contribution de chaque coordonnée de x dans

la plus petite valeur propre. De plus, seules les coordonnées non nulles de x induisent une itération qui augmente les perturbations correspondantes.

Pour remédier aux faiblesses numériques de la méthode de sous-gradient, on s'intéresse aux conditions nécessaires de convexité. Soit une perturbation α, un vecteur propre normalisé x associé à $\lambda_{\min}(Q_\alpha) < 0$ et δ la modification des perturbations que nous souhaitons effectuer pour obtenir $Q_{\alpha+\delta} \succcurlyeq 0$. Une condition nécessaire à cette dernière inégalité est $x^T Q_{\alpha+\delta} x \geq 0$, soit après simplifications :

$$\sum_i \delta_i x_i^2 \geq \lambda_{\min}(Q_\alpha).$$

On peut alors chercher une solution à cette contrainte qui minimise l'erreur de convexité supplémentaire induite par δ : $\sum_i \delta_i g_i$. Pour simplifier les raisonnements et notations, on suppose (quitte à effectuer un changement de variables) que $g_i = 1$. Le programme linéaire obtenu est alors de type sac-à-dos, dont la solution optimale est donnée par

$$\delta_m = -\frac{\lambda_{\min}(Q_\alpha)}{x_m^2},$$

pour un indice m satisfaisant $x_m^2 = \max_i x_i^2$, et $\delta_i = 0$ pour les autres coordonnées. Autrement dit, cette approche compense la non-convexité de Q_α avec une seule coordonnée, en y appliquant une perturbation qui peut être grande devant la plus petite valeur propre courante. Appliquée itérativement, cette méthode n'est pas compétitive : elle donne lieu à des relaxations mal conditionnées, et à des bornes inférieures de mauvaise qualité (comparées à celles de la méthode de la plus petite valeur propre par exemple). On tente alors de corriger le problème en ajoutant des contraintes du type $\delta_i \leq k(-\lambda_{\min}(Q_\alpha))$, qui interdisent des perturbations trop importantes relativement à ce qu'aurait fait la méthode de la plus petite valeur propre. Pour donner la nouvelle solution optimale, on suppose que les (x_i) sont triés par valeurs absolues décroissantes. On a alors

$$\delta_1 = \cdots = \delta_{K-1} = -\frac{\lambda_{\min}(Q_\alpha)}{\sum_{i=1}^K x_i^2}, \quad \text{et} \quad \delta_{K+1} = \cdots = \delta_n = 0,$$

où $K = \min\{q \mid \sum_{i=1}^q x_i^2 \geq \frac{1}{k}\}$ et δ_K est choisi de façon à saturer l'inégalité. Borner l'augmentation des perturbations diagonales revient donc à répartir la convexification sur les coordonnées y participant le plus.

Finalement, l'analyse qualitative des deux approches que nous venons de voir nous apprend plusieurs choses. Premièrement, la contribution de chaque variable à la plus petite valeur propre doit être prise en compte dans la modification de la perturbation diagonale. Ensuite, de nombreux schémas itératifs peuvent être déduits de conditions nécessaires ou suffisantes de convexité (par exemple, injecter le sous-gradient obtenu précédemment dans une condition nécessaire pour obtenir un pas de déplacement). En cherchant une méthode qui donne à la fois des relaxations bien conditionnées, et qui ne nécessite pas plus d'une dizaine ou vingtaine d'itérations pour converger, nos expérimentations numériques ont abouti à la règle de mise à jour suivante :

$$\delta_i = -\lambda_{\min}^-(Q_\alpha)\frac{|x_i| - x^{\text{med}}}{x^{\max} - x^{\text{med}}} \in [0, -\lambda_{\min}^-(Q_\alpha)],$$

où $x_{\max}$ désigne le maximum et x_{med} la médiane des valeurs absolues des coordonnées non nulles de x. Le calcul du quatrième et dernier terme de convexification est alors résumé dans l'algorithme 1.

Algorithme 1 Heuristique de convexification minimisant l'erreur de convexité.

fonction CONVEXIFY()
  Soit $\alpha = 0$, $k = 0$;
  Calculer $\lambda_{\min}(Q_\alpha)$ et x un vecteur propre normalisé associé ;
  tant que $\lambda_{\min}(Q_\alpha) < 0$ et $k \leq 20$ **faire**
    $\alpha_i \leftarrow \alpha_i - \lambda_{\min}(Q_\alpha)\frac{|x_i| - x^{\text{med}}}{x^{\max} - x^{\text{med}}}$;
    Calculer $\lambda_{\min}(Q_\alpha)$ et x un vecteur propre normalisé associé ;
    $k \leftarrow k + 1$;
  fin tant que
  $\alpha_i \leftarrow \alpha_i + \max(0, -\lambda_{\min}(Q_\alpha))$;
fin fonction

Cet algorithme correspond au cas où les (g_i) sont uniformes, et est donc naturel sur les instances où les domaines des variables sont de même tailles. Il pourrait être modifié pour prendre en compte le cas général. Cependant, puisque nous sommes en train de faire deux choses en même temps (approcher la convexité et maintenir l'erreur induite la plus petite possible) sous la contrainte d'un faible nombre d'itérations, cela nécessite des expérimentations numériques conséquentes pour généraliser notre règle de mise à jour, expérimentations que nous n'avons pas eu l'occasion de faire.

Comparaison numérique des stratégies de convexification

Pour chaque expression quadratique, nous avons quatre termes de convexification valides, résumés dans le tableau 2.5. On définit aussi la stratégie virtuelle BEST, définie sur chaque instance comme la meilleure, en nombre de nœuds nécessaire à la résolution, des quatre stratégies de convexification.

Stratégie	Formule	Calcul
LMIN	(2.4)	*root node*
HEUR	Algorithme 1	*root node*
DIAG	(2.2)	*root node*
SDIAG	(2.3)	à chaque nœud

Tableau 2.5 – *Les quatre stratégies de convexification utilisées.*

Pour analyser la performance des différentes stratégies, on considère les 952 instances de plus petite taille de notre benchmark où la convexification d'une fonction quadratique a lieu. On exécute alors le module dual de LocalSolver avec une durée maximale de 30 secondes, avec chacune des stratégies décrites ci-dessus. Le tableau

2.6 présente les résultats qualitatifs des différentes stratégies, et nous apprend deux choses importantes : la stratégie de convexification a peu d'impact sur le nombre d'instances résolues, et aucune stratégie ne domine ou n'est dominée par les autres.

Stratégie	LMIN	HEUR	DIAG	SDIAG	BEST
Instances résolues	593	610	574	604	633
Seul à résoudre	4	14	1	16	
Seul à ne pas résoudre	2	1	6	4	

Tableau 2.6 – *Impact de la stratégie de convexification sur les instances résolues à l'optimum global, sur un* benchmark *de 952 instances.*

Si les instances résolues par les différentes stratégies sont globalement les mêmes, ce n'est pas le cas du temps de résolution de ces instances. Pour le voir, on s'intéresse aux 560 instances résolues par les quatre stratégies, auxquelles on retire les instances où le nombre de nœuds est le même, nous laissant 380 instances à analyser. Le tableau 2.7 donne le nombre moyen de nœuds nécessaires à la résolution à l'optimum, calculé sous la forme d'une moyenne géométrique translatée de 10 nœuds, soit pour une série (x_i) de n entrées :

$$N = \prod_{i=1}^{n} (x_i + 10)^{1/n} - 10.$$

L'utilisation d'une moyenne géométrique permet de limiter la participation des plus grandes entrées, et la translation permet de limiter celle des plus petites entrées. En particulier, quand les instances sont résolues rapidement, gagner ou perdre un facteur 2 a moins d'importance. La valeur de la translation, ici de 10 nœuds, reste cependant arbitraire.

	LMIN	HEUR	DIAG	SDIAG	BEST
Meilleure stratégie	50	115	102	235	380
Nombre de nœuds moyen	276	180	236	152	115

Tableau 2.7 – *Comparaison de la taille de l'arbre de branchement selon la stratégie de convexification, sur les 380 instances où celle-ci a un impact (c'est-à-dire les instances où le nombre de nœuds diffère pour au moins deux stratégies).*

Cette fois-ci, les différences de performance entre les stratégies sont plus nettes. Par exemple, SDIAG nécessite en moyenne presque deux fois moins de nœuds que LMIN. On note aussi que notre heuristique itérative HEUR fournit une amélioration notable par rapport à LMIN, puisque l'on a une réduction moyenne du nombre de nœuds de 35%. De plus, SDIAG n'apporte en moyenne qu'une réduction de 16% du nombre de nœuds par rapport à HEUR. Le tableau 2.7 montre ainsi que la stratégie SDIAG est la meilleure stratégie individuelle, car elle est la meilleure dans 62% des cas et la meilleure en moyenne. Malgré son apparente simplicité, elle est difficile à battre, y compris avec des méthodes significativement plus complexes,

comme HEUR. Des remarques analogues ont déjà été effectuées dans différentes publications, comme par exemple [152].

2.3.3 Améliorations

Si les différentes stratégies de convexification permettent de résoudre les mêmes problèmes, les variations de temps de résolution peuvent être importantes. Nous terminons cette section par trois améliorations apportées aux convexifications que nous venons de présenter.

Pénalisation des contraintes linéaires dans l'objectif

En présence de contraintes d'égalité linéaires, pénaliser ces dernières dans l'objectif a pour effet d'y ajouter un terme convexe nul sur l'ensemble admissible. Ceci constitue alors une relaxation du problème, et peut réduire les non-convexités de Q. Cette idée a été introduite dans l'approche QCR [123]. Si l'on note $Ax + b = 0$ les contraintes d'égalité, ajouter $\frac{1}{2}\rho\|Ax+b\|^2$ à $q(x)$ ne change pas l'ensemble admissible ni la valeur de q sur celui-ci. Ceci revient à remplacer Q par $Q + \rho A^T A$, c'est-à-dire à ajouter un terme convexe à Q. Malheureusement, comme nous le verrons à la fin du développement, cette approche ne permet pas d'améliorer les convexifications DIAG et SDIAG est donc limitée au stratégies LMIN et HEUR. Dans [123], le facteur de pénalisation ρ est obtenu par résolution d'un programme semi-défini. Cette approche ne passant pas à l'échelle, on utilise une approche heuristique. Pour cela, on commence par donner quelques propriétés liées à la fonction objectif pénalisée, notée Q_ρ.

Premièrement, $\rho \mapsto \lambda_{\min}(Q_\rho)$ est croissante, comme prévu, mais aussi bornée :

$$\forall\ 0 \leq \rho_1 \leq \rho_2,\ \lambda_{\min}(Q) \leq \lambda_{\min}(Q_{\rho_1}) \leq \lambda_{\min}(Q_{\rho_2}) \leq \bar{\lambda},$$

où $\bar{\lambda} = \lambda_{\min}(Z^T Q Z)$, avec Z une base orthonormale de $\ker A$. Ainsi, pénaliser les égalités linéaires permet d'annuler toutes les non-convexités de Q présentes dans le sous-espace $\ker(A)^\perp$. De plus, si l'on utilise une perturbation uniforme $\alpha_i = -\bar{\lambda}$, Q_α devient convexe dans l'ensemble admissible $Ax + b = 0$. Le calcul de la matrice Z n'est cependant pas acceptable dans notre cas, puisque qu'elle est généralement pleine, même si Q est creuse.

Une seconde propriété qui nous intéresse est que :

$$\exists \rho^*, \forall \rho \geq \rho^*, \lambda_{\min}(Q_\rho) = \lambda_{\min}(Q_{\rho^*}) = \bar{\lambda}.$$

Il n'est donc pas nécessaire de faire tendre le facteur de pénalisation vers $+\infty$ pour atteindre la valeur optimale : il suffit de prendre ρ suffisamment grand. Ceci pose un autre problème, en particulier lors de la résolution de la relaxation convexe. Si $\rho \mapsto \lambda_{\min}(Q_\rho)$ est croissante et bornée, ce n'est pas le cas de $\rho \mapsto \lambda_{\max}(Q_\rho)$, puisque l'on a

$$\lambda_{\max}(Q_\rho) \underset{\rho \to +\infty}{\sim} \rho \lambda_{\max}(A^T A).$$

Ceci implique que le conditionnement de Q_ρ devient arbitrairement mauvais. Dans ce cas, la résolution des sous-problèmes devient plus difficile, car plus longue avec

des méthodes itératives et moins précise avec des méthodes directes. Il faut donc trouver un équilibre entre réduction maximale des non-convexités et détérioration du conditionnement.

L'heuristique utilisée commence par calculer les valeurs propres extrêmes de Q : $\text{sp}(Q) \subset [\lambda_{\min}, \lambda_{\max}]$ et de A^TA : $\text{sp}(A^TA) = [0, \sigma_{\max}]$. On utilise alors une pénalisation de la forme

$$\bar{\rho} = \omega \frac{\lambda_{\max} - \lambda_{\min}}{2\sigma_{\max}}, \tag{2.5}$$

où ω est un facteur de proportionnalité déterminé numériquement. L'idée derrière cette formule est un équilibre entre convexification et conditionnement. Pour la justifier, on utilise les formules de Weyl [164]. Pour convexifier q, on souhaite que $\lambda_{\min}(Q + \rho A^TA) \geq 0$. Or, les formules de Weyl donnent

$$\lambda_{\min}(Q + \rho A^TA) \leq \lambda_{\min}(Q) + \rho\lambda_{\max}(A^TA) = \lambda_{\min} + \rho\sigma_{\max}.$$

Ceci donne alors la condition nécessaire $\rho \geq \rho_1 = \dfrac{-\lambda_{\min}}{\sigma_{\max}}$.

On inspecte ensuite le conditionnement κ de $Q+\rho A^TA$. Celui-ci fait intervenir les valeurs propres extrêmes en valeur absolue, et peut avoir un comportement instable si des valeurs propres sont proches de zéro. Pour l'analyser, on se limite donc au cas où Q et Q_ρ sont convexes. Toujours avec Weyl, on a :

$$\kappa(Q + \rho A^TA) = \frac{\lambda_{\max}(Q + \rho A^TA)}{\lambda_{\min}(Q + \rho A^TA)} \leq \frac{\lambda_{\max}(Q) + \rho\lambda_{\max}(A^TA)}{\lambda_{\min}(Q)} = \kappa(Q) + \rho\frac{\sigma_{\max}}{\lambda_{\min}}.$$

Il est ensuite naturel de demander que le conditionnement ne soit pas détérioré de plus d'un facteur $\gamma > 1$. Si l'on veut $\kappa(Q+\rho A^TA) \leq \gamma\kappa(Q)$, les inégalités précédentes nous donnent une condition suffisante :

$$\kappa(Q) + \rho\frac{\sigma_{\max}}{\lambda_{\min}} \leq \gamma\kappa(Q).$$

Avec $\kappa(Q) = \dfrac{\lambda_{\max}}{\lambda_{\min}}$ et $\gamma = 2$, ceci devient $\rho \leq \rho_2 = \dfrac{\lambda_{\max}}{\sigma_{\max}}$.

Le candidat ρ_1 tente de convexifier le problème, et le candidat ρ_2 limite la détérioration du conditionnement. On utilise finalement un multiple de la moyenne des deux :

$$\rho = \omega\frac{\rho_1 + \rho_2}{2} = \omega\frac{\lambda_{\max} - \lambda_{\min}}{2\sigma_{\max}}.$$

On détermine la valeur de ω numériquement, et $\omega = 4$ donne les meilleurs résultats en moyenne.

Nous terminons ce paragraphe avec une illustration numérique de la pénalisation sur deux exemples. Le premier est une instance du problème du k-cluster, voir par exemple [121], et le second est l'instance `crossdock_15x17` de la MINLPLib [11]. Ce sont deux instances en variables booléennes, dont les informations sont résumées dans le tableau 2.8.

instance	n_{var}	n_{eq}	n_{ineq}	ub	lb_1	lb_2	lb_3
kcluster	160	1	0	-204	-398	-411	-234
crossdock	215	30	14	14409	0	-113642	-1849.3

Tableau 2.8 – *De gauche à droite : nom de l'instance, nombre de variables, d'égalités linéaires, d'inégalités linéaires, meilleure solution connue, borne inférieure par linéarisation, borne inférieure de la relaxation quadratique sans pénalisation, borne inférieure de la relaxation quadratique avec pénalisation.*

La relaxation linéaire utilisée ici est celle obtenue par le modèle factorisé linéaire : les termes carrés et bilinéaires sont remplacés par des couplages non linéaires, de sorte que les contraintes quadratiques deviennent linéaires. Pour l'instance k-cluster, l'ajout de la pénalisation réduit l'écart à l'optimum de la relaxation non linéaire de 100% à 15%, qui devient alors meilleure que la relaxation linéaire. Pour l'instance crossdock, si la borne issue de la relaxation quadratique est améliorée par l'ajout de la pénalisation, la relaxation linéaire reste meilleure. L'impact de la pénalisation sur les valeurs propres, sur la valeur de la relaxation convexe, ainsi que sur son temps de résolution est résumé dans les tableaux 2.9 et 2.10.

ρ	$\lambda_{\min}(Q+\rho A^TA)$	$\lambda_{\max}(Q+\rho A^TA)$	v_ρ	iter
0	-10.297	4.101	-411.88	36
10^{-4}	-10.282	4.101	-411.45	36
10^{-3}	-10.143	4.101	-407.32	53
10^{-2}	-8.768	4.101	-366.73	79
10^{-1}	-3.930	6.597	-234.23	267
$\bar{\rho} = 0.122$	-3.924	9.889	-234.1	247
1	-3.909	150.1	-233.82	234
10^1	-3.908	1590	-233.79	240
10^2	-3.908	15990	-233.79	240
10^3	-3.908	159990	-233.79	280
10^4	-3.908	$1.6\,10^6$	-233.79	305

Tableau 2.9 – *Impact de la pénalisation sur une instance de k-cluster. La pénalisation heuristique utilisée est $\bar{\rho}$, et est donnée par* (2.5)*. La borne inférieure v_ρ est obtenue par la résolution de la relaxation convexe utilisant une perturbation uniforme $\alpha_i = -\lambda_{\min}(Q_\rho)$. Enfin, iter désigne le nombre d'itération qu'une méthode itérative a requise pour résoudre la relaxation.*

La matrice A^TA contient au moins $n_{\max}^2$ non-zéros, où $n_{\max}$ est le support de la plus grande contrainte pénalisée. Ainsi, même si A contient peu de non-zéros (relativement à Q), A^TA peut être pleine. En pratique, on ne forme jamais la matrice, on se contente d'opérations utilisant des produits matrice-vecteur. Dans ce cas, seules les convexifications utilisant la plus petite valeur propre sont utilisées. En effet, on ne peut pas accéder en même temps au coefficient q_{ij} et a_ia_j de façon efficace dès que l'on utilise des structures creuses.

ρ	$\lambda_{\min}(Q+\rho A^TA)$	$\lambda_{\max}(Q+\rho A^TA)$	v_ρ	iter
0	-5067.6	5067.6	-113642	408
10^{-3}	-5067.6	5067.6	-113642	387
10^{-2}	-5067.5	5067.6	-113641	450
10^{-1}	-5066.9	5067.6	-113624	436
1	-5060.6	5074.6	-113463	389
10^1	-4997.7	5137.4	-111847	430
10^2	-4369.3	5766.3	-95693	438
10^3	-709.13	12061	-1849.3	1081
$\bar{\rho}=2171$	-709.13	20256	-1849.3	1310
10^4	-709.13	75057	-1849.3	1426
10^5	-709.13	705056	-1849.3	Fail

Tableau 2.10 – *Impact de la pénalisation sur* `crossdock_15x17`. *La pénalisation heuristique utilisée est* $\bar{\rho}$, *et est donnée par* (2.5). *La borne inférieure* v_ρ *est obtenue par la résolution de la relaxation convexe utilisant une perturbation uniforme* $\alpha_i = -\lambda_{\min}(Q_\rho)$. *Enfin,* iter *désigne le nombre d'itération qu'une méthode itérative a requise pour résoudre la relaxation.*

Cette amélioration touche peu d'instances, à peine plus de 5%, mais apporte généralement des améliorations importantes. Sur les 213 instances de notre *benchmark* sur lesquelles des contraintes linéaires ont été pénalisées, la borne du *root node* est améliorée en moyenne de 9%. Pour un temps limite d'une minute, 30% d'instances supplémentaires ont pu être résolues, et celles qui l'étaient déjà nécessitent en moyenne 25% de nœuds en moins.

Sélection dynamique du terme de convexification

Puisqu'aucune stratégie individuelle ne domine les autres, comme montré par le tableau 2.6, une possibilité est de considérer les quatre termes de convexification, sous la forme de quatre contraintes quadratiques. Si cette approche est pertinente dans un contexte de séparation, lors de la génération de coupes par exemple, elle a pour effet de multiplier par quatre le nombre de contraintes quadratiques dans le solveur non linéaire. Pour cette raison, nous avons plutôt cherché à améliorer le terme de convexification utilisé.

On s'intéresse alors à l'écart, dans le tableau 2.7, entre SDIAG et BEST. Si SDIAG est la meilleure stratégie individuelle, sélectionner la meilleure stratégie pour chaque instance permettrait de réduire le nombre de nœuds moyen de 24%. De plus, chaque stratégie individuelle peut devenir arbitrairement mauvaise par rapport à BEST. Pour le voir, on considère, pour chaque stratégie, les rapports des nombres de nœuds nécessaires à la résolution avec la stratégie et avec BEST. Ces rapports sont ensuite triés par ordre croissant pour chaque stratégie, puis tracés dans la figure 2.14.

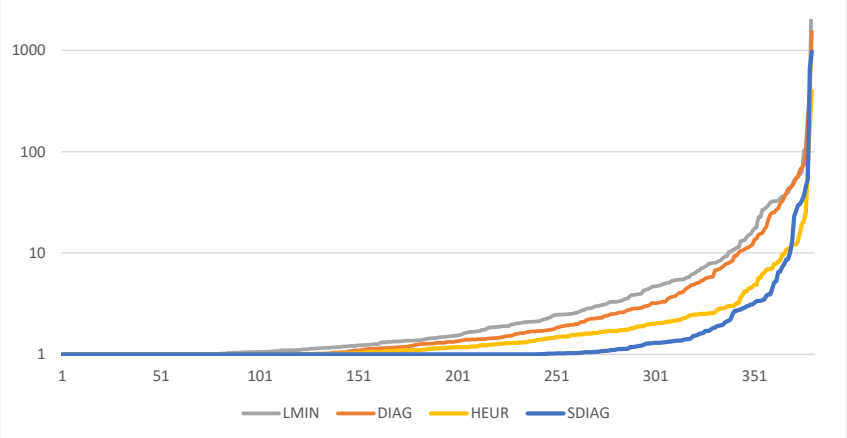

FIGURE 2.14 – *Performance relative de chaque stratégie par rapport à BEST, évaluée par le ratio des nombres de nœuds nécessaires à la résolution. Les tris par ratios croissants sont différents pour chaque stratégie.*

La figure 2.14 montre que pour chaque stratégie individuelle, il existe des instances où cette stratégie nécessite beaucoup plus de nœuds (plus de 1000 fois plus) que BEST. Cette variabilité montre aussi qu'il est probable qu'au sein d'une même instance, la meilleure stratégie varie au cours de la recherche.

On implémente alors une dernière stratégie, qui consiste à sélectionner le terme de convexification avant la résolution de chaque nœud. Pour cela, on considère l'erreur de convexité dans le pire cas pour le sous-problème courant. Pour un terme de convexification α, celui-ci s'écrit $g^T\alpha$, toujours avec $g_i = (u_i - \ell_i)^2$. On définit alors DYN, la stratégie dynamique consistant à choisir la convexification minimisant $g^T\alpha$. Les tableaux 2.11 et 2.12 reprennent les résultats des tableaux 2.6 et 2.7, en y ajoutant la stratégie de sélection dynamique.

Stratégie	LMIN	HEUR	DIAG	SDIAG	BEST	DYN
Instances résolues	593	610	574	604	633	617
Seul à résoudre	4	14	1	16		0
Seul à ne pas résoudre	2	1	6	4		0

Tableau 2.11 – *Impact de la stratégie de convexification sur les instances résolues à l'optimum global, sur un* benchmark *de 952 instances.*

	LMIN	HEUR	DIAG	SDIAG	BEST	DYN
Meilleure stratégie	50	115	102	235	380	204
Nombre de nœuds moyen	276	180	236	152	115	128

Tableau 2.12 – *Comparaison de la taille de l'arbre de branchement selon la stratégie de convexification sur les 380 instances où celle-ci a un impact (c'est-à-dire les instances où le nombre de nœuds diffère pour au moins deux stratégies).*

Les tableaux 2.11 et 2.12 montrent que DYN se situe entre les meilleures stratégies individuelles et BEST. Par rapport à SDIAG, DYN permet de réduire le nombre de nœuds moyen d'environ 16%, soit les deux-tiers que ce que l'on espérait.

On termine ce paragraphe en comparant, comme sur la figure 2.14, la performance de BEST et de DYN. La courbe est tracée dans la figure 2.15. Deux remarques importantes permettent de conclure sur l'intérêt d'une sélection dynamique. Premièrement, DYN ne devient pas arbitrairement mauvaise par rapport à BEST, puisque le rapport des nombres de nœuds ne dépasse 10 que pour une seule instance. Enfin, pour environ 22% des instances, DYN nécessite moins de nœuds que BEST, montrant ainsi que la meilleure convexification varie au cours de la recherche arborescente.

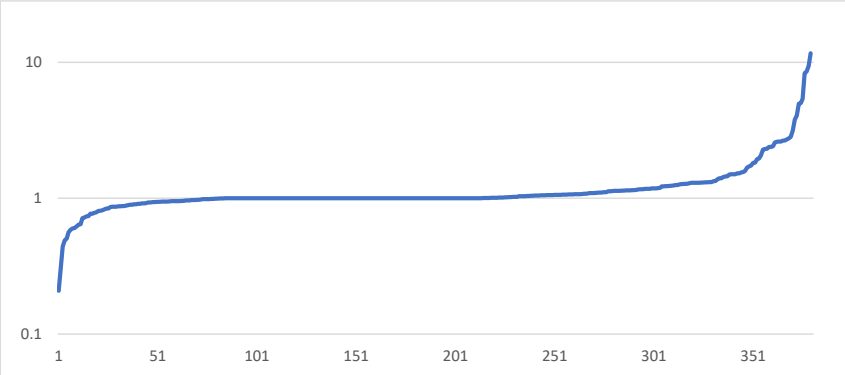

FIGURE 2.15 – *Performance relative de DYN par rapport à BEST, évaluée par le ratio des nombres de nœuds nécessaires à la résolution, sur les instances où la résolution par les quatre stratégies de base a requis des nombres de nœuds différents. À comparer avec la figure 2.14.*

Prise en compte de la solution de la relaxation

Lors de la génération de chaque relaxation convexe, le terme de convexification minimisant l'erreur de convexité maximale est choisi. Une amélioration possible à cette approche est de ne pas considérer uniquement l'erreur dans le pire cas, mais aussi l'erreur courante, c'est-à-dire celle commise par x^*, la solution optimale de la relaxation convexe résolue lors du nœud parent. L'erreur de convexité commise dans le pire cas est atteinte au milieu du domaine, c'est-à-dire pour $m = (u + \ell)/2$. Les deux erreurs que l'on cherche à minimiser en sélectionnant le terme de convexification sont donc

$$g(m) = \sum_i \alpha_i \left(\frac{u_i - \ell_i}{2}\right)^2 \quad \text{et} \quad g(x^*) = \sum_i \alpha_i (x^* - \ell_i)(u_i - x_i^*).$$

Ce sont deux fonctions linéaires, dont on peut minimiser une moyenne. Les meilleurs résultats ont été obtenus en sélectionnant le terme de convexification minimisant $g^T\alpha$ avec

$$g_i = \frac{1}{2}\left(\frac{u_i - \ell_i}{2}\right)^2 + \frac{1}{2}(x^* - \ell_i)(u_i - x_i^*),$$

c'est-à-dire en considérant la moyenne arithmétique entre la pire erreur et l'erreur courante. Cette modification a cependant peu d'impact sur les résultats numériques, puisqu'en reprenant les résultats des tableaux 2.11 et 2.12, elle permet de résoudre deux instances supplémentaires, et réduit le nombre moyen de nœuds de 2%.

Convexification optimale et coefficients de mise à l'échelle

Nous terminons ce chapitre avec une remarque sur la stratégie SDIAG, dont le terme de convexification est donné par (2.3). Dans notre méthode de reconvexification, lors de la génération de la relaxation convexe d'un nœud, un des quatre termes est sélectionné s'il minimise une certaine fonction linéaire approximant l'erreur de convexité (dans le pire cas, ou, comme au paragraphe précédent, en la solution optimale du nœud parent). On s'intéresse alors au cas à deux variables, dans l'espoir de trouver la convexification optimale pour la fonction de coût utilisée. Cette analyse n'a pas conduit à une amélioration de performance, comme nous l'espérions, mais complète notre intuition sur les raisons de la bonne performance de la stratégie SDIAG, en particulier pour le choix utilisé des coefficients de mise à l'échelle des variables.

Soient $g_1 > 0$ et $g_2 > 0$ les coefficients de l'objectif linéaire à minimiser, ainsi que trois réels q, q_1 et q_2 représentant la matrice quadratique à convexifier. Pour une perturbation $\alpha = (\alpha_1, \alpha_2)$, on a l'équivalence

$$\begin{pmatrix} q_1 + \alpha_1 & q \\ q & q_2 + \alpha_2 \end{pmatrix} \succcurlyeq 0 \iff (q_1 + \alpha_1)(q_2 + \alpha_2) \geq q^2,$$

et l'on cherche donc à résoudre

$$\min_{\alpha_1, \alpha_2} \; g_1\alpha_1 + g_2\alpha_2$$
$$\text{s.c.} \begin{cases} \alpha_1 \geq 0, \\ \alpha_2 \geq 0, \\ (q_1 + \alpha_1)(q_2 + \alpha_2) \geq q^2. \end{cases}$$

Pour cela on fait la remarque suivante : si l'un des α_i est non nul (c'est-à-dire si Q n'est pas déjà convexe), alors la contrainte d'inégalité est saturée. En effet, dans le cas contraire, il serait possible de réduire l'un des α_i non nuls jusqu'à saturer l'inégalité ou la contrainte de positivité de ce α_i. On peut alors utiliser la contrainte pour éliminer l'un des α_i, puis l'analyse réelle pour trouver le minimum de la fonction résultante. La solution optimale est finalement donnée par

$$\alpha_1 = \max\left(0, -q_1 + |q|\sqrt{\frac{g_2}{g_1}}\right) \quad \text{et} \quad \alpha_2 = \max\left(0, -q_2 + |q|\sqrt{\frac{g_1}{g_2}}\right).$$

On commence par remarquer que si $g_i = (u_i - \ell_i)^2$, c'est-à-dire si l'on cherche à minimiser l'erreur de convexité dans le pire cas, on retrouve la formule du terme SDIAG utilisant les coefficients $s_i = u_i - \ell_i$. Dans [21], où ce terme est introduit, ainsi que dans quelques autres articles sur les αBB sous-estimateurs [152, 22], ce choix est décrit comme arbitraire et est fait par intuition. Nous avons montré ici

qu'en dimension 2, le choix de la fonction linéaire approximant l'erreur de convexité détermine celui des coefficients de mise à l'échelle. Puisque les performances de DYN et de SDIAG dans le tableau 2.12 sont différentes, on sait que ceci ne se généralise pas en dimension quelconque (sinon SDIAG serait systématiquement sélectionnée, puiqu'étant le minimiseur de l'erreur de convexité), mais cela reste une explication aux excellentes performances pratiques de la méthode SDIAG. Finalement, pour une approximation linéaire $g^T\alpha$ de l'erreur de convexité, les coefficients conseillés de mise à l'échelle pour SDIAG sont définis par $s_i = \sqrt{g_i}$.

Chapitre 3

Résolution des relaxations non linéaires

La maturité numérique et algorithmique de l'algorithme du simplexe dual est telle qu'il est aujourd'hui possible de résoudre les relaxations linéaires avec rapidité et robustesse, en particulier dans un contexte de *branch-and-bound*. Pour cette raison, les solveurs d'optimisation globale exploitent principalement des relaxations linéaires pour l'obtention des bornes inférieures, et la résolution de programmes non linéaires est généralement limitée aux heuristiques primales. Dans LocalSolver, si les relaxations utilisées sont linéaires, celles-ci sont résolues avec le simplexe dual du logiciel.

Le cas des relaxations non-linéaires est plus complexe, car il n'existe pas de méthode générique privilégiée pour résoudre les problèmes non linéaires convexes. Si les algorithmes de points intérieurs permettent de résoudre efficacement un grand nombre de problèmes convexes, leur performance pratique repose sur la connaissance de barrières auto-concordantes spécifiques. De plus, leur utilisation au sein d'un *branch-and-bound* est un sujet encore ouvert. L'objet principal de ce chapitre est ainsi l'implémentation d'un solveur non linéaire sur-mesure, dont l'objectif sera de résoudre nos relaxations non linéaires.

Étant donnée une relaxation convexe du modèle factorisé, toute borne inférieure de cette dernière est aussi une borne inférieure du problème initial. Ainsi, si résoudre les relaxations non linéaires s'avère trop difficile, on souhaite quand même pouvoir les borner. Le second objectif de ce chapitre est de mettre en place des techniques permettant de borner des problèmes convexes, qu'ils soient linéaires ou non. Ces techniques permettront aussi d'obtenir des bornes et certificats d'inconsistance numériquement fiables.

Plan du chapitre

3.1 Conditions d'optimalité et dualité

On considère dans cette section un problème convexe et différentiable, de la forme

$$\begin{aligned} (\mathrm{P}) \quad & \min_{x \in \mathbb{R}^n} f(x) \\ & \text{s.c.} \left\{ g(x) \leq 0 \in \mathbb{R}^m. \right. \end{aligned} \tag{3.1}$$

En optimisation convexe différentiable, les conditions de Karush-Kuhn-Tucker permettent de déclarer l'optimalité d'une solution. Elles sont suffisantes, et nécessaires dès que les contraintes sont qualifiées. Pour un point primal-dual $(x, \mu) \in \mathbb{R}^n \times \mathbb{R}^m_+$, celles-ci sont constituées de trois groupes de contraintes, ayant chacune une interprétation géométrique précise. La première condition est la stationnarité du lagrangien $\nabla_x \mathcal{L}(x, \mu) = 0$, où $\mathcal{L}(x, \mu) = f(x) + \mu^T g(x)$. Cette condition s'appelle l'admissibilité duale : dès que (x, μ) est un point stationnaire du lagrangien, $\mathcal{L}(x, \mu)$ fournit une borne duale au problème. La seconde condition est l'admissibilité primale $g(x) \leq 0$: dès que x est admissible, $f(x)$ fournit une borne primale au problème. La dernière condition est la complémentarité $\mu^T g(x) = 0$. Si la solution (x, μ) est primale-duale admissibile, elle fournit un couple de bornes primales-duales dont le saut de dualité est égal à la complémentarité $\mu^T g(x)$.

Le calcul de bornes inférieures de (P) se ramène ainsi à la recherche de points stationnaires du lagrangien. Si l'on souhaite résoudre (P) à l'optimum, il faut de plus tendre vers l'admissibilité primale, et atteindre cette dernière en un point où les écarts complémentaires s'annulent. Même en optimisation convexe, ces tâches peuvent être difficiles et sont sujettes à des erreurs numériques. Les conditions de KKT ne sont jamais exactement vérifiées, et des tolérances, nécessairement arbitraires, sont généralement utilisées. Dans le cadre de l'optimisation non linéaire non convexe, c'est-à-dire la recherche de points admissibiles localement optimaux, ceci

pose peu de problèmes, puisqu'il existe dans ce cas un oracle déclarant l'admissibilité d'une solution. Dans ce cas, la stationnarité et la complémentarité sont utiles au contrôle de la convergence, mais pas indispensables. Dans notre contexte en revanche, c'est la stationnarité du lagrangien qui nous intéresse le plus, puisque c'est elle qui constitue le certificat d'une borne inférieure valide. Dans tout algorithme numérique qui la remplace par une condition approchée utilisant une tolérance, la validité des bornes inférieures obtenues ne peut être garantie. Les erreurs numériques et les tolérances présentes dans les solveurs a donc motivé, dans la communauté de l'optimisation exacte, l'utilisation de techniques permettant de rendre les bornes valides.

Un travail important sur le sujet est l'obtention de bornes inférieures et certificats d'inconsistance fiables dans le cas des programmes linéaires [133], ainsi que leur généralisation au cas non linéaire convexe [96]. Nous avons redécouvert ce second résultat pendant la thèse, en cherchant cependant autre chose : des relaxations non contraintes de (P) qui pourraient être données directement à la recherche locale de LocalSolver. Dans [96], le résultat est obtenu en considérant une relaxation linéaire de (P), ce qui permet ensuite d'utiliser le raisonnement de [133]. Une autre preuve est présentée dans [124], utilisant cette fois des sous-estimateurs convexes, et dont l'idée correspond aux arguments donnés ici au paragraphe « Interprétation du dual ». Dans ces deux articles, tout comme dans l'article original sur le cas des problèmes linéaires, beaucoup d'importance est donné à l'évaluation de la borne inférieure avec des techniques d'arithmétique d'intervalle, ainsi qu'à la prise en compte d'incertitudes dans les données du problème. Nous généralisons la portée du résultat, en montrant qu'il est plus qu'une astuce algébrique permettant de s'affranchir des tolérances. Celui-ci est formalisé sous la forme d'un théorème de dualité, et nous proposons des interprétations et applications plus générales. Notre technique de preuve est de plus différente, et traite les cas linéaires et non linéaires de la même façon.

3.1.1 Dualité sous contraintes de bornes

Dual de Wolfe

Le résultat principal que nous allons utiliser est un théorème de dualité dû à Wolfe [166]. Il concerne les programmes non linéaires convexes et nous le rappelons ci-dessous.

Théorème 3.1.1. [Théorème 1 de [166]] *Soient des fonctions* $f : \mathbb{R}^n \to \mathbb{R}$ *et* $g : \mathbb{R}^n \to \mathbb{R}^m$ *convexes et différentiables. Soit le couple de problèmes primals-duals suivant :*

$$\text{(P)} \min_{x \in \mathbb{R}^n} f(x) \quad s.c. \begin{cases} g(x) \leq 0. \end{cases} \qquad \text{(D)} \max_{(x,\mu) \in \mathbb{R}^n \times \mathbb{R}^m_+} f(x) + \mu^T g(x) \quad s.c. \begin{cases} \nabla f(x) + \sum_i \mu_i \nabla g_i(x) = 0. \end{cases}$$

Alors la dualité faible a lieu et on a $v(\text{D}) \leq v(\text{P})$*. En particulier, toute solution admissible de* (D) *fournit une borne inférieure à* (P)*.*

On peut comparer ce résultat à la dualité lagrangienne faible, qui s'écrit :

$$\begin{array}{lll} \min\limits_{x\in\mathbb{R}^n} & f(x) & \geq \max\limits_{\mu\geq 0} \min\limits_{x\in\mathbb{R}^n} f(x)+\mu^T g(x). \\ \text{s.c.} & \Big\{ g(x)\leq 0 & \end{array}$$

Le théorème explique que pour des fonctions convexes différentiables, on peut passer la condition $\nabla_x \mathcal{L}(x,\mu)=0$, nécessaire et suffisante à la réalisation de $\min\limits_x$, dans les contraintes du max, alors pris en (x,μ) :

$$\begin{array}{lll} \max\limits_{\mu} & \min\limits_{x} f(x)+\mu^T g(x) & = \max\limits_{x,\mu\geq 0} f(x)+\mu^T g(x) \\ \text{s.c.} & \Big\{ \mu\geq 0 & \text{s.c.} \Big\{ \nabla f(x)+\sum_i \mu_i \nabla g_i(x)=0. \end{array}$$

Un second résultat obtenu par Wolfe est la dualité forte, sous hypothèse de qualifications des contraintes.

Théorème 3.1.2. [Théorème 2 de [166]] *En reprenant les hypothèses et notations du théorème 3.1.1, si de plus les contraintes $g(x)\leq 0$ sont qualifiées en tout point admissible, alors pour toute solution optimale x^* de* (P)*, il existe μ^* tel que (x^*,μ^*) est une solution optimale de* (D)*. De plus, la dualité forte a lieu : $v(\mathrm{D})=v(\mathrm{P})$.*

Utilisation des contraintes de borne

Une hypothèse importante de notre modèle factorisé est la présence de bornes finies pour chaque variable. Dans [133], la procédure permettant d'obtenir des bornes inférieures valides pour un programme linéaire suppose aussi la présence de bornes finies pour chaque variable. Le théorème 3.1.3 formalise l'idée en obtenant explicitement le dual d'un problème d'optimisation convexe différentiable sous contraintes de bornes.

Théorème 3.1.3. *Soient $f:\mathbb{R}^n\to\mathbb{R}$ une fonction convexe différentiable et ℓ et u dans $\mathbb{R}^n$ tels que $\forall i,\ \ell_i\leq u_i$. Soit le problème de minimisation*

$$\begin{array}{lll} (\mathrm{P}) & \min\limits_{x\in\mathbb{R}^n} f(x) \\ & s.c. \Big\{ \ell\leq x\leq u. \end{array}$$

Alors en notant

$$\theta(x)=f(x)+(\ell-x)^T\nabla f(x)^+ +(u-x)^T\nabla f(x)^-, \tag{3.2}$$

le dual de Wolfe de (P) *est*

$$(\mathrm{D})\ \max_x \theta(x),$$

et la dualité forte a lieu : $v(\mathrm{D})=v(\mathrm{P})$.

Démonstration. On note μ_ℓ et μ_u les multiplicateurs des contraintes de borne. D'après le théorème 3.1.1, le dual de (P) est

$$\begin{array}{lll} (\mathrm{D}) & \max\limits_{x,\mu_\ell,\mu_u} f(x)+\mu_\ell^T(\ell-x)+\mu_u^T(x-u) \\ & \text{s.c.} \begin{cases} (\mu_\ell,\mu_u)\geq 0, \\ \nabla f(x)-\mu_\ell+\mu_u=0. \end{cases} \end{array} \tag{3.3}$$

Pour x fixé, ce problème est séparable en chaque (μ_ℓ^i, μ_u^i). Pour l'un de ces couples, le problème de maximisation s'écrit

$$\max_{\mu_\ell^i, \mu_u^i} \mu_\ell^i(\ell_i - x_i) + \mu_u^i(x_i - u_i)$$
$$\text{s.c.} \begin{cases} (\mu_\ell^i, \mu_u^i) \geq 0, \\ \mu_\ell^i - \mu_u^i = \nabla_i f(x). \end{cases}$$

Ce problème est linéaire, et on peut le résoudre en analysant les conditions de KKT. Nous proposons cependant une autre approche : en inspectant l'ensemble admissible de chaque contrainte, comme dans les figures 3.1 et 3.2, et en notant $a^+ = \max(0, a)$ et $a^- = \min(0, a)$ les parties positives et négatives de a, on peut voir que l'ensemble des solutions admissibles est décrit par la demi-droite

$$\begin{pmatrix} \mu_\ell^i \\ \mu_u^i \end{pmatrix} \in \begin{pmatrix} \nabla_i f(x)^+ \\ -\nabla_i f(x)^- \end{pmatrix} + \mathbb{R}^+ \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

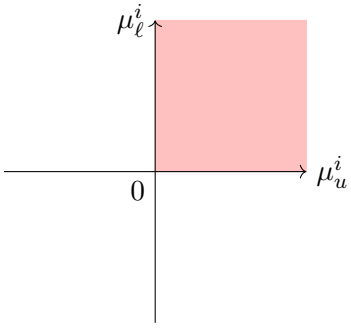

FIGURE 3.1 – *Domaine admissible de* $(\mu_\ell^i, \mu_u^i) \geq 0$.

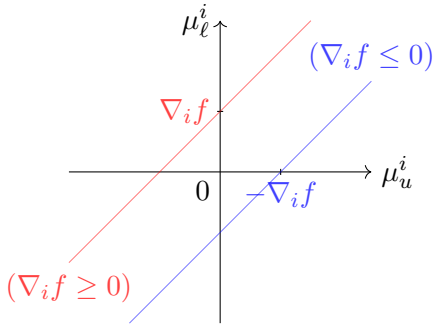

FIGURE 3.2 – *Domaine admissible de* $\mu_\ell^i - \mu_u^i = \nabla_i f$ *selon le signe de* $\nabla_i f$.

On pose alors $t_i \in \mathbb{R}_+$ et on définit $\mu_\ell^i = t_i + \nabla_i f(x)^+$ et $\mu_u^i = t_i - \nabla_i f(x)^-$. Ainsi, quand t_i parcourt $\mathbb{R}^+$, (μ_ℓ^i, μ_u^i) parcourt l'ensemble des solutions admissibles. En substituant ces expressions dans l'objectif, le problème dual en (μ_ℓ^i, μ_u^i) devient :

$$\max_{t \geq 0} \nabla_i f(x)^+(\ell_i - x_i) - \nabla_i f(x)^-(x_i - u_i) + t_i(\ell_i - u_i).$$

Puisque $\ell_i - u_i \leq 0$, la solution optimale est donnée par $t_i = 0$. On peut alors éliminer les variables (μ_ℓ, μ_u) dans (3.3), obtenant ainsi

$$\text{(D)} \max_x f(x) + \sum_i \nabla_i f(x)^-(u_i - x_i) + \nabla_i f(x)^+(\ell_i - x_i),$$

qui est bien la forme cherchée.

Par hypothèse, l'ensemble admissible est non vide, et (P) est donc consistant. Puisque les contraintes de bornes décrivent un compact non vide de $\mathbb{R}^n$ et qu'elles sont qualifiées, la dualité forte a lieu. □

Corollaire 3.1.4. [Ajout de contraintes d'inégalité] *Soient des fonctions $f : \mathbb{R}^n \to \mathbb{R}$ et $g : \mathbb{R}^n \to \mathbb{R}^m$ convexes et différentiables. Soit le problème de minimisation*

$$(\mathrm{P}) \min_{x \in \mathbb{R}^n} f(x) \quad s.c. \begin{cases} \ell \leq x \leq u, \\ g(x) \leq 0. \end{cases}$$

Pour $\mu \in \mathbb{R}^m$, on note $\mathcal{L}(x, \mu) = f(x) + \mu^T g(x)$. En posant

$$\theta(x, \mu) = \mathcal{L}(x, \mu) + (\ell - x)^T \nabla \mathcal{L}(x, \mu)^+ + (u - x)^T \nabla \mathcal{L}(x, \mu)^-,$$

le dual de Wolfe de (P) *est*

$$(\mathrm{D}) \max_{x, \mu} \theta(x, \mu), \quad s.c. \begin{cases} \mu \geq 0 \end{cases}$$

et la dualité faible a toujours lieu. Si de plus (P) *est consistant et si les contraintes $g(x) \leq 0$ sont qualifiées, alors $v(\mathrm{D}) = v(\mathrm{P})$.*

Démonstration. Soit $\mu \geq 0$. En appliquant le résultat du théorème 3.1.3 au problème $\min_{\ell \leq x \leq u} \mathcal{L}(x, \mu)$:

$$\min_{\ell \leq x \leq u} \mathcal{L}(x, \mu) = \max_x \mathcal{L}(x, \mu) + (\ell - x)^T \nabla \mathcal{L}(x, \mu)^+ + (u - x)^T \nabla \mathcal{L}(x, \mu)^-,$$

puis en prenant le maximum des deux membres de l'égalité sur $\mu \geq 0$:

$$\max_{\mu \geq 0} \min_{\ell \leq x \leq u} \mathcal{L}(x, \mu) = \max_{x, \mu \geq 0} \mathcal{L}(x, \mu) + (\ell - x)^T \nabla \mathcal{L}(x, \mu)^+ + (u - x)^T \nabla \mathcal{L}(x, \mu)^-,$$

on voit que le membre de droite est le problème (D), et le membre de gauche est le dual lagrangien classique. Les dualités faible et forte s'appliquent donc à (D) sous les mêmes hypothèses que dans le cadre de la dualité lagrangienne. □

Ce corollaire nous est utile pour résoudre (P) avec une méthode itérative, tout en obtenant des bornes le long de la recherche. Quelle que soit la solution primale-duale courante (x, μ), le corollaire 3.1.4 nous donne en effet une solution duale admissible, et donc une borne inférieure.

Interprétation du dual

La preuve du théorème 3.3 utilise la structure des contraintes de bornes. Puisque les multiplicateurs associés apparaissent de façon séparable dans la condition de stationnarité du lagrangien, leur résolution explicite est possible. La formule de dualité faible peut aussi s'obtenir par un raisonnement primal, en utilisant le sous-estimateur linéaire de l'objectif convexe f :

$$\forall x, y \in \mathbb{R}^n, \ f(y) \geq f(x) + \nabla f(x)^T (y - x), \tag{3.4}$$

ainsi que la structure séparable des contraintes de borne. Pour $x \in \mathbb{R}^n$, en prenant le min sur $\ell \leq y \leq u$ dans (3.4), on obtient :

$$
\begin{aligned}
v(\mathrm{P}) &\geq \min_{\ell \leq y \leq u} f(x) + \nabla f(x)^T (y - x) \\
&\geq f(x) - x^T \nabla f(x) + \min_{\ell \leq y \leq u} \sum_i \nabla_i f(x) y_i \\
&\geq f(x) - x^T \nabla f(x) + \sum_i \min \left(\nabla_i f(x) \ell_i, \nabla_i f(x) u_i \right) \\
&\geq f(x) - x^T \nabla f(x) + \frac{1}{2} \sum_i \nabla_i f(x) \ell_i + \nabla_i f(x) u_i - |\nabla_i f(x)|(u_i - \ell_i) \\
&\geq f(x) + \sum_i \nabla_i f(x)^- (u_i - x_i) + \nabla_i f(x)^+ (\ell_i - x_i) \\
&\geq \theta(x).
\end{aligned}
$$

Cette interprétation permet d'illustrer le théorème 3.3 dans le cas unidimensionnel. Dans la figure 3.3, on a tracé la relation de dualité entre f et θ dans deux cas, selon que x^* est dans l'intérieur du domaine ou sur un bord.

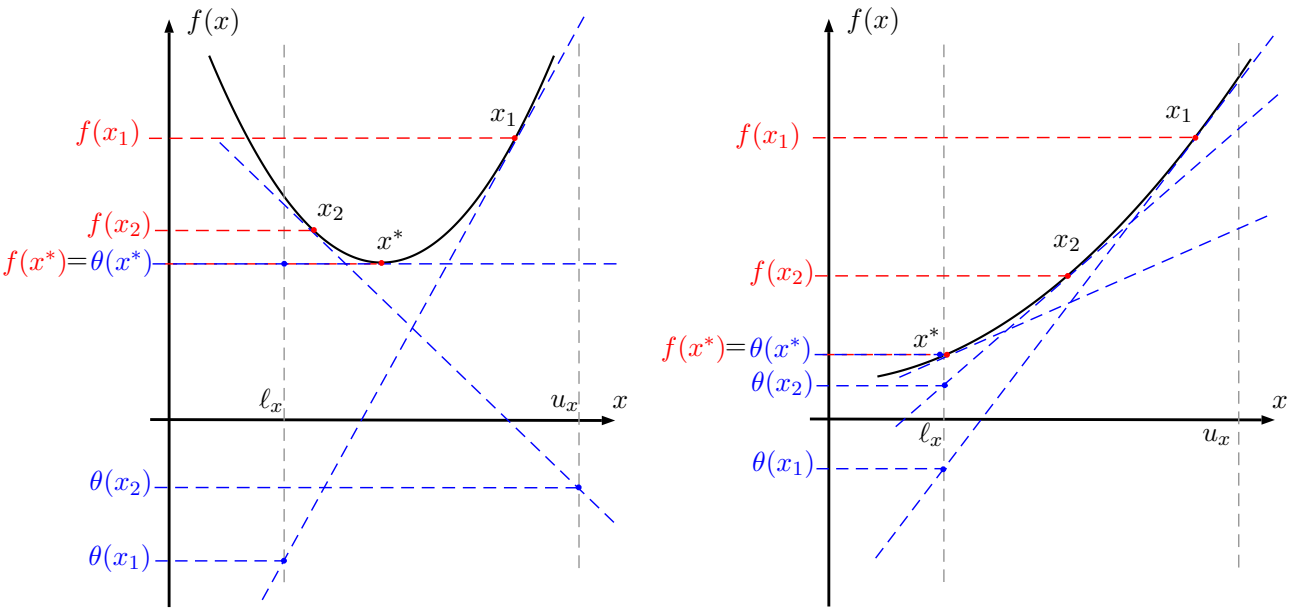

FIGURE 3.3 – *Dualité sur un problème unidimensionnel. À gauche, x^* est dans l'intérieur du domaine et à droite, sur le bord.*

3.1.2 Applications

Optimisation globale par recherche locale

La première application du théorème 3.1.3, qui était en fait notre premier objectif en s'intéressant à la dualité, est d'utiliser LocalSolver pour le calcul de bornes. En effet, si le dual obtenu dans le corollaire 3.1.4 est non convexe et sous-différentiable uniquement, son ensemble admissible n'est constitué que de contraintes de signe.

Dans ce cas, la recherche locale n'a aucun mal à explorer le paysage d'optimisation associé, fournissant à chaque itération une borne inférieure valide.

Malheureusement, cette approche n'est pas compétitive avec la résolution directe de la relaxation convexe. En effet, la recherche locale implémentée dans LocalSolver a été pensée pour les problèmes combinatoires. En l'absence de contraintes, c'est la diversification aléatoire qui permet d'explorer un paysage d'optimisation. Dans ce cas, le fléau de la dimension fait que les performances se dégradent rapidement lorsque le nombre de variables augmente. De meilleurs résultats ont été obtenus avec un algorithme de recherche directe : le simplexe de Nelder-Mead [131]. Si celui-ci est spécialisé pour la minimisation sans contraintes d'une fonction à variables réelles, il s'étend au cas des contraintes de bornes, par exemple par projection. Il est efficace pour intensifier la recherche autour d'un point, mais ne passe pas à l'échelle (encore à cause du fléau de la dimension).

S'il est élégant de résoudre des problèmes à l'optimum global en utilisant uniquement des méthodes de recherche directe, nous sommes limités à des instances de quelques variables. De plus, les temps de résolution sont plusieurs ordres de grandeur au-dessus de ceux de la résolution de la relaxation convexe par un solveur dédié. Cette approche a finalement été abandonnée dans le cas général, mais certains cas particuliers restent prometteurs : ceux où le terme $f(x)-x^T\nabla f(x)$ se simplifie. C'est le cas des programmes linéaires et quadratiques. Par exemple, soient les problèmes suivants :

$$\text{(LP)} \min_{x\in\mathbb{R}^n} c^T x \quad \text{s.c.} \begin{cases} -1 \leq x \leq 1, \\ Ax + b = 0. \end{cases} \qquad \text{(QP)} \min_{x\in\mathbb{R}^n} \frac{1}{2}x^T Q x + c^T x \quad \text{s.c.} \begin{cases} -1 \leq x \leq 1, \\ Ax + b = 0. \end{cases}$$

Les bornes des variables ont été normalisées par changement de variables, afin de simplifier les calculs. Les duals de ces problèmes s'écrivent, après simplification des parties positives et négatives en valeurs absolues :

$$\text{(D}_{\text{LP}}\text{)} \max_{\lambda} \lambda^T b - \|A^T\lambda + c\|_1, \qquad \text{(D}_{\text{QP}}\text{)} \max_{x,\lambda} -\frac{1}{2}x^T Q x + b^T\lambda - \|Qx + c + A^T\lambda\|_1.$$

Ces duals sont convexes, mais non différentiables. Une approche envisagée est d'utiliser une relaxation régulière. Dans le cas du programme linéaire, une relaxation $\mathcal{C}^\infty$ du dual est par exemple :

$$\text{(D}_\varepsilon\text{)} \max_{\lambda} \lambda^T b - \sum_i \sqrt{(a_i^T\lambda + c_i)^2 + \varepsilon^2},$$

avec $\varepsilon > 0$ et a_i la i-ème colonne de A. Il fournit une borne duale de (LP) en tout point et sa résolution pour une suite de $\varepsilon \to 0$ permet de résoudre (D_{LP}).

Un prototype d'un tel algorithme améliore significativement la situation par rapport à l'utilisation de la recherche locale. Pour un programme linéaire mal conditionné de la NETLib, `pilot87`, le simplexe dual de LocalSolver est en difficulté et l'approche décrite plus haut permet d'obtenir la valeur optimale en un temps similaire. Sur des instances où le simplexe ne rencontre pas de difficulté, ou si l'on utilise

une version de l'état de l'art telle que CPLEX, résoudre la relaxation convexe reste toujours plus intéressant. L'implémentation d'une méthode dédiée pour résoudre ces formulations des duals est donc prometteuse, surtout pour le cas du problème quadratique. En particulier, un démarrage à chaud d'un tel algorithme ne nécessiterait que l'augmentation du paramètre de régularisation ε, contrairement aux points intérieurs qui doivent aussi modifier les variables.

Ce travail dépasse cependant le cadre de cette thèse, car il reste spécifique. Dès qu'une contrainte devient non linéaire, on perd la convexité du dual : par exemple pour une contrainte quadratique, des termes non convexes de la forme $\eta_i q_{ij} x_j$ apparaissent dans la fonction duale, où η_i est le multiplicateur de la contrainte en question.

Certificats d'optimalité

Le corollaire 3.1.4 fournit une borne inférieure à la relaxation convexe quelque soit la solution (x, μ) courante, alors que dans une méthode itérative de résolution de (3.7), les conditions de KKT ne sont satisfaites qu'à la fin de la recherche. De plus, les multiplicateurs des contraintes de borne font généralement partie de la solution primale-duale. Le corollaire 3.1.4 signifie plus précisément que pour toute solution primale-duale d'un problème contenant des contraintes de borne pour chaque variable, il est possible de résoudre explicitement le dual en les multiplicateurs de bornes, et que la solution optimale de ce dual partiel fournit toujours une solution duale faisable. En particulier, il est ainsi possible de réparer la condition de stationnarité du lagrangien en ne modifiant que les multiplicateurs des contraintes de bornes.

Dans [133], la formule analogue à (3.2) pour les programmes linéaires est évaluée avec des techniques d'arithmétique validée, telles que celles présentées au chapitre 4. On utilise plutôt une arithmétique basée sur la sommation de Kahan [98], dans laquelle la précision d'évaluation de (3.2) est satisfaisante.

La réparation des certificats de KKT est la plus utile lorsque les solveurs échouent à résoudre la relaxation convexe d'un nœud. Dans ce cas, au lieu de n'obtenir aucune borne inférieure, la réparation en fournit une, parfois quasi-optimale. Par exemple, sur l'instance `ex7_3_5` de la MINLPLib, le simplexe dual échoue à résoudre la relaxation du *root node*. L'objectif dual renvoyé est -16854.2, alors que la borne obtenue par réparation du certificat de KKT est de -7.49. D'après CPLEX, la valeur optimale de la relaxation est 0. Finalement, l'utilisation du théorème 3.1.3 permet de résoudre le problème à l'optimum en 3 secondes et 1663 nœuds, au lieu 7 secondes et 2925 nœuds. Sur des problèmes de plus grande taille, un échec de résolution au *root node* a d'autant plus d'impact que peu de nœuds seront visités par la suite.

Certificats d'inconsistance

Dans un algorithme de *branch-and-bound*, la détection des sous-problèmes inconsistants est importante. Cela évite de visiter inutilement certaines branches de l'arbre, et de rester bloqué à un nœud que l'on n'arrive pas à résoudre puisqu'il est inconsistant. L'inconsistance d'un sous-problème peut être détectée en appliquant

des techniques de réduction de bornes (voir le prochain chapitre), ou en montrant que la relaxation convexe est elle-même inconsistante. Pour un problème convexe $\min \{f(x) \mid g(x) \leq 0\}$, un certificat assurant l'inconsistance est un couple primal-dual (x, μ) tel que $\mu \geq 0$, $\mu^T g(x) > 0$, et $\sum_i \mu_i \nabla g_i(x) = 0$. En effet, (x, μ) est dans ce cas une solution duale faisable de $\min \{0 \mid g(x) \leq 0\}$, et la dualité faible montre que $\min \{0 \mid g(x) \leq 0\} \geq \mu^T g(x) > 0$ et donc $\{g(x) \leq 0\}$ est inconsistant.

Ici encore, la condition de stationnarité $\sum_i \mu_i \nabla g_i(x) = 0$ est difficile à satisfaire et nécessite en pratique l'utilisation d'une tolérance du type $\| \sum_i \mu_i \nabla g_i(x) \| \leq \varepsilon$. Malheureusement, il n'existe pas de choix générique satisfaisant pour ε. Par exemple, pour $ax + b \leq 0$, un certificat d'inconsistance avec tolérance est (x, μ) tel que $\mu \geq 0$, $\mu(ax + b) > 0$ et $|\mu a| \leq \varepsilon$. En posant $\mu = \varepsilon/|a| > 0$ et x tel que $ax + b > 0$ (qui existe toujours), on a montré que notre tolérance ε donne inconsistantes toutes les inégalités linéaires.

Le théorème 3.1.3 nous permet d'obtenir des certificats d'inconsistance n'utilisant pas de tolérance, comme résumé dans le corollaire suivant.

Corollaire 3.1.5. [Certificat d'inconsistance] *Soient des fonctions $f : \mathbb{R}^n \to \mathbb{R}$ et $g : \mathbb{R}^n \to \mathbb{R}^m$, convexes et différentiables. Soient ℓ et u deux vecteurs de $\mathbb{R}^n$ tels que $\forall i,\ \ell_i \leq u_i$. Soit le problème de minimisation*

$$\text{(P)} \min_{x \in \mathbb{R}^n} f(x) \quad \text{s.c.} \begin{cases} \ell \leq x \leq u, \\ g(x) \leq 0. \end{cases}$$

Pour $x \in \mathbb{R}^n,\ \mu \in \mathbb{R}^m_+$, soit

$$\Phi(x) = \sum_i \mu_i \nabla g_i(x),$$

alors l'inégalité

$$\mu^T g(x) + (\ell - x)^T \Phi(x)^+ + (u - x)^T \Phi(x)^- > 0 \tag{3.5}$$

est un certificat d'inconsistance pour (P).

Démonstration. Soit le problème de faisabilité suivant :

$$\text{(F)} \min_{x \in \mathbb{R}^n} 0 \quad \text{s.c.} \begin{cases} \ell \leq x \leq u, \\ g(x) \leq 0. \end{cases}$$

En appliquant le corollaire 3.1.4 à (F), l'inégalité (3.5) pour $\mu \geq 0$ signifie que $\mathrm{v}(F) > 0$, et que (F), et donc (P), sont inconsistants. □

3.1.3 Optimisation non linéaire

Dans cette partie, on s'intéresse aux méthodes de résolution de (3.1). Puisque le problème est convexe, tout algorithme convergeant en théorie vers une solution

localement optimale sera ici globalement convergent. De plus, si une certaine relaxation non linéaire est trop difficile à résoudre, on souhaite quand même obtenir des bornes, en particulier grâce aux résultats obtenus dans les sections 3.1.1 et 3.1.2.

Méthodes de résolution

Il existe trois approches principales permettant de résoudre (P) dans le cas général. Ces trois approches sont à la base de la majorité des solveurs industriels, et sont les méthodes de points intérieurs, de programmation quadratique successive, et de lagrangien augmenté.

Les méthodes de points intérieurs sont aujourd'hui considérées comme l'état de l'art en optimisation non linéaire. Elles consistent à trouver des solutions approchées à des conditions de KKT perturbées, pour une suite de perturbations tendant vers 0. Elles sont particulièrement efficaces sur certaines classes de problèmes convexes (LP, QP, QCQP, SOCP), comme la théorie le prédit [132], mais aussi en pratique sur des NLP quelconques [2]. La propriété théorique principale dont elles bénéficient dans le cas convexe est le fait que la convergence est qualifiée par une borne sur le nombre d'itérations nécessaires, et non pas par une vitesse de convergence asymptotique [132, 37]. De plus, et comme pour l'algorithme du simplexe, la performance pratique semble encore meilleure que la garantie théorique. Dans le cas de l'optimisation linéaire par exemple, une analyse statistique permet de réduire le nombre d'itérations attendu de $O(\sqrt{k})$ à $O(\log(k))$, où k est le nombre de variables dans l'espace primal-dual [154]. La limitation principale des méthodes de points intérieurs est la difficulté à démarrer l'optimisation à chaud. Les solveurs KNITRO/IP [60] et IPOPT [162] sont basés sur un algorithme de points intérieurs.

Les méthode de SQP, pour *sequential quadratic programming*, sont une extension de la méthode de Newton au cas contraint. Le lagrangien est approximé par son développement limité au second ordre, et les conditions de KKT sont linéarisées. Le sous-problème équivalent ainsi obtenu est alors quadratique sous contraintes linéaires, et sa résolution fournit une direction de recherche. Cette direction est ensuite explorée pour minimiser une fonction de mérite. La vitesse de convergence asymptotique est quadratique, comme celle de la méthode de Newton. De plus, les performances pratiques des méthodes de SQP sont bonnes : presque aussi robustes que les algorithmes de points intérieurs, et compétitifs en temps de résolution pour des problèmes de taille moyenne [83]. Si le choix du point initial a un impact important sur la vitesse de convergence, la qualité de démarrage à chaud des méthode SQP est bien meilleure que celle des points intérieurs. La limitation principale de ces algorithmes est qu'ils utilisent une forme d'activation des contraintes, et sont donc sensibles à la combinatoire associée. Pour cette raison, lorsque le nombre d'inégalités devient grand, ou si le point initial est de mauvaise qualité, la performance des méthodes de SQP est dégradée. Des solveurs de référence implémentant des algorithmes de SQP sont SNOPT [82] et KNITRO/ACTIVE [60]

Enfin, la troisième approche est celle des méthodes de lagrangien augmenté. Elles consistent à pénaliser certaines contraintes dans le lagrangien, afin de se ramener à la résolution d'une suite de sous-problèmes plus simples. Si la vitesse de convergence de l'algorithme est seulement linéaire, la méthode est robuste et simple à implémenter.

Comme pour les méthodes de SQP, la capacité de démarrage à chaud sans surcoût est une caractéristique importante du lagrangien augmenté. Plus important encore, la résolution de chaque sous-problème fournit un point stationnaire du lagrangien, et donc une borne inférieure au problème. Sous des formes différentes, cet algorithme est à la base des solveurs MINOS [128], LANCELOT [63] et ALGENCAN [26].

Utilisation de solveurs non linéaires en optimisation globale

La plupart des solveurs globaux intègrent des codes d'optimisation non linéaire, dont l'utilisation principale reste cependant limitée aux heuristiques primales. L'utilisation de solveurs non linéaires pour résoudre les relaxations convexes est parfois disponible *via* la modification de paramètres, mais n'est jamais activée par défaut. Une exception est le solveur αBB [28], qui utilise des relaxations non linéaires deux fois différentiables, et les résout avec MINOS [128]. Deux publications récentes (2018) traitent ce sujet au sein des solveurs SCIP [160] et BARON [155].

Dans [127], les auteurs étudient les performances de SCIP si différents solveurs NLP locaux sont utilisés pour la résolution des relaxations convexes. La comparaison porte sur deux codes SQP et deux codes de points intérieurs, tous reconnus pour leurs performances, et nous en résumons les conclusions. Si les méthodes de points intérieurs sont plus robustes et permettent de résoudre plus de problèmes, l'utilisation d'un solveur SQP permet d'obtenir les meilleures performances en temps. Les auteurs notent que ceci semble être dû en grande partie à la vitesse de détection de l'inconsistance et à l'absence de phases de secours, appelées uniquement si l'algorithme principal échoue. De façon générale, la gestion de l'inconsistance et la rapidité de l'échec sont deux points clés pour la performance globale. Les auteurs notent même que « Paramétrer un solveur pour trouver plus de solutions localement optimales détériore considérablement la performance moyenne ». Si une méthode de SQP est celle obtenant les meilleurs résultats en moyenne, la sélection du meilleur solveur à chaque nœud permet un gain d'un facteur 2 par rapport à celle-ci. La variabilité de la performance, au sein d'une même instance, est donc importante et montre que les solveurs non linéaires utilisés ont une marge de progrès certaine sur les problèmes non linéaires convexes.

Dans [101], les auteurs décrivent la mise en place, dans le solveur BARON, d'un paradigme hybride LP/NLP pour la résolution des relaxations convexes. Trois ingrédients principaux participent aux résultats obtenus : 30% d'instances supplémentaires résolues sur des librairies classiques, dans le cas d'un temps limite de 500 secondes. Premièrement, la détection automatique de la convexité de la relaxation continue du problème sous forme de DAG. En effet, si celle-ci est convexe pour un nœud donné, c'est elle qui est résolue pour trouver une borne inférieure. L'idée est que la relaxation convexe du modèle factorisé contient plus de variables, et a généralement détruit la convexité par ajout d'égalités non linéaires. Deuxièmement, un apprentissage dynamique du meilleur solveur local, pour résoudre le problème de la variabilité de performance de chacun d'entre eux au cours du *branch-and-bound*. Enfin, en cas de difficultés numériques ou d'absence de gains à résoudre la relaxation non linéaire, un changement de stratégie automatique vers des relaxations polyédrales. On peut noter de plus quelques spécificités d'implémentation, comme

la résolution d'une première relaxation polyédrale pour initialiser le solveur non linéaire, ou le fait que l'appel aux solveurs NLP ne se fait que si la relaxation continue est convexe. Il est noté que pour des questions de robustesse, les certificats de KKT retournés par les solveurs sont systématiquement vérifiés (pour l'optimalité et pour l'inconsistance). Parmi les solveurs locaux utilisés, on trouve des représentants des trois catégories décrites plus haut, avec des différences de performance modestes (dans le cas où la sélection dynamique du meilleur solveur local est désactivée).

Approche retenue

Notre objectif n'est pas d'utiliser le meilleur solveur non linéaire disponible, mais d'en développer un sur-mesure. De plus, pour l'intégration à LocalSolver, des questions légales de licences se posent lors de l'utilisation de tout code externe. Il n'est pas non plus de développer un solveur tout-en-un idéal, qui dominerait les autres pour résoudre une relaxation convexe donnée. On souhaite plutôt mettre en place un algorithme guidant le calcul de bornes inférieures aux relaxations convexes du modèle factorisé. De plus, le contexte du *branch-and-bound* nous impose de résoudre une suite de relaxations proches les unes des autres. Enfin, l'objectif est aussi de mettre en évidence le fait qu'en exploitant la structure du modèle factorisé et le contexte du calcul de bornes, une approche simple peut déjà concurrencer l'utilisation de relaxations linéaires et du simplexe. Des pistes pour améliorer les performances avec des techniques de l'état de l'art pourront toujours être explorées plus tard, et certaines seront évoquées lors des perspectives.

La méthode la plus prometteuse pour la résolution des relaxations convexes non linéaires est l'approche par SQP. De bons résultats expérimentaux ont déjà été obtenus, et la vitesse de convergence asymptotique quadratique, couplée aux capacités de démarrage à chaud, en font le meilleur candidat. La difficulté est alors l'implémentation, puisque nous ne disposons pas de méthode de résolution des problèmes quadratiques. Les méthodes de points intérieurs sont les plus intéressantes si l'on souhaite résoudre un unique problème non linéaire convexe. Dans ce cas, l'approche idéale serait de résoudre le *root node* avec une méthode de points intérieurs dédiés, puis les nœuds de l'arbre avec une méthode de SQP. Enfin, si la vitesse de convergence théorique de l'algorithme du lagrangien augmenté est inférieure à celles des deux autres, il est le seul à fournir des bornes inférieures pendant la recherche (en théorie, tout point du chemin central fournit aussi une borne inférieure, mais les algorithmes de points intérieurs performants en pratique n'y restent pas, et se limitent plutôt à un voisinage de celui-ci).

Finalement, l'approche retenue est la méthode du lagrangien augmenté. Ce choix a été motivé, en plus des remarques précédentes, par deux éléments principaux. Premièrement, les résultats de dualité présentés en début de chapitre ont été obtenus et formalisés après l'implémentation du solveur. Ces résultats permettent d'obtenir une borne à chaque itération pour les trois approches, mais sans ceux-ci, seul le lagrangien augmenté visite explicitement des points stationnaires du lagrangien. Ensuite, le temps dédié au développement du solveur non linéaire est limité. L'optimisation globale est un assemblage de multiples composants, dont la diversité prime sur la performance. Pour calculer des bornes sur le plus de problèmes possibles, et assurer

à LocalSolver la fiabilité numérique de l'ensemble, il a été nécessaire de limiter le temps passé à la mise en place d'une méthode de résolution des relaxations non linéaires.

3.2 Relaxations lagrangiennes augmentées

Dans cette section, nous proposons une introduction à la méthode du lagrangien augmenté [137, 140]. En particulier, on s'intéresse à sa genèse et aux différentes méthodes dont elle est issue. La relation particulière de ces méthodes, et donc de l'algorithme du lagrangien augmenté, avec les bornes inférieures justifie en partie le choix de son utilisation. Des œuvres et implémentations de référence sont [44, 50, 63, 128].

3.2.1 Origine et intérêt de la méthode

Pénalisation extérieure

La méthode de pénalisation extérieure a pour but la résolution de problèmes non linéaires contraints. On suppose pour le moment que le problème est de la forme

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} f(x), \\ &\text{s.c.} \Big\{ g(x) = 0. \end{aligned}$$

L'idée est de se ramener à l'optimisation sans-contraintes, en pénalisant les violations des contraintes dans l'objectif. On commence par introduire une fonction de pénalisation $p(x)$, typiquement la pénalisation quadratique $p(x) = \frac{1}{2}\|g(x)\|^2$, ainsi qu'un facteur de pénalisation $\rho > 0$. On résout alors une suite de problèmes sans contraintes, paramétrée par ρ, pour $\rho \to +\infty$:

$$\min_{x\in\mathbb{R}^n} f(x) + \frac{\rho}{2}\|g(x)\|^2.$$

Ce problème pénalisé constitue une relaxation du problème initial, et sa résolution à l'optimum fournit une borne inférieure. La méthode de la pénalisation extérieure est donc une première source potentielle de bornes inférieures. Sous les hypothèses adéquates, et en notant x_ρ la solution du problème pénalisé par ρ, $(x_\rho, \rho g(x_\rho))$ converge vers une solution primale-duale du problème initial. Ce résultat découle de la similarité entre les conditions de stationnarité du problème pénalisé $(\nabla f(x) + \sum_i (\rho g_i(x)) \nabla g_i(x) = 0)$ et du problème initial $(\nabla f(x) + \sum_i \lambda_i \nabla g_i(x) = 0)$. Le principal défaut de cette méthode est que l'on a besoin de faire tendre le facteur de pénalisation vers l'infini pour obtenir la convergence. L'objectif pénalisé devient mal conditionné et de plus en plus difficile à minimiser.

Dualité lagrangienne

En reprenant les notations du paragraphe précédent, on introduit λ, les multiplicateurs associés aux contraintes d'égalité. Le lagrangien du problème s'écrit

$\mathcal{L}(x, \lambda) = f(x) + \lambda^T g(x)$, et donne une formulation min-max du problème :

$$\min_{g(x)=0} f(x) = \min_x \max_\lambda \mathcal{L}(x, \lambda).$$

Cette formulation est ensuite transformée par dualité en le problème de maximisation concave

$$\max_\lambda \theta(\lambda),$$

où $\theta(\lambda) = \min_x \mathcal{L}(x, \lambda)$ est la fonction duale. Cette dernière est concave et sur-différentiable, et ses sur-gradients sont donnés par les $g(x_\lambda)$, pour les x_λ réalisant l'argmin dans la définition de θ.

Ici aussi, le calcul de θ revient à minimiser une fonction différentiable sans contraintes, et la solution optimale fournit une borne inférieure. La convergence pour le problème initial s'obtient en appliquant un algorithme d'optimisation non différentiable à θ (méthode de sur-gradient, de faisceaux, ...). Le principal problème réside alors dans le fait que ces méthodes ne passent pas à l'échelle sur les tailles de problèmes que l'on souhaite résoudre.

La méthode des multiplicateurs

La méthode des multiplicateurs a été introduite dans un but algorithmique et numérique, afin de résoudre les problèmes pratiques des deux méthodes que nous venons de décrire. Elle consiste à ajouter au lagrangien la fonction de pénalisation quadratique extérieure. Pour un facteur de pénalisation $\rho > 0$, le lagrangien augmenté s'écrit alors :

$$\mathcal{L}_\rho(x, \lambda) = f(x) + \lambda^T g(x) + \frac{\rho}{2}\|g(x)\|^2, \text{ et}$$

$$\nabla \mathcal{L}_\rho(x, \lambda) = \nabla f(x) + \sum_i (\lambda_i + \rho g_i(x)) \nabla g_i(x). \tag{3.6}$$

On alterne ensuite des itérations de minimisation de $\mathcal{L}_\rho$ sur $\mathbb{R}^n$ et des mises à jour des multiplicateurs. La dualité min-max a lieu de la même manière qu'avec le lagrangien simple, et la méthode des multiplicateurs fournit une fois de plus une solution duale faisable pour chaque sous-problème résolu. La formule de mise à jour des multiplicateurs peut s'obtenir de deux façons, exposées plus bas, selon que l'on interprète la méthode du lagrangien augmenté dans le paysage d'optimisation primal, où elle est une amélioration de la méthode de pénalisation, ou dans le paysage dual, où elle est une amélioration de l'algorithme de sur-gradient donné plus haut. La prise en compte des multiplicateurs dans la méthode de pénalisation extérieure permet d'obtenir la convergence pour un facteur de pénalisation borné, et donne un pas de déplacement naturel pour leur mise à jour.

Interprétation primale

Dans la méthode de pénalisation extérieure, le gradient de la pénalisation $\frac{\rho}{2}\|g(x)\|^2$ est nul sur l'ensemble admissible. Cette remarque, couplée à la condition de stationnarité de l'objectif pénalisé, est à la source de l'hypothèse $\rho \to +\infty$, nécessaire à la

convergence. L'idée permettant de résoudre le problème est d'utiliser une pénalisation à deux paramètres : un facteur de pénalisation, et un facteur de translation. Le but de ce dernier est de pénaliser l'ensemble admissible translaté, afin que le gradient de la pénalisation ne soit plus nul pour les points admissibles. Le lagrangien augmenté est un exemple de telle pénalisation à deux facteurs. Pour le voir, on réarrange les termes :

$$\mathcal{L}_\rho(x,\lambda) = f(x) + \frac{\rho}{2}\|g(x) + \lambda/\rho\|^2 - \frac{1}{2\rho}\|\lambda\|^2,$$

on sort la constante $\rho\|\lambda\|^2/2$ et on interprète $s = \lambda/\rho$ comme le facteur de translation :

$$\mathcal{L}_{\rho,s}(x) = f(x) + \frac{\rho}{2}\|g(x) + s\|^2.$$

Pour obtenir la règle de mise à jour des multiplicateurs, on analyse les conditions de stationnarité du problème initial : $\nabla f + \sum_i \lambda_i \nabla g_i(x) = 0$, et du problème pénalisé : $\nabla f + \sum_i \rho(s + g_i(x))\nabla g_i(x)$. L'approximation des multiplicateurs est donc donnée par $\rho(s + g_i(x))$, soit $\lambda_i + \rho g_i(x)$, une fois la définition de s substituée. La règle de mise à jour des multiplicateurs est donc $\lambda_i \leftarrow \lambda_i + \rho g_i(x)$.

Interprétation duale

Dans le paysage dual, l'ajout de la pénalisation dans la définition de la fonction duale s'interprète comme l'utilisation de l'algorithme proximal à la place d'une méthode de sur-gradients. Dans ce cas, pour un paramètre $r_k > 0$, la mise à jour des multiplicateurs est donnée par une itération de l'algorithme proximal :

$$\lambda_{k+1} \in \operatorname{argmax}\ \theta(\lambda) - \frac{1}{2r_k}\|\lambda - \lambda_k\|^2.$$

Ceci se réécrit par concavité $0 \in \partial\theta(\lambda_{k+1}) - (\lambda_{k+1} - \lambda_k)/r_k$, puis se simplifie en $\lambda_{k+1} \in \lambda_k + r_k\partial\theta(\lambda_{k+1})$. On ne connaît cependant pas d'élément de $\partial\theta(\lambda_{k+1})$. En effet, la définition de θ nous assure uniquement que $g(x_k) \in \partial\theta(\lambda_k)$. On procède alors par analyse-synthèse pour obtenir à la place $g(x_k) \in \partial\theta(\lambda_{k+1})$. Une solution à l'itération de l'algorithme proximal est dans ce cas $\lambda_{k+1} = \lambda_k + r_k g(x_k)$. La condition $g(x_k) \in \partial\theta(\lambda_{k+1})$ signifie alors que x_k est stationnaire pour le lagrangien utilisant λ_{k+1}, soit

$$0 = \nabla f(x_k) + \sum_i \lambda_{k+1}\nabla g(x_k) = \nabla f(x_k) + \sum_i (\lambda_{k,i} + r_k g_i(x_k))\nabla g_i(x_k).$$

Cette condition est celle de stationnarité du lagrangien augmenté (3.6) utilisant un facteur de pénalisation r_k. Finalement, dérouler l'algorithme proximal pour la maximisation de θ avec une suite de paramètres (r_k) revient à minimiser le lagrangien augmenté pour la suite de facteurs de pénalisation (r_k).

Hiérarchie de bornes

L'utilisation de relaxations lagrangiennes augmentées et du résultat de dualité donné au théorème 3.1.4 nous donne accès à une hiérarchie de bornes inférieures, de plus en plus faciles à obtenir. Pour un problème $\min \{f(x) \mid g(x) = 0,\ h(x) \leq 0\}$, de relaxation convexe $\min \{\tilde{f}(x) \mid \tilde{g}(x) = 0,\ \tilde{h}(x) \leq 0\}$, la hiérarchie des bornes inférieures utilisées est illustrée dans la figure 3.4.

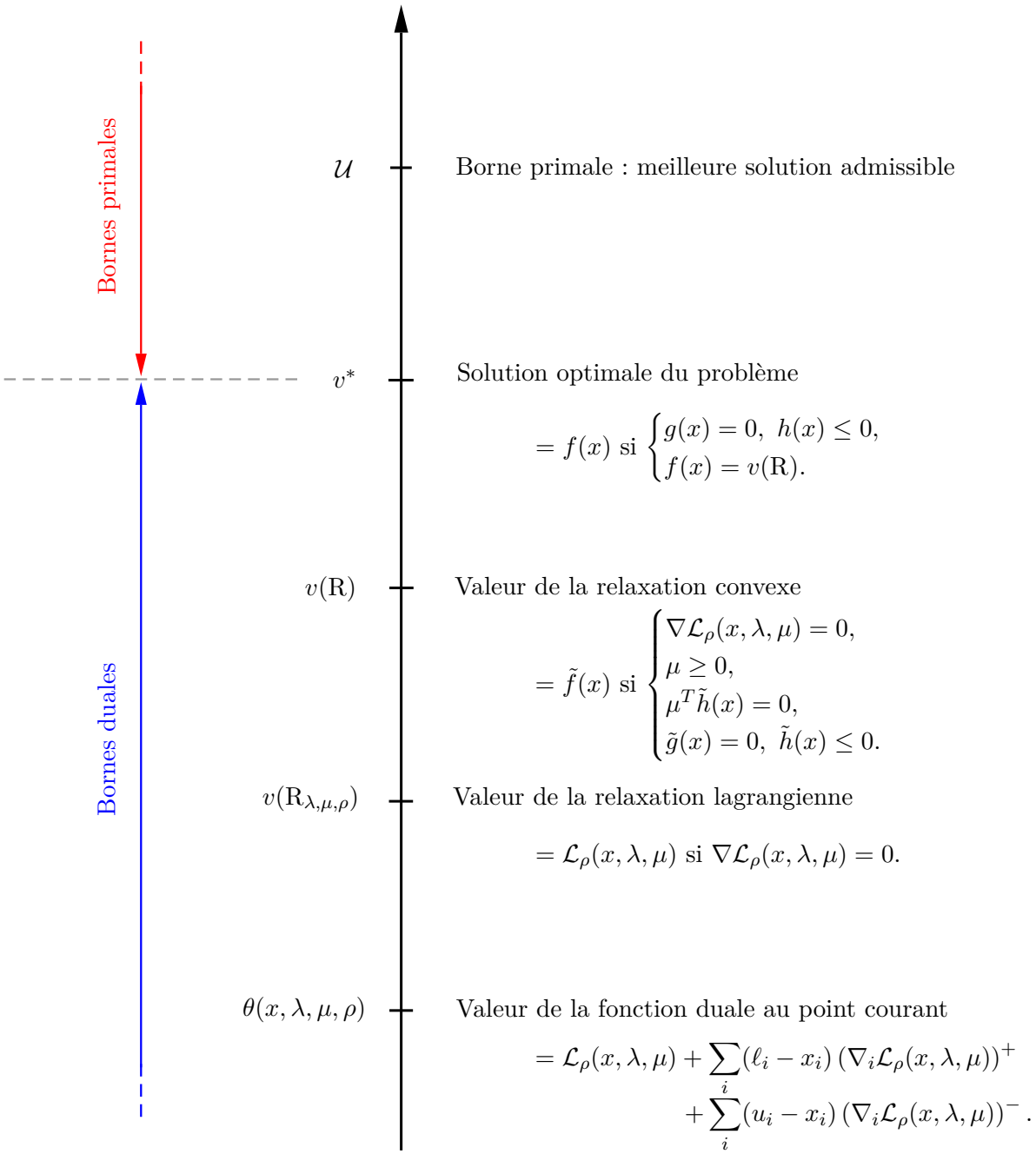

FIGURE 3.4 – *Hiérarchie des bornes inférieures utilisées*

3.2.2 L'algorithme du lagrangien augmenté

On s'intéresse maintenant à l'implémentation de la méthode des multiplicateurs pour la résolution d'un problème de minimisation convexe contenant des égalités (forcément linéaires) et des inégalités :

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} f(x), \\ &\text{s.c.} \begin{cases} g(x) = 0 \in \mathbb{R}^p, \\ h(x) \leq 0 \in \mathbb{R}^q. \end{cases} \end{aligned} \tag{3.7}$$

Schéma itératif général

La méthode décrite dans les paragraphes précédents se généralise au cas des contraintes d'inégalité. En introduisant des variables d'écart signées, on commence par se ramener au cas des égalités seules. Le problème de minimisation du lagrangien augmenté est séparable en ces variables d'écart, qui peuvent être éliminées par minimisation explicite. À une constante près, l'algorithme revient alors à minimiser

$$\mathcal{L}_\rho(x,\lambda,\mu) = f(x) + \frac{\rho}{2}\left\|g(x) + \lambda/\rho\right\|^2 + \frac{\rho}{2}\left\|(h(x) + \mu/\rho)^+\right\|^2,$$

et utilise la règle de mise à jour $\begin{cases} \lambda_{k+1} = \lambda_k + \rho_k g(x_k), \\ \mu_{k+1} = (\mu_k + \rho_k h(x_k))^+. \end{cases}$

Jusqu'à maintenant, toutes les contraintes ont été dualisées dans le lagrangien, et les sous-problèmes résolus sont des problèmes de minimisation sans contraintes. Dans le cas général, il est possible de partitionner les contraintes en deux, et de conserver l'une des deux parties dans les sous-problèmes. À partir de maintenant, on note $x \in X$ les contraintes non dualisées, qui sont donc présentes dans les sous-problèmes.

Une condition importante pour la convergence de la méthode du lagrangien augmenté est la bornitude des multiplicateurs et du facteur de pénalisation. Pour assurer ceci en pratique, on utilise un garde-fou par-dessus les règles de mise à jour des multiplicateurs présentée plus haut. On sélectionne une valeur maximale M, et on impose à tous les multiplicateurs de rester inférieurs à M (en valeur absolue), *via* l'utilisation de $\mathcal{P}_{[a,b]}$, le projecteur orthogonal sur l'intervalle $[a,b]$. La méthode de résolution de (3.7) utilisant le lagrangien augmenté est résumé dans l'algorithme 2. Si le problème est consistant, sous les hypothèses de régularité adéquates, l'algorithme converge vers une solution optimale du problème [50]. La vitesse de convergence asymptotique est de plus linéaire.

Choix des contraintes non dualisées

Lorsqu'un ensemble X de contraintes n'est pas dualisé, les sous-problèmes résolus sont de la forme

$$\min_{x\in X} \mathcal{L}(x,\lambda,\mu).$$

Algorithme 2 Algorithme de résolution de (3.7) par la méthode des multiplicateurs.

1 : **fonction** AUGMENTEDLAGRANGIAN
2 : Choisir X un ensemble de contraintes non dualisées ;
3 : Initialiser les variables $(x_0, \lambda_0, \mu_0) \in \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{R}^q_+$;
4 : Choisir un facteur de pénalisation $\rho_0 > 0$;
5 : $k \leftarrow 0$;
6 : **tant que** les conditions d'arrêt ne sont pas vérifiées **faire**
7 : Résoudre le sous-problème $x_{k+1} \in \underset{x \in X}{\operatorname{argmin}}\ \mathcal{L}_{\rho_k}(x, \lambda_k, \mu_k)$;
8 : $\lambda_{k+1} \leftarrow \mathcal{P}_{[-M,M]}\Big(\lambda_k + \rho_k g(x_{k+1})\Big)$;
9 : $\mu_{k+1} \leftarrow \mathcal{P}_{[0,M]}\Big(\mu_k + \rho_k h(x_{k+1})\Big)$;
10 : Mettre à jour le facteur de pénalisation ρ_{k+1} ;
11 : $k \leftarrow k + 1$;
12 : **fin tant que**
13 : **fin fonction**

Le choix de X est donc limité par les méthodes de résolution des sous-problèmes correspondants. On peut citer cinq approches viables pour X.

La première option est de dualiser toutes les contraintes, et de résoudre des sous-problèmes non contraints. L'intérêt principal de cette approche est que l'on peut appliquer directement les méthodes d'optimisation sans contraintes. En particulier, on peut appliquer directement la méthode de Newton et ses variantes. Cette approche est celle utilisée dans le solveur PENNON [103], même si la forme du lagrangien utilisé diffère de l'approche traditionnelle présentée précédemment.

La seconde option est de dualiser toutes les contraintes, sauf celles de borne. Grâce à leur structure séparable et orthogonale, il est facile de projeter sur le polyèdre défini par celles-ci. Les sous-problèmes sont résolus par des méthodes d'activation de contraintes et contiennent une partie combinatoire. Un autre intérêt des sous-problèmes avec contraintes de bornes est que l'application de nos résultats de dualité y est la plus favorable. Cette approche est la plus fréquemment implémentée, par exemple dans les solveurs LANCELOT [63] et ALGENCAN [26].

La troisième option est de dualiser toutes les contraintes, sauf les égalités, puisque celles-ci sont linéaires dans le cas convexe. Cette approche peut être envisagée car il est possible d'obtenir une expression explicite du projecteur orthogonal sur un sous-espace affine. Pour un point initial admissible, projeter les directions de recherche permet de maintenir la faisabilité des itérés. Malheureusement, les projecteurs orthogonaux font intervenir une algèbre linéaire qui est généralement pleine, introduisant ainsi un coût mémoire supplémentaire. Si des méthodes existent pour obtenir des projecteurs creux, celles-ci ont l'effet de détériorer le conditionnement du problème.

La quatrième option est de ne dualiser que les contraintes non linéaires. Dans

ce cas, les sous-problèmes consistent à minimiser une fonction non linéaire dans un polyèdre. On peut alors utiliser les techniques d'algèbre linéaire développées pour les algorithmes de simplexe, avec quelques adaptations puisque l'optimum ne se trouve pas nécessairement sur un sommet du polyèdre, ni même sur sa frontière. Cette approche est implémentée dans le solveur MINOS [128].

Enfin, il est possible de garder une seule contrainte dans les sous-problèmes, pourvu que l'on ait un projecteur orthogonal facile à calculer. C'est le cas par exemple des contraintes simpliciales $\sum_i x_i = 1$, ou encore des contraintes quadratiques convexes $\|Ax + b\|^2 \leq \delta$. Cette approche est cependant limitée à certains problèmes particuliers.

Dans le cadre de cette thèse, nous nous sommes limités à la première approche, c'est-à-dire à la résolution de sous-problèmes non contraints. Les raisons de ce choix sont avant tout la facilité d'implémentation, le contrôle de la géométrie des sous-problèmes, et une volonté de pousser la robustesse au maximum dans ce cas simple. Les sous-problèmes sous contraintes de bornes restent cependant attractifs, mais leur composante combinatoire peut être problématique. De plus, la généralisation de la méthode de Newton au cas des contraintes de bornes se fait généralement avec l'utilisation d'une méthode par régions de confiance, plus complexes et difficiles à régler numériquement que les méthodes de recherche linéaire, dont l'implémentation est l'objet de la dernière section de ce chapitre.

Gestion de la pénalisation

Pour des fonctions convexes deux fois différentiables, les conditions sur ρ nécessaires à la convergence de l'algorithme 2 sont faibles. Par exemple, un facteur $\rho > 0$ constant est suffisant pour l'obtenir. L'interprétation duale de l'algorithme donne quelques indices, mais les conditions théoriques de convergence de l'algorithme proximal, concernant la nature convergente ou divergente de certaines séries, ont peu d'intérêt pratique dans notre contexte. L'interprétation primale ne donne quant à elle aucune règle naturelle pour la mise à jour du facteur de pénalisation. Par conséquent, les formules utilisées dans la littérature sont largement heuristiques. Dans des cas spécifiques, comme celui de la programmation quadratique convexe, le facteur de pénalisation peut être relié au taux de convergence de l'algorithme [70, 62]. Selon la vitesse de convergence souhaitée, des formules explicites sont alors possibles pour ρ.

Dans le cas général, le choix du paramètre de pénalisation fait l'objet d'un compromis. Un facteur de pénalisation plus élevé donnera une mise à jour plus précise des multiplicateurs, et le nombre de sous-problèmes résolus sera réduit. En contrepartie, les sous-problèmes sont moins bien conditionnés, et plus longs à résoudre avec des méthodes itératives. L'heuristique universellement utilisée est de suivre l'évolution de la violation des contraintes par la solution des sous-problèmes. Le facteur de pénalisation est augmenté lorsqu'une certaine mesure d'infaisabilité n'a pas suffisamment décru, et laissé constant dans le cas contraire. En notant $\Psi(x) = (g(x), h^+(x))$ le vecteur des violations des contraintes, des mesures de faisabilité naturelles sont $\|\Psi(x)\|_p$, pour $p = 1, 2$ ou ∞. On choisit alors $\tau > 1$, le taux de mise à jour du facteur de pénalisation, et $\gamma < 1$, la décroissance cible pour la mesure d'infaisabilité.

Dans ce cas, une mise à jour typique du facteur de pénalisation est :

$$\rho_{k+1} = \begin{cases} \rho_k, \text{ si } \|\Psi(x_{k+1})\|_p \leq \gamma \|\Psi(x_k)\|_p, \\ \tau \rho_k, \text{ sinon.} \end{cases}$$

Deux améliorations classiques sont ensuite apportées à ce schéma élémentaire. La première est d'utiliser un facteur de pénalisation par contrainte. L'idée est de limiter la détérioration du conditionnement si la faisabilité est longue à atteindre, en ne pénalisant pas outre mesure les contraintes faciles à satisfaire. On note donc désormais $\rho \in \mathbb{R}_+^{p+q}$ les facteurs de pénalisation.

Pour converger vers une solution satisfaisant les conditions de KKT, il faut aussi prendre en compte la violation de la complémentarité. On considère par exemple une inégalité $h_i(x) \leq 0$, dont la variable d'écart implicite est une fonction de x, $s_i(x) = \max(0, -h_i(x) - \mu_i/\rho_i)$, puisqu'éliminée de la fonction duale. Pour être homogène avec le cas des égalités, la quantité à suivre n'est pas $h_i^+(x)$, mais plutôt $h_i(x) + s_i(x)$, c'est-à-dire $\max(h_i(x), -\mu_i/\rho_i)$. La convergence de cette quantité vers 0 suffit alors à garantir la convergence vers une solution admissible et complémentaire. En effet, si $h_i^+(x)$ et la translation μ_i/ρ_i tendent tous deux vers 0, la complémentarité aura lieu. Pour le voir, on peut distinguer les cas $h_i(x) \to 0$ et $h_i(x) \to h_i^* < 0$. Dans le premier, la complémentarité a lieu quel que soit μ_i. Dans le second, $h_i(x) + \mu_i/\rho_i \to h_i^*$, $\mu_i + \rho_i h_i(x)$ est équivalent à $\rho_i h_i^*$ et devient donc négatif, auquel cas la règle de mise à jour de μ_i implique que $\mu_i \to 0$ et la complémentarité a lieu.

En notant ρ_k^i le facteur de pénalisation de la contrainte i à l'itération k de l'algorithme 2, la procédure de mise à jour des facteurs de pénalisation est résumée dans l'algorithme 3. Comme pour les multiplicateurs, on impose aux facteurs de pénalisation de rester bornés à l'aide d'une valeur maximale $\rho_{\max}$.

De nombreux travaux ont cherché à améliorer les règles de mise à jour du facteur de pénalisation, dont nous donnons ici deux exemples. Dans [66], les auteurs augmentent celui-ci si la minimisation du modèle quadratique du lagrangien augmenté n'améliore pas suffisamment la faisabilité. Pour cela, ils comparent la décroissance observée de la mesure de faisabilité à celle obtenue en minimisant directement cette dernière dans une certaine région de confiance. L'impact sur un algorithme de lagrangien augmenté élémentaire est net, mais plus modeste une fois intégré à un code industriel (LANCELOT). Dans [47], des règles de mise à jour non monotones sont mises au point. Une fois encore, les gains observés restent modestes.

Détection de l'inconsistance

Comme expliqué dans la partie 3.1.2, la gestion des sous-problèmes inconsistants est cruciale en optimisation globale. La convergence de la méthode du lagrangien augmenté dans le cas inconsistant a été étudiée et qualifiée, par exemple dans [50, 70]. En modifiant les règles de mise à jour des multiplicateurs pour les remettre à zéro lorsque ceux-ci deviennent trop grand, l'algorithme 2 converge vers un minimum de la fonction $\|g(x)\|^2 + \|h^+(x)\|^2$. Nous utilisons plutôt le corollaire 3.1.5, qui nous donne un certificat d'inconsistance robuste aux erreurs numériques.

Algorithme 3 Mise à jour des facteurs de pénalisation après la résolution d'un sous-problème.

fonction UPDATEPENALIZATION
  pour $(i = 1..p)$ **faire**
    $\rho^i_{k+1} \leftarrow \begin{cases} \rho^i_k, \text{ si } |g_i(x_{k+1})| \leq \gamma |g_i(x_k)|, \\ \tau \rho^i_k, \text{ sinon.} \end{cases}$
    $\rho^i_{k+1} \leftarrow \min(\rho^i_{k+1}, \rho_{\max})$
  fin pour
  pour $(i = 1..q)$ **faire**
    $\rho^{p+i}_{k+1} \leftarrow \begin{cases} \rho^{p+i}_k, \text{ si } |\max(h_i(x_{k+1}), -\frac{\mu^i_{k+1}}{\rho^{p+i}_{k+1}})| \leq \gamma |\max(h_i(x_k), -\frac{\mu^i_k}{\rho^{p+i}_k})|, \\ \tau \rho^{p+i}_k, \text{ sinon.} \end{cases}$
    $\rho^{p+i}_{k+1} \leftarrow \min(\rho^{p+i}_{k+1}, \rho_{\max})$
  fin pour
fin fonction

Dans la figure 3.5, on a tracé le nombre et la proportion des nœuds déclarés inconsistants par les solveurs lors du *branch-and-bound*. On y voit que des nœuds inconsistants sont identifiés par les solveurs sur plus de 30% des instances. De plus, les capacités de détection d'inconsistance de notre solveur non linéaire semblent être au niveau de celles de l'algorithme du simplexe dual.

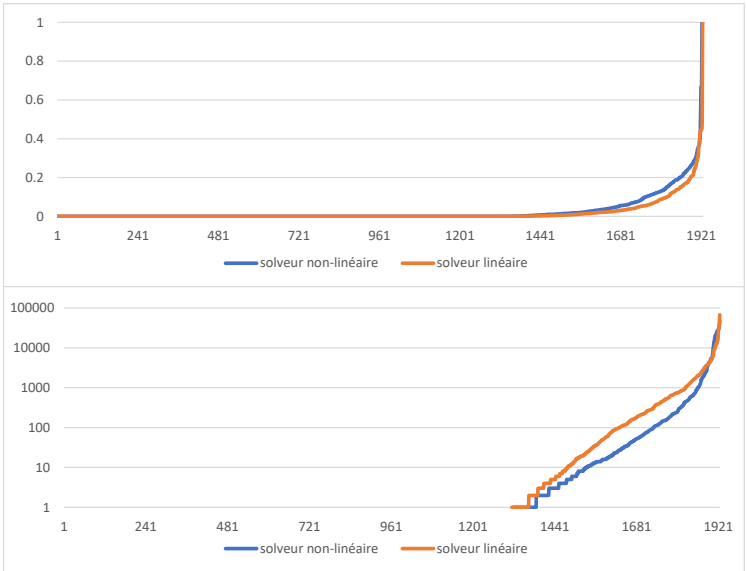

FIGURE 3.5 – *Nœuds déclarés inconsistants par les solveurs, en haut en proportion, en bas en nombre total. Les instances considérées sont celles où au moins un nœud a été visité.*

La figure 3.6 montre l'importance de la détection des nœuds inconsistants sur la performance globale. On y considère 760 instances non résolues, que la détection de l'inconsistance soit activée ou non, et on compare le nombre de nœuds visités selon que cette détection soit active ou non. Le résultat est clair, ne pas détecter l'inconsistance détériore la performance d'énumération du *branch-and-bound*.

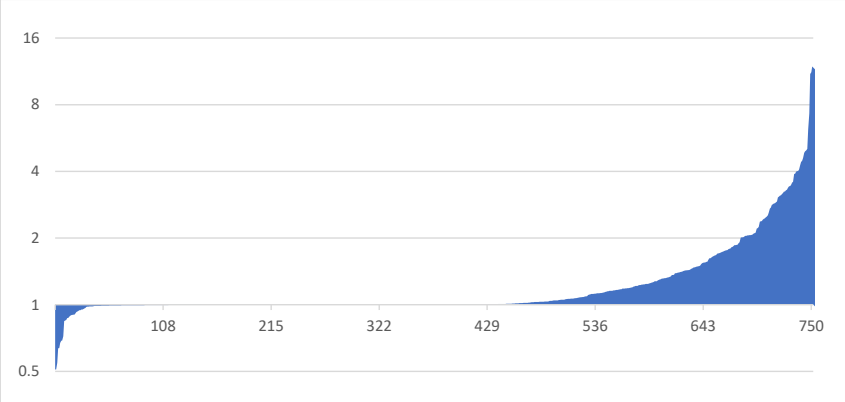

FIGURE 3.6 – *Sur les instances non résolues, ni avec, ni sans détection de l'inconsistance, impact de la détection de l'inconsistance sur la vitesse d'énumération (en nombre de nœuds visités).*

Critères d'arrêt

Dans l'algorithme 2, il reste à expliquer les conditions d'arrêt évoquées à la ligne 6. Généralement, puisque la stationnarité du lagrangien est l'objet de la résolution du sous-problème, l'algorithme est interrompu dès que la faisabilité et la complémentarité sont atteintes, à une tolérance près. Cependant, les conditions de KKT ne sont pas nécessaires en l'absence de qualification des contraintes [122]. Si l'on considère par exemple le problème d'optimisation convexe

$$\begin{aligned} &\min_{\mathbb{R}} x \\ &\text{s.c.} \left\{ x^2 \leq 0, \right. \end{aligned}$$

la complémentarité donne $\mu x^2 = 0$, soit $\mu x = 0$. Avec la stationnarité $1 + 2\mu x = 0$, ceci implique $1 = 0$. Les conditions de KKT n'ont donc pas de solution sur ce problème. La convergence de la méthode du lagrangien augmenté a été étudiée dans le cas où la qualification des contraintes n'a pas lieu [27, 50]. En résumé, si l'on est capable de résoudre les sous-problèmes, la convergence a lieu. Détecter cette dernière et atteindre les niveaux de tolérance souhaités n'est cependant pas toujours facile. Pour cette raison, si l'algorithme n'a pas convergé au bout de 100 itérations (et donc 100 sous-problèmes), un échec est déclaré. Si 10 sous-problèmes consécutifs échouent, un échec est aussi déclaré. Enfin, nous utilisons un critère d'arrêt supplémentaire, basé sur le corollaire 3.1.4, qui mesure l'optimalité en valeur.

Le problème (3.7) est une relaxation convexe à résoudre. On note $\overline{f}$ la borne primale de cette relaxation, c'est-à-dire la meilleure solution admissible visitée. De

même, on note $\underline{f}$ la borne duale de cette relaxation, obtenue par l'application du corollaire 3.1.4 à chaque itération de résolution de chaque sous-problème. La solution admissible associée à $\overline{f}$ est déclarée optimale, et la résolution interrompue, dès que

$$\frac{\overline{f} - \underline{f}}{1 + |\overline{f}|} \leq \varepsilon_{\text{gap}}, \tag{3.8}$$

où ε_{gap} désigne la précision souhaitée sur la valeur optimale de (3.7). Ce critère peut être amélioré, en remarquant qu'il n'y a pas d'intérêt à résoudre les relaxations qui n'ont aucune chance d'améliorer la borne inférieure du sous-problème dont (3.7) constitue une relaxation. Si l'on note L cette borne, ceci revient à interrompre la résolution dès que $\overline{f} \leq L$. L'importance du critère d'arrêt (3.8) est illustré dans la figure 3.7. On y compare sa fréquence d'activation à celle du critère usuel, basé sur des conditions de KKT relâchées par des tolérances. L'impact sur la vitesse de branchement (c'est-à-dire sur le nombre de nœuds visités) est illustrée dans la figure 3.8. Le critère (3.8) permet de détecter l'optimalité plus tôt, et donc d'accélérer la recherche arborescente.

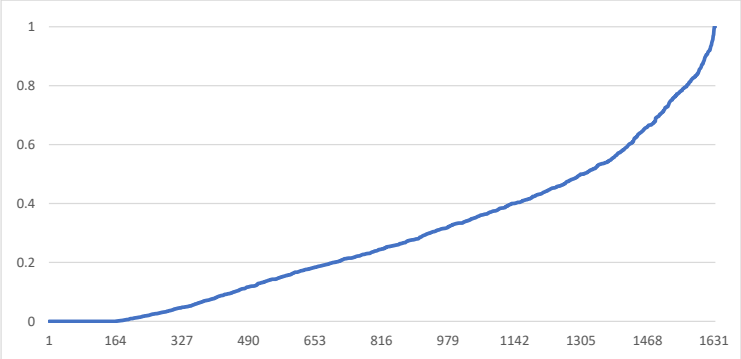

FIGURE 3.7 – *Proportions des relaxations où la solution finale est déclarée optimale par le critère d'arrêt* (3.8)*, triées par ordre croissant pour les instances où au moins une relaxation a été résolue.*

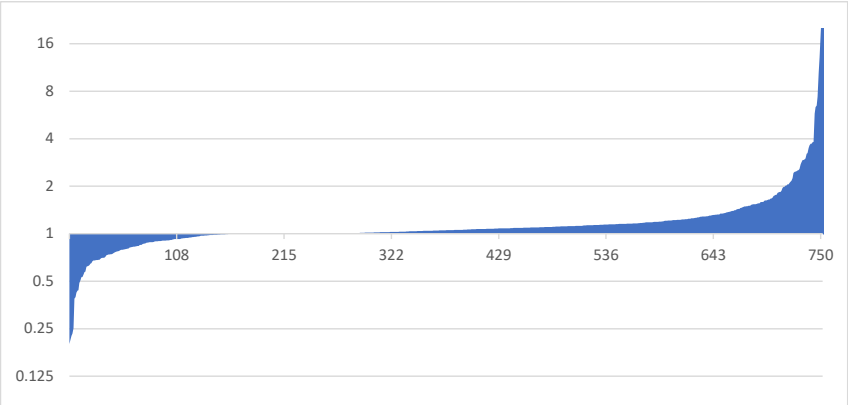

FIGURE 3.8 – *Impact du critère d'arrêt* (3.8) *sur la vitesse d'énumération (rapport des nombres de nœuds visités), instances non résolues.*

Impact du résultat de dualité sur la robustesse et la stabilité

Avant de parler des spécificités des sous-problèmes résolus dans le contexte du modèle factorisé, puis de présenter leur méthode de résolution, nous présentons quelques statistiques sur l'impact des corollaires 3.1.4 et 3.1.5. Le tableau 3.1 compare la borne inférieure finale, sur les instances non résolues, dans le cas où la dernière borne $\theta(x, \lambda, \mu, \rho)$ de la hiérarchie donnée dans la figure 3.4 est désactivée. Pour deux stratégies fournissant deux bornes L_1 et L_2, la différence relative de performance est mesurée avec $(L_1 - L_2)/(1 + |L_1| + |L_2|)$. Ce tableau est à comparer avec les tableaux 4.4, 4.5, 4.6 et 4.7 du chapitre 5, et montre que l'utilisation du dual de Wolfe à chaque itération a un impact similaire à une technique de réduction de bornes avancée telle que la SBBT, exposée en partie 4.3.2.

Impact	Proportion des instances
Détérioration supérieure à 10%	0.8%
Détérioration supérieure à 1%	5.3%
Détérioration inférieure à 1%	10.8%
Pas de changement de la borne	33.2%
Amélioration inférieure à 1%	25.4%
Amélioration supérieure à 1%	19.1%
Amélioration supérieure à 10%	5.4%

Tableau 3.1 – *Impact de la désactivation de la borne $\theta(x, \lambda, \mu, \rho)$ de la figure 3.4 sur la borne inférieure finale, pour les instances non résolues.*

Sur les instances résolues indépendamment leur utilisation, les résultats de dualité et d'inconsistance ont un impact sur le coût de résolution et sur la robustesse. Le tableau 3.2 mesure cet impact sur le coût de résolution. Le tableau 3.3 mesure cet impact sur la proportion des instances sujettes à au moins un échec du solveur ou du sous-solveur. Le sous-solveur est défini comme l'algorithme de résolution des sous-problèmes de la méthode lagrangien augmenté. Par nature de cette dernière, le sous-solveur est indépendant du reste de l'algorithme et peut être vu comme une brique interchangeable. La définition et les conditions d'un échec du sous-solveur est donnée dans la dernière section de ce chapitre.

	Temps de résolution	Évaluations de f	Systèmes linéaires résolus
Sans le corollaire 3.1.4	+14.9%	+31.1%	+12.7%
Sans le corollaire 3.1.5	+4%	+6.7%	+9.9%

Tableau 3.2 – *Sur les instances résolues, impact des résultats de dualité et d'inconsistance sur le coût de résolution, en moyenne géométrique translatée.*

	Stratégie par défaut	Sans le corollaire 3.1.5	Sans le corollaire 3.1.4
Échec(s) du solveur	8.7%	27.1%	15.6%
Échec(s) du sous-solveur	23.3%	30.6%	35.6%

Tableau 3.3 – *Sur les instances résolues, fréquence d'échec du solveur et du sous-solveur, selon l'activation des corollaires obtenus.*

3.3 Application au modèle factorisé

Comme expliqué dans la section précédente, nous avons choisi d'utiliser des sous-problèmes non contraints dans la méthode du lagrangien augmenté. Puisque nous connaissons la forme analytique des relaxations convexes du modèle factorisé, nous pouvons étudier ces sous-problèmes en détail. L'objectif est d'analyser les paysages d'optimisation associés, afin d'identifier et de traiter les problèmes numériques qui peuvent y apparaître. En particulier, on cherche à éliminer les problèmes de définition, de régularité et de mauvais conditionnement qui peuvent survenir.

3.3.1 Analyse des relaxations convexes

Nous commençons par analyser et régler certains problèmes posés par la formulation des relaxations convexes non linéaires.

Les opérateurs $\sqrt{\cdot}$ et log

Une contrainte dure est une contrainte implicite, causée par le domaine de définition d'un certain opérateur présent dans le modèle. Par exemple, l'expression $\sqrt{x-y}$ impose la contrainte $x - y \geq 0$, au sens où le lagrangien du problème ne peut être évalué dans le cas contraire. Généralement, les contraintes dures qui ne sont pas des bornes sont ignorées. En effet, en prenant la convention que l'objectif est identiquement égal à $+\infty$ si une contrainte dure est violée, ces points seront toujours rejetés par l'algorithme. Ceci pose ici cependant question, puisque les sous-problèmes ne sont plus réellement sans contraintes. Dans notre cas, traiter toutes les contraintes dures nous impose l'utilisation de sous-problèmes comportant des contraintes de bornes. La connaissance de l'expression analytique du modèle factorisé nous permet cependant d'éliminer ces contraintes dures, grâce à des reformulations.

Dans la plupart des cas, les contraintes dures viennent des opérateurs usuels non définis sur $\mathbb{R}^-$, c'est-à-dire $x \mapsto \sqrt{x}$ et $x \mapsto \log x$. Ces opérateurs sont cependant des bijections sur leur domaines de définition. Il est alors possible de les reformuler en utilisant leur bijection réciproque. Par exemple, grâce à la présence de la contrainte de borne $x \geq 0$, la contrainte convexe $y \leq \log(x)$, peut se reformuler en $e^y \leq x$, alors définie sur $\mathbb{R}^2$. De la même manière, $y = \sqrt{x}$ implique $y \geq 0$, et $y \leq \sqrt{x}$ est alors reformulée en $y^2 \leq x$.

La fonction puissance

L'opérateur racine carrée est un cas particulier de la fonction puissance $x \mapsto x^a$, qui peut elle aussi imposer une contrainte dure, ainsi que des problèmes de régularité. Par exemple si $0 < a < 1$, cette dernière n'est pas définie sur $\mathbb{R}^-$, et n'est de plus pas dérivable en 0. On distingue plusieurs cas pour la fonction puissance $x \mapsto x^a$, en supposant que $a \notin \{0, 1, 2\}$. Le cas $a < 0$ est particulier, puisque son domaine de définition varie selon a. Dans le modèle factorisé, $y = x^a,\ a < 0$ est reformulé en $z = x^{-a},\ y = 1/z$ dès la construction du modèle. On suppose donc que l'on a toujours $a > 0$, et quatre cas se présentent alors : a entier pair, a entier impair, $a > 1$ et $0 < a < 1$.

Si a est entier, la relation non linéaire ne pose pas de problème particulier, puisqu'elle est définie et $\mathcal{C}^\infty$ sur $\mathbb{R}$. À l'inverse, une puissance a non entière impose une contrainte dure $x \geq 0$. Puisque cette borne est présente dans le modèle factorisé, on étend par prolongement continu la définition de la fonction puissance sur $\mathbb{R}^-$, en définissant la fonction `safePow` :

$$\text{safePow}(x, a) = x^a \text{ si } x > 0 \text{ et } 0 \text{ sinon.}$$

Remplacer la fonction puissance par notre version prolongée ne pose pas de problème car les deux coïncident si les contraintes de bornes sont respectées, comme c'est le cas pour toute solution admissible. La nature géométrique du couplage par la fonction puissance prolongée est illustrée dans la figure 3.9.

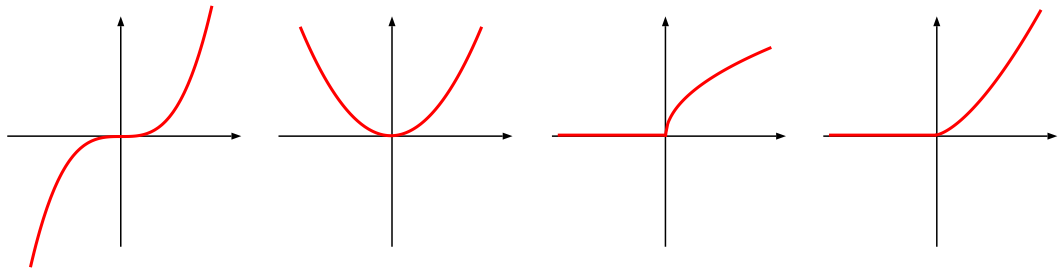

FIGURE 3.9 – *Les 4 cas de couplage non linéaire par la fonction puissance. Pour $k \in \mathbb{Z}$ et $a \notin \mathbb{Z}$, on a de gauche à droite : $y = x^{2k+1}$, $y = x^{2k}$, $y = x^a,\ 0 < a < 1$ et $y = x^a,\ a > 1$.*

Une fois les contraintes dures éliminées, il reste quelques problèmes numériques posés par la fonction puissance. Si $0 < a < 1$, le gradient a une singularité en $x = 0$. Cette remarque est à la base de travaux de régularisation, par exemple [172], dont l'objectif est de supprimer cette dernière. Un branchement cubique est effectué dans un voisinage de 0, afin d'obtenir une relaxation régulière. On utilise une autre approche, et encore une fois le fait que la fonction puissance est bijective sur $\mathbb{R}^+$. Comme pour la racine carrée, on peut donc reformuler $y = x^a$ en $x = y^{1/a}$ et se ramener au cas $a > 1$ Si $1 < a < 2$ et une fois prolongée, la relaxation $x^a - y \leq 0$ est convexe et différentiable sur $\mathbb{R}$, mais c'est la hessienne qui a un pôle en 0. Pour l'éliminer, on peut introduire une nouvelle variable z, ainsi que les contraintes $z = x^{ka}$ et $z = y^k$, avec $k = \lceil 2/a \rceil$. Cette reformulation modifie

cependant le nombre de variables et n'a pas été retenue dans l'implémentation. En effet, une singularité de la hessienne pose moins de problèmes numériques que pour le gradient.

Finalement, seules les relaxations convexes de $y = x^{2k}$, $y = x^{2k+1}$, $k \in \mathbb{N}^*$ et $y = \text{safePow}(x, a)$, $a > 1$, $a \notin \mathbb{N}$ peuvent apparaître dans les sous-problèmes. Ces relaxations n'imposent pas de contraintes dures, ni de problèmes de régularité car elles sont $\mathcal{C}^1$ sur $\mathbb{R}$ et $\mathcal{C}^\infty$ presque partout.

Domaine de validité des relaxations convexes

Les zones de convexité de f peuvent imposer des contraintes dures sur les estimateurs convexes utilisées dans les relaxations. Par exemple, si f est convexe dans $[\ell_x, u_x]$, $f(x) - y \leq 0$ fait partie de la relaxation convexe. Si f n'est pas convexe sur $\mathbb{R}$, on a alors une contrainte dure imposant à x de rester dans le domaine de convexité de f, puisque dans le cas contraire, il est possible qu'un point stationnaire du lagrangien ne soit pas son minimum.

Ceci se produit par exemple pour la relaxation sur un intervalle de $\mathbb{R}^+$ des couplages $y = 1/x$ et $y = x^{2k+1}$, mais aussi pour la relaxation des fonctions trigonométriques (même si l'on utilise que des relaxations linéaires pour le moment), et pour d'autres fonctions fréquentes que l'on ne traite pas encore, comme $x \mapsto xe^x$. La solution pour éliminer ce type de contraintes dures est d'effectuer un prolongement différentiable de l'estimateur convexe en question. Le principe est illustré sur la figure 3.10, et la relaxation convexe est alors donnée par :

$$\frac{f(u_x) - f(\ell_x)}{u_x - \ell_x} x + \frac{u_x f(\ell_x) - \ell_x f(u_x)}{u_x - \ell_x} \geq y \geq \underline{f}(x) = \begin{cases} f(x), & \ell_x \leq x \leq u_x, \\ \tan_{f,\ell_x}(x), & x < \ell_x, \\ \tan_{f,u_x}(x), & x > u_x. \end{cases}$$

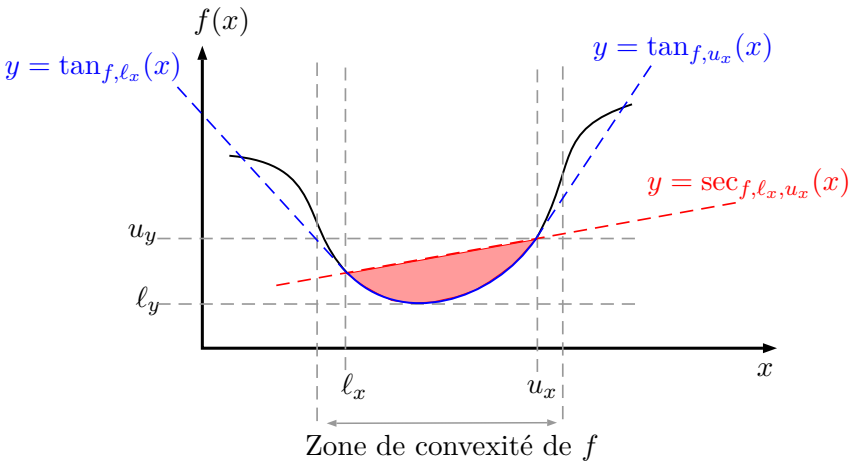

FIGURE 3.10 – *La fonction f est ici convexe dans le domaine de x, mais pas sur $\mathbb{R}$. En bleu, sous-estimateur convexe dans le domaine de x, prolongé par continuité sur $\mathbb{R}$. En rouge, sur-estimateur et enveloppe convexes.*

Le branchement différentiable peut introduit une discontinuité dans la hessienne du lagrangien, mais ceci n'a pas d'impact sur la résolution [139]. De telles discontinuités sont de plus introduites par la façon dont les inégalités sont prises en compte dans le lagrangien augmenté. Effectuer un branchement $\mathcal{C}^2$ en utilisant des paraboles pour prolonger le sous-estimateur permettrait de résoudre le problème si celui-ci devenait limitant.

Contrôle de la taille des contraintes

La structure des non-zéros du jacobien des relaxations non linéaires utilisées est importante. En effet, toutes les contraintes issues des relaxations des couplages non linéaires sont très creuses, puisque comportant deux ou trois non-zéros uniquement. En revanche, les contraintes linéaires et quadratiques peuvent être globales, toucher autant de variables que possible. Dans ce cas, le jacobien contient des lignes denses.

Un problème spécifique de l'algorithme du lagrangien augmenté, que nous n'avons jamais vu formulée dans la littérature, est que la hessienne du lagrangien augmenté fait intervenir, pour une contrainte $g(x) \leq 0$ active, des termes de la forme $\nabla g \nabla g^T$. Si le gradient de g contient k non-zéros, alors $\nabla g \nabla g^T$ est, à une permutation près, une matrice pleine de taille k. Ainsi, la hessienne du lagrangien augmenté contient des blocs denses dont la taille suit celle des contraintes. En particulier, si une contrainte est globale, c'est-à-dire qu'elle touche toutes les variables, alors la hessienne peut être une matrice pleine.

Deux solutions sont possibles à ce problème. La première est de ne pas former la hessienne explicitement, mais de résoudre les sous-problèmes avec des algorithmes ne nécessitant que des produits hessienne-vecteurs, telle que la méthode de Newton tronquée utilisant le gradient conjugué. En effet, si $\nabla g \nabla g^T$ est dense, le coût d'un produit avec un vecteur d reste linéaire en le nombre de non-zéros. La seconde est de découper les contraintes globales, *via* l'ajout de nouvelles variables et égalités. C'est l'action de *presolve* « découpe de contraintes » évoquée dans la partie 2.1.2. Ceci permet de conserver l'utilisation d'une méthode directe utilisant la hessienne pour résoudre les sous-problèmes.

3.3.2 Mise à l'échelle des contraintes

Motivation et état de l'art

Une fois la géométrie du paysage d'optimisation des sous-problèmes du lagrangien augmenté maîtrisée, on s'intéresse à leur conditionnement, défini comme celui de la hessienne du lagrangien augmenté. Le conditionnement d'une matrice est infini si celle-ci est singulière, et égal au rapport de la plus grande sur la plus petite valeur absolue d'une valeur propre dans le cas contraire. Le conditionnement détermine le coût de résolution d'un système linéaire par une méthode itérative, et la précision de résolution par une méthode directe. Il a un impact direct sur la performance de la méthode de Newton et sur son bassin d'attraction. Améliorer le conditionnement d'un problème revient à en accélérer la résolution, ainsi qu'à réduire les erreurs numériques présentes lors de cette résolution.

Une approche générique permettant d'améliorer la situation consiste à effectuer une mise à l'échelle, des variables, de l'objectif ou des contraintes. Pour un problème défini sur l'ensemble admissible $\{g(x) = 0,\ h(x) \leq 0\}$, la mise à l'échelle des contraintes revient à remplacer celui-ci par $\{w_g g(x) = 0,\ w_h h(x) \leq 0\}$, où w_g et w_h sont des vecteurs de scalaires strictement positifs. Aux tolérances près, les solutions admissibles des deux formulations sont les mêmes. L'objectif de cette partie est de déterminer w_g et w_h tels que la formulation mise à l'échelle soit mieux conditionnée. Pour ne pas modifier les conditions d'arrêt de faisabilité et de complémentarité, celles-ci sont évaluées dans la formulation initiale. Le choix des coefficients de mise à l'échelle en programmation non linéaire est un problème considéré comme difficile et encore ouvert, les méthodes utilisées étant largement heuristiques.

Pour évaluer le conditionnement des contraintes, on peut analyser les coefficients du Jacobien. Une méthode s'est cependant imposée au sein des solveurs industriels, qui consiste à normaliser les lignes de ce dernier en norme infinie. Un choix typique de coefficients suivant cette approche est par exemple $w_{g_i} = 1/\max(1, \|\nabla g_i(x_0)\|_\infty)$. Celui-ci est utilisé par exemple dans IPOPT [162], LANCELOT [63] ou encore ALGENCAN [26]. On retrouve le même choix dans WORHP [75], avec cette fois la norme 2. Cette formule semble presque universelle : les articles présentant les solveurs non linéaires les plus performants l'utilisent explicitement, ou alors ne parlent pas de mise à l'échelle des contraintes.

Les expériences numériques menées pendant cette thèse vont dans le sens de l'approche présentée ci-dessus, puisque l'analyse des gradients et hessiennes du lagrangien augmenté sur des problèmes mal conditionnés fait effectivement émerger une renormalisation des gradients des contraintes comme mise à l'échelle naturelle. L'utilisation de la norme initiale des gradients est une approximation de cette idée, adaptée au début de la recherche mais qui peut devenir arbitrairement mauvaise plus tard. Dans notre contexte, la connaissance explicite des contraintes du modèle factorisé nous permet de choisir des coefficients de mise à l'échelle plus précis. De plus, notre expérience numérique nous indique que la définition donnée plus haut du conditionnement est trop restrictive. En effet, la hessienne du lagrangien augmenté est souvent singulière, mais ceci ne pose pas systématiquement de problèmes numériques. En pratique, nous avons observé qu'un des meilleurs prédicteurs des problèmes numériques est la valeur propre maximale de la hessienne.

Choix de la norme : analyse des contraintes linéaires

Le cas d'une contrainte linéaire $a^T x + b = 0$ est le plus facile, puisque son gradient, a, est constant. On se demande quelle norme utiliser pour la normalisation de a. Les deux choix naturels que l'on retrouve dans la littérature sont la norme infinie et la norme 2, avec une prédominance de la norme infinie. On suppose que le multiplicateur et le facteur de pénalisation de la contrainte sont égaux à 1. Dans ce cas, les contributions de la contrainte dans le lagrangien augmenté, son gradient et sa hessienne sont

$$a^T x + b + \frac{1}{2}(a^T x + b)^2,\ a + (a^T x + b)a, \quad \text{et} \quad aa^T.$$

On rappelle que la norme de Frobenius d'une matrice $A = (a_{ij})$ est $\sqrt{\sum_{ij} a_{ij}^2}$. Les normes 2 et ∞ sont équivalentes en dimension finie, mais leur différence dépend du nombre de variables n. Ainsi, pour $a = (1, \cdots, 1)^T$, la norme infinie est égale à 1 et la norme 2 à $\sqrt{n}$. La plus grande valeur propre de aa^T est n, obtenue par exemple pour $x = (1/\sqrt{n}, \cdots, 1/\sqrt{n})$. Si la norme ∞ est utilisée pour normaliser a, la norme et la plus grande valeur propre de la hessienne sont toutes deux égales à n. Si c'est la norme 2 qui est utilisée, la norme de la hessienne est $\sqrt{n}$, et sa plus grande valeur propre est 1.

La norme 2 permet d'équilibrer les contributions des contraintes dans le gradient et la hessienne du lagrangien augmenté indépendamment de la dimension du problème. Un autre argument en faveur de la norme 2 est que celle-ci mesure la distance à l'ensemble admissible : la distance de x à $a^T x + b \leq 0$ est $(a^T x + b)^+ / \|a\|_2$. Pénaliser une contrainte selon la distance à l'ensemble admissible semble être une bonne approche. Ceci permet d'obtenir des sous-problèmes qui dépendent uniquement de la géométrie de l'ensemble admissible, et non pas des expressions algébriques qui le définissent. Pour ces deux raisons, nous utilisons la norme 2.

Extension aux contraintes non linéaires

Les gradients des contraintes non linéaires étant non constants, leurs normes peuvent varier au cours de la résolution. En utilisant la connaissance de la forme analytique de ces contraintes, on peut déterminer leurs gradients par intervalles. Ceci nous permet alors de calculer des coefficients de mise à l'échelle plus précis qu'avec le point initial uniquement.

Le gradient d'une contrainte quadratique $x^T Q x + c^T x + a \leq 0$ est $2Qx + c$. Par arithmétique d'intervalle, on obtient

$$2Qx + c \in [L, U] \quad \text{si} \quad l \leq x \leq u.$$

On définit alors $s_i = |\mathcal{P}_{[L_i, U_i]}(0)|$, la plus petite valeur absolue que la dérivée partielle i peut prendre. On utilise finalement une mise à l'échelle basée sur la plus petite valeur que peut prendre la norme du gradient :

$$w = \frac{1}{\max(1, \|s\|_2)}.$$

Le gradient d'une contrainte non linéaire $f(x) - y \leq 0$ est $(f'(x), -1)^T$. De la même manière, on obtient $f'(x) \in [L, U]$, et on utilise une mise à l'échelle de

$$w = \frac{1}{\sqrt{1 + \mathcal{P}_{[L,U]}(0)^2}}.$$

La mise à l'échelle des couplages non linéaires est particulièrement importante dans le cadre d'une recherche arborescente. En effet, il est fréquent que sur des problèmes où les domaines des variables sont grands, une décision de branchement donne lieu à des situations où L, dans la formule précédente, soit grand. Par exemple, pour $y = x^4$ avec $x \in [10^4, 10^5]$, la formulation naturelle $x^4 - y \leq 0$ est mal conditionnée. De façon générale, l'utilité de la mise à l'échelle est la plus nette lorsque $\min_{\ell \leq x \leq u} \|x\|_2 \ll 1$.

Impact de la mise à l'échelle sur la résolution

La mise à l'échelle des contraintes est l'un des composants du solveur non linéaire dont l'impact est le plus important. Pour l'illustrer, on considère dans ce paragraphe les 1633 instances de plus petites tailles du *benchmark*. Le module dual est exécuté seul, sans heuristiques primales, avec le modèle factorisé quadratique et le solveur non linéaire. La recherche est interrompue dès que 1000 nœuds ont été visités, si le temps limite de 30 secondes est atteint. On définit alors un score sur chaque instance, mesurant l'impact de la mise à l'échelle. Si l'instance était résolue sans mise à l'échelle, mais ne l'est plus avec, le score est de 0. Si l'instance n'était pas résolue sans mise à l'échelle, mais l'est avec, le score est de ∞. Pour les instances résolues avec et sans mise à l'échelle, le score est défini comme le rapport des coûts d'exécution sans et avec mise à l'échelle. Le coût d'exécution n'est pas mesuré en temps, puisqu'instable et non déterministe, mais en nombre de systèmes linéaires résolus pour la résolution de tous les sous-problèmes. Pour les instances non résolues, ni avec, ni sans mise à l'échelle, le score est défini comme le rapport des nombres de nœuds visités avec et sans mise à l'échelle. Dans les deux cas, plus le score est élevé, plus la mise à l'échelle a eu un impact positif. Un score de 1 signifie que la mise à l'échelle n'a pas eu d'impact. Le tableau 3.4 montre alors que la mise à l'échelle est largement bénéfique.

Score de la mise à l'échelle	Proportion des instances
< 0.1	0.8%
< 0.5	4.8%
< 0.9	20.8%
$[0.9, 1.1]$	15.1%
> 1.1	35%
> 2	17.5%
> 10	6%

Tableau 3.4 – *Répartition des instances selon le score de la mise à l'échelle,* benchmark *de* 1633 *instances.*

Le tableau 3.5 donne quelques statistiques générales sur l'impact de la mise à l'échelle, pour les instances où le statut final (optimal ou non) n'est pas impacté par l'activation de la mise à l'échelle. Il montre que la robustesse est améliorée, puisque le nombre d'échecs du solveur est divisée par plus de deux au *root node*, et réduite de plus de 40% en général. Le coût de résolution d'un nœud (en nombres d'évaluations de l'objectif et de systèmes linéaires résolus) est réduit, même si le nombre d'itérations du lagrangien augmenté est légèrement augmenté. Ceci signifie que la mise à l'échelle accélère la résolution des sous-problèmes, mais ralentit la convergence du lagrangien augmenté vers la faisabilité. Puisque le coût de résolution est entièrement compris dans la résolution des sous-problèmes, cette augmentation est sans conséquences.

Mesure	Impact de la mise à l'échelle
Échecs au *root node*	−59%
Échecs pendant le *branch-and-bound*	−42%
Nombre moyen d'évaluations	−8%
Nombre moyen de systèmes linéaires résolus	−7%
Itérations du lag. aug. moyennes par nœud	+10%

Tableau 3.5 – *Impact de la mise à l'échelle sur la robustesse et le coût de résolution pour* 943 *instances où le statut final n'est pas impacté par cette dernière.*

On s'intéresse maintenant aux instances non résolues, avec ou sans mise à l'échelle. La figure 3.11 montre que celle-ci permet de visiter bien plus de nœuds. Ceci signifie que le solveur non linéaire reste moins bloqué, ou est moins ralenti, par les problèmes difficiles à résoudre car mal conditionnés. Enfin, la réduction du taux d'échecs a pour conséquence que les instances résolues le sont parfois plus vite, comme le montre la figure 3.12 sur les instances résolues à l'optimum par les deux stratégies.

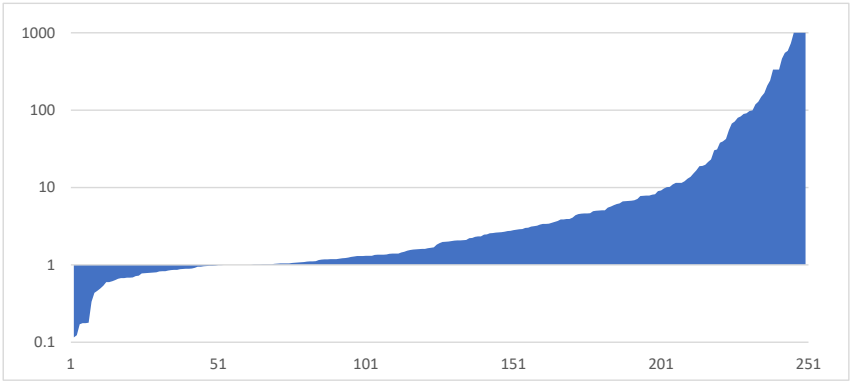

FIGURE 3.11 – *Pour les instances non résolues, ni avec, ni sans mise à l'échelle, rapports des nombres de nœuds visités avec et sans cette dernière.*

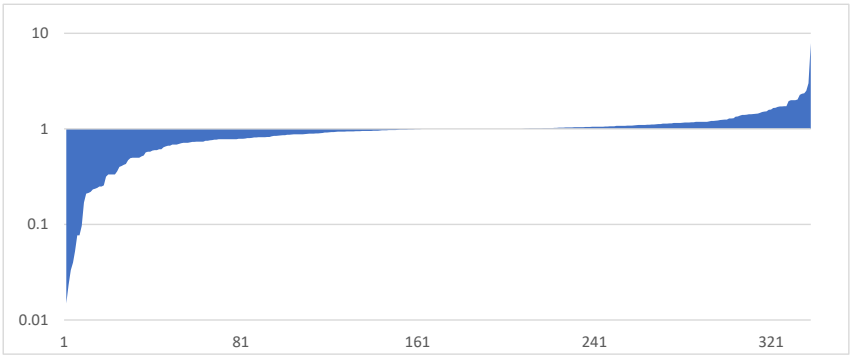

FIGURE 3.12 – *Pour les instances résolues à l'optimum avec et sans mise à l'échelle, rapports des nombres de nœuds nécessaires à la résolution.*

3.3.3 Reformulation des contraintes

Les formules de mise à l'échelle que nous utilisons sont conservatrices. Elles considèrent en effet le meilleur conditionnement possible pour chaque contrainte. En pratique, si la norme du gradient d'une contrainte est loin de sa valeur minimale, la mise à l'échelle n'est pas suffisante pour réduire les problèmes de mauvais conditionnement. Inversement, une mise à l'échelle utilisant la plus grande norme possible est trop agressive, écrase la contrainte et ralentit la résolution (en particulier, la faisabilité est difficile à atteindre). Une solution possible est d'utiliser une mise à l'échelle dynamique dépendant du point courant, et sera l'objet d'expérimentations numériques futures.

Nous avons suivi une autre approche pour améliorer le conditionnement des sous-problèmes non contraints. Celle-ci s'intéresse cette fois à la formulation des contraintes. L'idée originale que nous proposons dans cette section est de reformuler les contraintes problématiques pour qu'elles ne nécessitent plus de mise à l'échelle. Cette approche fait d'une pierre deux coups, car elle permettra aussi d'éliminer les dernières contraintes dures du paysage.

Motivation

Pour illustrer les problèmes de conditionnements, on considère la relaxation convexe de $y = 1/x$. Pour $x \geq 0$, celle-ci s'écrit $1/x - y \leq 0$. Cette relaxation est convexe sur $\mathbb{R}_+^*$ uniquement, et est donc prolongée en la borne inférieure de x. Les variables étant toutes bornées et $x \mapsto 1/x$ étant bijective sur $\mathbb{R}^+$, x et y ont toutes deux un domaine de définition de la forme $[1/M, M]$. La contrainte présente dans la relaxation convexe du problème est donc :

$$y \geq \underline{f}(x) = \begin{cases} 1/x, \ x \geq 1/M, \\ \tan_{1/x,1/M}(x), \ x < 1/M. \end{cases}$$

Le coefficient directeur de la branche linéaire est égal à $-1/M^2$. Ainsi, pour des bornes du type $y \leq 10^5$, on obtient $x \geq 10^{-5}$ et un coefficient directeur de 10^{10} : pour x proche de 0, la relaxation est mal conditionnée. Si le prolongement a permis de supprimer le pôle et la contrainte dure, il n'a pas résolu le problème de conditionnement. La mise à l'échelle n'en est pas capable non plus, car la norme du gradient tend vers 1 quand x augmente.

L'autre problème que nous cherchons à résoudre est celui des *overflow*. En arithmétique flottante 64 bits, un *overflow* arrive lorsqu'un nombre dépasse $1,8 \times 10^{308}$. En effet, tout nombre dépassant cette limite est assimilé à $+\infty$. Un opérateur sujet à ce problème impose une contrainte dure numérique, puisque la situation est similaire à celle, décrite plus haut, d'un opérateur non-défini. L'exemple le plus criant est celui de la fonction exponentielle $y = \mathrm{e}^x$, dont la relaxation convexe est $\mathrm{e}^x - y \leq 0$. Si la fonction exponentielle est bien définie sur $\mathbb{R}$, pour $x \geq \log(1,8 \times 10^{308}) \simeq 710$, $\mathrm{e}^x = +\infty$. Ainsi, $x \leq 710$ est une contrainte dure numérique.

Représentation algébrique d'un convexe

On se donne une contrainte non linéaire $g(x) \leq 0$ active, dont le multiplicateur est nul et le facteur de pénalisation égal à 1. La contribution de g dans la pénalisation se simplifie alors en :

$$p_g(x) = \frac{\rho}{2}\|(g(x) + \mu/\rho)^+\|^2 = \frac{1}{2}g(x)^2.$$

Les dérivées de cette contribution sont

$$\nabla p_g(x) = g(x)\nabla g(x) \quad \text{et} \quad \nabla^2 p_g(x) = \nabla g(x)\nabla g(x)^T + g(x)\nabla^2 g(x). \tag{3.9}$$

Puisque la hessienne contient un terme dépendant du gradient, on se limite à l'analyse du conditionnement de celle-ci.

L'idée centrale pour résoudre ces problèmes a déjà été évoquée lors de la mise à l'échelle des contraintes linéaires. Une contrainte est une représentation d'une certaine propriété géométrique que doit vérifier une solution du problème. Elle signifie que x doit appartenir à un certain convexe $\mathcal{C}$. Cette contrainte géométrique doit cependant être fournie sous forme algébrique, c'est-à-dire *via* une représentation par équation :

$$x \in \mathcal{C} \Leftrightarrow g(x) \leq 0.$$

Or, la représentation algébrique n'est pas unique. De plus, les expressions (3.9), et donc le conditionnement des sous-problèmes du lagrangien augmenté, dépendent de g. On se pose donc la question du choix de cette représentation.

L'idée la plus naturelle consiste à utiliser une pénalisation basée sur des notions géométriques. La magnitude de la violation serait ainsi égale à la distance à $\mathcal{C}$, et la direction (le gradient de g) serait vers le projeté orthogonal de x sur $\mathcal{C}$. Ces grandeurs sont cependant rarement calculables de façon analytique, même pour des convexes $\mathcal{C}$ simples.

Une autre possibilité est d'utiliser des représentations algébriques non convexes. Par exemple, $1/x - y \leq 0$ est équivalent à $xy \geq 1$, même si la seconde formulation n'est pas convexe. Un résultat de [107] assure alors que si la géométrie d'un problème (l'objectif et l'ensemble admissible) est convexe, toute représentation algébrique de ce dernier, convexe ou non, vérifie la propriété que les conditions de KKT sont suffisantes (sous des hypothèses de non-dégénérescence et de qualification de cette représentation). Il est donc possible d'attaquer un problème convexe par une représentation non convexe et un solveur local. Si ce résultat est intéressant, il nous empêche cependant d'utiliser la dualité mise en place en début de chapitre.

Pénalisation directionnelle

Le calcul du projeté orthogonal d'un point sur un convexe est difficile dans le cas général. Pour certains convexes, il existe cependant une approximation de la direction de projection. Plus précisément, il existe une direction constante, dont l'angle avec la direction de projection reste dans $[-\frac{\pi}{4}, \frac{\pi}{4}]$. Pénaliser la contrainte suivant cette direction revient alors à effectuer une rotation du repère canonique, et améliore significativement le conditionnement du problème. Le principe est illustré dans la figure 3.13, pour les contraintes $1/x - y \leq 0$ et $\mathrm{e}^x - y \leq 0$.

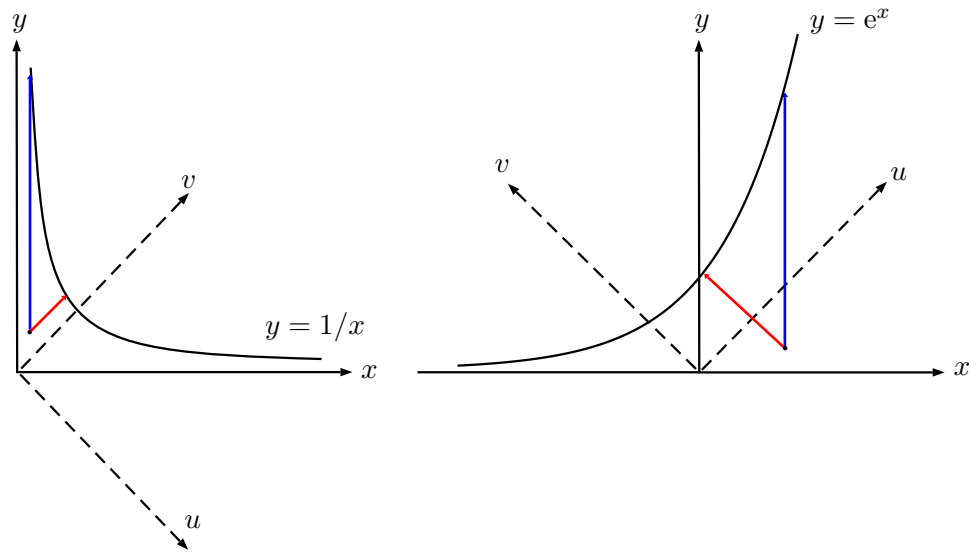

FIGURE 3.13 – *Pénalisation directionnelle pour* $1/x - y \leq 0$ *et* $e^x - y \leq 0$. *En bleu, pénalisation naturelle de la contrainte. En rouge, pénalisation dans la direction approximant la direction de projection :* $(1,1)$ *pour* $1/x - y \leq 0$ *et* $(-1,1)$ *pour* $e^x - y \leq 0$. *Ces directions correspondent à l'expression de la contrainte dans un certain repère* (u,v).

Relations $y = 1/x$

Pour x et y positifs, on considère la contrainte non linéaire

$$g_1(x,y) = \frac{1}{x} - y \leq 0,$$

et l'expression (3.9) qui nous intéresse est alors

$$\nabla^2 p_{g_1}(x,y) = \begin{pmatrix} (3-2xy)/x^4 & 1/x^2 \\ 1/x^2 & 1 \end{pmatrix}.$$

Pour $x = 10^{-3}$ et $y = 1/x - 1 = 999$ (et donc une violation de la contrainte de 1), les valeurs propres sont environ $\{0.002, 10^{12}\}$.

On décide alors de pénaliser suivant la direction $(1,1)$, ce qui revient à écrire la contrainte dans le repère canonique auquel on a appliqué une rotation de $-\pi/4$. Dans ce repère, la contrainte s'écrit $\phi(u,v) = \sqrt{u^2+2} - v \leq 0$, avec les formules de changement de repère

$$x = \frac{u+v}{\sqrt{2}}, \quad y = \frac{v-u}{\sqrt{2}}.$$

En substituant les formules de changement de repère dans l'expression de la contrainte, puis en multipliant cette dernière par $\sqrt{2}$, on obtient finalement la reformulation suivante, que nous utilisons dans le lagrangien augmenté :

$$g_2(x,y) = \sqrt{4 + (x-y)^2} - (x+y) \leq 0.$$

Une rotation du repère revient à multiplier certaines grandeurs par des matrices orthonormales, et ne change pas le conditionnement de la hessienne. Pour simplifier

les calculs, on étudie la hessienne de cette seconde formulation dans le repère (u, v), qui s'écrit

$$\nabla^2 p_\phi(u,v) = \begin{pmatrix} 1 - 2v/(u^2+2)^{3/2} & -u/\sqrt{u^2+2} \\ -u/\sqrt{u^2+2} & 1 \end{pmatrix}.$$

En reprenant le cas précédent avec $x = 10^{-3}$ et $y = 999$, on obtient $u \simeq -706$ et $v \simeq 706$. Dans ce cas, les valeurs propres de $\nabla^2 p_\phi$ sont environ $\{4.10^{-12}, 2\}$. Le conditionnement de la hessienne de la pénalisation et, surtout, sa plus grande valeur propre, ont été largement améliorés.

Relations $y = \mathrm{e}^x$

On considère maintenant une contrainte non linéaire

$$g_1(x,y) = \mathrm{e}^x - y \leq 0.$$

et l'expression (3.9) qui nous intéresse est alors

$$\nabla^2 p_{g_1}(x,y) = \begin{pmatrix} \mathrm{e}^x(2\mathrm{e}^x - y) & -\mathrm{e}^x \\ -\mathrm{e}^x & 1 \end{pmatrix}.$$

Pour $x = 10$ et $y = 0$, les valeurs propres sont environ $\{0.5, 9.7 \times 10^8\}$.

On décide alors de pénaliser suivant la direction $(-1, 1)$, ce qui revient à écrire la contrainte dans le repère canonique auquel on a appliqué une rotation de $-\pi/4$. Dans ce cas, le changement de variables s'écrit

$$x = \frac{u-v}{\sqrt{2}}, \quad y = \frac{u+v}{\sqrt{2}}.$$

Pour obtenir l'expression de la contrainte dans le repère (u, v), on part de

$$\mathrm{e}^{\frac{u-v}{\sqrt{2}}} = \frac{u+v}{\sqrt{2}},$$

que l'on multiplie par $\mathrm{e}^{\frac{u+v}{\sqrt{2}}}$ pour obtenir

$$\frac{u+v}{\sqrt{2}} \mathrm{e}^{\frac{u+v}{\sqrt{2}}} = \mathrm{e}^{\sqrt{2}u}.$$

La solution de cette équation est donnée par

$$\frac{u+v}{\sqrt{2}} = W(\mathrm{e}^{\sqrt{2}u}),$$

où W est la fonction définie comme la branche principale de la fonction de Lambert $W : [-\frac{1}{\mathrm{e}}, +\infty[\to [-1, +\infty[$, définie comme la réciproque de $w(x) = x\mathrm{e}^x$ sur $[-1, +\infty[$. La contrainte peut donc se reformuler en

$$g_2(x,y) = \sqrt{2}\left(W(\mathrm{e}^{x+y}) - y\right) \leq 0.$$

En divisant par $\sqrt{2}$ et en utilisant le fait que $\omega : x \mapsto W(\mathrm{e}^x)$ est la réciproque de $x \mapsto x + \ln x$, la contrainte se simplifie en

$$g_2(x,y) = \omega(x+y) - y \leq 0.$$

En utilisant $\omega' = \omega/(1+\omega)$, les dérivées de g_2 sont

$$\nabla g_2(x,y) = \frac{1}{1+\omega(x+y)} \begin{pmatrix} \omega(x+y) \\ -1 \end{pmatrix}, \quad \nabla^2 g_2(x,y) = \frac{\omega(x+y)}{(1+\omega(x+y))^3} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

En reprenant l'exemple précédent où $x = 10$ et $y = 0$, on a $\omega(x+y) \simeq 7.9$. La hessienne de la pénalisation est cette fois

$$H_2 = \begin{pmatrix} 0.9 & -0.01 \\ -0.01 & 0.1 \end{pmatrix},$$

et ses valeurs propres sont $\{0.1, 0.9\}$. Le conditionnement de la pénalisation en $x = 10$ et $y = 0$ passe donc (en ordre de grandeur) de 10^9 à 10^1.

Dans notre *benchmark*, un peu moins de 10% des instances contiennent des logarithmes ou exponentielles. Dans 72% des cas, le solveur visite des points où les contraintes dures $x \leq 710$ sont violées, et au moins un *overflow* a lieu avant le temps limite. Une fois la reformulation mise en place, nous n'avons plus constaté aucun *overflow* sur ces instances.

Remarque : la fonction ω est calculée par une méthode de Newton et une approximation initiale. Après des tests numériques intensifs, la convergence la plus lente observée est de cinq itérations (et donc cinq calculs de logarithme), soit une augmentation raisonnable par rapport à la formulation naturelle.

3.4 Résolution des sous-problèmes

La méthode du lagrangien augmenté présentée précédemment est particulièrement robuste. En effet, dès lors que l'on est capable de résoudre tous les sous-problèmes, la convergence vers une solution admissible (ou un certificat d'inconsistance) a lieu. La robustesse finale du solveur non linéaire est ainsi déterminée par celle de résolution des sous-problèmes. Dans cette section, on se concentre sur un problème de minimisation sans contraintes, de la forme

$$\min_{x \in \mathbb{R}^n} f(x), \tag{3.10}$$

où la fonction f est supposée $\mathcal{C}^1$ et $\mathcal{C}^\infty$ presque partout. Puisque nous avons travaillé sur la formulation des sous-problèmes, f peut aussi être supposée définie sur $\mathbb{R}^n$.

Le problème (3.10) est l'un des cas les plus élémentaires de l'optimisation non linéaire. Il s'agit d'un paysage d'optimisation à explorer, afin d'en trouver le point le plus bas. Il existe deux approches itératives de résolution : les méthodes de descente avec recherche linéaire, et les méthode par régions de confiance [134, 65]. Les méthodes par régions de confiance convergent sous des hypothèses plus faibles que celles de recherche linéaire, et sont plus pertinentes pour traiter des non-convexités. Dans notre cas, la convexité et la régularité de f nous simplifient la tâche : les deux méthodes convergent, en théorie.

Le problème (3.10) a été beaucoup étudié dans la littérature. Les méthodes de résolution sont connues et font aujourd'hui partie du folklore. Deux difficultés

pratiques se posent pour une résolution efficace. D'abord, le conditionnement du problème détermine la vitesse de résolution par les méthodes itératives citées ci-dessus. Ensuite, les erreurs numériques peuvent être assimilées à un bruit aléatoire qui s'ajoute à f. Ceci ne pose pas de problèmes si l'amplitude du bruit est petit devant la valeur minimale de f, mais peut complètement empêcher la convergence dans le cas contraire. Une fois la formulation du problème travaillée, comme nous l'avons fait, pour améliorer le conditionnement, il n'existe pas d'autre solution à ces deux problèmes que d'augmenter la précision des flottants utilisés. Pour cette raison, l'optimisation non linéaire est considérée comme un « art », et non pas comme une « technologie » ([56], page 9). Dans notre cas cependant, f est convexe, et on peut espérer atteindre des niveaux de robustesse satisfaisants.

L'objectif de cette section est de décrire avec précision la géométrie en présence, ainsi que les outils utilisés pour l'attaquer. Ceux-ci ayant été mis en place et réglés par expérimentation numérique, nous ne justifions pas précisément tous les choix effectués, mais essayons de donner tous les ingrédients en présence. L'approche retenue est celle utilisant des recherches linéaires. Après expérimentations, celle-ci converge plus rapidement sur les instances où les problèmes numériques ne se posent pas, même si elle est moins robuste sur les autres. Les solutions les plus prometteuses à ces problèmes ne sont cependant pas l'utilisation d'une région de confiance, et seront évoquées lors des perspectives. La méthode de résolution de (3.10) peut être résumée par l'algorithme 4. Celui-ci est constitué de trois composants principaux : le calcul de la direction de recherche, la recherche linéaire ainsi que les critères d'arrêt utilisés.

Algorithme 4 Algorithme de descente avec recherche linéaire pour résoudre (3.10)

fonction UnconstrainedLineSearch(x_0, ε)
  $k \leftarrow 0$;
  tant que les critères d'arrêt ne sont pas remplis **faire**
    Calculer une direction de descente d ;
    Résoudre approximativement $t^* \in \underset{t \geq 0}{\operatorname{argmin}}\ f(x_k + td_k)$;
    $x_{k+1} \leftarrow x_k + t^* d_k$;
    $k \leftarrow k + 1$;
  fin tant que
fin fonction

3.4.1 Directions de descente

Une direction de descente est un vecteur d tel que $d^T \nabla f(x) < 0$. En inspectant un développement limité de f, d'après sa régularité, on est alors assuré qu'il existe $t > 0$ tel que $f(x + td) < f(x)$. Cette condition est instable, et on utilise à la place une condition sur l'angle formé entre d et $\nabla f(x)$:

$$\frac{d^T \nabla f(x)}{\|d\| \|\nabla f(x)\|} \leq -\varepsilon_{\text{dir}}. \tag{3.11}$$

Si une direction d ne satisfait pas cette condition, il faut la modifier, ou alors utiliser la direction de plus forte pente : $d = -\nabla f(x)$ (qui la satisfait toujours).

Directions utilisées

Cinq directions de recherche ont été implémentées, testées et réglées. On utilise les notations $g_k = \nabla f(x_k)$ et $y_k = g_{k+1} - g_k$. Premièrement, la direction de plus forte pente utilise l'opposé du gradient :

$$d_k = -g_k. \tag{SD}$$

Cette direction minimise l'approximation de la décroissance de f au premier ordre. Deuxièmement, le gradient conjugué utilise une combinaison de la direction de plus forte pente et de la direction utilisée à l'itération précédente :

$$d_{k+1} = -g_{k+1} + \beta_k d_k. \tag{CG}$$

Cette direction généralise au cas non linéaire la méthode du même nom permettant de résoudre les problèmes quadratiques strictement convexes en n itérations. Nous utilisons la correction de Daï-Yuan-Hestenes-Siefel [68] :

$$\beta_k^{DY} = \frac{\|g_k\|^2}{y_k^T d_{k-1}}, \text{ et } \beta_k^{HS} = \frac{y_k^T g_k}{y_k^T d_{k-1}},$$

$$\beta_k^{DY-HS} = \max\left(0, \min(\beta_k^{DY}, \beta_k^{HS})\right).$$

Troisièmement, la méthode de quasi-newton BFGS à mémoire limitée [116]. Celle-ci construit une approximation B de la hessienne expliquant la variation de l'information du premier ordre expliquant la variation de l'information du premier ordre

$$g_{k+1} - g_k = B(x_{k+1} - x_k),$$

des m dernières itérations, où $m = 5$ est la mémoire de l'algorithme (généralement comprise entre 3 et 30). La direction utilisée est alors

$$d_k = -B^{-1} g_k, \tag{LBFGS}$$

où B^{-1} n'est jamais formée, et où le produit avec celle-ci est obtenu avec un algorithme utilisant des formules de récurrence et ayant une complexité de $O(nm)$. Quatrièmement, la méthode de Newton consiste à minimiser l'approximation de la décroissance de f, comme la direction de plus forte pente, mais au second ordre. Celle-ci s'écrit

$$f(x+d) - f(x) = d^T \nabla f(x) + \frac{1}{2} d^T \nabla^2 f(x) d + o(\|d\|^2), \tag{3.12}$$

et la direction d est obtenue en résolvant les conditions de KKT ce de problème de minimisation convexe :

$$\nabla^2 f(x_k) d_k + g_k = 0. \tag{EN}$$

Enfin, la direction de Newton tronquée [71, 130] consiste à calculer une direction en minimisant directement (3.12) par une méthode itérative : le gradient conjugué.

Cette méthode est adaptée pour les problèmes de très grande taille, ou quand la hessienne contient des blocs denses liés aux contraintes pleines, car elle ne nécessite que des produits hessienne-vecteur. La direction de Newton tronquée déroule le gradient conjugué sur (3.12), et s'interrompt dès que

$$\|\nabla^2 f(x_k)d_k + g_k\| \leq \varepsilon_k^{\text{tn}}\|g_k\|, \tag{TN}$$

où dès que la géométrie attendue lors de l'exécution du gradient conjugué n'est plus vérifiée [170, 171]. La séquence $\varepsilon_k^{\text{tn}}$, dite de forçage, contrôle la précision du calcul de la direction, ainsi que la vitesse de convergence asymptotique théorique. Le réglage de cette séquence est délicat, en particulier pour un paysage issu d'un lagrangien augmenté, où l'activation des contraintes peut changer d'une itération à l'autre, rendant la suite des $\|g_k\|$ non monotone, voire erratique.

Pour les problèmes de moins de 1000 variables, on utilise par défaut la méthode de Newton. Le système (EN) est résolu en effectuant une factorisation de Cholesky avec LAPACK [25], l'état de l'art sur les opérations d'algèbre linéaires denses. Au delà de 1000 variables, la même approche est utilisée, cette fois avec MUMPS [24] (en version séquentielle), l'un des outils d'algèbre linéaire creuse les plus performants. Si la hessienne contient plus de 10^6 non-zéros, on utilise la méthode de Newton tronquée.

Correction de la hessienne

La méthode de Newton et ses variantes nécessitent la stricte convexité de f, qui n'a pas toujours lieu dans notre cas. Si le système (EN) est indéfini, la factorisation de la hessienne peut échouer. De plus, pour des problèmes de convexité, d'erreurs numériques, ou de mauvais conditionnement, la condition de descente peut ne pas être satisfaite par la direction obtenue. Dans ce cas, deux stratégies existent pour calculer une direction de Newton approchée : utiliser une factorisation spécialisée pour traiter la singularité de la hessienne, ou convexifier le modèle quadratique de f. La première approche nécessite d'utiliser des librairies propriétaires, telles que la *Harwell Subroutine Library* [94], puisque les outils d'algèbre linéaire ne disposent pas de ces fonctionnalités. La seconde approche, celle retenue, consiste à régulariser $\nabla^2 f(x)$, et résout une suite de systèmes linéaires

$$\left(\nabla^2 f(x) + (2^k - 1)\varepsilon_{\text{diag}} I\right) d = -\nabla f(x),$$

avec $0 < \varepsilon_{\text{diag}} \ll 1$. Pour $k = 0$ et $D = 0$, il s'agit de la méthode de Newton, et pour $k \to +\infty$, d devient proportionnel à $\nabla f(x)$. En théorie, le système devient défini positif dès $k = 1$. Une correction de $\varepsilon_{\text{diag}}$ peut cependant ne pas être suffisante pour contrer les erreurs d'arrondi liées au mauvais conditionnement du système. On résout donc ces systèmes jusqu'à ce qu'une solution satisfaisant la condition (3.11) soit obtenue.

Une autre amélioration consiste à détecter les problèmes de courbure nulle avant de résoudre le premier système. Pour cela, on effectue une première correction de la

hessienne en y ajoutant une matrice diagonale D définie par

$$D_i = \begin{cases} 1, & \text{si } \nabla^2_{ii} f(x) = 0 \text{ et } \nabla_i f(x) = 0, \\ \varepsilon_{\text{indef}}, & \text{si } \nabla^2_{ii} f(x) = 0 \text{ et } \nabla_i f(x) \neq 0, \\ 0, & \text{si } \nabla^2_{ii} f(x) > 0. \end{cases}$$

Ces corrections rejoignent l'approche des méthodes par régions de confiance. En effet, minimiser le modèle quadratique de f (3.12) sur une contrainte de norme $\|d\|_2 \leq \Delta$ revient à résoudre $\left(\nabla^2 f(x) + \delta I\right) d = -\nabla f(x)$ pour un certain $\delta > 0$.

Comparaison des directions

On peut comparer les performances globales du solveur lorsque différentes directions sont utilisées. Le tableau 3.6 résume les résultats obtenus sur les 2211 plus petites instances du *benchmark*. On y observe une claire différence de robustesse entre les méthodes d'ordre 1 ((SD), (CG) et (LBFGS)) et les méthodes d'ordre 2 ((EN) et (TN)). Les premières ont une vitesse de convergence asymptotique linéaire, alors que les secondes au moins superlinéaire (pour (TN)), voire quadratique (pour (EN)), mais ce n'est pas ce qui explique la robustesse. En effet, le paysage associé au lagrangien augmenté est une agrégation de l'objectif et de toutes les contraintes. Dans ce cas, l'information du second ordre est importante pour descendre efficacement. C'est aussi pour cette raison que la méthode de Newton exacte fonctionne mieux que sa version tronquée.

Direction	Instances résolues	Nœuds visités	*Root node* non résolu	Au moins un échec	Temps moyen de résolution
(EN)	64.4%	693	3.6%	16.2%	1
(TN)	60.7%	190	9.0%	23.3%	1.17
(LBFGS)	57.4%	103	16.4%	47%	1.35
(CG)	53.4%	200	23.2%	53.4%	1.67
(SD)	52.8%	42	25.4%	54.6%	2.32

Tableau 3.6 – *Comparaison des différentes directions de recherche. De gauche à droite, proportion des instances résolues à l'optimum, nombre moyen de nœuds visités sur les instances qui n'ont été résolues par aucune stratégie, échec du solveur ou temps limite atteint au* root node*, instances où le solveur a échoué au moins une fois, temps moyen de résolution sur les instances résolues par toutes les stratégies, relatif à la stratégie par défaut utilisant la direction de Newton exacte.*

Une analyse complémentaire sur la robustesse permet d'illustrer la situation avec plus de précision. La figure 3.14 trace le taux d'échec de résolution des relaxations convexes des cinq stratégies utilisant les cinq directions, ainsi que celle utilisant des relaxations linéaires, résolues avec le simplexe dual. On y observe une différence qualitative entre les méthodes d'ordre 1 et celles d'ordre 2, ainsi qu'un taux d'échec significativement plus faible pour l'algorithme du simplexe dual. Malgré tous les outils et idées présentés dans ce troisième chapitre, la marge de progrès, en termes

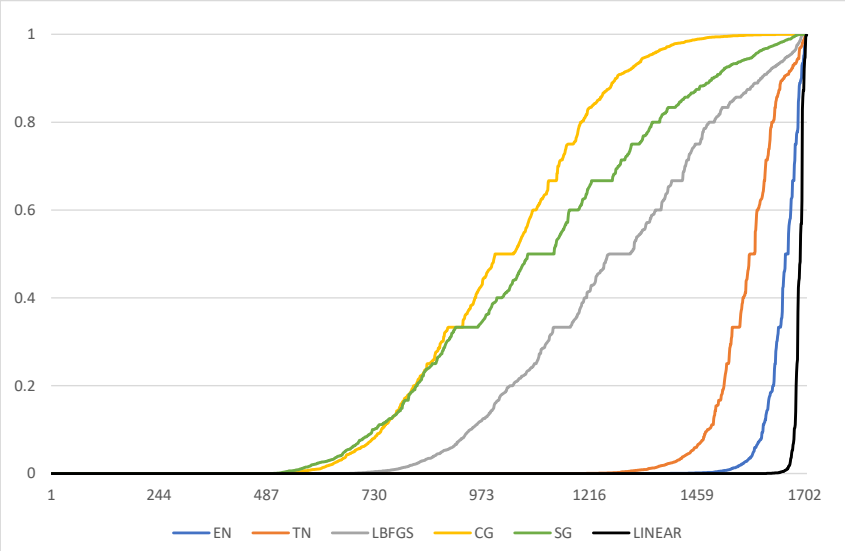

FIGURE 3.14 – *Taux d'échecs (en pourcentage des relaxations où le solveur échoue) des solveurs linéaire et non linéaire avec différentes directions, triés par taux croissants pour chaque stratégie. Seules les instances où au moins une relaxation a été résolue sont considérées.*

de robustesse, est donc encore importante. Heureusement, les causes des échecs du solveur non linéaire, ainsi que des solutions à ceux-ci, sont connus, et seront présentés lors des perspectives.

3.4.2 Recherche linéaire

On s'intéresse dans cette partie à l'algorithme de recherche linéaire utilisé pour minimiser $\phi(t) = f(x+td)$ sur la demi-droite $t \geq 0$. On suppose qu'une direction de descente d vérifiant $d^T \nabla f(x) \leq -\varepsilon_{\text{dir}} \|d\| \|\nabla f(x)\|$ a été calculée. On suppose aussi que les directions n'ayant pas de premier pas privilégié ont été normalisées, de sorte que $t = 1$ est toujours le premier candidat.

Géométrie de la fonction ϕ

Une partie importante de la littérature sur les recherches linéaires se concentre sur le cas non convexe. Ces algorithmes se basent sur la géométrie de la fonction à minimiser, et leur adaptation au cas convexe permet de les simplifier et d'en améliorer les performances. On commence donc par une rapide analyse de la géométrie de ϕ. D'après nos hypothèses de régularité et de convexité sur f, on a :

— ϕ est définie sur $\mathbb{R}^+$, est $\mathcal{C}^1$ et $\mathcal{C}^\infty$ presque partout,
— ϕ est convexe, et donc $\phi'(t) = d^T \nabla f(x+td)$ est croissante
— $\phi(0) = f(x)$, et $\phi'(0) = d^T \nabla f(x) < 0$.

D'après le théorème de la limite monotone, ϕ' a une limite en $+\infty$. Puisque les bornes des variables sont finies et qu'une pénalisation quadratique de leur violation

est présente dans f, $\lim_{t\to+\infty}\phi(t) = +\infty$ et $\lim_{t\to+\infty}\phi'(t) = +\infty$. Le théorème des valeurs intermédiaires affirme alors qu'il existe $t^* > 0$ tel que $\phi'(t^*) = 0$, ainsi que $t_0 > 0$ tel que $\phi(t_0) = f(x)$. Par les accroissements finis, puisque $\phi'(0) < 0$, on a $\phi(t^*) < f(x)$ et $t_0 > t^*$. La figure 3.15 résume ces propriétés géométriques.

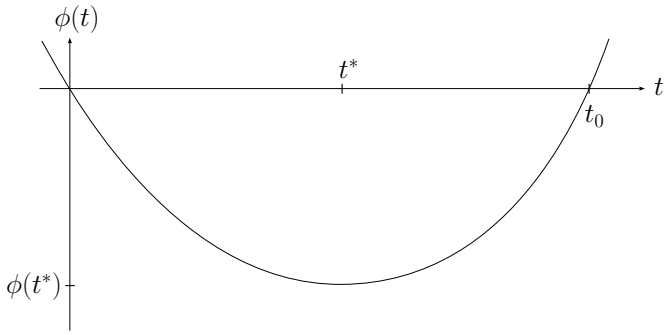

FIGURE 3.15 – *Géométrie de la fonction unidimensionnelle à minimiser. ϕ décroît strictement jusqu'à un minimum (qui n'est pas forcément unique, on peut avoir un plateau), puis croît strictement jusqu'à $+\infty$.*

L'algorithme de recherche linéaire utilisé

Notre algorithme de recherche linéaire spécialise les méthodes classiques (par exemple [134]) à la géométrie de ϕ, en particulier à sa convexité et à son unimodularité. La contribution est principalement technique, mais nous décrivons cette procédure en détail, car elle constitue la brique de base de tout le solveur non-linéaire. C'est un algorithme qui permet de résoudre le problème unidimensionnel de façon exacte, et qui est interrompu dès qu'un certain critère d'arrêt est vérifié. L'algorithme comporte de deux phases, dont la première a pour but de trouver un intervalle contenant le pas optimal. La seconde explore cet intervalle jusqu'à obtenir un point satisfaisant. Pour un équilibre entre performance (nombre de points évalués pour trouver un pas satisfaisant) et robustesse (capacité à trouver un tel pas), les points à évaluer sont obtenus alternativement selon des formules de dichotomie ou par interpolation.

La première étape est la phase dite d'encadrement (*bracket*, donnée dans l'algorithme 5) du pas optimal. L'objectif est d'obtenir un intervalle $[a, b]$ contenant tous les pas optimaux. Pour cela, il suffit que $\phi'(a) < 0$ et $\phi'(b) > 0$. Si $\phi'(1) > 0$, cette étape n'est pas nécessaire car $[0, 1]$ répond à nos besoins. On peut noter un détail important : l'extrapolation quadratique n'est pas effectuée avec les valeurs de ϕ, mais avec celles de ϕ'. Ceci permet d'être plus robuste aux erreurs numériques, puisque le bruit qu'elles induisent est plus faible sur les dérivées [90]. Les formules utilisées à la ligne 7 sont donc :

$$\begin{aligned} \alpha &= \frac{\phi'(b) - \phi'(a)}{2(b-a)}, \\ \beta &= \frac{\phi'(a) + \phi'(b)}{2} - \alpha(a+b), \\ c &= -\frac{\beta}{2\alpha}. \end{aligned}$$

Algorithme 5 Recherche d'un intervalle contenant les pas optimaux à partir d'un candidat $[0, \delta]$.

```
1 : fonction BRACKET(δ)
2 :     a ← 0 ;
3 :     b ← δ ;
4 :     k ← 0 ;
5 :     tant que (φ'(b) < 0 and k ≤ k_max) faire
6 :         si k est pair alors
7 :             c ← QuadraticMinimum(a, b, φ'(a), φ'(b)) ;
8 :             c ← max(c, ϑb) ;
9 :         sinon
10 :            c ← 2b ;
11 :        fin si
12 :        si StepRules(c) alors ;
13 :            retourner [c, c] ;
14 :        sinon
15 :            si (φ'(c) > 0) alors ;
16 :                retourner [b, c] ;
17 :            sinon
18 :                a ← b ;
19 :                b ← c ;
20 :            fin si
21 :        fin si
22 :        k ← k + 1 ;
23 :    fin tant que
24 :    retourner échec ;
25 : fin fonction
```

Une fois un intervalle $[a, b]$ contenant les pas optimaux trouvé, on utilise la fonction `zoom(a,b)`, donnée dans l'algorithme 6. Cette dernière réduit progressivement l'intervalle $[a, b]$ en maintenant $\phi'(a) < 0$ et $\phi'(b) > 0$. La fonction `CubicMinimum` calcule le minimum d'un polynôme cubique qui interpole ϕ et sa dérivée en a et en b. Avec les signes de ces dérivées, on est assuré que $c \in]a, b[$. Le minimum est calculé avec les formules suivantes (voir [134] par exemple) :

$$\begin{aligned} d_1 &= \phi'(a) + \phi'(b) - 3\frac{\phi(b) - \phi(a)}{b - a}, \\ d_2 &= \sqrt{d_1^2 - \phi'(a)\phi'(b)}, \\ c &= b - (b - a)\frac{\phi'(b) + d_2 - d_1}{\phi'(b) - \phi'(a) + 2d_2}. \end{aligned}$$

Algorithme 6 Algorithme convergeant vers un pas optimal à partir d'un intervalle contenant les pas optimaux ($\phi'(a) < 0$ et $\phi'(b) > 0$), et qui s'interrompt dès que les conditions d'acceptation sont remplies.

```
1 : fonction Zoom(a, b)
2 :     k ← 0 ;
3 :     tant que (b − a > ε_zoom) faire
4 :         si k%2 = 0 alors
5 :             c ← CubicMinimum(a, b, φ(a), φ'(a), φ'(b)) ;
6 :             c ← P_[a+θ(b−a),b−θ(b−a)](c) ;
7 :         sinon
8 :             c ← (a + b)/2 ;
9 :         fin si
10 :        si StepRules(c) alors ;
11 :            retourner c ;
12 :        sinon si (φ'(c) > 0) alors ;
13 :            b ← c ;
14 :            k ← k + 1 ;
15 :        sinon
16 :            a ← c ;
17 :            k ← k + 1 ;
18 :        fin si
19 :    fin tant que
20 :    retourner échec ;
21 : fin fonction
```

Finalement, notre procédure de recherche linéaire peut se résumer avec le diagramme donné en figure 3.16, dans le cas où le premier pas est δ_0.

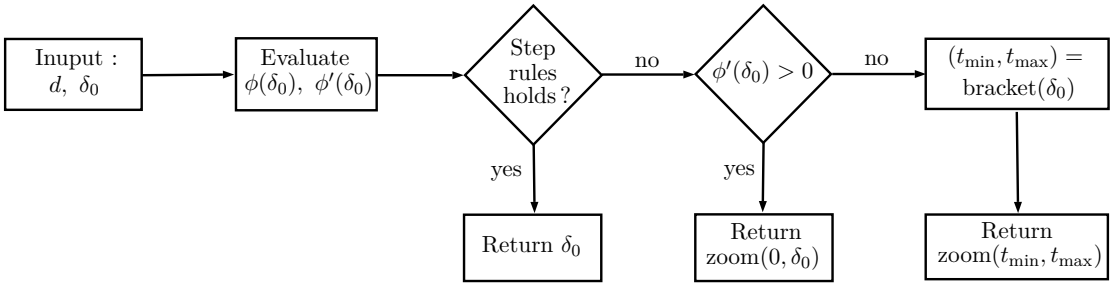

FIGURE 3.16 – *Déroulement de l'algorithme de recherche linéaire selon le résultat du premier pas.*

Impact des règles d'acceptation

L'algorithme 3.16 fait intervenir des règles d'acceptation d'un pas de déplacement. Ces règles ont surtout pour objectif de permettre les preuves de convergence des méthodes de descente les utilisant. Celles-ci comprennent la règle d'Armijo [31],

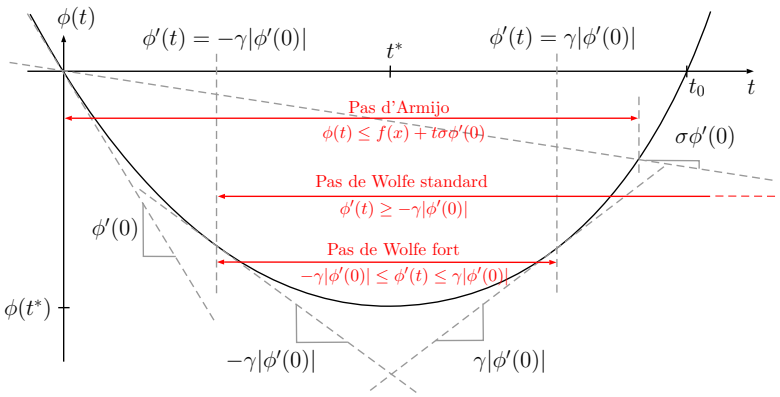

FIGURE 3.17 – *Pas acceptables pour les règles d'Armijo, de Wolfe et de Wolfe forte.*

qui demande

$$\phi(t) \leq \phi(0) + t\sigma\phi'(0), \tag{3.13}$$

pour une constante $0 < \sigma \ll 1$, les conditions de Wolfe standard [167, 168], constituées de la règle d'Armijo, et de

$$\phi'(t) \geq \gamma\phi'(0), \tag{3.14}$$

où γ est un réel dans $]0, 1[$, et les conditions de Wolfe fortes, qui remplacent (3.14) par :

$$|\phi'(t)| \leq \gamma|\phi'(0)|. \tag{3.15}$$

La condition (3.13) empêche les pas ne produisant pas suffisamment de décroissance de f car trop proches de t_0. Elle est utile en théorie, mais n'apporte rien en pratique, puisque sur un ordinateur, la distance de tout nombre strictement positif à zéro est bornée inférieurement. La condition (3.14) empêche les pas trop petits, et la condition (3.15) contrôle directement la précision de la recherche linéaire, puisque $\gamma = 0$ implique que la minimisation doit être exacte et $\gamma = 1$ implique que toute solution de $[0, t^*]$ est acceptée. L'impact de ces règles sur les pas acceptables est résumé dans la figure 3.17.

Chaque direction de recherche utilise les règles d'acceptation prescrites par la littérature. Pour la direction de Newton, si seule la condition d'Armijo est nécessaire, nous étudions la performance globale avec chaque jeu de règles. Nous utilisons de plus une règle d'Armijo non monotone, qui remplace (3.13) par

$$\phi(t) < f^r, \tag{3.16}$$

où f^r est une valeur de référence. Celle-ci est définie comme la valeur maximale de f visitée lors des m dernières itérations (on utilise $m = 10$). Ainsi, au lieu d'imposer la décroissance à chaque itération, on impose la décroissance d'un maximum glissant pris sur les itérations précédentes. Cette règle permet de réduire l'impact du bruit

Décroissance simple au lieu de non monotone	
Échec d'une recherche linéaire	+33%
Nœuds visités sur les instances non résolues	-8%
Règles de Wolfe standard au lieu d'Armijo	
Échec d'une recherche linéaire	+61%
Échec du solveur	+27%
Nœuds visités sur les instances non résolues	+4.9%
Règles de Wolfe fortes au lieu de standard	
Temps de résolution	+11%
Évaluations de f	+80%
Résolution de systèmes linéaires	-2.1%
Nœuds visités sur les instances non résolues	-13%

Tableau 3.7 – *Quelques statistiques sur l'impact des règles d'acceptation d'un pas de déplacement dans l'algorithme de recherche linéaire.*

lié aux erreurs numériques, et peut même accélérer la convergence [49, 87, 88]. L'impact des règles d'acceptation est résumé dans le tableau 3.7. La différence du nombre d'instances résolues est faible, mais l'intérêt de la règle non monotone est évident, tout comme le caractère délétère des règles de Wolfe fortes. Si celles-ci permettent de résoudre les relaxations en résolvant légèrement moins de systèmes linéaires, le nombre d'évaluations de f est presque doublé. L'intérêt des règles de Wolfe standard est plus subtil. Si le nombre d'échecs du solver est plus élevé, plus de nœuds sont visités. Finalement la règle utilisée est (3.16) uniquement, puisqu'activer (3.14) réduit le nombre d'instances résolues de 0.5% en moyenne.

3.4.3 Critère d'arrêt

Le dernier ingrédient à expliquer dans l'algorithme 4 est les critères d'arrêt évoqués à la ligne 3. Comme pour les conditions d'arrêt de l'algorithme du lagrangien augmenté, nous utilisons plusieurs conditions différentes.

Critères usuels

Les critères d'arrêt habituels pour la résolution de (3.10) sont basés sur des conditions de KKT approchés par des tolérances. Ceux que nous utilisons sont listés ci-dessous :

— tolérance absolue sur la mesure d'optimalité : $\|\nabla f(x)\| \leq \varepsilon_{\text{abs}}$,
— tolérance relative sur la mesure d'optimalité : $\|\nabla f(x)\| \leq \varepsilon_{\text{rel}}|f(x)|$,
— la mesure d'optimalité à suffisamment décru : $\|\nabla f(x)\| \leq \varepsilon_{\text{decr}}\|\nabla f(x_0)\|$.

Pour des problèmes mal conditionnés, la géométrie des lignes de niveau et le caractère inexact des opérations sur les flottants peuvent conduire à une suite de points x_k telle que $f(x_k)$ converge bien vers f^*, la valeur optimale, alors que $\|\nabla f(x_k)\|$ ne décroît pas vers zéro, mais vers un réel positif potentiellement grand devant ε_{abs} ou f^*. Le

critère absolu est dans ce cas généralement incapable de détecter la convergence, et les critères relatifs sont indispensables.

D'autres critères sont parfois utilisés, comme une tolérance relative à la magnitude de la solution :

$$\|\nabla f(x)\| \leq \varepsilon_{\text{magn}}\|x\|,$$

ou une stagnation de la valeur de l'objectif :

$$|f(x_{k+1}) - f(x_k)| < \varepsilon_{\text{decr}}|f(x)|.$$

Le premier n'est malheureusement pas applicable dans un *branch-and-bound*, car le minimum de la norme de x peut être grand pour certains nœuds. Dans ce cas, toute solution x satisfait la condition et est déclarée optimale. Le second doit être adapté pour prendre en compte le caractère non monotone de la recherche, mais nous préférons utiliser une autre approche pour détecter la stagnation, décrite en dernier paragraphe.

Interprétation géométrique

La famille de critères d'arrêt présentée ci-dessus est basée sur des conditions de KKT approchés. Une autre approche consiste à s'intéresser à l'optimalité de la solution en valeur, auquel cas ces conditions (au moins la première) ont une interprétation géométrique précise. On dit que x est ε optimale si $f(x) \leq f^* + \varepsilon$: l'erreur commise en valeur est inférieure à ε. La remarque importante est que x peut être ε-optimale, alors que $\|\nabla f(x)\|$ est n'est pas proche de 0. Plus précisément, on peut interpréter le critère $\|\nabla f(x)\| \leq \varepsilon$ de façon analogue à l'interprétation primale du théorème 3.1.3. Pour une solution x donnée, on s'intéresse à l'écart à l'optimum entre $f(x)$ et la solution optimale dans un certain voisinage. On définit donc

$$\text{gap}(f, x, \Delta) = f(x) - \min_{\|d\| \leq \Delta} f(x + d),$$

l'erreur d'optimalité commise dans un voisinage égal à une boule de norme Δ centrée sur x. En utilisant le sous-estimateur linéaire de f de x, on obtient

$$\text{gap}(f, x, \Delta) \leq \max_{\|d\| \leq \Delta} \nabla f(x)^T d.$$

L'analyse des conditions de KKT du problème de droite permet d'obtenir

$$\text{gap}(f, x, \Delta) \leq \Delta\|\nabla f(x)\|.$$

Ceci signifie que pour garantir l'optimalité à ε près dans une boule de rayon Δ, une condition suffisante est $\|\nabla f(x)\| \leq \varepsilon/\Delta$.

Ainsi, le critère $\|\nabla f(x)\| \leq \varepsilon$ peut s'interpréter comme : x est ε-optimal dans une boule de rayon 1. Une autre possibilité est de dire qu'alors, x est $\sqrt{\varepsilon}$-optimal dans une boule de rayon $1/\sqrt{\varepsilon}$. Si la distance entre x et l'ensemble des solutions optimales est grande, il peut être satisfait par des solutions dont l'erreur d'optimalité est arbitrairement grande.

Critère avec dualité

Les critères décrits plus hauts ont deux problèmes. Ils peuvent ne pas détecter la convergence, ou alors déclarer optimales des solutions qui sont loin de l'être. Si le second problème n'a pas vraiment de solution (à part ne pas utiliser ces critères, ce qui aurait des conséquences pratiques trop délétères), le premier peut être résolu grâce au théorème 3.3.

Lors de la résolution de (3.10), on note $(x_1, \cdots, x_k)$ la suite de points visités. On définit alors les bornes primales et duales de (3.10) par

$$\overline{f} = \min_{i=1..k} f(x_k),$$

et

$$\underline{f} = \max_{i=1..k} \theta(x_k),$$

où θ est la fonction définie par (3.2). La solution x définissant $\overline{f}$ est alors prouvée ε-optimale dès que

$$\frac{\overline{f} - \underline{f}}{1 + |\overline{f}|} \leq \varepsilon, \tag{3.17}$$

et l'on peut alors interrompre la résolution du sous-problème.

Ce raisonnement serait valide si des contraintes de bornes étaient présentes dans le sous-problème, ce qui n'est pas le cas. Il est alors possible que $\overline{f} < \underline{f}$. Cependant, le critère (3.17) est quand même utilisé et améliore la situation. La figure 3.18 montre la proportion des sous-problèmes interrompus par le critère (3.17). L'explication vient du fait que si $\overline{f} < \underline{f}$, alors x^*, la solution optimale de (3.10), est nécessairement en dehors des bornes $\ell \leq x \leq u$. Ce critère d'arrêt à donc pour conséquence que les multiplicateurs et facteurs de pénalisation sont mis à jour jusqu'à ce que la solution du sous-problème vérifie les contraintes de bornes. Pour des questions de stabilité, ce critère n'est pas activé pendant les $k = 2$ premières itérations de chaque sous-problème, afin d'imposer au moins deux itérations avant d'interrompre la recherche. Ceci a pour effet de limiter l'augmentation des facteurs de pénalisation lorsque c'est ce critère qui interrompt la recherche.

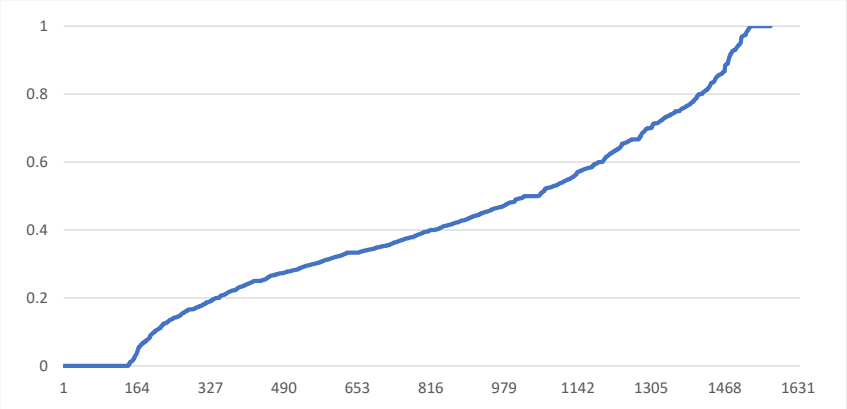

FIGURE 3.18 – *Proportion des sous-problèmes du lagrangien augmenté interrompus par le critère* (3.17).

Détection prématurée des échecs

Une dernière série de critères d'arrêt sont utilisés pour détecter la stagnation de la recherche. En l'absence de stricte convexité, la performance de la méthode de Newton avec correction de la hessienne peut devenir aussi mauvaise que celle d'une descente de gradient, nécessitant parfois des millions d'itérations pour résoudre le sous-problème. Deux garde-fous sont alors utilisés. Premièrement, une étude statistique sur les sous-problèmes dont la résolution se passe bien montre qu'au-delà de quelques centaines d'itérations de la méthode de Newton, la probabilité de succès plonge vers 0. Ainsi, une itération de l'algorithme du lagrangien augmenté est déclenchée au bout de 500 itérations, indépendamment des autres critères. Ensuite, plutôt que de détecter la stagnation en termes de valeur de l'objecif, nous utilisons deux grandeurs : la norme du déplacement $\|x_{k+1} - x_k\|$ et le pas de déplacement $\|x_{k+1} - x_k\|/\|d_k\|$ si l'une de ces deux quantités est inférieure à une certaine tolérance pendant $k = 10$ itérations consécutives, une itération extérieure est déclenchée.

Enfin, les sous-problèmes interrompus par ces considérations retournent un statut spécial. Celui-ci indique au solveur que des difficultés numériques surviennent, et permet accélérer la détection des échecs de l'algorithme du lagrangien augmenté. Si 10 sous-problèmes consécutifs retournent ce statut, un échec de résolution de la relaxation est retourné.

Chapitre 4

Calcul de bornes inférieures

La reformulation du problème, la génération, puis la résolution d'une relaxation convexe permettent d'obtenir une borne inférieure. Si celle-ci n'est pas optimale, ou si la solution de la relaxation n'est pas admissible, il faut ensuite utiliser des techniques permettant de l'améliorer, ou de réduire l'infaisabilité de la solution associée. Dans le modèle factorisé, la qualité des relaxations convexes dépend des tailles des domaines des variables. Si ces domaines tendent vers des singletons, alors l'erreur commise en formant la relaxation convexe tend vers zéro. Par conséquent, améliorer la borne inférieure obtenue sans changer les relaxations convexes nécessite de resserrer les bornes des variables.

Une technique universelle remplissant cet objectif, et permettant de plus de converger vers la solution optimale, est la recherche arborescente. Les variables participant aux non-convexités pouvant être mixtes, nous utilisons une technique de partitionnement de type *branch-and-bound* spatial. Une autre famille d'algorithmes, appelées techniques de réduction de bornes, exploite diverses informations pour réduire les domaines des variables et de l'objectif. Ces techniques interagissent avec la recherche arborescente, en détectant des nœuds inconsistants et en s'intercalant entre les résolutions de relaxations convexes.

Lorsque la résolution de relaxations convexes et les techniques de réduction de bornes sont les deux outils principaux d'une recherche arborescente, celle-ci est dite de type *branch-and-reduce*. Ce chapitre a pour objet l'implémentation d'une telle approche. Nous y présentons la cinématique, les règles de partitionnements ainsi que les techniques de réduction de bornes utilisées.

Plan du chapitre

4.1 L'algorithme *branch-and-reduce*

4.1.1 Une méthode de recherche arborescente

Les méthodes de séparation et évaluation, plus connues sous leur nom anglais de *branch-and-bound*, sont un outil central en programmation mathématique. Ce sont des méthodes énumératives basées sur le principe de diviser pour régner, notamment utilisées pour la résolution de problèmes combinatoires. Plus généralement, elles permettent de résoudre tous les problèmes en variables mixtes dont on sait résoudre la relaxation continue. Par exemple, en présence d'une contrainte d'intégrité $x_i \in \mathbb{Z}$ violée par une solution x^* de la relaxation continue, l'ensemble admissible peut être partitionné en les deux sous-domaines, selon que $x_i > x_i^*$ ou $x_i < x_i^*$. En utilisant l'intégrité de x_i pour resserrer ses bornes, x^* n'est alors plus admissible dans ces deux sous-domaines. Après un nombre fini de partitionnements, toutes les variables entières sont fixées et les contraintes d'intégrité sont satisfaites.

Les méthodes de *branch-and-bound* spatiales ont été introduites dans le contexte de l'optimisation globale [95], et généralisent le principe de partitionnement aux variables continues. Les différentes relaxations convexes exposées au chapitre 2 ont toutes en commun que l'erreur de convexité commise sur chaque contrainte et sur l'objectif dépend directement des bornes des variables. Par exemple, pour un couplage bilinéaire $z = xy$ avec $x \in [\ell_x, u_x]$ et $y \in [\ell_y, u_y]$, la violation maximale de la contrainte par une variable z satisfaisant les inégalités de Mc Cormick est $|z - xy| \leq (u_x - \ell_x)(u_y - \ell_y)/4$. De même, pour un terme de convexification $\phi_{\alpha,\ell,u}(x) = \sum_i \alpha_i(x_i - \ell_i)(x_i - u_i)$, l'erreur maximale de convexité induite est $\frac{1}{4}\sum_i \alpha_i(u_i - \ell_i)^2$. Pour les autres relations, et pour les preuves, voir par exemple [155]. Réduire le domaine des variables limite donc l'erreur commise lors de la formation de la relaxation convexe, et la réduit à zéro lorsque les domaines tendent vers des singletons. Une recherche arborescente permet donc de converger vers une solution admissible, ou vers une preuve d'inconsistance. L'algorithme *branch-and-reduce*, introduit dans [144], nous permet donc d'obtenir une méthode convergente pour la résolution à l'optimum global du problème reformulé (2.1), défini au second

chapitre :

$$\min_x \; x^T Q x + c^T x + a$$
$$\text{s.c.} \begin{cases} \ell \leq x \leq u, \\ x \in \mathcal{P} \cap \mathcal{Q} \cap \mathcal{R} \cap \mathcal{B} \cap \mathcal{M} \cap \mathcal{C}, \\ \forall i \in I, \; x_i \in \mathbb{Z}. \end{cases}$$

Dans une méthode de *branch-and-bound*, le partitionnement de l'hyperrectangle $\ell \leq x \leq u$ a pour objectif de réduire l'infaisabilité de la solution de la relaxation convexe, mais aussi d'améliorer la borne obtenue. Dans notre contexte, c'est surtout ce second aspect qui nous intéresse : nous avons volontairement laissé les heuristiques primales de côté, en décidant de découpler entièrement les recherches primales et duales. Ainsi, les autres modules de LocalSolver, et en particulier la recherche locale, ont la charge de trouver les solutions admissibles. Notre implémentation du *branch-and-reduce* s'en trouve simplifiée, puisqu'uniquement dédiée au calcul de bornes. Si la solution d'une relaxation convexe se trouve être admissible, cette solution est envoyée aux autres modules de LocalSolver, qui tenteront de l'améliorer.

Un nœud est défini par la donnée de deux vecteurs représentant un sous-domaine des contraintes de bornes du problème initial. Ainsi, pour $\ell \leq \tilde{\ell}$ et $\tilde{u} \leq u$, $\tilde{\ell} \leq x \leq \tilde{u}$ définit un nœud de la recherche. Une partition du domaine initial des variables, notée $\mathcal{P} = \{n_1, \cdots, n_k\}$ et composée des nœuds $n_1, \cdots, n_k$, fournit une borne inférieure au problème. Cette borne est celle portée par le nœud courant, celui dont la borne inférieure est la plus petite. L'arbre de branchement est défini comme l'organisation des nœuds visités pendant la recherche en un graphe dont les arrêtes représentent leurs relations de paternité. Les nœuds ouverts sont ceux qui n'ont pas encore été partitionnés, c'est-à-dire ceux qui n'ont pas d'enfants, soit les feuilles de l'arbre. Dans notre cas, on ne stocke pas toute la structure de l'arbre, mais seulement la partition induite, constituée des feuilles. Celles-ci sont organisées en une file de priorité, qui les trie par bornes inférieures croissantes, et qui permet l'accès au nœud courant en temps constant. L'algorithme *branch-and-reduce*, introduit dans [144] et résumé dans la figure 4.1, alterne partitionnement du nœud courant et réduction de bornes sur les sous-parties obtenues.

4.1.2 Les fonctions *branch* et *reduce*

Réduction cyclique d'un nœud : la fonction *reduce*

La fonction *reduce* est la fonction d'évaluation de l'algorithme *branch-and-reduce*. Son objectif est de réduire les bornes de l'objectif, mais aussi les bornes des variables. Pour cela, elle commence par propager la réduction obtenue lors du partitionnement du domaine de la variables sur laquelle on vient de brancher (sauf pour le *root node*), puis elle alterne entre techniques de réduction de bornes et relaxations convexes. L'intérêt d'atteindre un point fixe approché pour les bornes du nœud est de réduire la taille de la partition. Dans LocalSolver, on démarre un nouveau cycle de réduction si celui qui vient d'être effectué a permis de réduire une certaine mesure de la taille des bornes (réduction d'un facteur $\gamma < 1$ du domaine de l'objectif ou de la somme des tailles des domaines des variables). Le déroulement de la fonction *reduce* est présenté

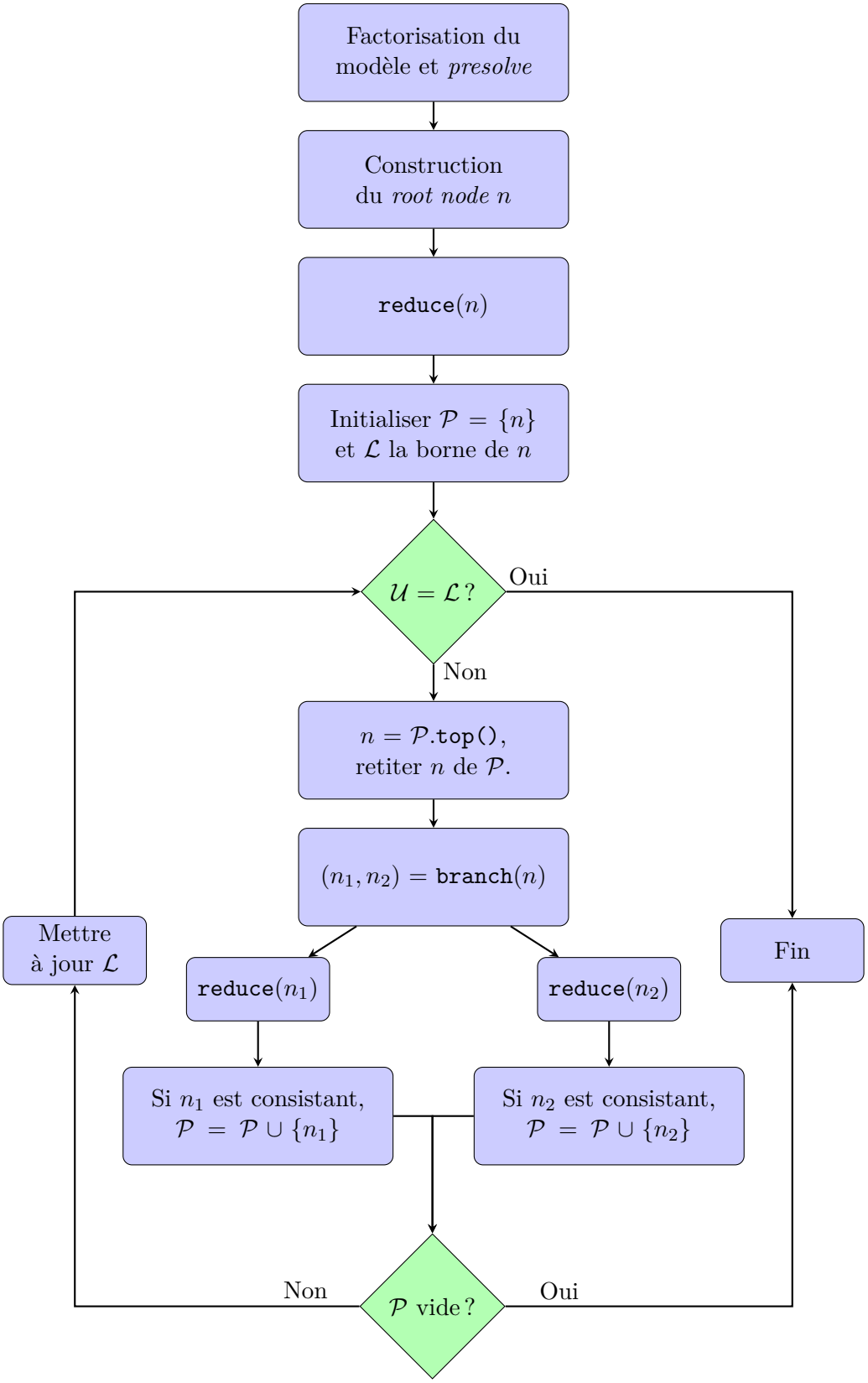

FIGURE 4.1 – *L'algrithme* branch-and-reduce. *On a noté $\mathcal{U}$ la borne supérieure (la meilleure solution admissible), $\mathcal{P} = \{n_1, \cdots, n_k\}$ la partition de l'ensemble admissible en nœuds ouverts, et $\mathcal{L}$ la borne inférieure (celle du premier élément de $\mathcal{P}$).*

dans la figure 4.2, les techniques de réduction de bornes qui y sont évoquées sont présentées dans les autres sections de ce chapitre.

Règles et scores de branchement : la fonction *branch*

La résolution à l'optimum global d'un MINLP non convexe est un problème NP-difficile. Même si la résolution des relaxations convexes est rapide, le coût en temps et en mémoire du *branch-and-reduce* explose rapidement. En contrepartie, les techniques de recherche arborescente sont robustes et permettent de résoudre la plupart des instances, si l'on leur laisse suffisamment de temps.

Une fois la fonction d'évaluation d'un nœud définie, la recherche se résume aux choix de branchements qui sont effectués. À chaque itération, le nœud portant la plus mauvaise borne inférieure est partitionné suivant une variable. Le choix de cette variable, ainsi que du point de partitionnement de son domaine, forment ce que l'on appelle une décision de branchement. Les règles de branchement, qui régissent la prise d'une décision de branchement, ont un impact majeur sur l'efficacité de la recherche, à la fois en temps et en nombre de nœud nécessaires à la résolution à l'optimum.

Les conditions théoriques sur les règles de branchement permettant d'assurer la convergence de la recherche, ainsi que le caractère fini de cette dernière, sont faibles. Sur un ordinateur, le domaine des variables $\ell \leq x \leq u$ est de toute façon un ensemble fini, qu'il suffit d'énumérer pour obtenir la convergence. L'implémentation de règles de branchement jouit donc d'une certaine liberté, et reste un domaine largement heuristique.

L'intérêt pratique du partitionnement est cependant limité aux instances de petite taille. L'exploration d'un grand nombre de nœuds n'est généralement pas possible pour des instances de grande taille. Si les règles de partitionnement ont un impact certain sur certains *benchmarks*, tels que la MINLPLib, ce n'est pas le cas sur d'autres. Pour cette raison, les règles de branchement implémentées sont élémentaires, et peuvent se définir comme un branchement par scores. Les formules utilisées ont été intuitées, complétées avec des idées issues de [36, 155] et réglées par expérimentation numérique.

Sélection de la variable de branchement

Pour sélectionner la variable sur laquelle brancher, nous calculons une liste de scores de branchement pour chacune variable. Cette liste est composée des scores associés à chaque contrainte violée dans laquelle la variable apparaît. Pour une contrainte d'intégrité, pour une variable entière $x_i \in [\ell_i, u_i]$, le score ajouté à la liste est donné par

$$\min(x_i^* - \lfloor x_i^* \rfloor, \lceil x_i^* \rceil - x_i^*),$$

où x_i^* est la valeur de la solution de la relaxation convexe. Pour une relation unaire $y = f(x)$, le score est

$$\frac{|y^* - f(x^*)|}{1 + \|\nabla f(x^*)\|_2}.$$

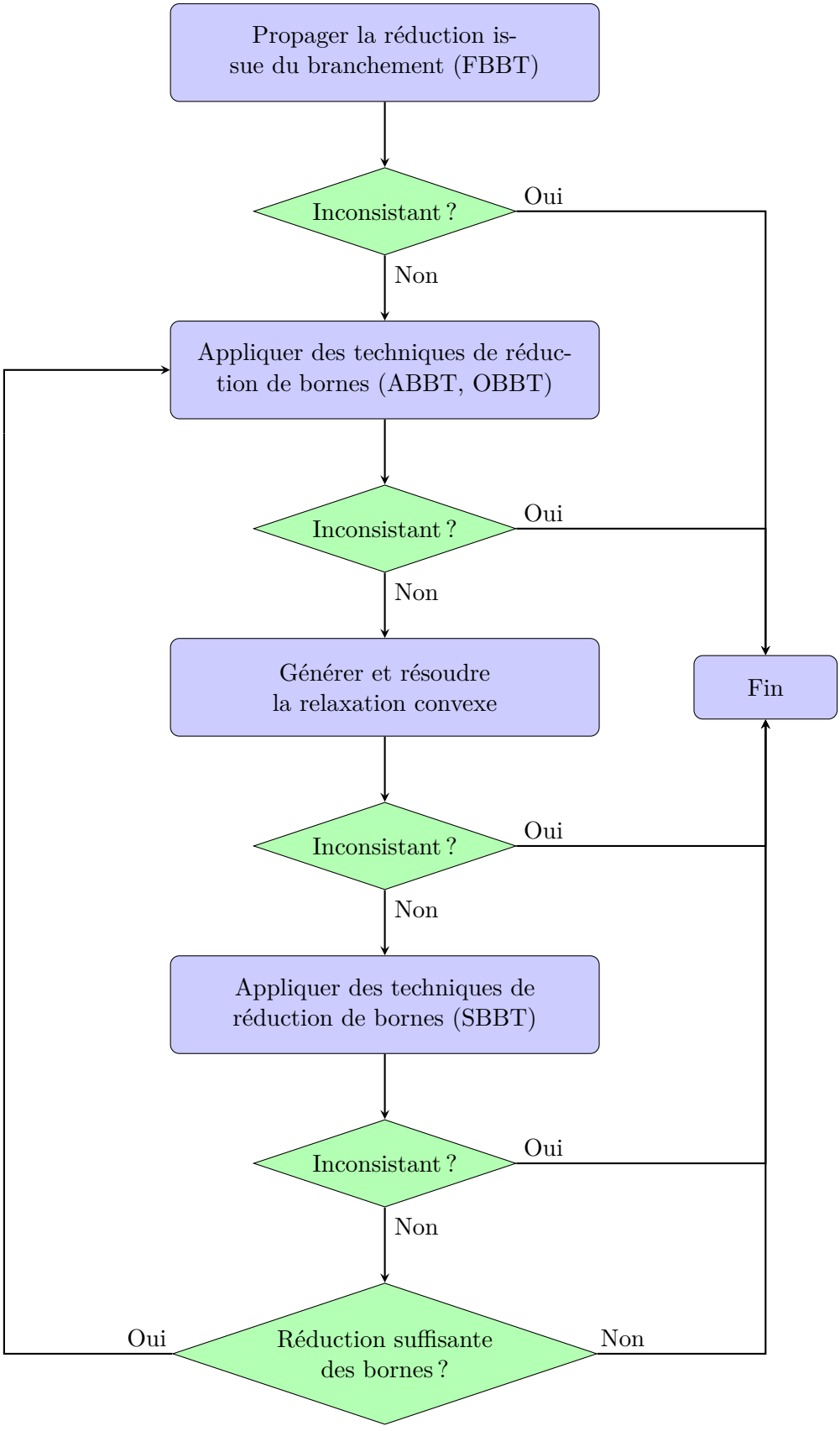

FIGURE 4.2 – *Déroulement de la fonction* reduce. *Les étapes FBBT, ABBT, OBBT et SBBT sont quatre techniques de réduction de bornes, présentées dans la suite du chapitre.*

Si f est bijective, ce score est attribué à x et y, car brancher sur l'un réduira aussi les bornes de l'autre, par propagation. Si f n'est pas bijective en revanche, ce score n'est attribué qu'à x. Pour un couplage bilinéaire $z = xy$, le score utilisé est

$$|z^* - x^* y^*|,$$

et il est attribué à x et à y. Enfin, dans le cas où l'on utilise des relaxations non linéaires, une dernière source de non-convexités est constituée des termes de convexification de l'objectif et des inégalités quadratiques. Ces erreurs de convexité sont les seules présentes dans le solveur αBB [21], mais peu de détails sont donnés sur les règles de branchement utilisées. Dans [28], l'auteur indique même utiliser par défaut une règle de partitionnement selon la variable de plus grand domaine. Si le terme de convexification associé est $\phi_{\alpha,\ell,u}(x) = \sum_i \alpha_i (x_i - \ell_i)(x_i - u_i)$, le score de x_i est défini par

$$\alpha_i (x_i^* - \ell_i)(u_i - x_i^*).$$

Si l'expression est l'objectif, le score représente l'amélioration espérée de la borne inférieure si l'on branche sur x_i. Pour une contrainte violée, comme sa relaxation convexe est satisfaite par le solveur, le score représente le gain de faisabilité espéré.

Enfin, tous ces scores sont agrégés en un unique réel. Premièrement le score maximal lié à une violation d'intégrité $s^{\text{int}}_{\max}$ est calculé. Le score d'intégrité de chaque variable est ensuite normalisé avec $s^{\text{int}}_i = s_i / s^{\text{int}}_{\max}$. Ensuite les scores liés aux couplages non linéaires et aux termes de convexification sont agrégés pour chaque variable. Si x_i possède les scores $(s_1, \cdots, s_k)$, le score non linéaire utilisé est $\sum_i s_i + \gamma \max_i s_i + \gamma \min_i s_i$, avec $\gamma = 0.1$. Le score non linéaire maximal $s^{\text{nl}}_{\max}$ est utilisé pour normaliser les scores non linéaires avec $s^{\text{nl}}_i = s_i / s^{\text{nl}}_{\max}$. L'agrégation finale est alors définie par :

$$s_i = (1 + s^{\text{int}}_i)(1 + s^{\text{nl}}_i) \in [1, 4].$$

La variable sélectionnée est finalement celle maximisant ce score.

Sélection du point de partitionnement

Dans [36], il est montré que les différentes stratégies de sélection du point de partitionnement ont peu d'impact sur la résolution. Nous utilisons donc des règles simples, basées uniquement sur la valeur de la solution et des bornes associées, plutôt que sur les contraintes violées qui ont déterminé le domaine $[\ell_i, u_i]$ à partitionner. Si $x_i \in \mathbb{Z}$, le point de partitionnement est x_i^*. Sinon, on utilise une combinaison convexe entre le point courant et le milieu de l'intervalle, avec

$$p = \frac{2x_i^* + \ell_i + u_i}{4}.$$

Si zéro est dans le domaine de x_i et si la distance entre p et 0 est inférieure à un pourcent de la taille du domaine, le partitionnement est effectué en 0. La raison de ceci est que certaines relations non linéaires voient leur relaxation convexe et propagation de bornes améliorées lorsque le signe des variables est fixé.

Branchement sur des contraintes

Une alternative au branchement par scores sur les variables est de considérer un branchement par contraintes. Par exemple, pour des relations du type $xy = 0$ ou $y = (x \geq a)$, des partitionnements possibles sont $x = 0$ ou $y = 0$ et $y = 1, x \geq a$ ou $y = 0, x < a$. Pour des contraintes de type $y = \min(x_i)$, on peut considérer des branchements non binaires, avec autant d'enfants que d'éléments dans le min. Dans chaque sous-partie obtenue, une contrainte du type $y = x_i$ est ajoutée. Ces stratégies non conventionnelles facilitent l'énumération des solutions pour certains types de contraintes. Trop peu d'expériences numériques (la MINLPLib ne contient pas ces opérateurs) sont cependant disponibles pour évaluer leur apport. Elles ont été implémentées dans l'optique d'orienter le branchement sur des instances où les scores et les partitions de domaines n'apportent que peu d'informations.

4.2 Propagation et inférence de bornes

La première technique de resserrage de bornes que nous allons présenter s'appelle *feasibility based bound tightening*, et nous y ferons référence par FBBT. La FBBT permet d'éliminer des régions ne contenant aucune solution admissible et potentiellement de détecter l'inconsistance sans avoir à faire appel à un solveur. L'idée est de visiter chaque expression du modèle et d'y appliquer deux principes, appelés *propagation* et *inférence* de bornes. La FBBT est généralement implémentée directement sur le graphe du problème, ou alors sur un modèle explicite, comme en programmation linéaire ou quadratique par exemple. Notre contribution est ici d'implémenter la FBBT sur le modèle factorisé, obtenant ainsi des gains significatifs par rapport à une implémentation basée sur un DAG. Notre implémentation est similaire à [17] (implémentation pour la PLNE), et nous nous attacherons à décrire tous les détails d'implémentation, dont certains sont souvent passés sous silence.

Dans un DAG, la propagation consiste à resserrer les bornes d'un nœud à partir de l'opérateur que celui-ci porte et des bornes de ses enfants. L'inférence de bornes consiste quant à elle à resserrer les bornes des enfants d'un nœud à partir des bornes de celui-ci. En particulier, l'inférence exploite le fait que les contraintes imposent des bornes sur certains nœuds. Dans le modèle factorisé, qui est un DAG à un seul rang, chaque expression est une contrainte, à l'exception de l'objectif. Dans le contexte d'un *branch-and-bound*, porte cependant une contrainte implicite, puisque l'on recherche une solution dont l'objectif est compris entre les bornes globales du problème. Ainsi, dans la FBBT, l'objectif est traité comme deux inégalités, linéaires ou quadratiques.

Dans le modèle factorisé, la propagation est effectuée en obtenant par manipulations algébriques des expressions de la forme $g_1(x_1, \cdots, x_k) \leq y \leq g_2(x_1, \cdots, x_k)$, où $y \notin \{x_i\}$. Si l'on est capable de calculer $\min_x g_1(x_1, \cdots, x_k)$ et $\max_x g_2(x_1, \cdots, x_k)$ sur $\ell \leq x \leq u$, on peut alors propager ces bornes sur y. Inversement, l'inférence est effectuée en partant d'une contrainte écrite sous la forme $g(x_1, \cdots, x_k) \in [L, U]$. Si l'on est capable d'obtenir que ceci implique $x_i \in [L_i, U_i]$, ces bornes sont inférées sur x_i.

Lors du déroulement de la FBBT, si une expression (variable, contrainte ou

expression intermédiaire) possède des bornes de propagation (induites par les bornes des variables) et des bornes d'inférence (induites par les bornes des contraintes) d'intersection vide, alors le problème peut être déclaré inconsistant.

4.2.1 Formules de propagation et d'inférence

Nous présentons maintenant les calculs permettant de propager et d'inférer de nouvelles bornes sur chacun des trois types d'expressions du modèle factorisé : les couplages non linéaires et les expressions linéaires et quadratiques (groupées sous le nom de sommes).

Couplages non linéaires

Pour les couplages non linéaires, les variables en présence dans une contrainte n'y apparaissent qu'une seule fois. Il est donc facile de calculer les bornes de propagation et d'inférence. Les calculs sont effectués à l'aide de l'arithmétique d'intervalle, dont l'implémentation sera présentée en fin de partie. L'ensemble des opérations de propagation et d'inférence est résumé dans les tableaux 4.1 et 4.2.

Sommes linéaires

Une expression linéaire, de la forme $a^T x + b = b + \sum_i a_i x_i$, est séparable en les variables présentes. Dans ce cas, les bornes de propagation s'obtiennent en sommant les bornes de chaque terme :

$$a^T x + b \in b + \sum_i a_i * [\ell_i, u_i] = \left[b + \sum_i \min(a_i \ell_i, a_i u_i), b + \sum_i \max(a_i \ell_i, a_i u_i) \right].$$

Pour l'inférence, on se donne une contrainte $L \leq b + \sum_i a_i x_i \leq U$. Dans le cas d'une égalité, $[L, U] = \{0\}$, et dans le cas d'une inégalité, $[L, U] =]-\infty, 0]$. En réarrangeant les termes on peut inférer que pour chaque variable présente dans la contrainte,

$$x_i \in \begin{cases} \left[\dfrac{L-b}{a_i} - \displaystyle\sum_{j \neq i} \max\left(\frac{a_j}{a_i}\ell_i, \frac{a_j}{a_i}u_i \right), \dfrac{U-b}{a_i} - \displaystyle\sum_{j \neq i} \min\left(\frac{a_j}{a_i}\ell_i, \frac{a_j}{a_i}u_i \right) \right], & \text{si } a_i > 0, \\ \left[\dfrac{U-b}{a_i} - \displaystyle\sum_{j \neq i} \max\left(\frac{a_j}{a_i}\ell_i, \frac{a_j}{a_i}u_i \right), \dfrac{L-b}{a_i} - \displaystyle\sum_{j \neq i} \min\left(\frac{a_j}{a_i}\ell_i, \frac{a_j}{a_i}u_i \right) \right], & \text{si } a_i < 0. \end{cases}$$

Sommes quadratiques

On se donne une expression quadratique de la forme

$$x^T Q x + c^T x + a = \sum_{ij} q_{ij} x_i x_j + \sum_i c_i x_i + a.$$

Contrainte	Sous-cas	Bornes propagées et inférées
$y = x^{2k}$	$\ell_x \geq 0$	$y \in [\ell_x^{2k}, u_x^{2k}]$
		$x \in [\ell_y^{1/2k}, u_y^{1/2k}]$
	$u_x \leq 0$	$y \in [u_x^{2k}, \ell_x^{2k}]$
		$x \in [-u_y^{1/2k}, -\ell_y^{1/2k}]$
	$0 \in]\ell_x, u_x[$	$y \in [0, \max\left(\ell_x^{2k}, u_x^{2k}\right)]$
		$x \in [-u_y^{1/2k}, u_y^{1/2k}]$
$y = x^{2k+1}$	-	$y \in [\ell_x^{2k+1}, u_x^{2k+1}]$
		$x \in [\text{sign}(\ell_y)\lvert\ell_y\rvert^{1/2k+1}, \text{sign}(u_y)\lvert u_y\rvert^{1/2k+1}]$
$y = \lvert x\rvert$	$\ell_x \geq 0$	$y \in [\ell_x, u_x]$
		$x \in [\ell_y, u_y]$
	$u_x \leq 0$	$y \in [-u_x, -\ell_x]$
		$x \in [-u_y, -\ell_y]$
	$0 \in]\ell_x, u_x[$	$y \in [0, \max(\lvert\ell_x\rvert, \lvert u_x\rvert)]$
		$x \in [-u_y, u_y]$
$y = x^a,\ a \notin \mathbb{N}$	$a > 1$, $\ell_x \geq 0$	$y \in [\ell_x^a, u_x^a]$
		$x \in [\ell_y^{1/a}, u_y^{1/a}]$
$y = \mathrm{e}^x$	-	$y \in [\mathrm{e}^{\ell_x}, \mathrm{e}^{u_x}]$
		$x \in [\log \ell_y,\ \log u_y]$
$x \in \mathbb{Z}$	-	$x \in [\lceil \ell_x \rceil, \lfloor u_x \rfloor]$
$y = (x \geq a)$	$(x \geq a) \vee (y = 1)$	$y \in \{1\}$
		$x \in [a, +\infty[$
	$(x < a) \vee (y = 0)$	$y \in \{0\}$
		$x \in]-\infty, a[$
$y = (x = a)$	$(x = a) \vee (y = 1)$	$y \in \{1\}$
		$x \in \{a\}$
	$\ell_x \geq a$	$y = (u_x = a)$
		$x \in]a, +\infty[$
	$u_x \leq a$	$y = (\ell_x = a)$
		$x \in]-\infty, a[$

Tableau 4.1 – *Propagation et inférence de bornes sur les couplages non linéaires*

Contrainte	Sous-cas	Bornes propagées et inférées
$y = \log x$	$\ell_x \geq 0$	$y \in [\log \ell_x, \log u_x]$
		$x \in [\mathrm{e}^{\ell_y},\ \mathrm{e}^{u_y}]$
$xy = 1$	$\ell_x u_x \geq 0$	$y \in [1/u_x, 1/\ell_x]$
		$x \in [1/u_y, 1/\ell_y]$
	$0 \in]\ell_x, u_x[$	$y \in [-\infty, 1/\ell_x] \cup [1/u_x, +\infty]$
		$x \in [-\infty, 1/\ell_y] \cup [1/u_y, +\infty]$
$z = xy$	-	$z \in [\ell_x, u_x] * [\ell_y, u_y]$
		$x \in [\ell_z, u_z]/[\ell_y, u_y]$
		$y \in [\ell_z, u_z]/[\ell_x, u_x]$
$y = \max(x_1, \cdots, x_k)$	-	$y \in [\max(\ell_{x_1}, \cdots, \ell_{x_k}), \max(u_{x_1}, \cdots, u_{x_k})]$
		$x_i \in [-\infty, u_y]$
$y = \min(x_1, \cdots, x_k)$	-	$y \in [\min(\ell_{x_1}, \cdots, \ell_{x_k}), \min(u_{x_1}, \cdots, u_{x_k})]$
		$x_i \in [\ell_y, +\infty]$
$y = \sin x$	-	$y \in \sin([\ell_x, \ell_y])$
		$x \in \arcsin([\ell_x, u_x] \bmod 2\pi)$
$y = \cos x$	-	$y \in \cos([\ell_x, u_x])$
		$x \in \arccos([\ell_y, u_y] \bmod 2\pi)$
$y = \tan x$	-	$y \in \tan([\ell_x, \ell_y])$
		$x \in \arctan([\ell_x, u_x] \bmod \pi)$

Tableau 4.2 – *Propagation et inférence de bornes sur les couplages non linéaires*

Dans le cas général, le calcul des bornes de propagation et d'inférence optimales d'une telle expression est NP-difficile. En effet, celui-ci se ramène à l'optimisation quadratique non convexe sous contraintes de bornes. Si Q est convexe ou concave, certaines de ces bornes peuvent être obtenues en temps polynomial, *via* l'utilisation d'un algorithme de points intérieurs par exemple. Puisque la FBBT est appliquée à chaque expression quadratique plusieurs fois par nœud, cette approche est trop coûteuse. On se contente donc, dans le cas général, de bornes sous-optimales rapides à obtenir. Il existe dans la littérature plusieurs approches pour calculer des bornes valides, dont aucune ne domine les autres.

Une première possibilité pour obtenir des bornes de propagation est de considérer chaque terme de la contrainte indépendamment, comme si celle-ci était séparable :

$$x^T Q x + c^T x + a \in a + \sum_i c_i * [\ell_i, u_i] + \sum_{i,j} q_{ij} * [\ell_i, u_i] * [\ell_j, u_j].$$

Cette formule naïve est rapide à évaluer, mais ne tient pas compte du fait qu'une même variable peut apparaître dans différents termes et ainsi contribuer aux bornes de propagation tantôt par sa borne inférieure, tantôt par sa borne supérieure.

Une première amélioration, notamment proposée dans [72], est de regrouper les termes carrés et linéaires pour chaque variable :

$$x^T Q x + c^T x + a = \sum_k (q_{kk} x_k^2 + c_k x_k) + \sum_{i \neq j} q_{ij} x_i x_j.$$

Cette formule permet de calculer les bornes optimales dans le cas séparable. Dans le cas général, [72] propose trois stratégies pour prendre en compte les termes bilinéaires. La première remplace $q_{ij} x_i x_j$ par le produit du coefficient et des intervalles des variables. La seconde remplace $q_{ij} x_i x_j$ par des estimateurs linéaires séparables, typiquement ceux de Mc Cormick. Enfin, la dernière encadre $q_{ij} x_i x_j$ par des termes quadratiques séparables en utilisant les identités suivantes, valides $\forall \beta \in \mathbb{R}^*$:

$$q_{ij} x_i x_j = \left(\frac{q_{ij}}{2\beta} x_i + \beta x_j\right)^2 - \frac{q_{ij}^2}{4\beta^2} x_i^2 - \beta^2 x_j^2 = -\left(\frac{q_{ij}}{2\beta} x_i - \beta x_j\right)^2 + \frac{q_{ij}^2}{4\beta^2} x_i^2 + \beta^2 x_j^2.$$

Une fois les termes non séparables remplacés par leurs signes, ceci donne

$$-\frac{q_{ij}^2}{4\beta^2} x_i^2 - \beta^2 x_j^2 \leq q_{ij} x_i x_j \leq \frac{q_{ij}^2}{4\beta^2} x_i^2 + \beta^2 x_j^2.$$

On a alors encadré l'expression quadratique initiale par des expressions quadratiques séparables, dont on peut alors obtenir les bornes de propagation optimales. On peut aussi choisir les β utilisés pour chaque terme bilinéaire afin d'obtenir les meilleures bornes. Dans [72], il est dit que la première approche fournit les meilleures bornes si celles de x_i et de x_j ne sont pas trop grandes. La troisième approche est la plus efficace dans le cas contraire et sous certaines conditions de signe des autres coefficients. Enfin, la seconde est utilisée dans certaines applications uniquement.

Dans le solveur GloMIQO [125], une autre approche est utilisée. L'idée est d'améliorer la formule naïve décrite plus haut en réarrangeant l'expression pour réduire le nombre d'occurrences d'une variable. Ainsi, les bornes de cette variable contribueront une seule fois aux bornes finales. L'approche utilisée par GloMIQO est de

factoriser l'expression par chaque variable qui y est présente. Par exemple pour x_k, on obtient :

$$x^T Q x + c^T x + a = x_k \left(c_k + \sum_i q_{ik} x_i \right) + q(x),$$

où $q(x)$ est une expression quadratique ne dépendant pas de x_k. Cette nouvelle expression est ensuite utilisée pour obtenir des bornes de propagation sur l'expression initiale et inférer des bornes sur x_k, en remplaçant chaque autre variable par son intervalle de définition. Avec cette approche, les bornes de propagation d'une expression quadratique sont l'intersection de toutes les bornes de propagation obtenues en factorisant l'expression quadratique par l'une des variables présentes.

Enfin, dans [160], le cas d'une expression quadratique à deux variables est traité de façon optimale. Il n'est pas précisé comment les autres termes sont bornés si la contrainte possède plus de deux variables. Nous n'avons pas cherché à implémenter cette approche, les calculs étant plus complexes, avec en particulier des disjonctions de cas importants.

Notre approche se rapproche de celle de GloMIQO, mais utilise des factorisations plus complètes :

$$\forall k,\ x^T Q x + c^T x + a = q_{kk} x_k^2 + A_k x_k + B_k,$$

avec

$$A_k = c_k + \sum_{i \neq k} q_{ik} x_i \text{ et } B_k = a + \sum_{i \neq k} c_i x_i + \sum_{i \neq k, j \neq k} q_{ij} x_i x_j.$$

On utilise de plus une structure de données permettant d'obtenir toutes ces factorisations en deux parcours de l'expression. On remplace ensuite A_k et B_k par des intervalles. A_k est une expression linéaire, on utilise donc la formule de propagation donnée plus haut. Pour les bornes de B_k, qui est quadratique, nous avons plusieurs possibilités. Par défaut, on utilise la méthode de [72] où les termes bilinéaires sont remplacés par le produit des intervalles et où les termes quadratiques et linéaires sont regroupés ensembles. Une fois ceci fait, notre contrainte quadratique prend la forme suivante :

$$L \leq q_{kk} x_k^2 + [\underline{A_k}, \overline{A_k}] x_k + [\underline{B_k}, \overline{B_k}] \leq U.$$

En utilisant des bornes de propagation et d'inférence optimales sur les trinômes à coefficients intervalles (voir par exemple [72] ou [160]), on peut calculer une borne de propagation pour l'expression quadratique et une borne d'inférence pour x_k. On termine en prenant les meilleures bornes de propagation parmi toutes celles obtenues.

4.2.2 Implémentation : aspects algorithmiques

La FBBT est constituée de deux phases. La première est la phase d'initialisation. Celle-ci est exécutée depuis l'extérieur puisque selon le contexte, on peut vouloir resserrer les bornes à partir de toutes les variables et contraintes (par exemple avant le *root node*), ou alors uniquement à partir d'une variable (par exemple après une décision de branchement). Une fois que la phase d'initialisation à communiqué les bornes qui ont changé, la seconde phase commence. Celle-ci s'appelle la phase de propagation.

Lors de la propagation, on visite les contraintes du modèle pour y appliquer la propagation et l'inférence de bornes, jusqu'à ce qu'un critère d'arrêt soit vérifié. Celui-ci contrôle le temps d'exécution et détermine quand le point fixe est atteint. La performance de la FBBT, c'est-à-dire le temps d'exécution de la propagation, ainsi que la qualité des bornes finales, dépend de plusieurs facteurs. Nous allons présenter maintenant quelques points-clés importants pour une implémentation performante.

Évaluation des formules d'inférence des sommes

Une amélioration importante de la performance de la FBBT est obtenue en s'intéressant à la façon dont sont évaluées les formules de propagation et d'inférence des sommes (linéaires et quadratiques). Par exemple pour une expression linéaire, si $a_i x_i$ est un terme de l'expression avec $a_i > 0$, les bornes inférées sur x_i sont

$$x_i \in \left[\frac{L-b}{a_i} - \sum_{j \neq i} \max\left(\frac{a_j}{a_i}\ell_i, \frac{a_j}{a_i}u_i \right), \frac{U-b}{a_i} - \sum_{j \neq i} \min\left(\frac{a_j}{a_i}\ell_i, \frac{a_j}{a_i}u_i \right) \right],$$

on se rend compte que l'intervalle inféré sur x_i est défini par une somme de taille $m-1$, m étant la taille de la contrainte. Générer naïvement ces intervalles sur chaque variable a donc une complexité quadratique en la taille de la contrainte. Pour résoudre ce problème on utilise les bornes de propagation de l'expression :

$$[L_p, U_p] = b + \sum_i a_i * [\ell_i, u_i].$$

Les formules d'inférence peuvent alors se réécrire en

$$x \in [L, U] - \frac{1}{a_i}\Big([L_p, U_p] \ominus (a_i * [\ell_i, u_i]) \Big),$$

où l'opérateur $\ominus$ n'est pas la soustraction classique des intervalles, mais l'opérateur qui retire la participation d'un intervalle dans une somme contenant ce dernier (voir le paragraphe sur l'arithmétique d'intervalle pour les formules). Cette formule s'évalue en temps constant et l'inférence sur l'expression linéaire a finalement une complexité linéaire en la taille de l'expression.

Pour les expressions quadratiques, la situation est similaire. On définit des bornes de propagation auxiliaires

$$[L_a, U_a] = \sum_k \texttt{Trinom}([\ell_k, u_k], q_{kk}, c_k) + \sum_{i \neq j} q_{ij} * [\ell_i, u_i] * [\ell_j, u_j],$$

où `Trinom` est la fonction qui calcule l'image d'un intervalle par un trinôme. À partir de ces bornes, obtenir pour chaque variable x_k la contrainte de la forme

$$L \leq q_{kk}x_k^2 + [\underline{A_k}, \overline{A_k}]x_k + [\underline{B_k}, \overline{B_k}] \leq U$$

se fait en m opérations, m étant le nombre de termes où x_k apparaît. Pour ce faire, on initialise $[\underline{B_k}, \overline{B_k}] = [L_a, U_a]$ puis on visite chacun de ces termes. On retire la participation du terme dans $[\underline{B_k}, \overline{B_k}]$ et ajoute celle dans q_{kk} et $[\underline{A_k}, \overline{A_k}]$. Finalement, on visite chaque non-zéro de l'expression deux fois pour obtenir toutes les bornes de propagation et d'inférence, soit encore une complexité linéaire.

Calcul incrémental des bornes de propagation des sommes

Les bornes auxiliaires utilisées pour accélérer l'inférence dans le paragraphe précédent sont aussi des bornes de propagation. Dans le cas linéaire, ce sont les bornes optimales et elles sont aussi utilisées pour la propagation. Dans le cas quadratique, elles sont aussi utilisées pour accélérer l'évaluation d'une famille de bornes de propagation plus serrées. Il est possible de calculer ces bornes lors de la visite des sommes : une première passe sur les non-zéros de la contrainte pour obtenir les bornes auxiliaires, une seconde pour en déduire les bornes de propagation et d'inférence.

Ceci peut être amélioré en maintenant à jour ces bornes de façon incrémentale, au fur et à mesure que les bornes des variables sont resserrées. Ceci à plusieurs intérêts : on ne fait qu'une seule passe sur les sommes lors de leur visite par la FBBT, si une seule variable est resserrée entre deux visites, la mise à jour est plus rapide que le recalcul, le démarrage à chaud de la FBBT est plus rapide puisque l'on peut mettre à jour les bornes auxiliaires au lieu de les calculer et, enfin, l'inconsistance est détectée plus tôt. On rappelle que les bornes de propagation des contraintes linéaires et quadratiques ne sont utiles que pour détecter l'inconsistance.

On initialise donc toutes les bornes auxiliaires une fois, au début, et on les maintient à jour. Pour cela, à chaque fois qu'une variable x_i est réduite de $[\ell_i, u_i]$ vers $[\ell'_i, u'_i]$, on visite la participation de x_i dans chaque contrainte linéaire et quadratique. Lors de cette visite, on remplace la participation de $[\ell_i, u_i]$ par celle de $[\ell'_i, u'_i]$ dans les bornes auxiliaire. Par exemple si x_i apparaît avec un coefficient a_i dans une expression linéaire, on met à jour les bornes de l'expression avec la formule suivante :

$$[L'_p, U'_p] = [L_p, U_p] \ominus a_i * [\ell_i, u_i] + a_i * [\ell'_i, u'_i].$$

Le temps de mise à jour de toutes les bornes des sommes où x_i apparaît est proportionnel au nombre de termes où x_i apparaît dans ces expressions.

Ordre de visite des contraintes

Les améliorations que nous venons de présenter ne changent pas le résultat de la FBBT : les réductions de bornes obtenues ne sont pas impactées. Le temps d'exécution de la FBBT dépend alors de deux facteurs : l'ordre de visite des contraintes et les conditions sous lesquelles on décide de les visiter. On commence par s'intéresser à l'ordre des contraintes.

La FBBT est généralement implémentée sur le DAG du problème. C'est l'approche retenue par la plupart des solveurs d'optimisation globale, comme Couenne [36] et SCIP [160]. Ceci permet d'alterner entre des phases de propagation, ou *bottom-up*, et des phases d'inférence, ou *top-down*. Lors de la propagation, on visite par rang croissant les nœuds dont les bornes ont été resserrées lors de la phase précédente. Lors de l'inférence, on visite par rang décroissant les nœuds dont les bornes des parents ont été resserrées lors de la phase précédente.

Dans notre cas, la structure du DAG n'est pas conservée, et on travaille directement sur le modèle factorisé. En factorisant le modèle, nous avons transformé les arcs du DAG en contraintes dans le modèle factorisé. De plus, l'alternance propagation/inférence n'est pas la meilleure approche, même dans un DAG (voir par exemple

[161]). On définit alors des objets de base à propager : les variables et les sommes (linéaires et quadratiques). Propager une variable consiste à visiter chaque couplage non linéaire où la variable apparaît. Propager une somme consiste à y appliquer l'inférence de bornes. Ces objets sont stockés dans une file de priorité, qui suit les règles suivantes :

— les variables sont prioritaires sur les contraintes linéaires et quadratiques,
— les variables sont traitées avec le principe *first in, first out*,
— les contraintes linéaires et quadratiques sont traitées avec le principe *last in, last out*.

Enfin, la FBBT consiste à propager le premier objet de la file jusqu'à ce que celle-ci devienne vide. L'utilisation de priorités différentes pour les variables et pour les sommes a un impact majeur sur la performance. Sur les instances comportant des sommes de grande taille, ces gains peuvent être arbitrairement grands, et dépasse souvent un facteur 10, 100, voire 1000.

Critères de propagation

La séquence de contraintes visitées par la FBBT peut être arbitrairement longue, puisque l'on n'a pas nécessairement la convergence en temps fini. L'exemple classique [33] qui montre ceci est

$$\min_{x,y} f(x,y)$$
$$\text{s.c.} \begin{cases} x, y \in [0,1], \\ x = ay, \\ y = ax, \end{cases}$$

avec $a > 1$. Il est prouvé qu'après une avoir visité successivement les deux contraintes k fois, les bornes des variables sont réduites à $x \in [0, 1/a^{2k+1}]$ et $y \in [0, 1/a^{2k}]$. Le seul point admissible est $[0, 0]$, mais on ne l'atteint pas en temps fini.

Notre objectif est d'obtenir des critères permettant de détecter que l'on a approximativement atteint le point fixe, et de limiter le temps d'exécution de la propagation. Il faut trouver un compromis entre cet objectif et la détérioration des réductions de bornes obtenues si l'on interrompt la propagation avant le vrai point fixe. Un tel équilibre, dans le cas de la programmation linéaire, est obtenu dans [17]. Nous améliorons et généralisons l'approche dans le cas de notre modèle factorisé. Nous nous intéressons dans ce paragraphe aux conditions sous lesquelles on ajoute un objet à la file de propagation, puisque la FBBT s'interrompt quand cette dernière devient vide.

On considère ici un objet (variable ou somme) dont on vient de prouver que les bornes (dans le cas des expressions, les bornes de propagation) ont été réduites de $[\ell, u]$ vers $[\ell', u']$. Une première idée est de ne propager l'objet que si la réduction des bornes est suffisante, par exemple si

$$|u' - \ell'| \leq (1 - \gamma)|u - \ell|,$$

avec $0 < \gamma \ll 1$. Ce critère n'est pas suffisant, puisque sur l'exemple précédent, si $1/a^2 < 1 - \gamma$, il n'est jamais vérifié. En fait, une réduction relative d'un intervalle

doit toujours être complétée par une réduction absolue minimale. On utilise donc plutôt :

$$\begin{aligned} \ell' - \ell &\geq \varepsilon + \gamma \min\left(1 + |\ell|, |u - \ell|\right) \text{ ou} \\ u - u' &\geq \varepsilon + \gamma \min\left(1 + |u|, |u - \ell|\right). \end{aligned}$$

Typiquement, on utilise $\gamma = 10^{-6}$ et $\varepsilon = 10^{-16}$. Dans le cas des variables, on propage aussi si le signe d'une des bornes change : $\operatorname{sign} \ell \neq \operatorname{sign} \ell'$ ou $\operatorname{sign} u \neq \operatorname{sign} u'$. L'intérêt est que la relaxation convexe, la propagation et l'inférence de certains couplages, par exemple $xy = 1$ ou $z = xy$, sont plus précises si le signe des variables est déterminé. De plus, un changement de signe ne pouvant arriver que deux fois par variable, le surcoût dans le pire cas est limité.

Résumé de l'algorithme

L'algorithmique de la phase de propagation de la FBBT peut être résumé par les deux fonctions suivantes. Premièrement, la fonction qui visite les contraintes jusqu'à atteindre le point fixe est donnée dans l'algorithme 7.

Algorithme 7 propagation de la FBBT

```
1 : fonction PROPAGATE()
2 :     tant que la file de propagation n'est pas vide faire
3 :         Prendre le prochain objet à propager ;
4 :         si cet objet est x_i alors
5 :             pour tous les couplages dans lesquels x_i apparaît faire
6 :                 Appliquer la propagation à ce couplage ;
7 :                 Appliquer l'inférence à ce couplage ;
8 :                 pour tout x_j ainsi réduit de [ℓ, u] vers [ℓ', u'] faire
9 :                     notifyReducedVariable(j, [ℓ, u], [ℓ', u']) ;
10 :                fin pour
11 :            fin pour
12 :        sinon
13 :            Appliquer l'inférence à la contrainte ;
14 :            pour tout x_j ainsi réduit de [ℓ, u] vers [ℓ', u'] faire
15 :                notifyReducedVariable(j, [ℓ, u], [ℓ', u']) ;
16 :            fin pour
17 :        fin si
18 :    fin tant que
19 : fin fonction
```

Ensuite, lorsqu'une variable est réduite, on utilise la fonction de notification donnée dans l'algorithme 8. À la ligne 7, on dit que le domaine d'une contrainte c est vide si les bornes naturelles (celles imposées par le fait que l'on a une contrainte

$c = 0$ ou $c \leq 0$) et les bornes de propagation (celles induites par les bornes des variables) sont d'intersection vide.

Algorithme 8 Fonction de réduction des bornes d'une variable et de propagation de cette réduction

```
1 : fonction NOTIFYREDUCEDVARIABLE(i, [ℓ, u], [ℓ', u'])
2 :     si x_i ∈ ℤ, [ℓ', u'] ← [⌈ℓ'⌉, ⌊u'⌋] ;
3 :     si la réduction de bornes est suffisante alors
4 :         pour toutes les sommes s où x_i apparaît faire
5 :             Retirer la contribution de [ℓ, u] aux bornes de propagation de s ;
6 :             Ajouter la contribution de [ℓ', u'] aux bornes de propagation de s ;
7 :             si le domaine de s est vide alors
8 :                 Déclarer le problème inconsistant ;
9 :             fin si
10 :            si la réduction des bornes de s est suffisante alors
11 :                si s n'est pas déjà dans la file de propagation alors
12 :                    Ajouter s à la fin de la file ;
13 :                fin si
14 :            fin si
15 :        fin pour
16 :        si x_i n'est pas déjà dans la file de propagation alors
17 :            Ajouter x_i au début de la file ;
18 :        fin si
19 :    fin si
20 : fin fonction
```

4.2.3 Implémentation : aspects numériques

La FBBT est utilisée de façon intensive tout au long de la recherche. En particulier, les autres techniques de réduction de bornes y font appel pour propager les réductions obtenues. Si un certain sous-problème est déclaré inconsistant à tort, ou si la solution optimale est coupée des bornes par la FBBT, les bornes inférieures obtenues peuvent être fausses.

Les calculs utilisés pour effectuer la propagation et l'inférence sont implémentés à l'aide de fonctions d'arithmétique d'intervalle. Ceci permet de les rendre robustes aux erreurs numériques d'arrondi. La façon dont on s'affranchit de ces erreurs dépend cependant des opérations concernées.

Le contrôle des erreurs d'arrondi, et plus généralement des erreurs numériques, dans la propagation et l'inférence représente la principale difficulté à implémenter l'ensemble des techniques de réduction de bornes présentées dans ce chapitre. Cette difficulté est souvent évoquée, mais rarement décrite en détail. Une fois de plus,

pour que ce manuscrit rapporte fidèlement le travail effectué et puisse servir de référence au code implémenté, nous détaillons les précautions à prendre pour une implémentation stable et robuste de la FBBT.

Arithmétique élémentaire

On appelle arithmétique classique les opérations élémentaires $+, \times, -, \div, .^2, \sqrt{}$. Quelques exemples de ces opérations sont résumés dans le tableau 4.3. L'implémentation de ces opérations sur des processeurs modernes utilise ce que l'on appelle l'arrondi directionnel, ou *directed rounding*. Il est ainsi possible de régler le processeur pour que tout calcul ne tombant pas juste en écriture binaire soit arrondi de la façon demandée par l'utilisateur. Ces sens d'arrondi sont :

— arrondi au plus proche,
— arrondi par excès (vers $+\infty$),
— arrondi par défaut (vers $-\infty$),
— arrondi vers 0.

Pour des calculs sur des intervalles, on aimerait utiliser des arrondis par excès pour la borne supérieure, ainsi que des arrondis par défaut pour la borne inférieure. Malheureusement, changer le mode d'arrondi est une opération coûteuse. On utilise donc toujours le sens d'arrondi par excès, et on exploite le fait que prendre l'opposé d'un nombre permet de passer d'un arrondi à l'autre sans introduire d'erreurs supplémentaires. Par exemple, voici l'écriture de quelques opérations d'arithmétique d'intervalle élémentaires si le sens d'arrondi est par excès :

$$\begin{aligned}
[a,b] + [c,d] &= [-(-a-c), b+d]\,, \\
a * [b,c] &= [-((-a)b), ac]\,, \text{ si a } > 0, \\
[a,b] * [c,d] &= [-\max(-((-a)c), -((-a)d), -((-b)c), -((-b)d), \max(ac,ad,bc,bd)]\,, \\
1/\,[a,b] &= [-1/(-b), 1/a]\,, \text{ si a } > 0.
\end{aligned}$$

Dans ces formules, certaines expressions sont simplifiables. Il faut donc régler le compilateur pour qu'il ne les modifie pas, sous peine d'obtenir des résultats faux.

Arithmétique transcendante

La politique d'arrondi du processeur permet d'effectuer de façon exacte la plupart des opérations décrites dans l'inférence et la propagation. L'implémentation standard de certaines fonctions ne supporte cependant pas la modification du sens d'arrondi, et retourne un résultat arrondi de façon indéfinie. C'est le cas des fonctions exp et log, des fonctions trigonométriques sin, cos et tan, ainsi que de la fonction puissance.

Dans ce cas, on utilise deux opérateurs qui permettent de naviguer sur la grille des nombres représentables : `succ` et `pred`, représentant le successeur et le prédécesseur d'un nombre. On node $\mathcal{D}$ l'ensemble des réels représentables, ainsi que $\mathrm{f}_{\mathcal{D}}$ l'opérateur f arrondi sur $\mathcal{D}$ (celui implémenté dans le processeur). Pour les fonctions $f \in \{\exp, \log, \sin, \cos, \tan, \mathrm{pow}\}$, on a la seule garantie que

$$\mathrm{f}(x) \in [\mathrm{pred}(\mathrm{f}_{\mathcal{D}}(x)), \mathrm{succ}(\mathrm{f}_{\mathcal{D}}(x))].$$

Opération	Sous-cas	Résultat
$a * [b, c]$	$a \geq 0$	$[ab, ac]$
	$a < 0$	$[ac, ab]$
$[a, b] + [c, d]$	-	$[a + c, b + d]$
$[a, b] * [c, d]$	-	$[\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$
$1/[a, b]$	$a \geq 0$	$[1/b, 1/a]$
	$b \leq 0$	$[1/b, 1/a]$
	$0 \in]a, b[$	$\mathbb{R}$
$[a, b]^2$	$a \geq 0$	$[a^2, b^2]$
	$b \leq 0$	$[b^2, a^2]$
	$0 \in]a, b[$	$[0, \max(a^2, b^2)]$
$[a, b] \ominus [c, d]$	-	$[a - c, b - d]$

Tableau 4.3 – *Opérations élémentaires d'arithmétique d'intervalle*

On utilise alors les opérateurs `succ` et `pred` afin de garantir que les erreurs d'arrondi ne puissent pas introduire d'erreurs dans les intervalles renvoyés. La situation est illustrée dans la figure 4.3. Finalement, dans le cas de la fonction exponentielle par exemple, l'image d'un intervalle est donnée par la formule :

$$\exp([a, b]) = [\texttt{pred}(\exp_{\mathcal{D}} a), \texttt{succ}(\exp_{\mathcal{D}} b)] \,.$$

La situation est similaire pour les autres opérateurs, mais les calculs peuvent être plus complexes. Dans le cas des fonctions trigonométriques, on a en plus une disjonction de cas sur les signes de f et de f' pour obtenir l'image exacte d'un intervalle. Enfin, dans le cas des puissances, par exemple $y = x^a$, il faut faire attention au sens d'arrondi de $1/a$ pour l'inférence $x = y^{1/a}$, puisque le sens de variation de $b \mapsto y^b$ n'est pas constant.

Bornes sous-optimales

L'utilisation d'arrondis vers l'extérieur permet de rendre les calculs de bornes robustes aux erreurs numériques. Un problème intervient cependant lorsque l'on applique ceci aux bornes de propagation et d'inférence des sommes. En effet, l'utilisation de bornes auxiliaires pour accélérer l'inférence augmente le nombre d'expressions intermédiaires dans les formules. Puisque chaque expression intermédiaire introduit une erreur numérique vers l'extérieur, l'erreur finale est plus importante. Par exemple, si $x \in [0, 1]$, $y \in [-10^{20}, 10^{20}]$ et $x + y = 1$, les formules naïves donnent bien $y \in [0, 1]$ alors que les formules utilisant les bornes de propagation donnent (environ) $y \in [-10^4, 1]$.

Ce constat est aussi fait dans [72], où la solution proposée est l'utilisation de

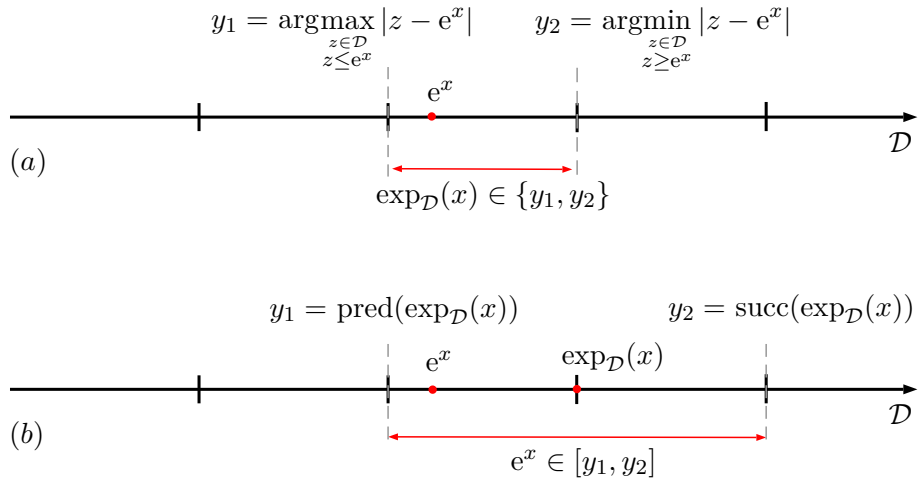

FIGURE 4.3 – (a) *valeurs possibles de* $\exp_{\mathcal{D}}(x)$*, quelque soit le mode d'arrondi et* (b) *intervalle contenant la valeur réelle calculée à partir de la valeur arrondie et des opérateurs* `succ` *et* `pred`*.*

formules plus complexes mais plus précises, faisant intervenir les indices des termes ayant les plus grandes bornes. Le maintien à jour des bornes de propagation nous donne une autre solution à ce problème. En effet, on est capable de détecter les situations où, comme dans l'exemple précédent, certaines bornes sont probablement sous-optimales. L'idée est que les erreurs d'arrondi, même toujours sommées dans le même sens, sont négligeables par rapport à la taille des bornes (l'erreur relative sur une opération élémentaire est de l'ordre de 10^{-16}). Cependant, si les bornes d'une certaine somme ont été significativement réduites, les erreurs d'arrondi introduites plus tôt peuvent ne plus être négligeables par rapport aux bornes actuelles. Ainsi, lors d'un calcul complet des bornes de propagation, une sauvegarde de ces bornes est effectuée. Lors de chaque mise à jour incrémentale de ces bornes, celles-ci sont comparées à leur sauvegarde. Si la taille de l'intervalle défini par ces bornes a été réduit de façon importante par rapport à la sauvegarde (en pratique, d'un facteur 1000), un calcul complet des bornes est relancé, suivi d'une nouvelle sauvegarde. Cette solution est moins efficace que les formules alternatives utilisées dans [72] dans des cas mal conditionnés, mais est souvent suffisante, comme dans l'exemple précédent. Elle peut aussi être plus efficace, en particulier quand les problèmes ne viennent pas uniquement d'un unique terme de grande magnitude. Elle a aussi l'avantage de ne pas détériorer la performance de l'inférence sur les sommes par l'utilisation de formules plus complexes.

Autres erreurs numériques

Si toutes les erreurs numériques ont pour origine des calculs inexacts sur les nombres réels, certains problèmes se présentant lors de la propagation et de l'inférence trouvent leur origine dans d'autres modules du solveur.

Une première source d'erreur est l'ensemble des modifications apportées au modèle lors de la factorisation et du *presolve*. Par exemple, quand on effectue une

substitution $x_i \leftarrow ax_j + b$ dans une expression linéaire, les coefficients modifiés peuvent devenir faux s'ils ne sont pas remplacés par des intervalles. Une fois dans la FBBT, ces erreurs peuvent se traduire par des affirmations fausses, typiquement de la forme $-10^{-14} \geq 0$. Si des précautions ne sont pas prises, le problème est alors déclaré inconsistant.

Le même type d'erreur peut venir des bornes (primales ou duales) renvoyées par les solveurs, y-compris par la recherche locale, en particulier à cause les tolérances sur les conditions d'optimalité et de faisabilité. Par exemple, la solution $x = -\varepsilon_{\text{feas}}$ est déclarée admissible pour le problème

$$\min_{x \in \mathbb{R}^n} x$$
$$\text{s.c.} \begin{cases} 0 \leq x \leq 1, \\ x = 0. \end{cases}$$

Une borne primale devient $-\varepsilon_{\text{feas}}$, qui permet d'inférer $x \leq -\varepsilon_{\text{feas}}$. Le problème est finalement déclaré inconsistant une fois ceci propagé aux bornes de x.

Comme expliqué en introduction, nous ne souhaitons pas rendre le solveur robuste à toutes les erreurs numériques. Les méthodes exactes ont leur intérêt, mais ne sont pas pertinentes dans notre contexte industriel. Il n'y alors pas de solution universelle aux problèmes que nous venons de décrire. Pour s'assurer que des erreurs numériques de faible amplitude n'interfèrent pas avec la propagation et l'inférence de bornes, un certain nombre de réglages numériques ont été effectués. Par exemple, les informations venant des solveurs sont systématiquement protégées par des tolérances relatives, et les critères de déclaration d'inconsistance sont assouplis. Par exemple, si $[L, U]$ sont les bornes de propagation d'une égalité linéaire, on n'utilise pas $[L, U] \cap \{0\} = \emptyset$ pour déclarer l'inconsistance, mais plutôt $[L, U] \cap [-\varepsilon_{\text{feas}}(1 + |b|), \varepsilon_{\text{feas}}(1 + |b|)] = \emptyset$, où b est le coefficient constant de la contrainte. Des précautions similaires sont prises lors des réductions de bornes des variables.

Performance et intérêt de la FBBT

Dans [33], il est remarqué que la durée d'exécution de la FBBT, appliquée au DAG du problème, peut être longue. L'approche utilisée pour réduire le temps d'exécution est cependant différente. La FBBT est appliquée à une relaxation linéaire du problème. Il est ensuite montré que dans ce cas, le point fixe peut être atteint en résolvant un unique programme linéaire bien choisi. La structure de ce programme est étudiée pour en réduire la taille. Enfin, en itérant génération de relaxation linéaire et résolution d'un certain programme linéaire, le point fixe de la FBBT est atteint significativement plus rapidement qu'avec une propagation et inférence dans le graphe. Les temps d'exécutions des problèmes les plus pathologiques de la MINLPLib sont reportés, et varient de 2 à 218 secondes. En moyenne, l'approche par point fixe et résolution de programmes linéaires apporte une accélération d'un facteur 50. Les temps d'exécutions de notre approche, appliquant la FBBT au modèle factorisé, varient sur ces instances de 0.6ms à 40ms, pour des facteurs d'accélération variant de 1000 à 10000. L'améliorations du matériel en six ans n'explique pas, à lui seul,

une telle différence. La même remarque s'applique par comparaison avec la propagation de LocalSolver, effectuée sur le DAG lors du *preprocessing*. Nous en concluons qu'une implémentation soignée de la FBBT sur la forme factorisée du modèle est significativement plus performante que son analogue sur le graphe.

La FBBT est un outil utilisé par toutes les autres techniques de réduction de bornes. Elle est aussi utilisée après la factorisation du modèle, afin d'obtenir des bornes finies sur toutes les variables. En ce sens, elle est indispensable au solveur d'optimisation globale. Nous mesurons quand même l'impact de la propagation de la variable réduite après une décision de branchement. Le *benchmark* considéré est celui composé des 2211 instances de plus petite taille, sur lequel 1431 instances sont résolues avec les réglages de LocalSolver par défaut. L'impact sur les modèles résolus avec et sans la propagation sur le nombre de nœuds est donné dans la figure 4.4, et est globalement positif. Sans la FBBT post-branchement, 5.7% d'instances de moins sont résolues, soit plus du double de l'impact des autres techniques de réduction de bornes. Sur les instances non résolues, on mesure aussi l'impact la FBBT post-branchement sur la borne finale obtenue. Cet impact est résumé dans le tableau 4.4. Comme n'importe quel composant, la FBBT est parfois bénéfique, parfois délétère. Le tableau 4.4 place cependant la FBBT parmi les composants les plus importants, et son activation permet de résoudre 5.7% d'instances supplémentaires.

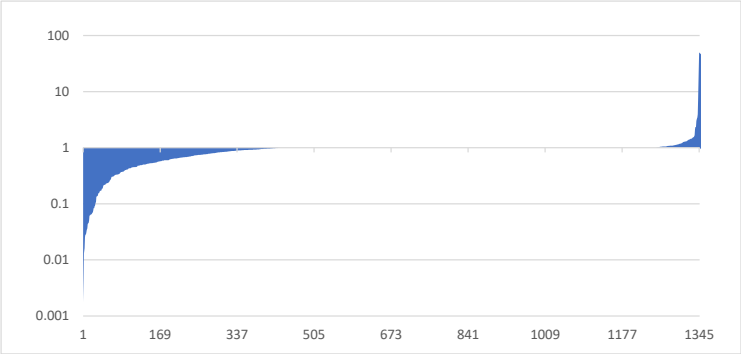

FIGURE 4.4 – *Pour les instances résolues avec ou sans la propagation et l'inférence de bornes consécutives aux décisions de branchement, impact de celles-ci sur le nombre de nœuds nécessaires à la résolution.*

Impact de la FBBT post-branchement sur la borne finale	Proportion des instances
Détérioration supérieure à 10%	2.2%
Détérioration supérieure à 1%	7.9%
Détérioration inférieure à 1%	22%
Pas de changement de la borne	25.1%
Amélioration inférieure à 1%	25.5%
Amélioration supérieure à 1%	14.5%
Amélioration supérieure à 10%	10.7%

Tableau 4.4 – *Pour les instances que ni la stratégie par défaut, ni la stratégie sans FBBT post-branchement n'ont résolues, comparaison des bornes inférieures finales.*

4.3 Techniques de réduction de bornes

4.3.1 Utilisation du *probing*

Principe

La FBBT ne capte pas la géométrie de l'ensemble admissible, définie par la conjonction des contraintes, puisqu'elle analyse ces dernières une à une. Une technique permettant d'améliorer les réductions de bornes obtenues est de partitionner le domaine des variables, d'appliquer la FBBT à chaque partie, puis à en déduire des informations valides sur le problème initial. Par exemple, sur le problème

$$\min_{x,y} f(x)$$
$$\text{s.c.} \begin{cases} 0 \leq x \leq 1000, \\ 0 \leq y \leq 1000, \\ x + y = 1, \\ x \leq y, \end{cases}$$

la FBBT permet de resserrer les bornes de (x, y) à $[0, 1]^2$. En partitionnant le domaine de x en $[0, 1/2] \cup]1/2, 1]$ et en appliquant la FBBT au sous-domaine $x \in]1/2, 1]$, on montre que celui-ci est inconsistant. Dans ce cas, on a réduit le domaine de x à $[0, 1/2]$, puis celui de y à $[1/2, 1]$ en propageant cette réduction.

Cette approche s'appelle le *probing*, et permet d'améliorer n'importe quelle technique de réduction de bornes. Les informations obtenues peuvent être des réductions supplémentaires, si certaines parties sont inconsistantes, ou des relations d'implications entre variables. Historiquement utilisé en programmation par contraintes, le *probing* a été introduit en optimisation dans le contexte des programmes linéaires en nombres entiers [149], en particulier pour améliorer le traitement des variables booléennes. Dans l'exemple précédent, si x ou y sont booléens, le *probing* permet de résoudre entièrement le problème. En optimisation globale, le *probing* est utilisé sur les variables entières, mais aussi sur celles apparaissant dans des termes non linéaires [155].

Formellement, on se donne une variable $x_i \in [\ell_i, u_i]$ et on cherche $\alpha \in]\ell_i, u_i[$ tel que $[\ell_i, \alpha]$ ou $[\alpha, u_i]$ est prouvé inconsistant par la FBBT. Dans certains solveurs [160, 36], le *probing* n'est effectué que si une solution primale a été trouvée. En effet, la connaissance d'un point admissible facilite la recherche d'un tel α (si un sous-domaine contient une solution admissible, il ne sera pas prouvé inconsistant). Puisque l'on se concentre sur la partie bornes inférieures de l'optimisation globale, on veut utiliser le *probing* même en l'absence de solution. Une façon de chercher un point de partitionnement satisfaisant sans la connaissance d'une solution primale est d'appliquer une dichotomie, dont on limite le nombre d'itérations. Le prototype de l'algorithme de *probing* cherchant à réduire u_i est donné dans l'algorithme 19.

Algorithme 9 Fonction de *probing* de la borne suppérieure de x_i.

```
1 : fonction probeUpperBound(i)
2 :     Réduire temporairement x_i de [ℓ_i, u_i] à [u_i, u_i] ;
3 :     Propager cette réduction avec la FBBT
4 :     si la FBBT a prouvé l'inconsistance alors
5 :         count ← 0 ;
6 :         α ← (u_i + ℓ_i)/2 ;
7 :         tant que count ≤ maxSteps faire
8 :             Réduire temporairement x_i de [ℓ_i, u_i] à [α, u_i] ;
9 :             Propager cette réduction avec la FBBT ;
10 :            si la FBBT à prouvé l'inconsistance alors
11 :                Réduire définitivement x_i de [ℓ_i, u_i] à [ℓ_i, α] ;
12 :                retourner ;
13 :            sinon
14 :                α ← (α + u_i)/2 ;
15 :            fin si
16 :            count++ ;
17 :        fin tant que
18 :    fin si
19 : fin fonction
```

Objectif et contribution

Malgré la performance de la FBBT, appliquer le *probing* naïvement s'avère parfois trop coûteux. De plus, on peut vouloir l'exécuter au *root node* (peut-être plusieurs fois afin d'en atteindre le point fixe) mais aussi durant la recherche arborescente. Peu de détails sont disponibles dans la littérature sur l'implémentation du *probing*. Cette technique est cependant considérée comme faisant partie des techniques trop coûteuses pour être exécutées de façon complète. Nous proposons donc de présenter les améliorations principales que nous avons effectuées par rapport à une implémentation élémentaire. Notre contribution est une implémentation performante du *probing*, qui montre qu'il s'agit en réalité d'une technique moins coûteuse que des opération telles que la résolution de la relaxation linéaire du *root node*. La comparaison du temps d'exécution du *probing* avec celui d'autres modules est donnée dans le dernier chapitre. Le *probing* est appliqué aux variables entières en PLNE, et sur les variables entières ou non linéaires en optimisation globale. Nous avons pour ambition de l'appliquer à toutes les variables du modèle factorisé, qu'elles contribuent aux non-convexités du modèle ou non.

L'objectif de cette partie est de réduire le temps d'exécution tout en maintenant les réductions de bornes obtenues : il s'agit donc d'un équilibre à trouver. Nous présentons les quelques détails d'implémentation dont l'impact est le plus significatif. Le temps d'exécution est mesuré avec une horloge de haute précision, intégrée directement dans le code. Les réductions de bornes sont quant à elles mesurées à partir du volume de l'ensemble admissible $\ell \leq x \leq u$, défini par :

$$\mathrm{vol}(\ell, u) = \prod_{i=1}^{n} \left(\max\left(10^{-8}, u_i - \ell_i\right) \right)^{1/n}.$$

La moyenne géométrique est présente pour empêcher le volume d'atteindre $+\infty$ (environ 2.10^{308} sur un processeur 64 bits) quand la dimension n du problème est grande. De même, le seuil 10^{-8} empêche le volume d'atteindre zéro si une variable devient fixée.

Communication incrémentale

Lors du *probing*, deux problèmes d'optimisation coexistent : le problème initial dont nous voulons réduire les bornes, et le sous-problème dont nous cherchons à prouver l'inconsistance. Ces problèmes ne sont pas uniquement définis à partir des bornes des variables, mais aussi à partir des bornes des contraintes. En effet, lors d'un démarrage à chaud de la FBBT, recalculer les bornes des expressions linéaires et quadratiques peut être plus long que la propagation elle-même. Un problème d'optimisation sur le modèle factorisé est alors défini par un vecteur de bornes $[L, U] = [\ell, u] \cup [L_{\text{lin}}, U_{\text{lin}}] \cup [L_{\text{quad}}, U_{\text{quad}}]$. On note $[L^{\text{glob}}, U^{\text{glob}}]$ les données du problème initial et $[L^{\text{loc}}, U^{\text{loc}}]$ les données du sous-problème en cours.

Ces problèmes communiquent grâce à deux opérations : le *rollback* et le *commit*. Lorsque l'on propage la restriction une variable à un sous-domaine avec la FBBT, il faut ensuite réinitialiser cette dernière avec les données globales afin de continuer le *probing* : c'est le *rollback*. Si le sous-domaine a effectivement été prouvé inconsistant, on a réduit une borne. Une fois cette réduction propagée à l'aide de la FBBT, on applique les changements obtenus aux données globales : c'est le *commit*. Une version naïve de ces deux opérations est donnée dans les algorithmes 10 et 11.

Algorithme 10 *Rollback* naïf.

fonction ROLLBACK()
  pour $i = 1$ to $n_{\text{var}} + n_{\text{lin}} + n_{\text{quad}}$ **faire**
    $L_i^{\text{loc}} \leftarrow L_i^{\text{glob}}$;
    $U_i^{\text{loc}} \leftarrow U_i^{\text{glob}}$;
  fin pour
fin fonction

Algorithme 11 *Commit* naïf.

fonction COMMIT()
  pour $i = 1$ to $n_{\text{var}} + n_{\text{lin}} + n_{\text{quad}}$ **faire**
    $L_i^{\text{glob}} \leftarrow L_i^{\text{loc}}$;
    $U_i^{\text{glob}} \leftarrow U_i^{\text{loc}}$;
  fin pour
fin fonction

Les algorithmes 10 et 11 induisent $O(n_{\text{var}}^2)$ opérations d'écriture pour le *probing*, puisque l'on a au moins deux *rollback* par variable). Cette approche reste viable pour les problèmes de petite et moyenne taille, grâce à la performance d'implémentation de l'opérateur de recopie des vecteurs en C++. Pour les problèmes de grande taille, on peut améliorer la complexité avec le fait que dans la plupart des cas, le support des changements est petit devant la taille des vecteurs de bornes (en général de l'ordre du pourcent, ponctuellement de la dizaine de pourcent).

Nous maintenons donc à jour, dans la FBBT, le support des bornes modifiées lors de la propagation et de l'inférence. On utilise un tas contenant les indices des bornes modifiées, ainsi qu'un vecteur de booléens permettant de ne pas y ajouter deux fois

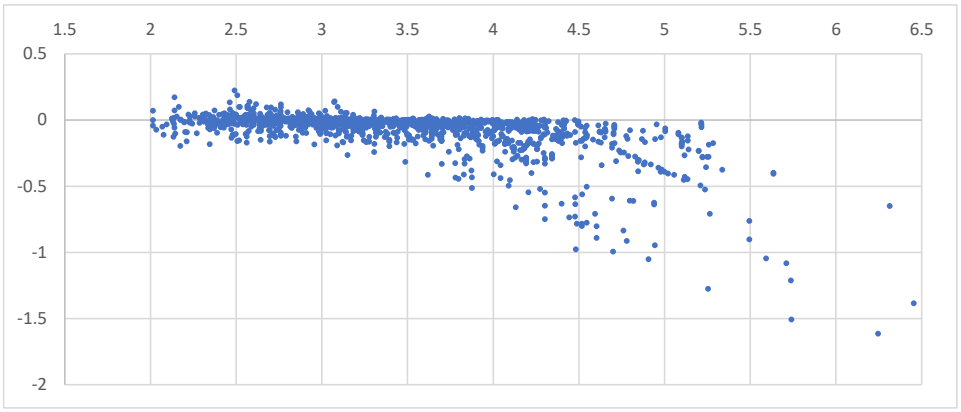

FIGURE 4.5 – *Pour les instances de plus de 100 variables, en double échelle logarithmique, gain relatif sur le temps d'exécution apporté par la communication incrémentale, en fonction du nombre de variables. Un gain de* -0.25 *correspond à une multiplication du temps d'exécution du* probing *par* $10^{-0.25} \simeq 0.56$.

la même borne. L'implémentation du tas n'est pas celle par défaut, car cette dernière utilise une mémoire dynamique. Puisque l'on connaît la taille maximale du tas, on utilise un vecteur de la taille en question ainsi qu'un pointeur vers le haut du tas. L'utilisation de ces structures de données permet de ne recopier que les bornes qui ont changé lors du *rollback* et du *commit*. L'implémentation du *rollback* incrémental est donnée dans l'algorithme 10, celle du *commit* incrémental est analogue.

Algorithme 12 *Rollback* incrémental utilisant le support des changements de borne.

```
1 : fonction IncrementalRollback()
2 :     Soit k le nombre de bornes modifiées de [L^loc, U^loc] ;
3 :     pour i = 1..k faire
4 :         Soit j la i-ème borne modifiée ;
5 :         Marquer [L_j^loc, U_j^loc] comme non modifiée ;
6 :         L_j^loc ← L_j^glob ;
7 :         U_j^loc ← U_j^glob ;
8 :     fin pour
9 :     Remettre à zéro le nombre de bornes modifiées ;
10 : fin fonction
```

L'impact de la communication incrémentale sur le temps d'exécution du *probing* est présenté dans la figure 4.5. L'impact sur les petites instances, jusqu'à 1000 variables, est négligeable. On y observe une certaine variabilité du temps d'exécution, mais celle-ci n'est pas due à des différences de performances (il faut plutôt accuser le système d'exploitation). Pour les instances de grande taille, l'apport peut être significatif, en particulier pour les instances où peu de bornes sont resserrées lors de chaque propagation.

Limitation des effets quadratiques

Après la communication des bornes entre données locales et globales, un autre effet quadratique se cache dans l'algorithme 19. Lorsque l'on initialise la FBBT avec une réduction de borne, celle-ci commence par mettre à jour les bornes auxiliaires des sommes. Si la réduction induite sur l'une d'entre elles est suffisante, la FBBT y appliquera l'inférence. On risque donc d'appliquer $|s|$ fois l'inférence à chaque somme s touchant $|s|$ variables, chaque inférence ayant un coût de $O(|s|)$ opérations. Ainsi, le *probing* a une complexité supérieure à $\sum_{s\in\mathcal{S}} |s|^2$, où $\mathcal{S}$ l'ensemble des contraintes de type somme.

En présence de contraintes globales, cette complexité devient rédhibitoire dès quelques milliers de variables. On cherche alors à la limiter, en restreignant l'inférence aux sommes de petite taille pendant le *probing*. Le calcul des bornes de propagation est maintenu, puisque ce sont elles qui permettent de déclarer l'inconsistance. La figure 4.6 montre l'impact de cette restriction sur le temps d'exécution et sur les réductions de bornes obtenues.

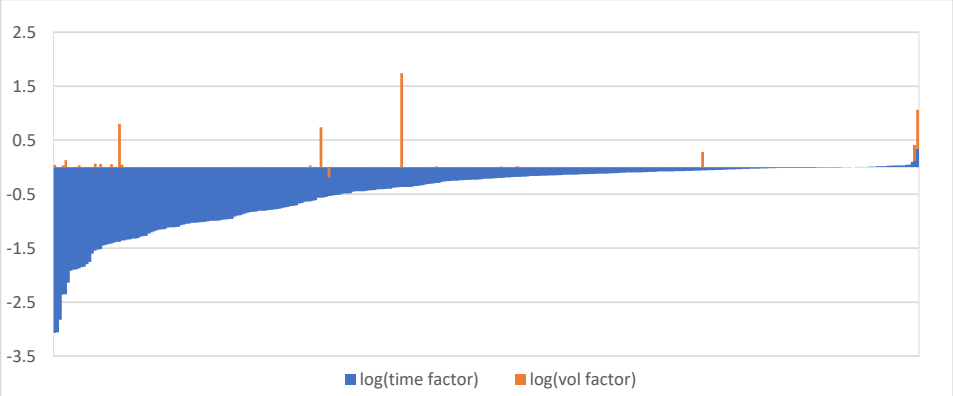

FIGURE 4.6 – *Gains relatifs en temps d'exécution (en bleu, triés par gains décroissants, échelle logarithmique) contre pertes de réductions de bornes (en orange, échelle logarithmique) lorsque l'on limite l'inférence aux sommes de moins de 100 termes lors du* probing. *Seules les instances comportant de telles sommes sont considérées.*

Le bilan de cette modification est positif. Le temps d'exécution du *probing* sur les instances concernées est significativement réduit, au prix d'une détérioration ponctuelle des réductions obtenues. Certaines contraintes globales, comme $\sum_i x_i = 1$ pour des booléens, sont importantes lors de l'inférence. Il semble cependant que dans la plupart des cas, une contrainte touchant beaucoup de variables apporte peu d'information sur chacune d'entre elles. C'est pour cette raison que nous limitons la taille maximale, en utilisant une limite à 100 au lieu de 1000 par exemple.

Contrôle du temps d'exécution de la FBBT

Une fois les principaux problèmes de complexité résolus, on s'intéresse au coût des appels à la FBBT. Notre objectif est d'estimer le nombre de contraintes que

la FBBT a besoin de visiter avant d'atteindre le point fixe. On mesure donc, sur chaque instance, le nombre d'itérations nécessaires à l'obtention du point fixe lors du premier appel à la FBBT, généralement le plus coûteux. On met ensuite en relation ce coût avec la taille des instances, en termes de variables ou de non-zéros. Cette analyse est présentée dans la figure 4.7.

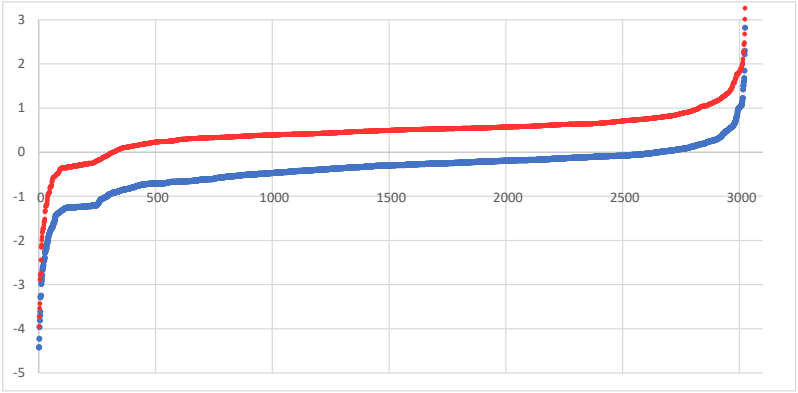

FIGURE 4.7 – *En rouge, logarithmes des rapports itérations sur nombre de variables, triés par ordre croissant. En bleu, logarithmes des rapports itérations sur nombre de non-zéros, triés par ordre croissant. Les deux tris ne sont pas les mêmes, les courbes semblent translatés puisque pour une majorité d'instances,* $n_{\text{nz}} \simeq 5n_{\text{var}}$.

Pour environ 95% des instances, la FBBT converge en moins de $10n_{\text{var}}$ itérations. De même, pour environ 97.5% d'entre elles, $2n_{\text{nz}}$ itérations sont suffisantes. Si ces limites ont peu d'effet lors de la propagation initiale, elles devraient en avoir encore moins pour une propagation démarrée à chaud. On impose donc un nombre maximal de $N = \max(10n_{\text{var}}, 2n_{\text{nz}})$ itérations pour chaque propagation effectuée lors du *probing*. La figure 4.8 montre l'impact de cette limitation sur le nombre total de contraintes visitées lors du *probing* et sur les réductions de bornes obtenues.

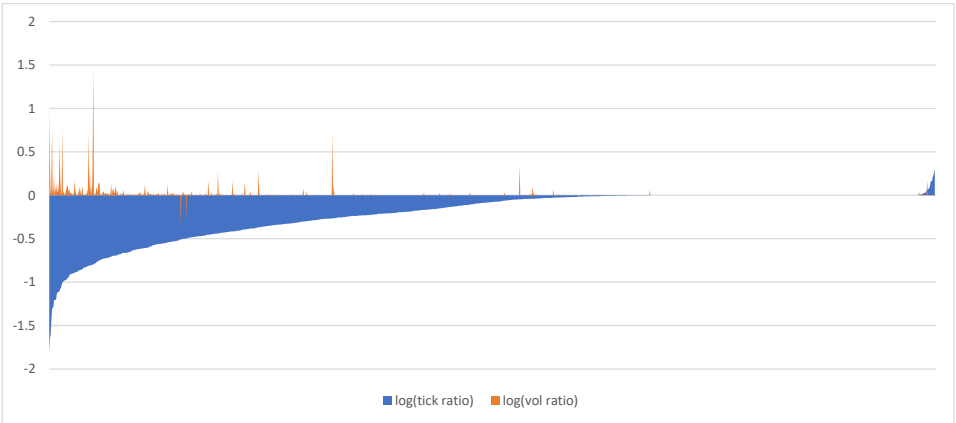

FIGURE 4.8 – *En bleu, gain relatif, en nombre total d'itérations, apporté par la limitation du nombre d'itérations lors d'un appel à la FBBT. En orange, perte relative de réductions, en rapport des volumes de l'ensemble admissible* $\ell \leq x \leq u$.

Une fois de plus, le bilan des modifications effectuées est positif. On observe un gain généralisé en nombre d'itérations pour des pertes limitées de réductions de bornes. Les instances qui bénéficient sont aussi plus nombreuses que prévu (comparer aux 5% et 2.5% impactées par la limitation lors de la propagation initiale). Ceci signifie que lors du *probing*, certaines propagations convergent lentement en visitant cycliquement un sous-ensemble de contraintes, comme sur l'exemple donné dans le paragraphe sur le critère d'arrêt de la FBBT. Enfin, imposer un budget maximal à la FBBT pour chaque propagation permet aussi de réduire la variance du temps d'exécution du *probing*, comme montré sur la figure 4.9.

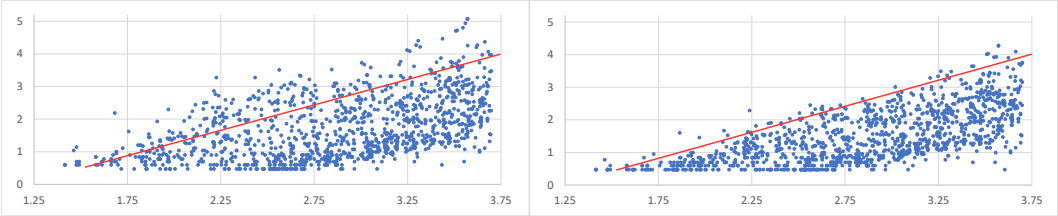

FIGURE 4.9 – *Logarithme du temps d'exécution (en milliseconde) du* probing *en fonction du logarithme de la taille du modèle (en non-zéros), à gauche sans contrôle du temps d'exécution de la FBBT, à droite avec contrôle. Le repère visuel est le même pour les deux réglages. À taille fixée, la variabilité est réduite de 3.5 ordres de grandeur à 2.5.*

***Probing* sur l'objectif**

Dans certains solveurs d'optimisation globale, l'objectif est remplacé par une variable, *via* l'ajout d'une égalité linéaire. Ceci permet d'appliquer le *probing* à l'objectif et d'en améliorer les bornes, pour un coût largement inférieur à la résolution du *root node*. On peut même généraliser le *probing* à toute expression bornée du modèle, mais l'intérêt est limité pour les contraintes. On se contente donc de l'appliquer à l'objectif, qui est de la forme $f(x)$, où f est une fonction linéaire ou quadratique. Les bornes globales peuvent se voir comme une contrainte $f(x) \in [L, U]$, dont on élimine des sous-domaines inconsistants avec la même approche que pour les variables. De plus, les bornes globales obtenues au *root node* et par *probing* ne sont généralement pas comparables. Ceci est du au fait que le *probing* ne travaille pas sur la relaxation convexe, mais sur le problème initial.

Impact du *probing*

Dans notre implémentation, le *probing* s'applique une fois, après les actions de *presolve*. Lorsque le modèle contient plus de 20000 variables, le *probing* n'est appliqué que sur les 20000 premières, afin de rester cohérent avec le temps de référence de la minute. Le *probing* sur l'objectif est quant à lui effectué à chaque nœud. Contrairement aux autres techniques de réduction de bornes, nous limitons l'analyse du *probing* aux instances où celui-ci a resserré au moins une borne lors de l'exécution

initiale (soit un peu plus de la moitié des instances). L'impact sur le nombre de nœuds est donné dans la figure 4.10, sur les instances résolues, et l'impact sur la qualité des bornes retournée est donné dans le tableau 4.5, sur les instances non résolues. Le *probing* permet de résoudre 1.6% d'instances supplémentaires, et améliore les bornes inférieures produites.

Nous mesurons aussi l'impact de l'activation du *probing* complet à chaque nœud. Si cette stratégie est globalement délétère (−6.1% d'instances résolues en moins), les instances résolues le sont avec moins de nœuds, comme montré dans la figure 4.11. Nous avons noté que l'impact du *probing* complet à chaque nœud a souvent un effet similaire sur la taille de l'arbre que l'utilisation de techniques de branchement de type *strong branching*, c'est-à-dire lorsque toutes les décisions possibles sont prises, évaluées, triées, puis la meilleure décision est utilisée.

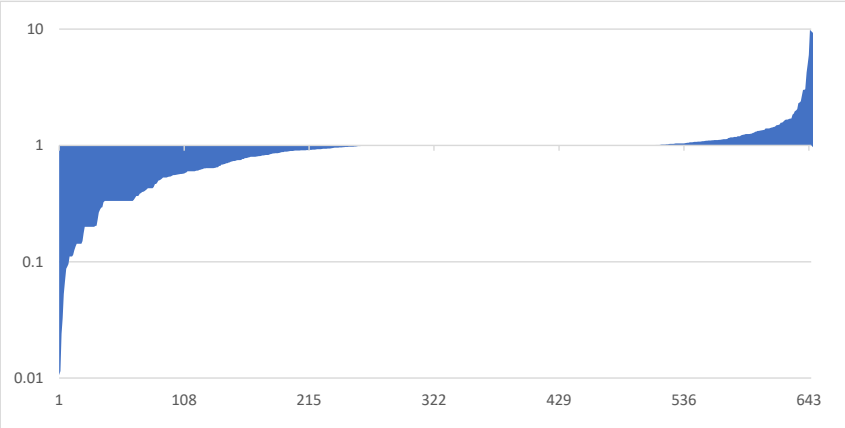

FIGURE 4.10 – *Pour les instances résolues avec ou sans* probing *et où celui-ci réduit au moins une borne, impact de ce dernier sur le nombre de nœuds nécessaires à la résolution.*

Impact du *probing* sur la borne finale	Proportion des instances
Détérioration supérieure à 10%	1.7%
Détérioration supérieure à 1%	7.2%
Détérioration inférieure à 1%	10.3%
Pas de changement de la borne	16.7%
Amélioration inférieure à 1%	15.8%
Amélioration supérieure à 1%	17.8%
Amélioration supérieure à 10%	30.5%

Tableau 4.5 – *Pour les instances que ni la stratégie par défaut, ni la stratégie sans* probing *n'ont résolues et telles que le* probing *a resserré au moins une borne, comparaison des bornes inférieures finales.*

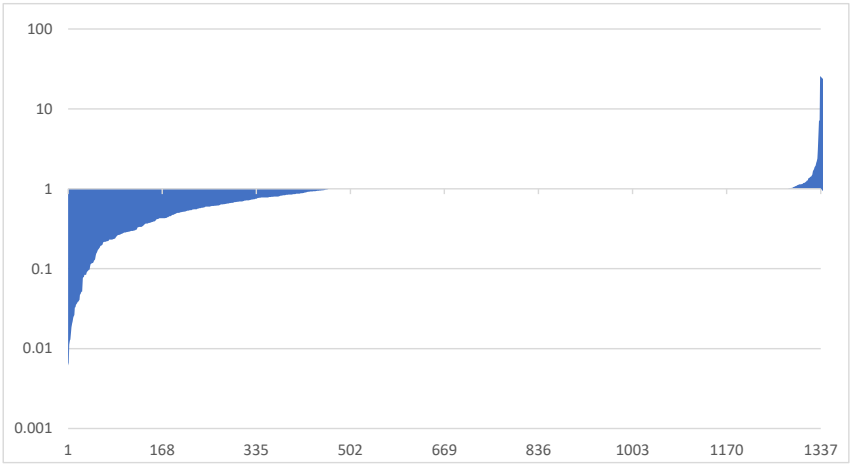

FIGURE 4.11 – *Pour les instances résolues avec et sans le* probing *à chaque nœud, impact de ce dernier sur le nombre de nœuds nécessaires à la résolution.*

4.3.2 Réductions de bornes par optimisation

Principe

Une remarque importante en optimisation globale, qui permet d'obtenir de nombreuses réductions de bornes, est que l'ensemble admissible de toute relaxation convexe contient celui du problème initial. Ainsi, toute réduction de bornes valide sur une relaxation convexe est aussi valide sur le problème initial. On s'intéresse donc, dans cette partie, aux techniques de réduction de bornes pour un problème d'optimisation convexe.

Étant donnée un ensemble convexe $\mathcal{C}$, l'algorithme OBBT, pour *optimization based bound tightening*, consiste à trouver le plus petit domaine $\ell \leq x \leq u$ contenant $\mathcal{C}$ tout entier [36, 138]. On définit alors $\mathcal{C}$ comme la relaxation convexe du problème initial, à laquelle on ajoute les contraintes de bornes globales $L \leq f(x) \leq U$ dans le cas où f est linéaire. Les solutions optimales de ces $2n_{\text{var}}$ problèmes d'optimisation convexe sont des bornes valides pour les variables en question, comme illustré dans la figure 4.12.

Puisque les relaxations convexes dépendent des bornes des variables, l'OBBT est un opérateur itératif convergeant vers un point fixe. Son coût d'exécution est cependant élevé, puisque nous devons résoudre de nombreux problèmes d'optimisation. Pour cette raison, on utilise exclusivement des programmes linéaires, auquel cas $\mathcal{C}$ est un polyèdre, et on ne cherche pas à atteindre le point fixe.

Filtrage des bornes

Le temps d'exécution de l'OBBT est trop important pour qu'elle soit appliquée telle quelle. Dans [84], l'amélioration principale de l'OBBT permettant d'en réduire le temps d'exécution est le filtrage de bornes. Un domaine filtré $\ell^{\text{filter}} \leq x \leq u^{\text{filter}}$ contenant toutes les solutions admissibles de $\mathcal{C}$ est maintenu à jour le long de l'OBBT.

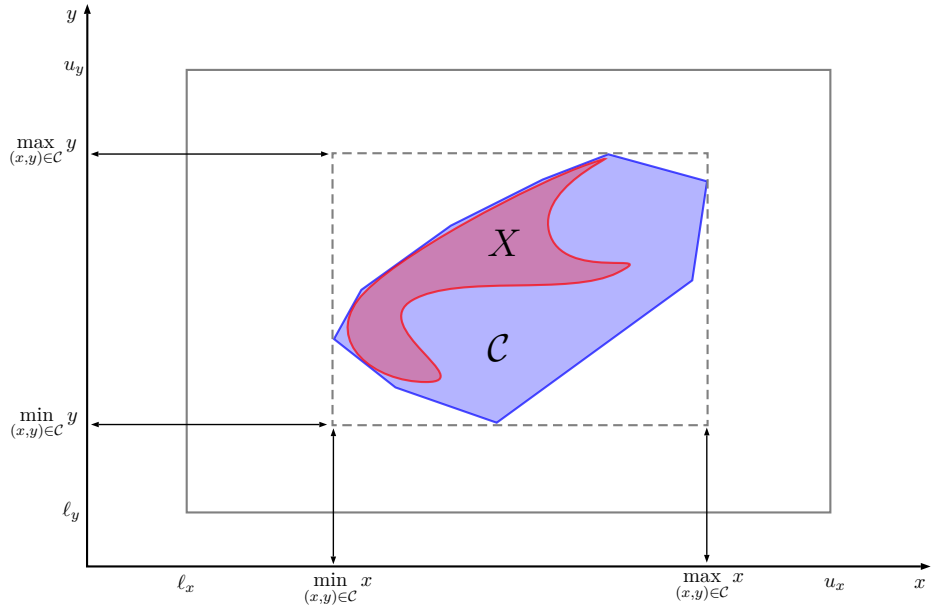

FIGURE 4.12 – *Pour une relaxation convexe $\mathcal{C}$ d'un ensemble admissible X, l'utilisation d'un solveur permet de trouver les réductions de bornes optimales pour $\mathcal{C}$, qui sont aussi valides pour X.*

Si les $x^k,\ k = 1..m$, sont les solutions obtenues, on définit $\ell_i^{\text{filter}} = \min_j x_i^j$, ainsi que $u_i^{\text{filter}} = \max_j x_i^j$. Ainsi, le resserrage d'une variable x_i est surestimé par $\ell_i^{\text{filter}} - \ell_i$ pour la borne inférieure et $u_i - u_i^{\text{filter}}$ pour la borne supérieure. On ne résout donc $\min_{\mathcal{C}} x_i$ que si $\ell_i^{\text{filter}} - \ell_i > \gamma(u_i - \ell_i)$ et $\max_{\mathcal{C}} x_i$ que si $u_i - u_i^{\text{filter}} > \gamma(u_i - \ell_i)$, avec γ un facteur définissant la réduction minimale valant la peine de résoudre un programme linéaire (typiquement 5% ou 10%). Le principe du filtrage de bornes est illustré dans la figure 4.13.

Enfin, il est souvent bénéfique de commencer l'OBBT par quelques passes de filtrage pur. L'objectif est de filtrer beaucoup de bornes en résolvant peu de programmes linéaires. Par exemple, pour les bornes supérieures, on définit l'ensemble des bornes non filtrées par

$$I^+ = \{i \mid u_i - u_i^{\text{filter}} \geq \gamma(u_i - \ell_i)\},$$

et la phase de filtrage consiste à résoudre

$$\max_{x\in\mathcal{C}} \sum_{I^+} x_i \quad \text{et} \quad \max_{x\in\mathcal{C}} \sum_{I^+} \frac{x_i}{1+|u_i|}$$

tant que leurs solutions optimales permettent de filtrer suffisamment de bornes (par exemple 10). On effectue de façon analogue le filtrage des bornes inférieures.

Gestion des problèmes numériques

Contrairement aux méthodes de réduction de bornes qui utilisent de l'arithmétique d'intervalle, l'OBBT n'est pas fiable numériquement. Si le solveur linéaire est

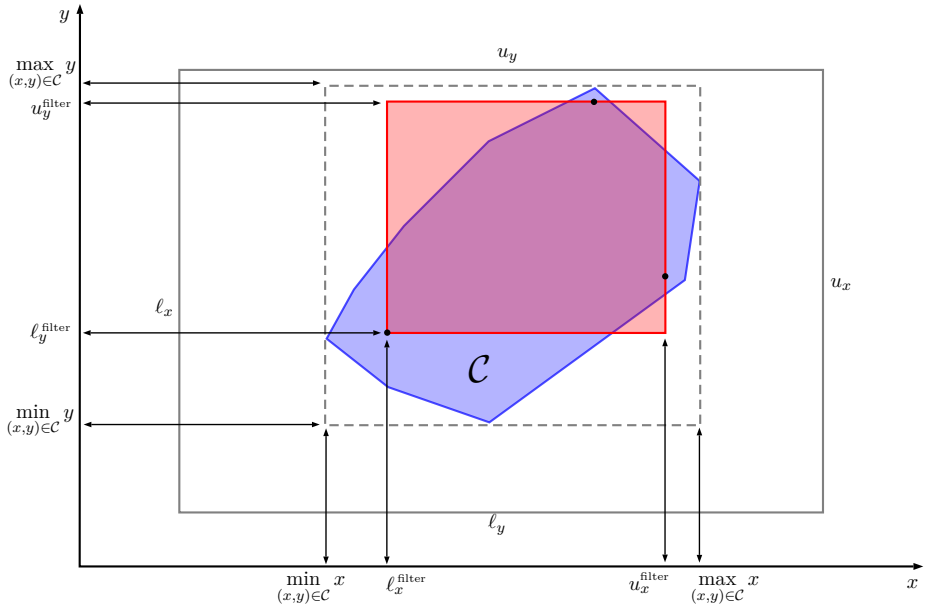

FIGURE 4.13 – *On cherche à appliquer l'OBBT à $\mathcal{C}$. Les points noirs sont des solutions admissibles de $\mathcal{C}$ et le rectangle rouge représente le domaine filtré. Les réductions de bornes ne pourront jamais être plus serrées que ce dernier. On voit par exemple qu'il n'y a pas grand-chose à gagner à résoudre* $\max_{\mathcal{C}} y$.

victime d'erreurs de précision numérique, comme cela peut être le cas sur des relaxations convexes mal conditionnées (par exemple, du fait de grandes bornes pour des variables présentes dans des couplages non linéaires), les nouvelles bornes obtenues sur les variables peuvent couper des solutions optimales. Il nous faut donc une procédure pour rendre l'OBBT robuste aux erreurs numériques.

Une fois de plus, on utilise les idées de dualité sur les contraintes de bornes. Soit un programme linéaire ainsi que son dual :

$$\text{(LP)} \quad \min_{x\in\mathbb{R}^n} c^T x \quad \text{s.c.} \begin{cases} \ell \le x \le u, \\ Ax + b = 0, \\ Cx + d \le 0. \end{cases}$$

$$\text{(D}_{\text{LP}}\text{)} \quad \max_{\lambda,\mu,\mu^+,\mu^-} b^T\lambda + d^T\mu + \ell^T\mu^- - u^T\mu^+ \quad \text{s.c.} \begin{cases} \mu \ge 0, \\ (\mu^+,\mu^-) \ge 0, \\ c + A^T\lambda + C^T\mu = \mu^- - \mu^+. \end{cases}$$

Une fois le programme linéaire résolu par le simplexe dual, on récupère les multiplicateurs des contraintes autres que les bornes : λ et μ. On calcule alors les coûts réduits des variables, non pas à partir des multiplicateurs de bornes ou en utilisant ceux renvoyés le simplexe, mais en calculant $c + A^T\lambda + C^T\mu$. Chaque coordonnée du vecteur des coûts réduits est calculée en utilisant une sommation de Kahan. Enfin, on définit les multiplicateurs de borne en utilisant

$$\mu^+ = -\min(0, c + A^T\lambda + C^T\mu) \quad \text{et} \quad \mu^- = \max(0, c + A^T\lambda + C^T\mu).$$

On utilise alors la valeur du lagrangien pour obtenir une borne valide pour la variable

courante :

$$L = b^T\lambda + d^T\mu + \ell^T\mu^- - u^T\mu^+.$$

Ce dernier calcul est aussi effectué avec une sommation de Kahan. Cette approche permet d'obtenir un certificat de KKT d'une grande précision pour la borne prouvée sur la variable, quels que soient les multiplicateurs renvoyés par le simplexe.

Impact de l'OBBT

Dans notre implémentation, l'OBBT s'applique une fois, après les actions de *presolve* et le *probing*. L'impact sur le nombre de nœuds est donné dans la figure 4.14, sur les instances résolues, et l'impact sur la qualité des bornes retournée est donné dans le tableau 4.6, sur les instances non résolues. Dans les deux cas, l'impact de l'OBBT est faible, et elle est significativement bénéfique sur peu d'instances. Son activation a permis de ne résoudre qu'une seule instance supplémentaire. Des expérimentations numériques plus poussées montrent plus précisément la chose suivante : pour les instances de petite et moyenne taille (jusqu'à quelques milliers de variables), les réductions obtenues avec l'OBBT sont dominées par celles obtenues par le *probing* (quel que soit leur ordre d'exécution). En revanche, pour les problèmes de plus grande taille, la situation s'inverse. Ceci est tout à fait logique, puisque le *probing* se base sur la FBBT qui n'exploite elle même que des informations liées à une contrainte à la fois, alors que l'OBBT exploite de l'information globale. L'importance relative de l'information portée par une unique contrainte par rapport à la géométrie globale du problème étant décroissante du nombre de contraintes, un tel comportement était à prévoir. Malheureusement, l'OBBT explose en temps justement à partir de 1000 ou 2000 variables. Pour cette raison, l'OBBT n'est pas activée par défaut dans LocalSolver. La possibilité d'utiliser une méthode de simplexe primal plutôt que dual devrait accélérer significativement l'OBBT, mais LocalSolver ne dispose pas de tel code.

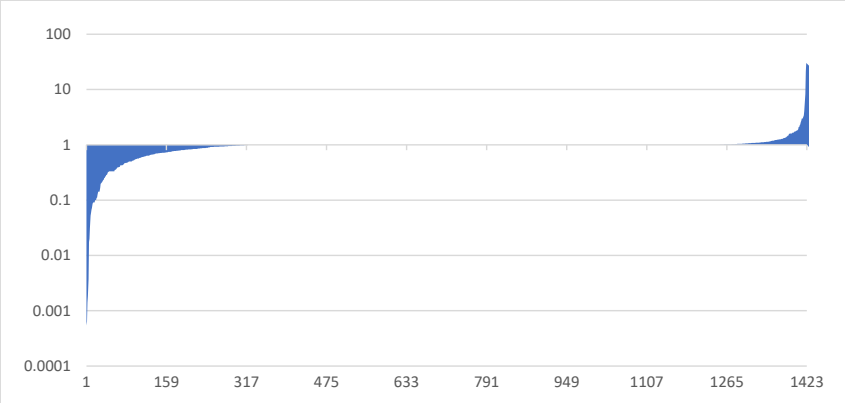

FIGURE 4.14 – *Pour les instances résolues avec ou sans l'OBBT, impact de cette dernière sur le nombre de nœuds nécessaires à la résolution.*

Impact de l'OBBT sur la borne finale	Proportion des instances
Détérioration supérieure à 10%	4.3%
Détérioration supérieure à 1%	9%
Détérioration inférieure à 1%	23.9%
Pas de changement de la borne	33.3%
Amélioration inférieure à 1%	14.2%
Amélioration supérieure à 1%	8.5%
Amélioration supérieure à 10%	6.3%

Tableau 4.6 – *Pour les instances que ni la stratégie par défaut, ni la stratégie sans l'OBBT n'ont résolues, comparaison des bornes inférieures finales.*

4.3.3 Réductions de bornes par analyse de sensibilité

Principe

Les techniques de réduction de bornes utilisant les informations duales ont été initialement développées pour l'optimisation linéaire en variables entières [169]. Dans ce contexte, elles sont plus connues sous le nom de *reduced cost tightening*. Elles ont été étendues au cas de la programmation non linéaire dans le cadre de l'optimisation globale, notamment dans [144]. L'idée générale est une technique de programmation par contraintes, qui consiste à exploiter des informations duales pour d'éliminer des sous-domaines qui ne contiendraient que des solutions sous-optimales [150]. On commence par rappeler le résultat principal que nous utiliserons.

Théorème 4.3.1. *[Extension de [144], Théorème 2] Soit un problème (P) et une relaxation convexe (R) de (P),*

$$\text{(P)} \min_{x\in\mathbb{R}^n} f(x) \quad s.c. \begin{cases} g(x) \leq 0, \end{cases} \qquad \text{(R)} \min_{x\in\mathbb{R}^n} \tilde{f}(x) \quad s.c. \begin{cases} \tilde{g}(x) \leq 0, \end{cases}$$

où f, $\tilde{f}$ ainsi que les g_i et $\tilde{g}_i$ sont supposées différentiables. Soient U une borne supérieure de (P), $x^* \in \mathbb{R}^n$ et $\mu \in \mathbb{R}^+_m$ tels que $\nabla\tilde{f}(x^*) + \sum_i \mu_i \nabla\tilde{g}_i(x^*) = 0$. Alors en posant $L = \tilde{f}(x^*) + \mu^T\tilde{g}(x^*)$, on a que pour tout i tel que $\mu_i > 0$, la borne

$$\tilde{g}_i(x) \geq \frac{L-U}{\mu_i},$$

est valide pour toute solution admissible x de (P) de valeur inférieure à U.

Démonstration. Dans [144], la preuve utilise la théorie de la dualité générale, avec des fonctions de perturbation. Nous proposons une preuve plus directe, utilisant uniquement des arguments géométriques.

L'idée est de partir de la contrainte valide $f(x) \leq U$. Par relaxation, cette contrainte passe sur $\tilde{f}$ puis, par convexité, sur la linéarisation de $\tilde{f}$ en x^*. En utilisant le fait que (x^*, μ) annule le gradient du lagrangien, cette inégalité passe de la

linéarisation de $\tilde{f}$ à celles des $\tilde{g}_i$. Enfin, encore par convexité, l'inégalité remonte aux $\tilde{g}_i$.

Plus précisément, $f(x) \leq U$ est valide pour toute solution améliorante. Or, puisque $\tilde{f}$ est une relaxation de f sur l'ensemble admissible, $\tilde{f}(x) \leq f(x)$. On a donc aussi $\tilde{f}(x) \leq U$. Par convexité, $\tilde{f}$ est sous-estimé par sa linéarisation en x^* et on a donc

$$U \geq \tilde{f}(x) \geq \tilde{f}(x^*) + \nabla\tilde{f}(x^*)^T(x - x^*),$$

en utilisant l'hypothèse que $\nabla\tilde{f}(x^*) = -\sum_i \mu_i \nabla\tilde{g}_i(x^*)$, on obtient

$$U \geq \tilde{f}(x) \geq \tilde{f}(x^*) + \sum_i \mu_i \nabla\tilde{g}_i(x^*)^T(x^* - x),$$

en utilisant la définition de L, $\tilde{f}(x^*) = -\sum_i \mu_i \tilde{g}_i(x^*)$, ce qui donne

$$U \geq L + \sum_i \mu_i \left(-\tilde{g}_i(x^*) + \nabla\tilde{g}_i(x^*)^T(x^* - x)\right).$$

Finalement, puisque les $\tilde{g}_i$ sont convexes on a $\tilde{g}_i(x^*) + \nabla\tilde{g}_i(x^*)^T(x - x^*) \leq \tilde{g}_i(x)$, ce qui permet d'éliminer x^* de l'inégalité :

$$U \geq L - \sum_i \mu_i \tilde{g}_i(x).$$

Puisque $g_i(x) \leq 0$ et donc $\tilde{g}_i(x) \leq 0$ sont satisfaites pour toute solution admisible, on a finalement

$$\forall i,\ \mu_i \tilde{g}_i(x) \geq L - U$$

est valide pour toute solution primale x de valeur inférieure à U. □

Remarque : l'inégalité $\sum_i \mu_i \tilde{g}_i(x) \geq L - U$ est plus serrée que la famille d'inégalités que nous utilisons. Dans le cas linéaire, c'est une contrainte linéaire et peut être utilisée pour de l'inférence de bornes. Cette idée est exploitée dans [165], en particulier lorsqu'elle constitue une preuve d'inconsistance. Dans le cas non linéaire, sa forme algébrique diffère de celles des contraintes du modèle factorisé. Son exploitation passe donc par l'implémentation de formules d'inférence dédiées, analogues à celles que nous avons mises en place pour les sommes quadratiques, approche non retenue pour le moment.

Gestion des erreurs numériques

Comme pour l'OBBT, les formules de réduction que nous utilisons ne sont pas robustes aux erreurs numériques. Si l'inférence utilisée ensuite pour resserrer les bornes des variables n'ajoute pas d'erreurs, il reste celles présentes dans L, U et les μ_i. Différentes idées ont été utilisées pour rendre la procédure robuste. Dans [108], le certificat de KKT est analysé et raffiné avec la méthode de Newton par intervalles. Ceci permet d'obtenir un encadrement serré d'une solution primale-duale optimale exacte. Les bornes sur les variables duales sont ensuite utilisées pour appliquer le cas le plus pessimiste de la formule. Dans [100], les formules de réduction ne sont utilisées que pour suggérer des réductions de bornes. Ces réductions sont ensuite prouvées

par propagation exacte, de façon similaire au *probing* que nous avons présenté. Si la propagation ne peut prouver l'inconsistance, aucune réduction n'est effectuée.

Si les deux approches ci-dessus sont exactes et sont implémentées dans des solveurs certifiés, nous n'avons pas besoin d'aller aussi loin. On utilise plutôt la même méthode que pour l'OBBT : on répare le certificat d'optimalité avec les multiplicateurs des contraintes de bornes et des sommes de Kahan. La borne L ainsi obtenue est plus précise, et les multiplicateurs des contraintes de bornes sont exacts. Par mesure de sécurité, on utilise aussi un coefficient de sous-estimation pour tous les multiplicateurs. En effet, les bornes impliquées sont d'autant plus serrées que les multiplicateurs associés sont grands. En remplaçant les formules par

$$\tilde{g}_i(x) \geq \frac{L - U}{\gamma \mu_i},$$

avec $0 < \gamma < 1$, on obtient un algorithme plus robuste aux erreurs numériques.

Impact de la SBBT

Les réductions de bornes par sensibilité, que nous désignons par SBBT, s'appliquent après la résolution de chaque relaxation convexe. L'impact sur le nombre de nœuds est donné dans la figure 4.15, sur les instances résolues, et l'impact sur la qualité des bornes retournée est donné dans le tableau 4.7, sur les instances non résolues. Pour les instances non résolues, l'impact de la SBBT reste modeste. Ceci n'est pas surprenant, puisque celle-ci utilise la donnée d'une solution primale. Comme nous le verrons dans le prochain chapitre, il existe une corrélation entre la capacité à trouver des solutions primales et la qualité des bornes retournées. Sur les instances résolues en revanche, l'impact de la SBBT est net, et est même plus significatif que celui de la FBBT post-branchement. Le nombre d'instances résolues augmente de 2.8% avec l'activation de la SBBT.

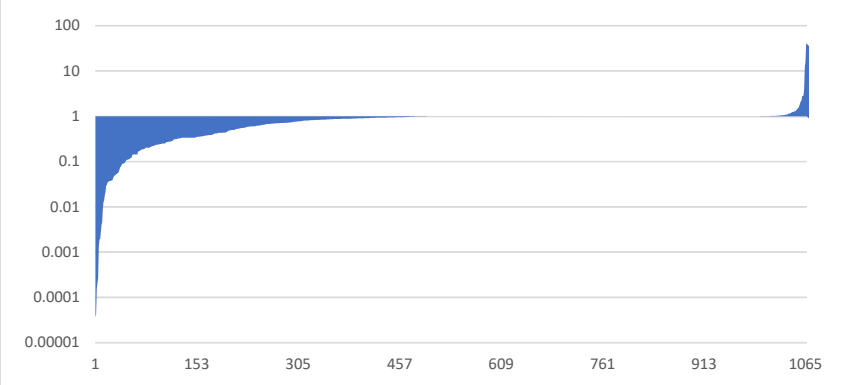

FIGURE 4.15 – *Pour les instances résolues avec ou sans la SBBT, impact de cette dernière sur le nombre de nœuds nécessaires à la résolution.*

Impact de la SBBT sur la borne finale	Proportion des instances
Détérioration supérieure à 10%	1.4%
Détérioration supérieure à 1%	9.2%
Détérioration inférieure à 1%	30%
Pas de changement de la borne	37%
Amélioration inférieure à 1%	11.6%
Amélioration supérieure à 1%	6.5%
Amélioration supérieure à 10%	4.2%

Tableau 4.7 – *Pour les instances que ni la stratégie par défaut, ni la stratégie sans la SBBT n'ont résolues, comparaison des bornes inférieures finales.*

Chapitre 5

Résultats, perspectives et conclusions

Les outils décrits dans les chapitres précédents - reformulations, génération et résolution de relaxations convexes, techniques de réduction de bornes et recherche arborescente - forment les composants d'un solveur d'optimisation globale. Dans notre cas, celui-ci est dédié au calcul de bornes inférieures, et fonctionne en interaction avec les autres modules de LocalSolver.

Le dernier chapitre de ce manuscrit commence par mesurer l'impact de ce nouveau module dual sur les résultats obtenus par LocalSolver. Nous analysons les bornes inférieures produites sur différents *benchmarks*, mais aussi la performance de LocalSolver en tant que solveur d'optimisation globale. Pour ce faire, nous comparons les résultats obtenus avec ceux de quelques solveurs de référence du domaine.

Cette comparaison, ainsi que l'analyse des avantages respectifs des relaxations linéaires et non linéaires, nous permettent de donner les perspectives futures de LocalSolver pour le calcul de bornes et l'optimisation globale. En reprenant les différents aspects évoqués dans cette thèse, nous proposons une feuille de route technique qui devrait permettre d'améliorer les résultats obtenus.

Enfin, l'abstraction de cette feuille de route, sa mise en relation avec d'autres approches de l'optimisation globale et la prise en compte de certaines méthodes plus spécifiques nous permettent de conclure en proposant une réflexion sur l'implémentation d'outils numériques pour l'optimisation globale.

Plan du chapitre

5.1 Résultats numériques

L'objectif de cette section est de décrire de façon concise l'apport du module dual implémenté, en particulier vis-à-vis de notre contexte industriel. L'optimisation globale étant l'assemblage d'un certain nombre d'outils géométriques et algorithmiques, il est délicat d'établir de façon objective les contributions individuelles de chaque composant. Une sélection de résultats et d'analyses numériques a déjà été présentée tout au long du développement, et nous nous concentrons ici sur les résultats finaux.

Pour des questions de ressources en puissance de calcul, mais aussi d'adéquation avec les besoins de LocalSolver, les analyses ont été effectuées en utilisant un temps limite de l'ordre de la minute. Si celui-ci est relativement faible (on trouve plus souvent dans la littérature des temps limite de l'ordre de l'heure), ce choix sera justifié de façon plus pragmatique par les conclusions de la partie 5.1.2. Par ailleurs, les calculs ont été effectués sur des ordinateurs de bureau relativement récents, qui peuvent varier selon les expériences mais qui sont toujours les mêmes pour une comparaison donnée.

La présentation des résultats numériques se veut factuelle et descriptive, les perspectives et analyses qui en découlent étant présentées dans la dernière section. Nous rappelons de plus que si le calcul de bornes inférieures se veut le plus universel possible, tout ensemble fini d'instances est nécessairement biaisé. Une comparaison de deux stratégies, ou de deux solveurs, sur un *benchmark* donné est toujours limitée à ce dernier, et ne peut se généraliser directement.

5.1.1 Calcul de bornes inférieures

Le module dual permet, grâce à la recherche arborescente et après un nombre fini de décisions de branchement, de résoudre les MINLP supportés à l'optimum global. Cependant, l'objectif premier de cette thèse reste le calcul de bornes inférieures. Si LocalSolver a toujours été capable de fournir de telles bornes, par propagation et inférence de bornes sur le DAG, il ne disposait pas de méthodes permettant de les améliorer.

Parmi les techniques implémentées et présentées précédemment, plusieurs sont susceptibles d'améliorer la borne inférieure courante. En suivant l'ordre d'exécution lors d'un appel à LocalSolver, six de ces bornes sont résumées dans le tableau 5.1. Au-delà des améliorations de bornes, c'est le compromis entre qualité et temps d'obtention qui est pertinent dans notre contexte. C'est pour cette raison que l'OBBT (partie 4.3.2) a d'ores-et-déjà été éliminée de nos comparaisons, puisque son temps d'exécution explose trop vite. De même, le coût mémoire est le facteur limitant pour certaines instances de grande taille. Pour cette raison, nous interrompons dans cette analyse la recherche arborescente après un nombre fini de décisions de branchement. Si les relaxations linéaires sont utilisées, on autorise 100 nœuds au maximum, contre 10 pour les relaxations non linéaires, afin d'équilibrer les temps de calcul. L'ensemble du *benchmark* est considéré dans cette analyse, pour une durée maximale de 15 minutes.

No	Nom	Moment d'exécution	Description	Présentation
1	FBBT-LS	*Préprocessing* du DAG	Propagation et inférence sur le DAG	
2	FBBT-D	*Presolve* du module dual	Propagation et inférence sur le modèle factorisé	Section 4.2
3	PROB-obj	*Presolve* du module dual	*Probing* sur les bornes de l'objectif	Partie 4.3.1
4	PROB-var	*Presolve* du module dual	*Probing* sur les bornes de toutes les variables	Partie 4.3.1
5	ROOT-relax	*Root node* du module dual	Résolution de la première relaxation convexe	Chapitre 2
6	BR-k	Recherche arborescente	Borne inférieure après k décisions de branchement	Section 4.1

Tableau 5.1 – *Techniques susceptibles d'améliorer la borne inférieure renvoyée par LocalSolver.*

Les figures 5.1 et 5.2 montrent l'amélioration relative de la borne inférieure calculée par chaque borne décrite dans le tableau 5.1 par rapport à FBBT-LS, la borne de propagation de LocalSolver, dans les cas linéaire et non linéaire. Deux bornes inférieures L_1 et L_2 sont comparées avec la formule

$$\frac{L_1 - L_2}{1 + |L_1| + |L_2|}. \tag{5.1}$$

Pour plus de 40% des instances, les techniques de réduction de bornes implémentées améliorent la borne duale. Une fois la relaxation convexe résolue, ce nombre passe à plus de 60%. Il est à la fois enthousiasmant, puisque les outils développés pendant cette thèse permettent d'améliorer les bornes finales de façon significative, et désespérant, car malgré toute la géométrie implémentée, plus d'un tiers des instances

voit sa meilleure borne obtenue par propagation uniquement. Le *preprocessing* de LocalSolver n'était pas capable de déterminer de borne sur 117 instances, car utilisant une librairie d'arithmétique d'intervalle incomplète, et retournait une valeur duale de $-\infty$. Ces cas ont tous disparu, puisque la finitude supposée des bornes des variables dans le module dual assure une borne finie sur chaque instance.

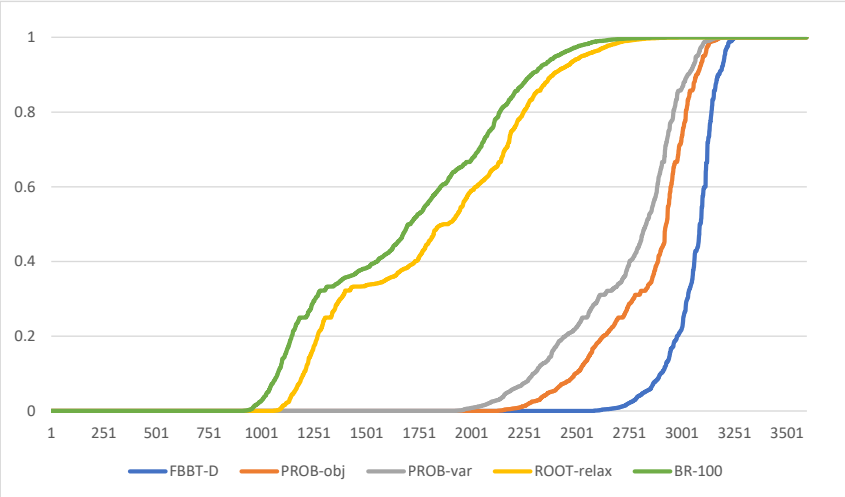

FIGURE 5.1 – *Amélioration relative de la borne fournie par LocalSolver après l'exécuction des différentes techniques, modèle factorisé linéaire. Chaque courbe utilise un tri différent.*

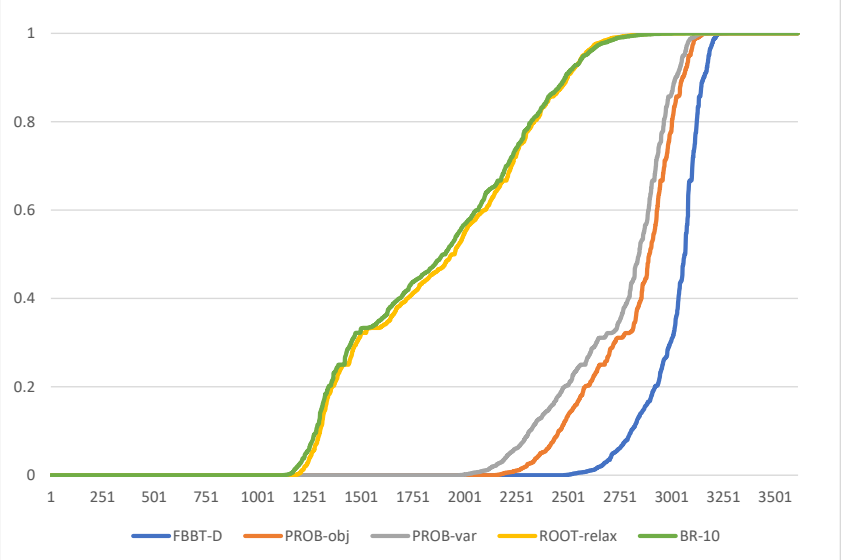

FIGURE 5.2 – *Amélioration relative de la borne fournie par LocalSolver après l'exécuction des différentes techniques, modèle factorisé quadratique. Chaque courbe utilise un tri différent.*

On s'intéresse maintenant au temps d'obtention des bornes inférieures. Le temps d'exécution du *probing* sur l'objectif est largement inférieur à celui des autres bornes, et n'est donc pas présent dans cette analyse. De plus, le temps d'exécution de FBBT-LS est défini comme celui du *preprocessing* complet, bien que la propagation et l'inférence n'en constituent qu'une partie. Ce dernier représente cependant une référence unitaire à laquelle on peut se comparer, puisqu'exécuté sur tous les modèles quelles que soient leurs tailles. Les temps d'exécution des différents modules sont tracés dans les figures 5.3 et 5.4.

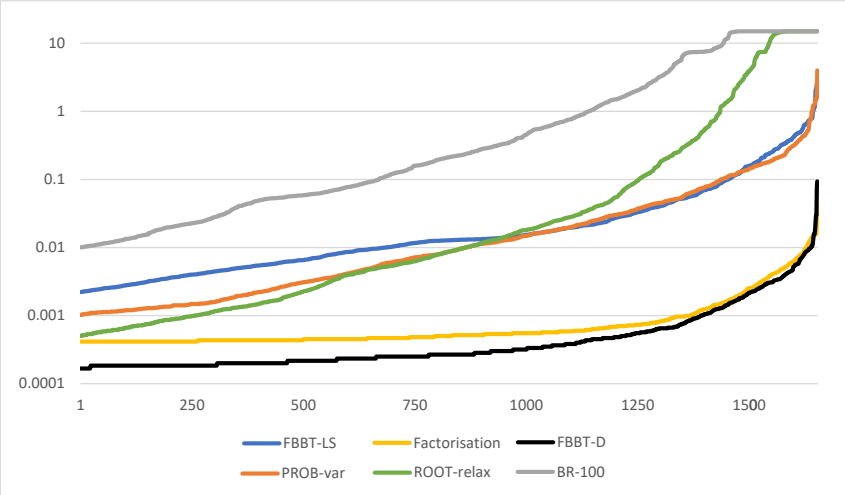

FIGURE 5.3 – *Temps d'obtention des différentes bornes, en minutes, avec le modèle factorisé linéaire. Seules les instances de plus grande taille sont considérées. Chaque courbe utilise un tri différent.*

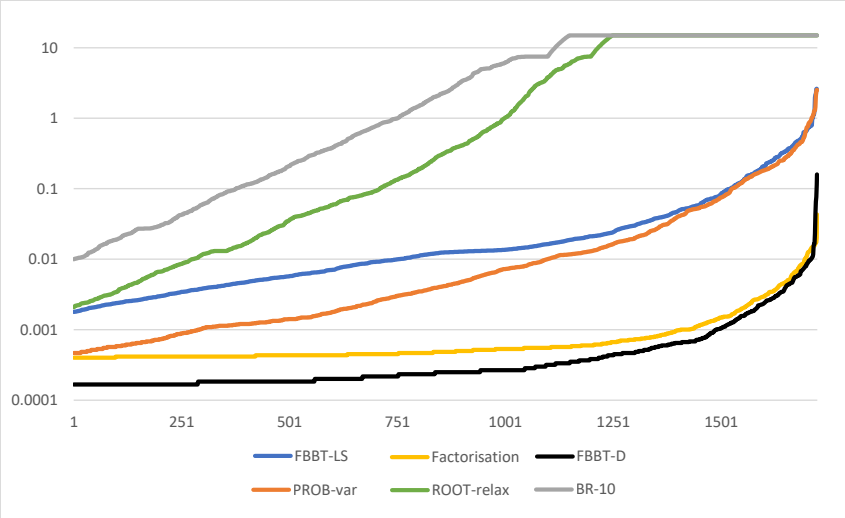

FIGURE 5.4 – *Temps d'obtention des différentes bornes, en minutes, avec le modèle factorisé quadratique. Seules les instances de plus grande taille sont considérées. Chaque courbe utilise un tri différent.*

Une première remarque est que les opérations de reformulation (courbes jaunes) et de FBBT initiale (courbes noires) sont peu coûteuses. L'augmentation rapide de leur temps d'exécution, à la droite des courbes, est due à la taille des DAG des instances du *benchmark*, qui n'augmente pas de façon linéaire, comme montré dans la figure 5.5. Pour les autres boenes, le passage à l'échelle en temps est également satisfaisant. En particulier, le *probing* semble passer à l'échelle à la même vitesse que le *preprocessing* de LocalSolver, et seules quelques instances nécessitent plus d'une minute pour l'effectuer. L'utilisation des solveurs pour résoudre des relaxations convexes est plus coûteux. Pour environ 2% des instances, les 15 minutes n'ont pas été suffisantes pour résoudre la relaxation linéaire du *root node*, contre 10% pour la relaxation non linéaire. La minute, référence de temps pendant ce travail, ne permet pas de résoudre la première relaxation linéaire pour 7% des instances. Dans le cas des relaxations non linéaires, c'est 18% des relaxations du *root node* qui ne sont pas résolues en une minute. Pour les temps qui intéressent LocalSolver, les relaxation linéaires restent les plus adaptées. Il est cependant intéressant de remarquer que la courbe augmentant le plus vite est celle du temps de résolution des relaxations linéaires. Enfin, on remarque sur la figure 5.2 que la borne BR-10, utilisant des relaxations non linéaires, n'améliore quasiment pas la borne du *root node*. Multiplier le nombre de nœuds par 10 donnerait des améliorations similaires à celles observées sur la figure 5.1 pour la borne BR-100 utilisant des relaxations linéaires, mais le coût de résolution des 10 nœuds non linéaires dépassant déjà celui des 100 nœuds linéaires, améliorer la borne inférieure par branchement avec des relaxations non linéaires semble une approche encore ambitieuse.

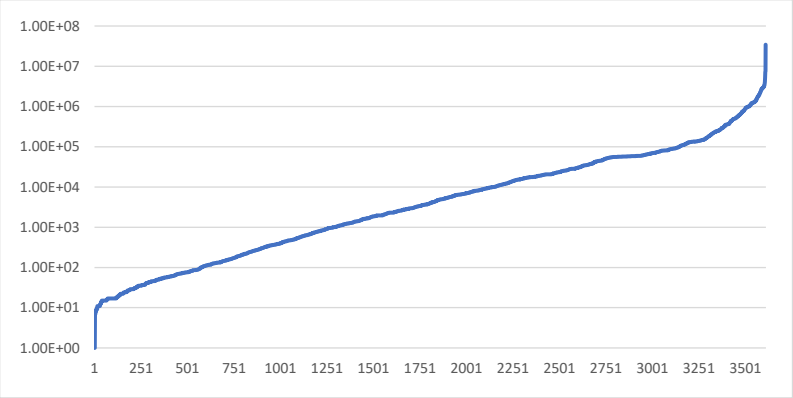

FIGURE 5.5 – *Tailles des instances, en non-zéros dans le DAG après* preprocessing.

Nous terminons l'analyse des bornes inférieures fournies par une comparaison entre modèle factorisé linéaire et quadratique. Si les temps de factorisation, de FBBT initiale et de *probing* sont légèrement meilleurs pour le modèle factorisé quadratique, la différence n'est pas significative et peut être ignorée. La figure 5.6 compare quelques-unes des bornes obtenues dans les deux cas. Les deux formulations ont leurs avantages respectifs, mais aucune ne domine l'autre, ni en termes de techniques de réduction de bornes, ni en termes de relaxations convexes. Si la FBBT semble plus performante sur la structure quadratique, cette différence s'amenuise après le *probing*, et disparaît même après la résolution du *root node*. La version quadratique du modèle factorisé permet d'exploiter plus de structure que sa version linéaire, et l'on

aurait pu s'attendre à de meilleures bornes, au moins pour la FBBT. En pratique, des subtilités d'implémentation causent des différences de comportement entre les deux, et donnent parfois l'avantage au linéaire. Des explications plus précises, sur les raisons de l'absence de domination du quadratique, ainsi que sur les solutions pour obtenir le meilleur des deux, sont données dans les perspectives.

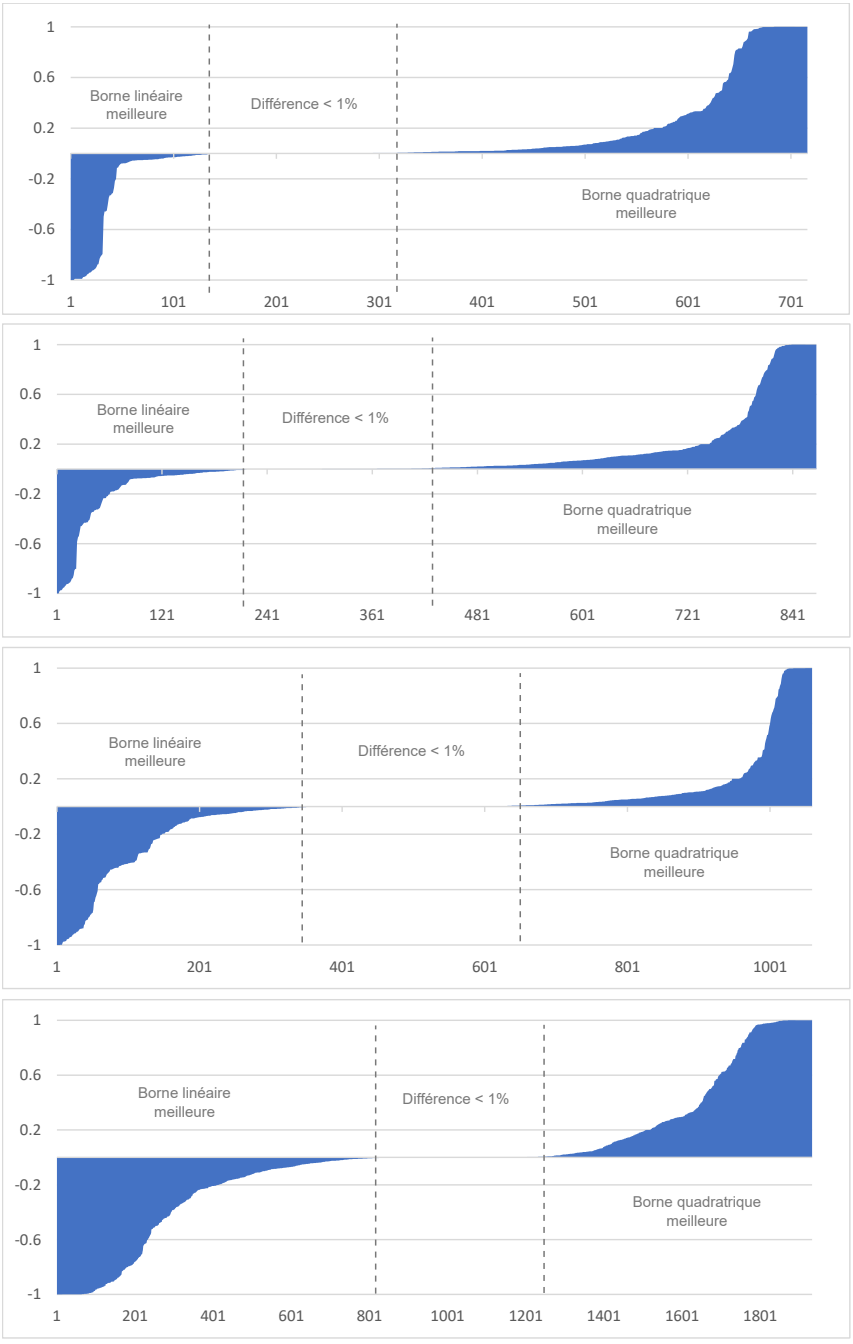

FIGURE 5.6 – *Comparaison des bornes obtenues par les modèles factorisés linéaire et quadratique, lorsque celles-ci sont différentes. On utilise la formule* (5.1)*. De haut en bas, FBBT-D, PROB-obj, PROB-var, ROOT-relax.*

5.1.2 Optimisation globale avec LocalSolver

Les bornes inférieures obtenus par le module dual de LocalSolver sont d'autant plus serrées que les relaxations convexes utilisées sont précises. D'un autre côté, plus les relaxations convexes sont précises, plus leurs solutions sont proches de la faisabilité primale. Pour cette raison, analyser les performances de LocalSolver en tant que solveur d'optimisation globale a plusieurs intérêts : tous les problèmes d'optimisation de l'industrie ne sont pas de grande taille (en particulier lors de phases de test ou de développement) et la validité des bornes inférieures obtenues est difficile à contrôler si l'on n'est pas capable d'aller au bout de la recherche sur des petites instances. Nous analysons différentes stratégies sur trois *benchmark* : celui des 2211 plus petites instances, la MINLPLib et sur 638 instances industrielles issues des clients de LocalSolver. Le module dual peut fonctionner sous cinq modes différents, selon les paramètres utilisés :

— DUAL(lin), utilisant le modèle factorisé et les relaxations linéaires,
— DUAL(nl), utilisant le modèle quadratique et les relaxations non linéaires,
— DUAL(hyb), une heuristique sélectionnant automatiquement l'un des deux,
— CP(lin), utilisant le modèle factorisé linéaire sans résoudre de relaxations,
— CP(nl), utilisant le modèle factorisé quadratique sans résoudre de relaxations.

Les deux modes CP convergent grâce aux techniques de réduction de bornes et à la recherche arborescente. Les bornes d'un nœud donné sont moins bonnes, mais l'énumération est plus rapide. L'heuristique utilisée pour DUAL(hyb) est élémentaire : le modèle factorisé quadratique est construit, puis conservé s'il contient exactement une expression quadratique (l'objectif ou une unique contrainte). Dans le cas contraire, le module dual est redémarré en mode linéaire. Des règles plus fines sont possibles, mais l'objectif est de montrer qu'une stratégie optimale est hybride, et que les relaxations linéaires et non linéaires doivent se compléter, et non pas s'opposer. De plus, nous définissons deux stratégies virtuelles pour évaluer les gains potentiels d'une stratégie hybride plus évoluée. La stratégie SPLIT divise le temps limite en deux et exécute chaque mode sur une moitié. Enfin, la stratégie VBEST consiste à prendre le résultat du meilleur mode (sur deux exécutions). Le module primal de LocalSolver est noté LS, et l'on compare LS seul, le module dual seul, ainsi que différentes combinaisons possibles. Les tableaux 5.2, 5.3 et 5.4 présentent les résultats, respectivement sur les *benchmarks* de petite taille, MINLPLib et industriel.

	Solution optimale	Solution admissible	Aucune solution
LS	25.8%	80.7%	19.2%
DUAL(hyb)	64.9%	75.9%	24.2%
LS + CP(lin)	43.1%	81.5%	18.4%
LS + CP(nl)	43.4%	81.7%	18.3%
LS + DUAL(lin)	65%	85.8%	14.1%
LS + DUAL(nl)	64.7%	84.1%	15.9%
LS + DUAL(hyb)	67.4%	85.7%	14.3%

Tableau 5.2 – *Capacités d'optimisation globale de LocalSolver sur les 2211 instances de plus petite taille. Temps limite de 60 secondes.*

L'analyse des résultats du tableau 5.2 montre que sur les petites instances, les capacités d'optimisation globale de LocalSolver ont été largement améliorées. Le nombre de solutions optimales est plus que doublé, et le nombre d'instances sans solutions est réduit de 25%. Sur les très petites instances (moins de 300 non-zéros dans le DAG), les stratégies LS + DUAL(lin) et LS + DUAL(nl) sont capables de résoudre 94% des problèmes en la minute impartie, et la stratégie LS + DUAL(hyb) porte ce chiffre à 96%. On peut noter que les sauts qualitatifs entre LS seul, LS + CP et LS + DUAL sont supérieurs à la variabilité liée à l'utilisation de tel ou tel couple relaxation-solveur. Enfin, on remarque que malgré la recherche arborescente, capable d'énumérer un grand nombre de nœuds sur cette taille d'instances, LocalSolver reste plus efficace pour trouver une solution admissible.

Stratégie	Solution optimale	Solution admissible
LS	4.7%	64.4%
DUAL(hyb)	38.7%	51.2%
LS + DUAL(lin)	37.4%	70.5%
LS + DUAL(nl)	37%	68.4%
LS + DUAL(hyb)	40.5%	70.4%
LS + SPLIT	41.4%	
LS + VBEST	42.8%	

Tableau 5.3 – *Capacités d'optimisation globale de LocalSolver sur 1511 instances de la MINLPLib. Temps limite de 60 secondes.*

Ces résultats sont confirmés par le tableau 5.3, sur la MINLPLib. Malgré l'absence de techniques primales dédiées aux MINLP, LocalSolver trouve plus de solutions que le module dual. Contrairement aux instances de petite taille, on voit un léger avantage pour les relaxations linéaires. Cet avantage est un effet numérique lié aux tolérances de faisabilité et à la remontée des solutions admissibles à LocalSolver. Dans les relaxations, le simplexe dual utilise des tolérances absolues, alors que le solveur non linéaire utilise des tolérances relatives. Ceci est dû à la nature directe de résolution des programmes linéaires, par factorisation, qui permet au simplexe d'être plus précis. Puisque le modèle est reformulé, puis prétraité par le *presolve*, des tolérances relatives peuvent empêcher la faisabilité d'une solution dans le DAG. Ainsi, les 2.1% de solutions admissibles trouvées en plus par l'approche linéaire se traduisent par 0.4% d'instances fermées supplémentaires. Enfin, on peut remarquer que notre heuristique DUAL(hyb) n'est pas si mauvaise, puisqu'elle comble plus de la moitié de la différence entre DUAL(lin) (ou DUAL(nl)) et VBEST.

Stratégie	Solution optimale	Solution admissible	Aucune solution
LS	21.5%	98%	2%
DUAL(lin)	9.9%	41.2%	58.8%
LS + DUAL(lin)	25.4%	98.3%	1.7%
LS + DUAL(nl)	24.8%	98.1%	1.9%

Tableau 5.4 – *Capacités d'optimisation globale de LocalSolver sur 638 instances industrielles des clients. Le temps limite est de 300 secondes.*

Le tableau 5.4 montre que la situation est différente sur le *benchmark* industriel. Ces instances ne sont pas vraiment des MINLP, puisqu'ils comportent rarement d'opérateurs non linéaires tels que exp, log ou sin. Les produits mis à part, ce sont les min, max, valeurs absolues, structures `if-then-else`, opérateurs logiques et relations de comparaisons qui y sont les plus fréquents. Ces instances ont été modélisés pour le module primal de LocalSolver, d'où les excellentes performances de ce dernier sur le nombre de solutions admissibles. Le tableau 5.4 montre deux choses importantes sur l'optimisation globale : celle-ci est limitée aux instances de petite et moyenne taille, et sans heuristiques primales performantes, elle n'est pas capable de fermer beaucoup d'instances.

Nous terminons par une remarque sur le coût de résolution à l'optimum global, en temps et en mémoire, des plus petites instances. Les figures 5.7 et 5.8 tracent ces coûts de résolution pour la stratégie LS + DUAL(lin), et montrent une explosion du coût mémoire et une augmentation rapide du temps de résolution.

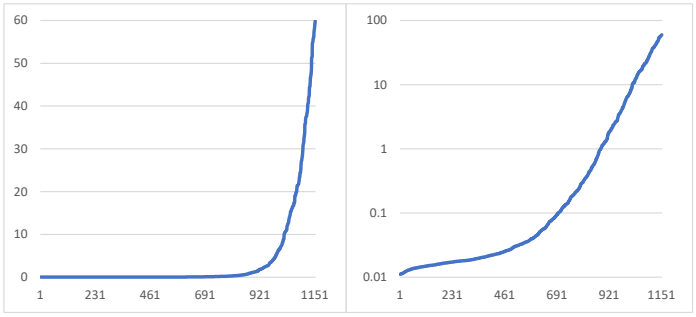

FIGURE 5.7 – *Temps de résolution en secondes, pour les 1160 plus petites instances résolues par LS + DUAL(lin) mais pas par LS seul. À gauche, en échelle normale et à droite, en échelle logarithmique.*

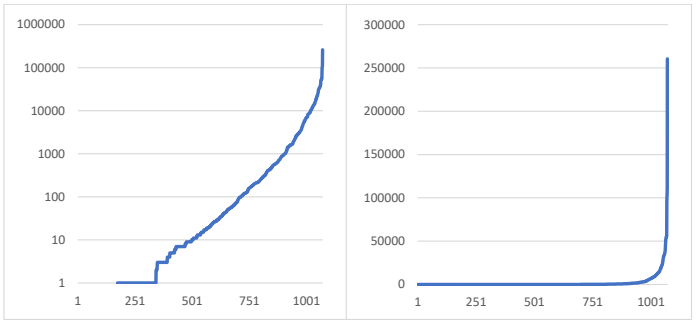

FIGURE 5.8 – *Taille de l'arbre de branchement (coût mémoire), pour les 1160 plus petites instances résolues par LS + DUAL(lin) mais pas par LS seul. À gauche, en échelle logarithmique et à droite, en échelle normale.*

En interpolant les courbes, on voit que doubler le temps de résolution permettrait de résoudre à peine 10% d'instances supplémentaires. Cette explosion du temps de résolution à l'optimum justifie l'utilisation de temps limite assez courts, de l'ordre de la minute, pour des instances de petite et moyenne taille. L'explosion en mémoire montre quant à elle que la recherche arborescente est un outil inadapté aux instances

de grande taille. Sur ces dernières, elle ne doit pas être utilisée en tant que squelette d'un solveur, mais plutôt en tant qu'outil prenant en argument un certain crédit (de nœuds, d'itérations des solveurs, ...).

5.1.3 Comparaison à BARON, COUENNE et SCIP

Les solveurs BARON [155, 144, 102, 101, 146, 147], SCIP [15, 165, 160, 159, 17, 127] et COUENNE [36] sont des solveur d'optimisation globale des MINLP non convexes. Les outils principaux (techniques de réduction de bornes, relaxations convexes et recherche arborescente) utilisées sont essentiellement les mêmes que ceux décrits dans cette thèse, même si l'on peut noter quelques différences. Aucun de ces solveurs ne résout les relaxations du modèle factorisé avec un solveur non linéaire. De plus, le traitement du primal diffère totalement. Dans LocalSolver, les recherches de solutions et de bornes sont entièrement découplées. Dans ces trois solveurs, des heuristiques primales sont appliquées lors du *presolve* ou pendant la recherche arborescente, en particulier par appel à différents solveurs locaux de NLP, PLNE ou MINLP. La recherche arborescente est de plus hybride primale-duale, ce qui a des conséquences sur les règles de branchement et de sélection des nœuds. Dans BARON et COUENNE, les relaxations linéaires sont résolues avec CLP [5], et dans SCIP, avec Soplex [6]. Si BARON peut utiliser CPLEX, nous préférons faire cette comparaison avec des solveurs linéaires les plus homogènes possibles.

Dans BARON, des relaxations non linéaires sont résolues, mais directement sur le DAG du problème, lorsque celui-ci est convexe pour un nœud donné [101]. De plus, des relaxations de type PLNE peuvent être utilisées [102], et plusieurs problèmes particuliers y sont traités de façon spécifique [155]. Enfin, des techniques de réduction de bornes plus avancées existent, qui généralisent par exemple les réductions OBBT et SBBT présentées en parties 4.3.2 et 4.3.3. Le cas de SCIP est différent : le logiciel est construit par extension de CIP [17], un solveur de PLNE rassemblant des techniques de divers domaines, tels que la programmation par contrainte, la programmation entière, ou encore les problèmes de satisfaction. Pour cette raison, il utilise intensivement les techniques issues de la PLNE, en particulier dans la géométrie de l'intégrité et dans la recherche arborescente. Enfin, COUENNE est le solveur global de COIN-OR [8], une bibliothèque d'outils libres pour la recherche opérationnelle. Contrairement aux deux premiers, qui sont les solveurs les plus performants, COUENNE n'est pas maintenu de façon intensive. Enfin, il est important de noter que les fonctions trigonométriques ne sont pas supportées par BARON ni SCIP, alors qu'une centaine d'instances de la MINLPLib en comportent.

Nous considérons la version de la MINLPLib, bibliothèque de référence, datant de mi-2018. Une fois les instances supportées par aucun solveur retirées, celle-ci comporte 1511 instances. Les trois solveurs BARON, COUENNE et SCIP ont été obtenus par une licence temporaire de GAMS [9], et leur version date de fin 2018. Depuis, les fonctionnalités que ces solveurs comportent a pu évoluer. Par exemple, un sous-ensemble des techniques mises en place au chapitre 2 sur les relaxations non linéaires des expressions quadratiques a récemment été intégré à BARON [147]. Notre objectif n'est cependant pas de classer les solveurs, mais d'en comparer la performance, afin de mettre en place une feuille de route pour l'amélioration des

bornes dans LocalSolver. Les hypothèses numériques que nous avons supposées sur les bornes des variables sont par exemple plus restrictives que celles que les autres s'imposent. De plus, le format et techniques de modélisation sont différents entre GAMS et LocalSolver, résultant en des DAG d'entrée légèrement différents. Le tableau 5.5 présente les résultats obtenus et les tableaux 5.6 et 5.7 s'intéressent aux différences et à la complémentarité des solveurs. La stratégie utilisée par LocalSolver est LS + DUAL(hyb).

Instances	LocalSolver	COUENNE	BARON	SCIP
Toutes (1511)	40.5%	34.2%	43.2%	44.8%
Sans intégrités (582)	51.5%	50.3%	54.3%	47.8%
Avec intégrités (929)	33.6%	24.1%	36.2%	42.9%
Sans non linéaire (238)	61.8%	58.4%	65.1%	60.1%
Avec non linéaire (1273)	36.5%	29.7%	39.0%	41.8%
Non linéaire sans intégrités (469)	44.3%	44.1%	48.0%	40.7%

Tableau 5.5 – *Nombre d'instances résolues par chaque solveur, selon la présence de contraintes d'intégrité ou d'opérateurs non linéaires (autre que des produits). Temps limite de 60 secondes.*

	LocalSolver	COUENNE	BARON	SCIP
Seul à résoudre	29	5	33	67
Seul à ne pas résoudre	32	63	5	31
Instabilité numérique	5	15	5	7
Solution admissible	1036	1091	1209	1244

Tableau 5.6 – *Quelques statistiques sur les instances résolues et les problèmes rencontrés par les différents solveurs sur les 1511 instance. Une instabilité numérique est déclarée si les bornes primale ou duale finales ne sont pas cohérentes avec les meilleures solutions connues ou avec celles des autres solveurs.*

Le tableau 5.5 montre que les capacités d'optimisation globale de LocalSolver sont inférieures à l'état de l'art, mais que nous sommes sur le bon chemin. On y voit cependant que la présence d'opérateurs non linéaires (du type $\exp$, $\sqrt{}$, ...) a moins d'impact sur les performances relatives des solveurs que la présence de contraintes d'intégrité. Le tableau 5.6 montre qu'aucun solveur ne domine ou n'est dominé par les autres. Une ou deux instances mises à part pour chaque solveur, les instabilités numériques observées sont de faible amplitude, et dues à des critères de faisabilité différents entre LocalSolver et GAMS, et à des DAG d'entrée parfois légèrement différents. Il est ainsi rassurant de voir que la robustesse numérique des solveurs est homogène, ce qui permet une entre-validation des résultats et indique les instances sur lesquelles travailler pour améliorer la fiabilité.

LocalSolver vs.	COUENNE	BARON	SCIP
Les deux ont résolu	466	522	519
Aucun n'a résolu	848	769	741
L'un des deux a résolu	663	742	770
Seul LocalSolver a résolu	$146^{(1)}$	$90^{(2)}$	$93^{(3)}$
Seul LocalSolver n'a pas résolu	$51^{(4)}$	$130^{(5)}$	$158^{(6)}$

Tableau 5.7 – *Différences et complémentarité entre LocalSolver et les trois autres solveurs, sur les 1511 instances de la MINLPLib.*

Le tableau 5.7 nous permet d'illustrer deux choses. Premièrement, l'utilisation de stratégies concurrentes donne des gains de performance plus significatifs que la plupart des composants de notre solveur. La stratégie virtuelle prenant le meilleur de LocalSolver-SCIP permet de résoudre 770 instances (contre 768 pour BARON-SCIP), et une stratégie répartissant le temps entre les deux permet d'en résoudre 737. Ceci représente une amélioration nette de 20% d'instances supplémentaires résolues pour LocalSolver, et de 9% pour SCIP. Ces variations sont supérieures à l'apport de la plupart des composants de l'un des solveurs. Ensuite, les différences entre les solveurs sont souvent dues en grande partie à des groupes d'instances où l'un des deux est plus efficace que l'autre. Ainsi, les deux dernières lignes du tableau 5.7 peuvent s'expliquer par une différence de performance sur un ou deux types d'instances, que nous résumons ici.

— (1) : dont 60 instances `smallinv*`,
— (2) : dont 48 instances `smallinv`*,
— (3) : dont 21 instances `smallinv`* et 21 instances `ex`*,
— (4) : dont 11 instances `cvxnonsep_`*,
— (5) : dont 17 instances `cvxnonsep_`* et 31 instances `syn`*/`rsyn`*,
— (6) : dont 12 instances `cvxnonsep_`* et 57 instances `syn*`/`rsyn*`.

Ainsi, entre le tiers et la moitié de la différence entre LocalSolver et les autres solveurs vient d'une ou deux familles d'instances. La même remarque s'applique d'ailleurs si l'on compare les 90 instances résolues par BARON et pas par SCIP (22 instances `ex`*), et les 116 instances résolues par SCIP et pas par BARON (25 instances `rsyn`* et 27 instances `smallinv`*).

Nous terminons cette comparaison par quelques illustrations complémentaires. Pour chaque couple de solveurs, pour les instances résolues par les deux membres du couple, on a tracé dans la figure 5.9 les rapports des temps de résolution. Ces courbes montrent une fois de plus que chaque solveur domine l'autre sur certaines instances, et qu'il reste beaucoup à faire - et beaucoup à gagner - pour obtenir un code dont la performance atteindrait le meilleur des deux.

La figure 5.10 s'intéresse à la robustesse des solveurs sur les petites instances. On y trace la proportion des instances résolues, en fonction du nombre d'instance considérées, et en triant les instances par taille croissante. LocalSolver est le plus performant sur les 300 plus petites instances, et le second après BARON si l'on enlève celles contenant des fonctions trigonométriques. Encore une fois, un phénomène rassurant se produit : les quatre plus petites instances non résolues par LocalSolver

ne sont résolues par aucun autre solveur. De même, les six plus petites instances non résolues par BARON ne contenant pas de fonctions trigonométriques ne sont résolus par aucun autre solveur.

Enfin, pour chaque couple de solveurs, pour les instances résolues par aucun des deux membres du couple, on a tracé dans la figure 5.11 la comparaison des bornes duales finales, avec la formule (5.1). On y a enlevé les instances où la borne imposée (-10^{50} pour COUENNE et BARON, -10^{20} pour SCIP et -10^{15} pour LocalSolver) de l'un des membres domine la borne finale de l'autre, sous peine de comparer des paramètres et non des résultats. Ces courbes sont à prendre avec précaution, puisque nos hypothèses numériques sur les bornes des variables sont plus fortes. Cependant, une analyse des instances où la différence est plus forte montre que la différence s'explique souvent par l'impact positif du *probing* complet. Sur certains groupes d'instances (par exemple `glider`*, `catmix`*, `popdyn`*, `rocket`*, `pinene`* ou encore certains problèmes de *pooling*), c'est le *probing* qui fournit la meilleure borne. De plus, cette borne est généralement difficile à atteindre par branchement. Pour `catmix100` par exemple, où la meilleure solution connue est -0.048, la borne inférieure prouvée par BARON passe de -8.30×10^8 à -8.19×10^8 en 60 secondes, alors que le *probing* complet de LocalSolver permet de prouver la borne -1.45 en 20 secondes. Le point fixe du *probing* (inactif par défaut) remonte cette borne à -0.647 en cinq exécutions.

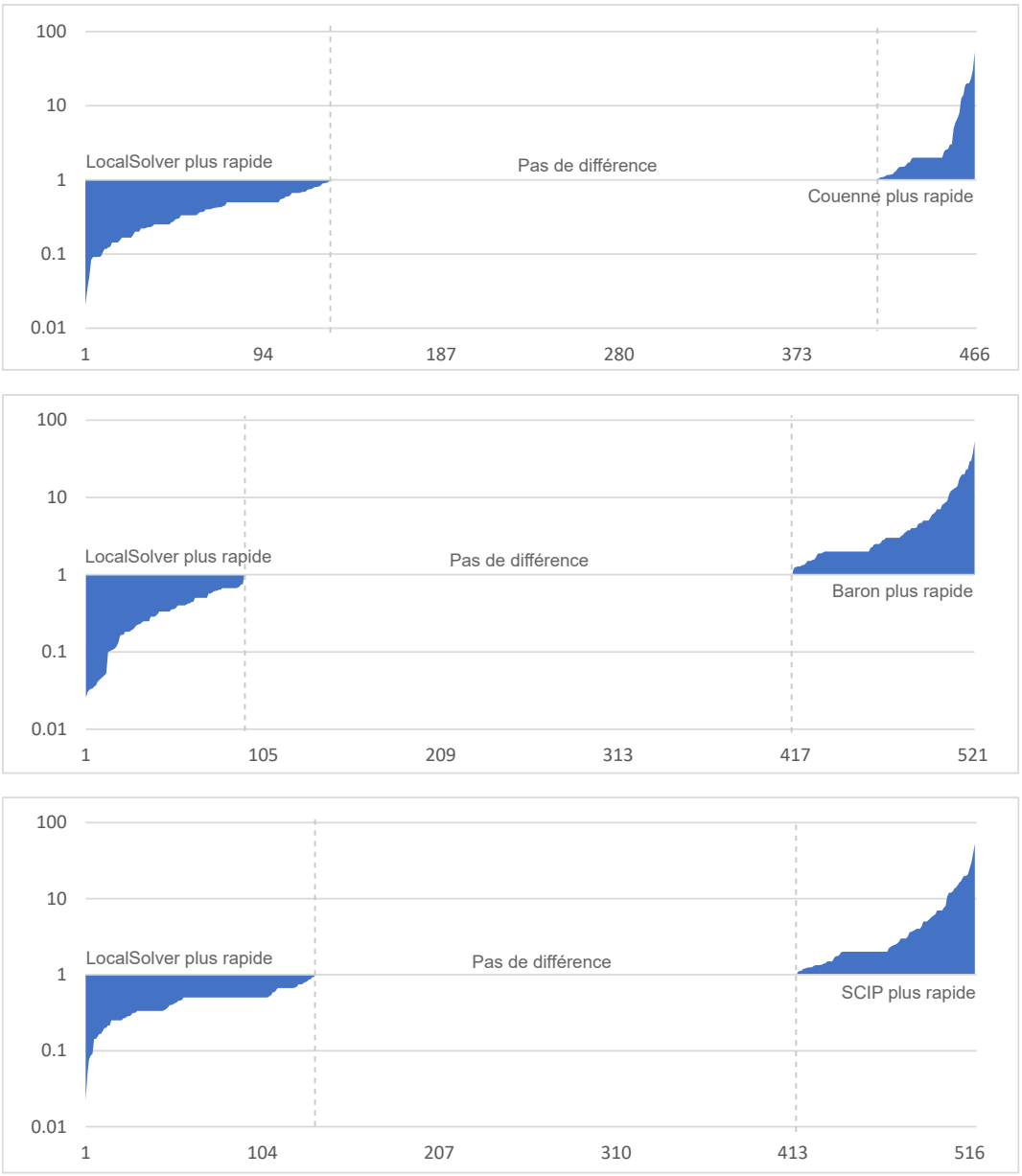

FIGURE 5.9 – *Pour chaque couple de solveurs LocalSolver-COUENNE, LocalSolver-BARON et LocalSolver-SCIP, pour les instances que les deux éléments du couple ont résolus, rapports des temps de résolution.*

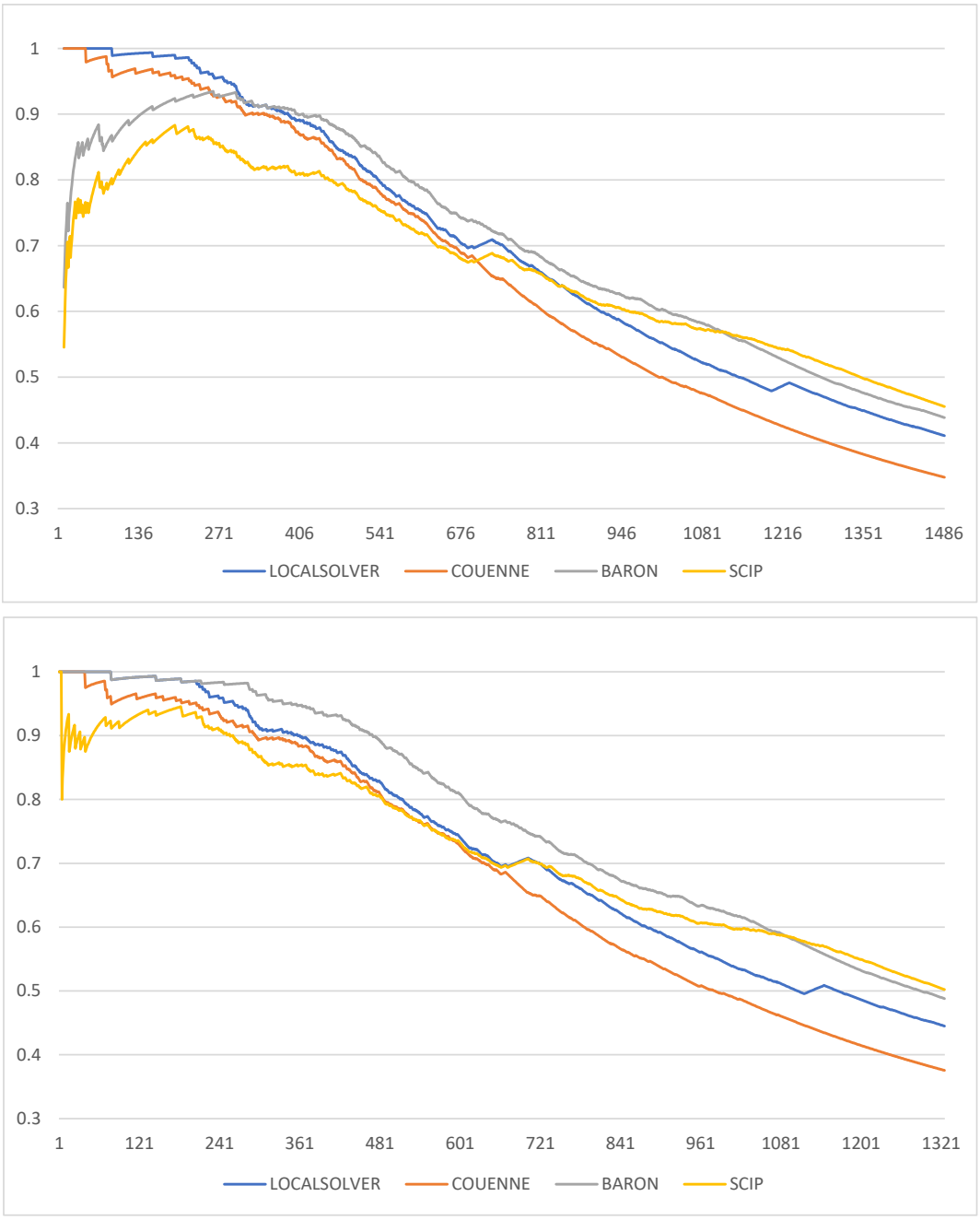

FIGURE 5.10 – *Proportion des instances résolues parmi les k plus petites instances, en fonction de k. En haut, toutes les instances et en bas, instances sans fonctions trigonométriques.*

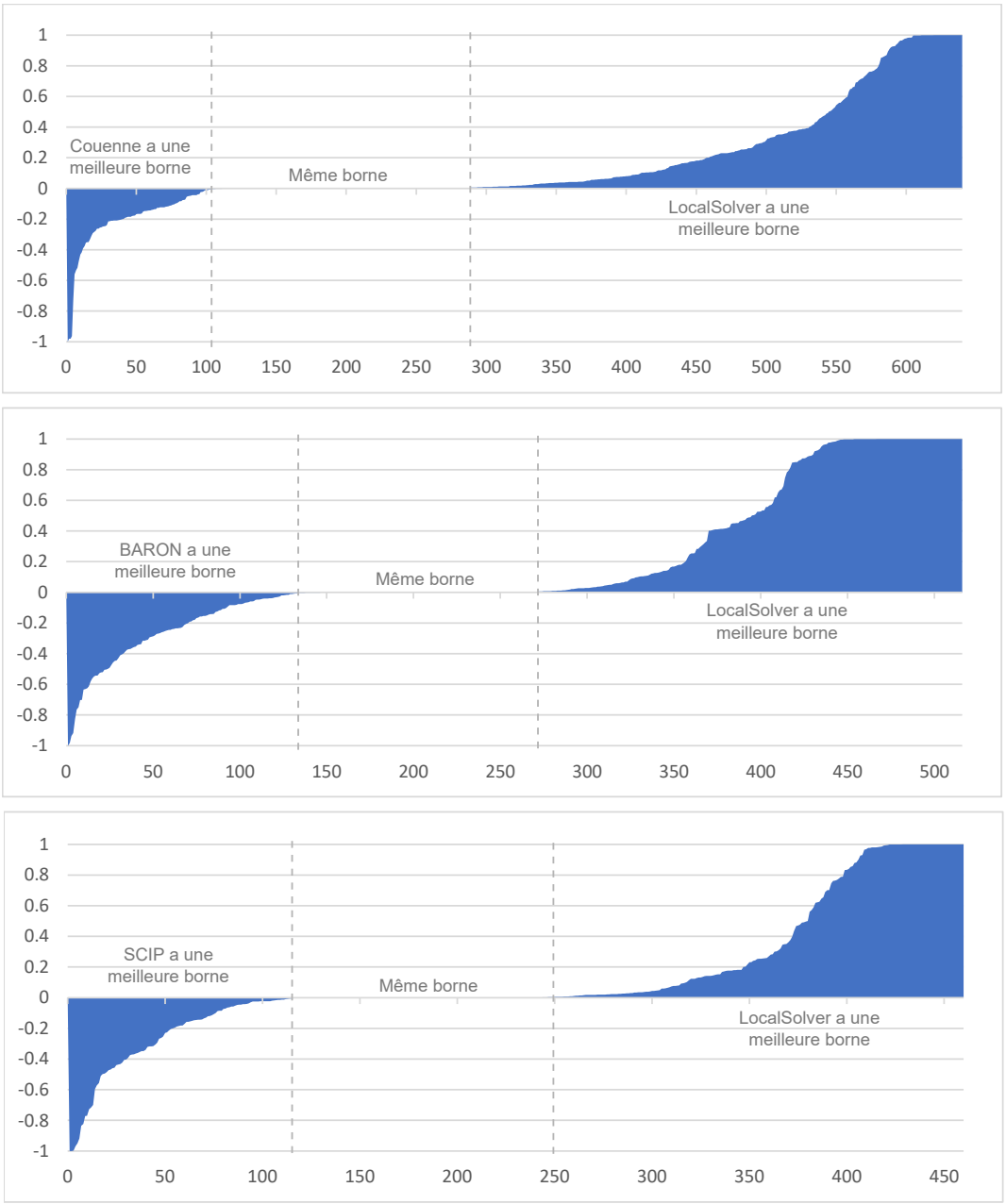

FIGURE 5.11 – *Pour chaque couple de solveurs, pour les instances consistantes, supportées par les deux membres, mais résolues par aucun des deux : comparaison des bornes duales finales, toujours avec la formule* (5.1).

5.2 Analyses et perspectives

Le solveur d'optimisation globale implémenté dans LocalSolver est constitué d'un éventail d'algorithmes, géométriques, numériques ou combinatoires, fonctionnant selon une certaine cinématique, inspirée de l'algorithme *branch-and-reduce* mais reflétant surtout le résultat de nombreuses expérimentations. Les chapitres 2, 3 et 4 de ce manuscrit ont donné une description des composants principaux, mais sans être exhaustifs. Certaines méthodes, dont l'impact est plus modeste, ont été omises. D'autres, plus nombreuses encore, ont été implémentées, réglées, testées, puis finalement abandonnées, car trop coûteuses en temps ou en mémoire, trop spécifiques ou encore inefficaces, voire délétères.

Comme souvent dans le domaine de la recherche, les idées et approches potentielles dépassent largement la capacité de mise en œuvre de ceux qui les imaginent. Ce déséquilibre, loin de se résorber par le travail, a tendance à s'amplifier avec le temps et a pour conséquence qu'une présentation complète des perspectives sur le calcul de bornes inférieures dans LocalSolver nécessiterait plusieurs chapitres. Heureusement, l'expérience acquise pendant ces trois ans sur la transformation de résultats géométriques en code fonctionnel, la prise en compte de la littérature ainsi que les comparaisons numériques effectuées nous permettent d'affiner notre intuition et de présenter une feuille de route adaptée aux besoins de LocalSolver.

Comme la comparaison à d'autres solveurs de l'état de l'art nous l'a montré, les gains les plus spectaculaires de performances sont obtenus en considérant des approches concurrentes. Plus qu'un élément technique à considérer dans le futur, cette idée pose des questions profondes sur l'implémentation d'outils d'optimisation globale, et est un sujet majeur pour le développement d'un code permettant de rassembler le meilleur des outils disponibles. Le *benchmark* officiel pour l'optimisation globale des MINLP non convexes est la MINLPLib. Celui-ci est utile pour évaluer la robustesse d'un solveur, pour contrôler les résultats obtenus ou encore pour comparer deux approches et déterminer leurs forces et faiblesses respectives. Résoudre le plus d'instances possibles de la MINLPLib en un temps donné et répondre aux besoins des clients de LocalSolver sont des objectifs liés, mais pas identiques. Le premier axe de nos perspectives illustre ce fait, en présentant les progrès qu'il reste à faire pour compléter l'adéquation des bornes calculées aux besoins du logiciel.

5.2.1 Des bornes à l'image de LocalSolver

Trois questions, centrales à la philosophie de LocalSolver, nous intéressent pour préparer nos développements futurs : quels sont les problèmes que l'on peut donner à LocalSolver, que celui-ci cherche-t-il à faire sur ces problèmes, et que signifie améliorer les bornes inférieures qu'il retourne ?

Des bornes sur tous les modèles

Si LocalSolver dispose désormais de meilleures capacités en termes de calcul de bornes, les MINLP ne représentent qu'une partie des problèmes d'optimisation rencontrés dans l'industrie. Chercher à calculer des bornes sur tous ces problèmes est

utopique, mais devient réalisable si l'on se limite aux problèmes traités par un formalisme donné. Les fonctions *black-box* mises à part, LocalSolver traite les problèmes représentables par un DAG contenant certains opérateurs. Lorsque ces opérateurs manipulent des valeurs numériques (flottant, entier, booléen), nous sommes capables de calculer des bornes (au moins sur le premier objectif). À l'inverse, dès qu'un opérateur manipule des valeurs ensemblistes (ensemble de valeurs numériques, ordonné ou non), nous n'en sommes pas capables.

La première amélioration à apporter au module de calcul de bornes est donc d'y implémenter le traitement des valeurs ensemblistes. Ceci demande de prendre en compte d'autres types de variables dans le modèle factorisé, portant leur compte à cinq : flottant, entier, booléen, ensemble et liste. Un ensemble et une liste (définis par une constante k) sont des variables représentant respectivement un sous-ensemble non ordonné et un sous-ensemble ordonné de $\{0, \cdots, k-1\}$. Plusieurs opérateurs y sont alors définis : `count` permet d'en obtenir la taille, `contains`(e) permet de savoir si un élément e y est présent, `disjoint`$(s_1, \cdots, s_m)$ permet de savoir si les ensembles ou listes $(s_1, \cdots, s_m)$ sont disjoints et `partition`$(s_1, \cdots, s_m)$ si ces derniers forment une partition de leur domaine de définition. Enfin, les listes possèdent aussi l'opérateur `at`(i), permettant d'obtenir la valeur du i-ème élément. De la même façon que nous avons implémenté les opérations élémentaires $+, \times, -, \div, \exp, \max, \cdots)$ sur les variables numériques dans le modèle factorisé, il faut implémenter leurs homologues ensemblistes. Nous pouvons y généraliser le calcul de relaxations convexes, les techniques de réduction de bornes et même la recherche arborescente. L'hybridation avec d'autres techniques est cependant nécessaire pour obtenir des résultats de l'état de l'art, comme par exemple les méthodes de génération de colonnes [76, 135, 117].

Une autre faiblesse des bornes obtenues par le module dual est qu'elles se limitent au premier objectif. Dans le *benchmark* industriel présenté en partie 5.1.2, plus d'un quart des instances présentent deux objectifs ou plus. Dans ce cas, c'est l'optimisation globale du premier objectif qui a été évaluée. Le traitement de ces problèmes multi objectifs dans LocalSolver étant lexicographique, la généralisation du modèle factorisé à ceux-ci est uniquement technique. En effet, lors de la factorisation du modèle, il est possible de stocker non pas un, mais plusieurs objectifs $(f_1, \cdots, f_m)$, tous linéaires ou quadratiques dans notre cas. La résolution à l'optimum sur un objectif i, suivi de l'ajout de la contrainte $f_i(x) = f_i^*$ permet d'attaquer le suivant, et donc de traiter l'optimisation globale multi objectif lexicographique.

Des solutions sur tous les modèles

La validité d'une borne duale repose sur la notion de certificat (certificat de KKT pour un problème convexe par exemple, ou arithmétique d'intervalle validée pour la propagation), celle d'une borne primale se vérifie par son évaluation dans le DAG. Si ce second cas est en théorie plus simple, la pratique montre que les bornes duales peuvent être calculées systématiquement, comme par exemple en suivant l'approche retenue dans cette thèse. En réalité, aucune heuristique primale ne peut obtenir de solution admissible sur toutes les instances où LocalSolver sait désormais calculer des bornes. Une borne inférieure, même de bonne qualité, est de plus peu pertinente si elle ne vient qualifier aucune solution. Ainsi, la réponse à la seconde question posée

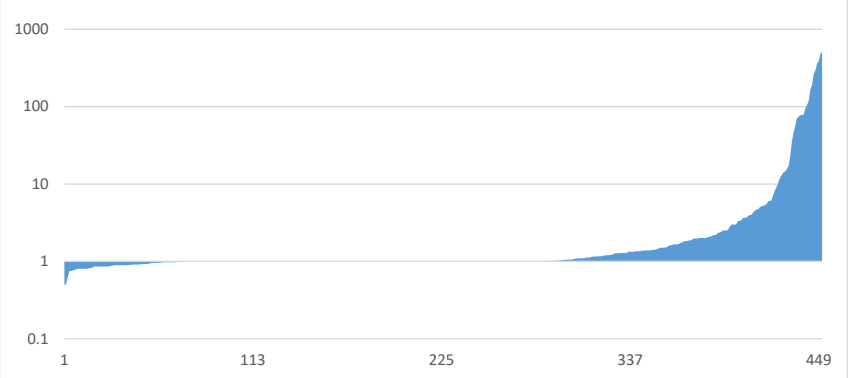

FIGURE 5.12 – *Sur les 760 instances où l'heuristique primale de points intérieurs s'active, 451 sont résolues à l'optimum avec et sans. Cette courbe représente les rapports des nombres de nœuds nécessaires à leur résolution : nœuds visités sans l'heuristique sur nœuds visités avec l'heuristique.*

plus tôt, sur ce que l'on attend de LocalSolver, est un critère bi-objectif primal-dual utilisant l'ordre lexicographique. La raison d'être du logiciel est d'abord de trouver des solutions admissibles, et ensuite de calculer des bornes les qualifiant.

Cette motivation philosophique des heuristiques primales se complète par des arguments plus pragmatiques liés au calcul de bornes. Comme les résultats respectifs de LocalSolver, Couenne, SCIP et BARON sur la MINLPLib l'ont montré, la performance primale (nombre de solutions trouvées) et la performance globale (le nombre d'instances fermées) sont corrélées. On peut illustrer ce principe plus en détail par une petite expérimentation numérique. LocalSolver dispose, depuis la version 9.0, d'une méthode de points intérieurs, utilisée en tant qu'heuristique primale pour les problèmes non linéaires. Nous avons intégré celle-ci au module dual (l'algorithme est donc déroulé sur le modèle factorisé), après la résolution de la relaxation convexe du *root node* pour profiter d'une solution initiale de qualité. On considère alors les 760 instances de plus petite taille où cette heuristique s'active. Celle-ci n'a pas d'impact sur le nombre d'instances résolues (moins de 1% de différence), mais influence le temps de résolution, comme illustré sur la figure 5.12. On en conclut que l'intégration d'heuristiques primales améliore le temps de résolution. Ceci s'explique par le fait que la solution optimale est trouvée plus tôt (parfois dès le *root node* par le points intérieurs), mais surtout par le fait que la donnée d'une solution primale permet de resserrer les bornes (par l'inférence sur l'objectif, par le *probing* ou encore par la SBBT. Si accélérer le temps de résolution à l'optimum n'est pas notre objectif (celui-ci est de toute façon bien trop important pour la plupart des instances), resserrer les bornes des variables au début de la recherche améliore les bornes duales obtenues.

Plus généralement, les heuristiques primales complémentaires à la recherche locale sont bénéfiques. Le cas le plus notable concerne la résolution de programmes linéaires en nombres entiers, sur lesquels la recherche locale n'est pas performante. Dans ce cas, LocalSolver reformule le DAG en un PLNE, dans l'optique d'y appliquer des heuristiques primales de type *feasibility pump* [142], parfois plus efficaces que

la recherche randomisée. Cette stratégie serait à généraliser au cas des programmes non linéaires (une autre classe de problèmes sur lesquels la recherche locale est en difficulté) et à leur version mixte, les MINLP. Le sujet est alors vaste et une large littérature existe, dont quelques exemples sont [69, 42, 113]. Comme toute recette numérique, ces heuristiques doivent cependant être calibrées sur les besoins de LocalSolver, en particulier sur la taille des instances résolues et sur les opérateurs présents.

Amélioration des bornes

Une fois le module dual capable de traiter tous les problèmes supportés par LocalSolver, se pose la question de l'amélioration des bornes fournies, et donc de ce que sont de « meilleures » bornes. On distingue alors deux types d'instances, selon que le nombre de non-zéros soit « grand » ou « petit ». Une instance de petite taille est une instance que l'on s'attend à résoudre à l'optimum global, grâce à l'exhaustivité de la recherche arborescente. Des exemples sont le *benchmark* souvent utilisé de nos 2211 plus petites instances, ainsi que la MINLPLib, si l'on lui retire le tiers des plus grosses instances. La limite se situe aujourd'hui à environ 300 non-zéros dans le graphe, taille en dessous de laquelle LocalSolver est capable de résoudre plus de 95% des instances. Au-delà, le coût en temps et en mémoire de la recherche arborescente explose trop vite.

La résolution des petites instances à l'optimum global revêt un intérêt certain. Ceci renvoie une première impression positive du logiciel lors d'un essai, permet de tester différents modèles, et de relever des erreurs lors du développement. Tout ceci peut se résumer en un meilleur confort de travail pour les ingénieurs en optimisation et en recherche opérationnelle. Enfin, pour les problèmes naturellement de petite taille, le module dual constitue une heuristique primale comme une autre, capable de fournir des solutions de qualité. Toutes ces raisons nous poussent à améliorer la recherche exhaustive, afin de repousser l'explosion du temps de calcul. Nous pouvons citer deux axes de recherche, peu explorés pendant cette thèse, permettant d'avancer dans cette direction. Le premier concerne les règles de branchement utilisées, présentées dans notre cas dans la partie 4.1.2. Notre approche est élémentaire et ne reflète pas les idées de l'état de l'art sur le sujet. Les améliorations possibles sont d'une part l'utilisation de fonctions de scores plus avancées, qui sont à la base de l'*hybrid branching* [16], et d'autre part le *reliability branching* [19, 36], approximant le *strong branching* par des opérations dont le coût est contrôlé. Le second concerne la détection et la prise en compte des symétries du DAG initial lors du branchement, et plus généralement des isomorphismes de graphe qui fournissent des problèmes équivalents, mais parfois plus rapide à résoudres. La réduction des symétries pour un DAG donné peut s'effectuer de différentes manières, soit par l'ajout de contraintes statiques, soit de façon dynamique lors du branchement. L'exemple le plus simple est le cas de deux variables x et y interchangeables : l'ajout de $x \leq y$ permet de briser la symétrie en réduisant l'espace de recherche. Des références sur les motivations et l'intérêt de cette approche sont par exemple [110, 112]. Plus généralement, les sujets des formulations et reformulations des problèmes d'optimisation n'a pas été exploré lors de cette thèse, mais constitue une source d'améliorations importantes.

Comme nous l'avons vu dans les résultats de la partie 5.1.1, la résolution d'un unique nœud est déjà une opération coûteuse. De plus, le coût mémoire pour stocker les nœuds ouverts ne permet pas d'en énumérer un grand nombre. Sans une révolution des ordinateurs quantiques ou une démocratisation des supercalculateurs, il ne faut donc pas se reposer sur le partitionnement du domaine des variables pour améliorer les bornes sur les instances de grande taille. Une première approche pour contourner la difficulté de l'énumération est de considérer des règles de partitionnement différentes. Le branchement utilisé dans les solveurs d'optimisation globale et dans les solveurs de PLNE se contente de partitionner le domaine d'une variable, mais d'autres approches sont possibles. Par exemple, le *probing* sur l'objectif, présenté en partie 4.3.1 et fournissant la borne PROB-obj dans la partie 5.1.1, peut être amélioré. En particulier, une fois les bornes de l'objectif restreintes à un sous-domaine et cette réduction propagée, si l'inconsistance n'est pas détectée, le *probing* s'arrête. Résoudre la relaxation convexe correspondante reviendrait alors à effectuer un partitionnement selon la valeur de l'objectif, et peut améliorer la borne duale à l'aide d'un petit nombre de nœuds visités. Une autre idée de partitionnement non conventionnel est donnée dans l'exemple suivant. On se donne un problème où toutes les variables ont pour bornes $[0, M]$, avec $M \gg 1$. Si l'on ne peut pas inférer de meilleures bornes sur les variables, les relaxations convexes peuvent être mauvaises. Une heuristique efficace est alors de réduire arbitrairement ces domaines à $[0, m]$, avec $m \ll M$. Si ce sous-problème est inconsistant, la coupe $\sum_i x_i \geq m$ peut être ajoutée au problème. Inversement, s'il ne l'est pas, on peut partitionner le domaine en deux, avec d'un côté, les bornes $[0, m]$, et de l'autre, les bornes $[0, M]$ et la coupe $\sum_i x_i \geq m$. Ce type de partitionnement fonctionne sur un grand nombre de problèmes où les bornes initiales des variables sont peu serrées et pas suffisamment améliorées par les techniques de réduction utilisées. Dans les deux cas décrits ci-dessus, la coupe est généralement profonde, et la réduction de toutes les bornes des variables resserre largement les relaxations convexes obtenues. Par exemple, l'instance difficile `minlphix` de la MINLPLib n'a été résolue que par Couenne en une minute. L'utilisation de cette heuristique avec $m = 10^5$ permet de résoudre le problème en moins de une seconde, et en 18 nœuds. Enfin, et comme l'ont montré les figures 5.3 et 5.4, le calcul de bornes duales « peu chères », et donc rapide à obtenir, ne doit pas être négligé, même si elles ne permettront pas de résoudre plus d'instances à l'optimum global.

Au-delà de la taille des instances, la quantité de structure que l'on est capable d'exploiter à l'intérieur de celles-ci reste le facteur le plus important. Une prise en compte plus fine des opérateurs présents et de leurs interactions permet d'améliorer les bornes sur toutes les instances, quelle que soit leur taille. Pour cette raison, la prochaine partie de nos perspectives se concentre sur l'axe d'amélioration principal : exploiter toujours plus de géométrie à l'intérieur du modèle factorisé.

5.2.2 Géométrie du modèle factorisé

Le dénominateur commun de toutes les bornes présentées dans la partie 5.1.1 est la géométrie du modèle factorisé. Plus la connaissance de cette dernière est précise, plus les bornes sont serrées. L'objectif de cette partie est d'entamer la liste des propriétés géométriques inexploitées dans le modèle factorisé. La tâche étant déjà

ambitieuse, on se limite à la forme standard utilisée aujourd'hui dans LocalSolver : un programme linéaire ou un programme quadratique sous contraintes quadratiques, couplé à un ensemble de relations non linéaires entre variables.

Géométrie de l'intégrité

Les contraintes d'intégrité sont utilisées pour arrondir les bornes dans la propagation et l'inférence, et dans la recherche arborescente pour partitionner le domaine des variables. La prise en compte de cette intégrité dans les relaxations convexes utilisées nous semble être le principal composant manquant du module dual implémenté. Les résultats de SCIP, dans la partie 5.1.3, sont les meilleurs justement grâce à une gestion fine de l'intégrité, *via* des techniques de *presolve* dédiées (techniques de resserrage de coefficients ou de réduction de bornes avancées) et de génération de coupes. Dans [102], la désactivation des relaxations de type PLNE dans BARON est le composant dégradant le plus les performances. La géométrie et l'interaction des polyèdres et contraintes d'intégrité est étudiée de façon intensive depuis plusieurs décennies, d'un point de vue théorique mais aussi pratique, avec des implémentations de plus en plus performantes au sein des solveurs de PLNE. La performance de ces derniers est telle que la nature de la prise en compte de l'intégrité dans le module dual pose question. L'approche idéale pour améliorer la gestion de l'intégrité est d'y réimplémenter les techniques de *presolve* et de génération de coupes. Au long terme, c'est ainsi que les gains les plus importants pourront être obtenus, même si le travail nécessaire est conséquent.

L'approche la plus facile à mettre en place, et qui apporte les gains les plus immédiats, est de conserver les contraintes d'intégrité dans les relaxations convexes linéaires. Chaque nœud nécessite alors la résolution d'un PLNE, plus long à résoudre mais fournissant une meilleure borne. C'est l'approche utilisée dans Antigone [126], et BARON peut utiliser ce type de relaxation [102]. On peut motiver cette dernière par quelques considérations sur la MINLPLib. Par exemple, deux familles de problèmes contiennent 96 instances (les instances `syn`* et `rsyn`*). LocalSolver en résout environ un tiers en une minute, BARON les deux tiers et SCIP la totalité. Après inspection de ces problèmes, la relaxation initiale couplée aux contraintes d'intégrité fournit un écart d'optimalité de l'ordre du pourcent, voire du dixième de pourcent, et nécessite un temps de résolution de l'ordre de la seconde pour un solveur de PLNE de l'état de l'art. Utiliser des relaxations de PLNE permettrait donc de combler les deux tiers du retard de LocalSolver sur SCIP sur ce *benchmark* officiel. Le point faible de cette approche est cependant sa difficulté d'intégration dans la recherche arborescente, puisqu'il n'existe pas de technique permettant de démarrer à chaud les algorithmes de résolution des PLNE. En particulier, si le problème initial contient d'autres sources de non-convexités que les contraintes d'intégrité, il n'y a pas toujours d'intérêt à résoudre ces relaxations à l'optimum et il est même probable que beaucoup de temps soit perdu au début de la recherche.

La solution aux problèmes posés par la résolution de PLNE à chaque nœud est d'interrompre prématurément cette résolution. L'idée est alors de résoudre la relaxation linéaire - comme ce qui est actuellement fait dans LocalSolver dans le cas du modèle factorisé linéaire - de laisser le solveur PLNE dérouler ses méthodes de

séparation et éventuellement de laisser celui-ci visiter un nombre prédéfini (10, 100, 1000, $\cdots$) de nœuds. Dans ce cas, il est possible de récupérer les coupes générées, afin de les utiliser pour les nœuds suivants par exemple. De plus, la recherche arborescente des solveurs PLNE a été améliorée par des décennies d'ingénierie logicielle, et est donc particulièrement rapide. Résoudre une dizaine de nœuds a généralement un coût inférieur à celui de résolution du *root node* par exemple. Enfin, puisque toute réduction des bornes des variables sur une relaxation du problème est valide sur le problème lui-même, il est possible de laisser le solveur PLNE effectuer son *presolve*, puis de récupérer ensuite les réductions de bornes obtenues par les téchniques dédiées à l'intégrité.

LocalSolver dispose d'un tel solveur de PLNE, et le module dual utilise le simplexe dual qui y est présent. Si la performance de ce dernier est satisfaisante (de l'ordre de celle des meilleurs solveurs libres, tels que CLP [5]), les autres modules sont en cours de débogage. Ainsi, ni le *presolve* linéaire, ni les coupes, ni la recherche arborescente ni les heuristiques primales n'ont pu être utilisées pour ce travail. Au moment où nous écrivons ces lignes, des ingénieurs de LocalSolver sont en train de résoudre ces problèmes, qui diminuent à vue d'œil. L'activation du *presolve* sur les relaxations linéaires promet une amélioration nette de la performance de résolution de celles-ci, de l'ordre de plusieurs dizaines de pourcents. Comme nous venons de le voir, les coupes et recherche arborescente de ce solveur permettront de fournir de meilleures bornes dès le *root node*, et, enfin, les heuristiques primales linéaires seront utilisées en sous-routine par des heuristiques primales plus complexes, adaptées aux MINLP.

Géométrie linéaire, bilinéaire et quadratique

L'opérateur non linéaire le plus présent dans les modèles d'optimisation est la multiplication. Les contraintes bilinéaires ($z = xy$), unaires quadratiques ($y = x^2$) et quadratiques sont ainsi les contraintes non linéaires les plus fréquentes dans notre modèle factorisé. Leur gestion est un facteur clé dans la performance d'un solveur d'optimisation globale, et une littérature riche existe sur le sujet (voir la partie 2.3.1). Dans notre cas, les contraintes quadratiques sont remplacées par des contraintes linéaires et des couplages bilinéaires (auquel cas des relaxations linéaires seront utilisées), ou alors elles sont laissées telles-quelles (et des relaxations quadratiques compactes sont alors utilisées). Pour bien comprendre le problème, il faut commencer par se rendre compte que dans notre implémentation, aucune de ces deux approches ne domine l'autre. La première étape d'affinage de la géométrie du modèle factorisé est ainsi d'obtenir le meilleur des deux mondes.

Les différences entre les deux façons de relâcher les contraintes ont été expliquées à la fin du chapitre 2. L'erreur de convexité dépend, en plus des bornes des variables, du nombre de termes bilinéaires pour les relaxations linéaires, et du nombre de variables dans ces termes pour les relaxations quadratiques. Comme les exemples donnés dans les tableaux 2.9 et 2.10 le montrent, ceci implique qu'aucune des deux relaxations ne domine l'autre. Le cas des techniques de réduction de bornes est plus subtil. Les formules de propagation et d'inférence sur les sommes quadratiques, données en partie 4.2.1, exploitent plus de structure que la propagation et l'inférence

utilisant les contraintes linéaires et termes bilinéaires. Cependant, la borne FBBT-D, utilisée dans la section sur les résultats numériques, n'est pas systématiquement meilleure lorsque les contraintes quadratiques sont conservées. De même, le *probing* sur les variables (la borne PROB-var) peut être plus serrée dans le cas linéaire, y compris dans le cas où la borne de propagation FBBT-D était initialement moins bonne. L'explication à ces deux observations contre-intuitives est que l'on ne stocke pas de bornes sur les structures intermédiaires dans le cas quadratique (en particulier, les termes de type xy et x^2 des expressions quadratiques n'ont pas de domaine de définition, contrairement aux variables). Ainsi, si un domaine $[l, u]$ est inféré sur un terme xy depuis une contrainte quadratique, il n'est pas garanti que celui-ci puisse d'inférer à son tour sur les bornes de x et de y. Dans le cas contraire, si xy est présent dans plusieurs contraintes quadratiques, un intervalle $[l, u]$ plus serré que les bornes de propagations naturelles du produit peut être prouvé dans une contrainte, mais ne sera pas utilisé pour la propagation et l'inférence sur une autre, comme déjà remarqué dans [125]. La situation est similaire pour le *probing* : puisque celui-ci est effectué sur les variables, il ne cherche pas à resserrer les bornes de xy dans le cas quadratique. Cette observation montre qu'il peut être avantageux d'effectuer le *probing* sur des structures intermédiaires autres que les variables, comme ici sur les termes bilinéaires.

La réponse à ces sous-optimalités est technique. La difficulté consiste à trouver une structure de données, pour les expressions quadratiques, permettant à la fois de générer nos deux types de relaxations et d'y appliquer le meilleur des bornes de propagation et d'inférence. La partie concernant les relaxations est relativement simple, et permettra même d'améliorer les relaxations convexes dans le cas linéaire. Par exemple, un DAG représentant une fonction quadratique $x^TQx + c^Tx + c$ est aujourd'hui transformé en une contrainte linéaire avec les couplages bilinéaires correspondants. Dans ce cas, les termes linéaires et quadratiques en une certaine variable x ne seront pas regroupés. Par exemple, $x^2 + x$ sera reformulé en $y + x$, $y = x^2$. La structure de données actuelle des contraintes quadratiques permet de voir qu'une meilleure reformulation est plutôt $z - 1/4$, $z = y^2$, $y = x + 1/2$, puisque fournissant une relaxation plus serrée. Une fois que les différentes relaxations convexes pourront être obtenues à partir d'un seul modèle factorisé, nous pourrons par exemple utiliser les relaxations quadratiques compactes pour générer des coupes dans le cas où l'on utilise des relaxations linéaires. Dans le solveur non linéaire, ajouter les contraintes linéaires et enveloppes de Mc Cormick correspondant à la linéarisation des contraintes quadratiques permettra aussi de resserrer la relaxation. Dans ce cas, l'augmentation significative du nombre de variables et de contraintes ne sera cependant pas toujours bénéfique. Une difficulté supplémentaire est donc de pouvoir maintenir l'utilisation de relaxations compactes, c'est-à-dire utilisant uniquement un sous-ensemble des variables et des contraintes du modèle factorisé. L'unification des deux cas pour la propagation et l'inférence de bornes est plus délicat. Obtenir une structure de données et l'algorithmique permettant d'atteindre le meilleur des deux cas est, d'après nos réflexions actuelles, impossible sans détériorer le temps d'exécution de la propagation et de l'inférence. Ce temps étant suffisamment faible à l'heure actuelle, et des améliorations de bornes étant de toute façon prioritaires, la détérioration nous semble cependant acceptable.

Une fois les traitements unifiés dans le cas des relaxations linéaires et non linéaires, un certain nombre d'autres techniques permettent de compléter et d'améliorer le traitement des expressions quadratiques. Les améliorations possibles à la propagation et à l'inférence de bornes présentées en partie 4.2.1 en sont un exemple. Les techniques de renforcement des relaxations linéaires présentées en partie 2.3.1 en sont d'autres. Ceux-ci comprennent l'implémentation de la désagrégation des produits, l'élimination de certains termes bilinéaires ou encore la génération de coupes de type *reformulation-linearization technique*. Un autre axe de progrès est la généralisation au cas quadratique de certaines actions de *presolve* aujourd'hui limitées au cas linéaire. Enfin, pour conclure sur ce sujet, la géométrie des contraintes quadratique est complexe, et n'est jamais traitée de façon optimale dans le cas général (à n variables, sans hypothèses de convexité). Cette remarque a une conséquence importante : au-delà du cas général, qui permet de traiter toutes les contraintes quadratiques, la prise en compte de cas particuliers est souhaitable, puisque plus faciles à caractériser. C'est l'idée derrière le traitement optimal des fonctions quadratiques à deux variables pour la propagation et l'inférence de bornes dans SCIP [160], et de l'utilisation des enveloppes convexes exactes pour une certaine classe de fonctions quadratiques à trois variables dans GloMIQO [125]. L'intérêt de ces cas particuliers est montré par le succès de GloMIQO, et pourrait se généraliser. Au même titre qu'une bibliothèque de fonctions non linéaires élémentaires est utilisée, une cartographie des contraintes quadratiques de petite dimension ou de forme parmi les plus courantes permettrait de caractériser la géométrie des problèmes avec plus de précision.

Bibliothèque géométrique

Lorsque les différents types de contraintes pris en compte dans le modèle factorisé permettent de recouvrir le DAG du problème, la factorisation du modèle est une reformulation. Dans le cas contraire, une première relaxation est effectuée lors de cette étape. La seule couverture des opérateurs du DAG par ceux présents dans le modèle factorisé n'est cependant pas l'objectif final. En effet, plus les sous-structures utilisées pour recouvrir le graphe seront complexes, plus le modèle factorisé sera proche de ce dernier. Au-delà du nombre de variables et de contraintes qui augmenteront moins, l'ensemble des techniques de resserrage de bornes et de relaxations convexes seront plus précises. C'est un long cheminement, ici entamé par la prise en compte des expressions quadratiques dans le modèle factorisé. Pour le moment, nous supportons trois familles de contraintes dans le modèle factorisé : les contraintes n-aires (linéaires, quadratiques, min et max), les contraintes unaires ($y = f(x)$) et les contraintes bilinéaires ($z = xy$). L'extension de cette classification, ainsi que la diversification de chacune des catégories, permet de capter des structures géométriques plus complètes, et donc de fournir des meilleures bornes.

La prise en compte des contraintes quadratiques est une grande avancée (même si loin d'être achevée, comme expliqué au paragraphe précédent) dans la catégorie des contraintes n-aires. Pour aller plus loin dans cette direction, la prise en compte des contraintes polynomiales de degré supérieur à 2 semble cependant trop ambitieuse pour en valoir la peine. La forme générique d'une expression cubique est plus complexe que celle d'une expression quadratique, et y implémenter des méthodes

dédiées de propagation et d'inférence de bornes serait une tâche fastidieuse. Par ailleurs, d'autres types de contraintes sont plus fréquentes que les polynômes, comme les contraintes basées sur une norme, par exemple de la forme $\|Ax+b\|_p \leq a$, $p > 0$. Les contraintes coniques du second ordre sont celles de la forme $\|Ax+b\|_2 \leq a^T x+b$, et sont le prochain type de contraintes n-aires à prendre en compte. En particulier, certaines contraintes quadratiques, en apparence non convexes, peuvent se reformuler comme tel. L'expression donnée ci-dessus est convexe, et peut donc être utilisée dans un solveur non linéaire, à condition que celui-ci sache gérer la non différentiabilité qui y est présente. Lorsque de telles contraintes sont présentes dans le modèle, l'impact de leur prise en compte sur le temps de résolution est souvent majeur, comme le montrent par exemple les résultats numériques obtenus dans [43]. Enfin, les expressions signomiales sont une des rares classes de contraintes n-aires non reformulées dans certains solveurs globaux, comme par exemple dans [126]. Ce sont les expressions de la forme $\sum_i c_i \prod_j x_j^{a_{ij}}$, utilisées par exemple en programmation géométrique (voir la section 4.5 dans [56]), et en ingénierie chimique. Leur pertinence reste cependant modérée, puisqu'elles ont été prises en compte pour répondre à un besoin dans le domaine de l'ingénierie chimique, et sont rarement présents dans les modèles des clients de LocalSolver.

Les contraintes unaires et binaires présentes dans le module dual de LocalSolver forment l'ensemble minimal nécessaire à la couverture des DAG des MINLP les plus fréquents. La liste complète des contraintes supportées est : $y = \exp x$, $y = \log x$, $y = \sin x$, $y = \cos x$, $y = \tan x$, $y = \sqrt{x}$, $y = x^{2k}$, $y = x^{2k+1}$, $y = x^a$, $a \notin \mathbb{Z}$, $y = (x = a)$, $y = (x \leq a)$, $y = (x \geq a)$, $y = a/x$ et $z = xy$. Si cette liste permet effectivement de couvrir tous les DAG souhaités, certaines fonctions non linéaires sont reformulées à l'aide de variables intermédiaires, ou par modification de leur forme initiale. Des exemples de telles fonctions sont $y = |x|^a$, $y = 1/x^a$ ou encore $z = x/y$. Le traitement explicite de ces dernières (en particulier le fait de ne pas reformuler les divisions en des produits) permettrait d'améliorer les performances du module dual sur les problèmes où elles apparaissent. Plus généralement, le module de calcul de bornes de LocalSolver serait d'autant plus efficace si toutes les fonctions unaires ou binaires présentes dans un modèle d'optimisation étaient prises en compte. Une approche possible pour avancer dans cette direction est d'analyser les DAG de tous les modèles disponibles, et de déterminer chaque fonction unaire ou binaire non prise en compte, ainsi que sa fréquence d'apparition. Cette approche a déjà été utilisée dans BARON pour améliorer la détection les régions de convexité et de concavité du DAG, en particulier pour l'utilisation de solveurs non linéaires [146, 101]. Les fonctions non linéaires ainsi supportées forment une bibliothèque d'informations géométriques. Chaque type de relation implémente plusieurs classes de fonctionnalités : fonctions de *presolve* (fonctionnalités de substitution de variables, résolution des auto-couplages), fonctions de propagation et d'inférence de bornes, fonctions de génération de relaxation convexes et d'intégration dans les solveurs (évaluation et différentiation de la relaxation non linéaire par exemple). À titre d'exemples, nous pouvons citer le début de la liste des fonctions non supportés, sans ordre particulier : les fonctions $y = |x|^a$, $y = 1/x^a$ et $z = x/y$, les plus importantes, quelques autres contraintes binaires : $z = x^a y^b$, $z = x^2/y$, $z = x/y^2$ et $z^2 = xy$, des fonctions unaires supplémentaires : $y = x\mathrm{e}^x$, $y = x \log x$ ou encore $y = \mathrm{e}^{-x^2}$, et, enfin, quelques fonctions de plus grande arité : $t = xyz$ ou $t = xy/z$. Enfin, des

cas particuliers de fonctions quadratiques à deux ou trois variables, comme expliqué au paragraphe précédent, seraient bénéfiques, tout comme des cas particuliers de polynômes à une ou deux variables.

L'un des facteurs qui a fait le succès de l'approche utilisant des relaxations non linéaires est l'utilisation de relaxations compactes. Ces relaxations sont inspirées de la méthode αBB, et s'appliquent au modèle factorisé, contrairement à la méthode originelle. Comme expliqué en introduction, il serait possible de généraliser cette approche, en exploitant dans le modèle factorisé des contraintes n-aires stockée sous la forme d'un petit DAG. En particulier, pour les contraintes non linéaires convexes, cette approche est séduisante, car elle ne modifie pas la taille du problème, et n'introduit pas d'erreur de convexité. Dans le cas général, le couplage de la factorisation du modèle et de l'utilisation de relaxations de type αBB sur les DAG des contraintes permettrait de resserrer les relaxations convexes, de la même manière que les relaxations quadratiques compactes que nous avons utilisées ont parfois resserré la relaxation linéaire. L'utilisation d'un solveur non linéaire demande de plus de porter une grande attention à la taille des relaxations utilisées, pour des questions de passage à l'échelle de leur résolution. L'utilisation d'un modèle hybride, avec un sous-graphe entièrement factorisé et un ensemble de petit graphes non factorisés pour certaines contraintes, permet de contrôler la taille des relaxations générées, et offre un degré de liberté supplémentaire pour obtenir des bornes inférieures.

Enfin, une dernière idée pour l'extension de la géométrie du modèle factorisé serait de ne plus séparer les contraintes selon leur forme, mais selon la structure physique qu'elles décrivent. En particulier, certaines structures combinatoires détectables dans le DAG du modèle concernent plusieurs contraintes et variables, et apporteraient de l'information sur la géométrie de l'ensemble admissible. Par exemple, les contraintes de type capacité, ou sac-à-dos, sont intensément utilisées dans les solveurs de PLNE pour obtenir de l'information sur l'ensemble admissible. D'autres exemples de structures plus complexes seraient les flots, les assignations, les couplages, les contraintes de couverture, de *packing* ou encore des structures de voyageur de commerce et de tournées de véhicules si l'on anticipe sur la prise en compte des variables ensemblistes dans le module dual. Cette approche offre la possibilité d'utiliser tous les résultats obtenus en recherche opérationnelle sur ces problèmes, en particulier certaines relaxations combinatoires (par exemple [92, 77]) ou résultats d'approximation.

5.2.3 Relaxations non linéaires

L'algorithme utilisé pour la résolution des relaxations non linéaires repose sur la transformation du problème de calcul de bornes inférieures en celui de la minimisation non contrainte d'une fonction convexe différentiable. C'est une approche générique, qui ne fait aucune hypothèse sur la géométrie des contraintes présentes. Un axe d'amélioration possible est alors d'utiliser une approche dédiée à chaque type particulier de problèmes. Ainsi, si la relaxation non linéaire du modèle factorisé est un programme quadratique, un programme quadratique sous contraintes quadratiques ou encore un programme conique du second ordre, l'utilisation d'un algorithme dédié à cette classe de problème fournira toujours des améliorations conséquentes sur

le temps de résolution. Si de tels solveurs ne sont pas disponibles, ou si la relaxation non linéaire n'a pas de structure particulière, il faut cependant disposer d'une approche générique telle que nous avons développée. Pour cette raison, nous nous concentrons dans ce paragraphe sur les améliorations naturelles du solveur implémenté, c'est-à-dire qui ne changent pas la nature de l'approche.

Amélioration de la robustesse

Sur le sous-ensemble des 900 plus petites instances, pour un temps limite de 20 secondes, l'approche utilisant les relaxations non linéaires permet de résoudre 94% des problèmes. Ce taux de réussite est le même que si l'on utilise des relaxations linéaires, ce qui est encourageant. Comme l'a montré la figure 3.14, le taux d'échec du solveur non linéaire reste cependant plus élevé que celui du simplexe dual. Les échecs, tels qu'ils ont été mesurés, indiquent que l'un des deux événements suivants a lieu : soit les hypothèses mathématiques nécessaires au contrôle de la convergence ne sont plus respectées à cause des erreurs numériques, soit le nombre d'itérations effectuées dépasse largement le nombre habituel nécessaire à la convergence, et quelque chose ne se passe donc pas comme prévu.

Pour expliquer les sources de ces problèmes ainsi que leurs solutions, nous analysons les deux instances de plus petite taille sur lesquelles la résolution du *root node* a échoué. La première est l'instance `ex8_1_3` de la MINLPLib. Dans la formulation factorisée, l'objectif est quadratique mais possède un unique terme, noté ici xy. Les bornes de x et de y n'ont cependant été resserrées par aucune technique de réduction de bornes, et sont donc toujours à leur valeur initiale, c'est-à-dire $[-10^{30}, 10^{30}]$. Le terme de convexification utilisé est $\phi(x) = \frac{1}{2}(x-10^{30})(x+10^{30})+\frac{1}{2}(y-10^{30})(y+10^{30})$. Après quelques itérations, l'objectif atteint le minimum de ϕ, soit environ -10^{60}. Le facteur de pénalisation nécessaire pour que la pénalisation des contraintes soit visible, dans le paysage optimisé, devant le terme de convexification est alors très grand, et la valeur maximale autorisée (10^{30}) est insuffisante. De plus, des nombres de cet ordre de grandeur conduisent à des erreurs d'arrondi importantes, puis à un échec de l'algorithme. Comme cet exemple le montre, la présence de termes de convexification quadratiques touchant des variables ayant un domaine de définition très grand est responsable d'une partie des problèmes de robustesse du solveur non linéaire implémenté. Une autre difficulté, plus rare, concerne la présence de contraintes non linéaires mal conditionnées, par exemple $x^6 - y \leq 0$ pour x grand. Ces deux cas mis à part, les problèmes numériques, liés à la précision finie des calculs effectués, ont tous été traités par les techniques décrites dans la section 3.3.

La seconde instance à laquelle on s'intéresse est `hs059` de la librairie CUTEr. Dans ce cas, l'échec est dû à un dépassement du nombre d'itérations autorisé. En retirant cette limite, le premier sous-problème est résolu en environ 5000 itérations et le troisième au bout de 68000 itérations. À partir de ce moment, la solution reste dans le bassin d'attraction quadratique de la méthode de Newton après les mises à jour des multiplicateurs et des facteurs de pénalisation. Les neuf derniers sous-problèmes sont résolus en un total de 24 itérations et une solution optimale est finalement trouvée après un peu plus de 68000 itérations. Comme dans la majorité des cas où la convergence est lente, ceci est causé par une absence de stricte

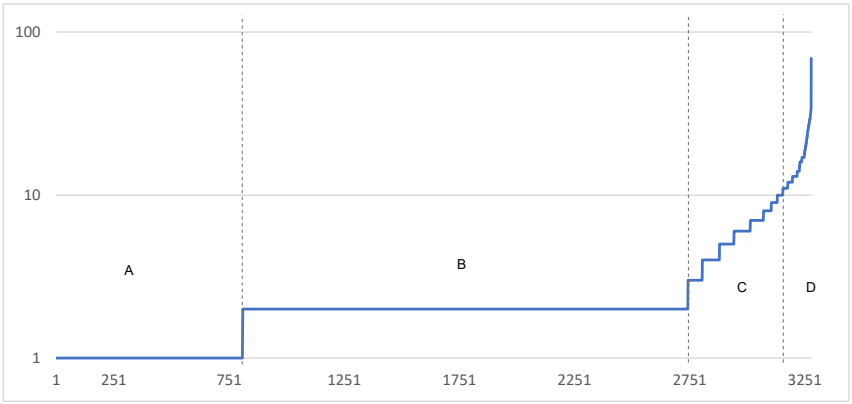

FIGURE 5.13 – *Répartition des instances selon $N_{\max}$, le nombre maximal de systèmes linéaires résolus pour calculer une direction de Newton pendant la résolution.*

$N_{\max}$	Échec du solveur	Échec d'un sous-problème	Échec d'une recherche linéaire	Coût moyen maximal d'un sous-problème
1 (809 instances A)	0.5%	0.6%	0.2%	16
2 (1934 instances B)	1.6%	11.6%	0.5%	425
3-10 (412 instances C)	30%	60.2%	5.6%	250
11 et plus (124 instances D)	70.1%	94.4%	14.5%	196

Tableau 5.8 – *Proportion des instances où certains évènements numériques arrivent, selon la correction maximale de la hessienne. La dernière colonne donne le coût moyen de résolution d'un sous-problème (en itérations) pour l'instance de la classe qui maximise ce coût moyen.*

convexité du lagrangien augmenté. Au début de la recherche, les valeurs propres de la hessiennes sont $\{0, 4.10^{-8}, \cdots, 970\}$ et une direction de courbure nulle est donc présente. La résolution du premier système de Newton échoue systématiquement, et la correction diagonale effectuée sur la hessienne a pour effet de ralentir fortement la convergence. L'absence de stricte convexité constitue en réalité la cause principale d'échecs du solveur. Si l'algorithme converge en théorie, le coût limite est souvent dépassé et le solveur est interrompu. Pour illustrer l'ampleur du problème, la figure 5.13 représente, pour chaque instance, le maximum du nombre de systèmes linéaires résolus pour calculer une direction de Newton ($N_{\max}$), pris sur tous les sous-problèmes résolus par le solveur non linéaire. Il s'agit d'une approximation empirique de la stricte convexité. Le tableau 5.8 compare ensuite la fréquence de certains évènements numériques et le coût moyen de résolution d'un sous-problème selon cette mesure.

Heureusement, des solutions à tous les problèmes que nous venons d'évoquer sont possibles, et nous commençons par les problèmes de conditionnement et de précision numérique. Les techniques de pénalisation directionnelle, développées en partie 3.3.3, se généralisent à toutes les contraintes non linéaires unaires du modèle factorisé. L'utilisation de différentes directions de pénalisation permet d'améliorer le conditionnement de n'importe quelle inégalité représentant un convexe de $\mathbb{R}^2$. Si ces techniques sont aujourd'hui implémentées pour $e^x - y \leq 0$ et pour $1/x - y \leq 0$, leur généralisation à $x^a - y \leq 0$, $x^{2k} - y \leq 0$ ainsi qu'à la relaxation convexe de $y = x^{2k+1}$ est possible et améliorerait les performances du solveur non linéaire sur les instances où ces contraintes sont présentes. Étendre l'idée de pénalisation directionnelle aux contraintes quadratiques quelconques permettrait aussi de régler les problèmes numériques liés aux termes de convexification des variables ayant un grand domaine de définition. Si cette tâche se révèle trop difficile, nous pouvons envisager d'utiliser une mise à l'échelle plus performante pour ces contraintes. En particulier, la prise en compte du pire cas dans le calcul des valeurs possibles la norme du gradient est trop pessimiste dans le cas général, mais serait parfois nécessaire. On peut alors imaginer utiliser une mise à l'échelle dynamique, calculée selon le point courant et mise à jour en même temps que les multiplicateurs et facteurs de pénalisation.

Le cas de la fonction objectif est plus délicat, et il faut introduire un nouveau paramètre de mise à l'échelle. Plus précisément, pour un problème non linéaire $\min \{f(x) \mid g(x) = 0,\ h(x) \leq 0\}$, la mise à l'échelle des contraintes consiste à remplacer l'ensemble admissible $\{g(x) = 0,\ h(x) \leq 0\}$ par $\{w_g g(x) = 0,\ w_h h(x) \leq 0\}$, où w_g et w_h sont des vecteurs de scalaires strictement positifs. La mise à l'échelle de l'objectif consiste à remplacer l'objectif f par wf, avec $w > 0$. Dans le cas de la méthode du lagrangien augmenté, la mise à l'échelle de l'objectif est en théorie équivalente à une modifications des coefficients de mise à l'échelle des contraintes et des facteurs de pénalisation. Cette équivalence reste cependant numériquement problématique, puisque la précision numérique absolue, définie comme l'inverse de la distance entre deux réels représentables successifs, n'est pas constante. La meilleure précision étant au voisinage de 1, il ne suffit pas d'augmenter les w_g et w_h pour équilibrer la valeur de l'objectif et de la pénalisation dans le lagrangien augmenté : il faut réduire $f(x)$. Cette approche est parfois utilisée pour accélérer la détection de l'inconsistance dans la méthode du lagrangien augmenté, et consiste à remplacer problème par $\min\{wf(x) \mid w_g g(x) = 0,\ w_h h(x) \leq 0\}$. Les coefficients w_g et w_h sont alors utilisés pour équilibrer les contraintes entre elles, et w est utilisé pour équilibrer l'objectif et le terme de pénalisation.

Les problèmes de stricte convexité sont plus complexes à résoudre, puisqu'ils sont intrinsèques à l'algorithme du lagrangien augmenté. La solution passe donc par une modification de l'algorithme mathématique utilisé, afin de rendre le paysage d'optimisation minimisé strictement convexe. Une première idée est d'exploiter des reformulations des contraintes de bornes. Le domaine $\ell \leq x \leq u$ des variables s'écrit aussi $(x - l)(x - u) \leq 0$, cette dernière formulation étant strictement convexe. Un terme $\phi_\alpha(x) = \sum_i \alpha_i (x_i - \ell_i)(x_i - u_i)$ dans l'objectif correspond alors au lagrangien associé à la seconde formulation, et rend le paysage minimisé strictement convexe. En interprétant les α_i comme des facteurs de pénalisation des contraintes de bornes

et en imposant leur stricte positivité, il est peut-être possible d'obtenir un schéma itératif, similaire à celui utilisé, où l'on minimise la somme $\mathcal{L}(x, \mu, \rho) + \phi_\alpha(x)$ à chaque itération. Modifier la règle de mise à jour des multiplicateurs et en obtenir une pour les α_i est une piste possible pour se ramener à une suite de problèmes strictement convexes (et même fortement convexes ici). Cependant, le terme ϕ_α utilisé ici est le même que celui utilisé pour convexifier les expressions quadratiques. Si des variables ont un grand domaine de définition, les mêmes problèmes numériques sont à prévoir : si une contrainte de borne est saturée par la solution optimale, le α_i correspondant devra tendre vers 0 pour atteindre l'optimum. Dans ce cas, la constante de forte convexité décroît vers 0 et il n'est pas garanti que des problèmes de courbure nulle (ou presque nulle) ne réapparaissent pas. Une autre possibilité est d'utiliser l'algorithme du lagrangien augmenté proximal [141]. Celui-ci consiste à minimiser, à chaque itération, la somme du lagrangien augmenté et d'un terme proximal :

$$\min_x \ \mathcal{L}(x, \mu, \rho) + \frac{1}{2\sigma}\|x - x^*\|,$$

où x^* est la solution optimale du sous-problème précédent et $\sigma > 0$. Sous certaines hypothèses sur la suite de σ utilisée, cet algorithme converge de la même façon que le lagrangien augmenté classique, avec la même vitesse asymptotique, et fait intervenir des sous-problèmes strictement convexes. La présence du terme proximal dégrade cependant généralement les performances pratiques de la méthode du lagrangien augmenté, comme décrit par exemple dans [73]. Finalement, la solution la plus prometteuse consiste à utiliser une autre forme de lagrangien augmenté, décrite dans le prochain paragraphe, qui résout le problème de la convexité tout en offrant des améliorations de performances, à la fois d'un point de vue théorique et d'un point de vue pratique.

Amélioration de la performance

Lorsque le lagrangien augmenté est strictement convexe, le nombre d'itérations nécessaires à la résolution des sous-problèmes diminue significativement, comme le montre le tableau 5.8. Une solution aux problèmes de courbures nulles évoqués ci-dessus permettrait de généraliser la performance obtenue sur les instances de type A (première ligne du tableau) aux autres catégories. L'absence de stricte convexité n'est cependant pas le seul problème rencontré par le solveur. Les discontinuités de la hessienne, liées au traitement des inégalités dans le lagrangien augmenté, peuvent détériorer les performances. Si la vitesse de convergence asymptotique reste en théorie quadratique [139], une stagnation reste possible en dehors du bassin d'attraction quadratique. La solution aux problèmes de convexité que nous envisageons permettra aussi de régler ce second problème.

La méthode du lagrangien augmenté peut se généraliser à d'autres pénalisations que celle utilisée historiquement, à savoir la pénalisation extérieure quadratique. Ces variantes s'interprètent à leur tour par certains schémas itératifs dans l'espace dual, et donnent lieu à des paysages d'optimisation différents. L'exemple historique s'appelle la méthode exponentielle des multiplicateurs (*exponential method of multipliers*, [158]), et la version qui nous intéresse est décrite par exemple dans [38].

L'idée fondamentale est l'équivalence de formulation suivante :

$$g(x) \leq 0 \Leftrightarrow \rho\phi(g(x)/\rho) \leq 0,$$

pour $g : \mathbb{R}^n \to \mathbb{R}$ et certaines fonctions ϕ. Par exemple, $\phi(t) = \mathrm{e}^t - 1$ donne la méthode exponentielle des multiplicateurs, et $\phi(t) = -\log(1-t)$ correspond à une barrière translatée utilisée dans certaines méthode de points intérieurs. Les fonctions utilisées dans les implémentations de cette approche sont obtenues par un branchement $\mathcal{C}^2$ entre un logarithme (à l'intérieur de l'ensemble admissible translaté), et une fonction quadratique (à l'extérieur). Le lagrangien augmenté ainsi obtenu s'écrit

$$\mathcal{L}(x, \mu, \rho) = f(x) + \sum_i \rho_i \mu_i \phi(g_i(x)/\rho_i),$$

et est dans notre cas fortement convexe et au moins aussi régulier que les contraintes. Un intérêt supplémentaire de cette approche est que les sous-problèmes résolus sont encore des relaxations du problème initial, puisque ϕ est négative uniquement sur $\mathbb{R}^+$. Les règles de mise à jour des multiplicateurs s'obtiennent comme pour la méthode des multiplicateurs, en inspectant les conditions d'optimalité du premier ordre :

$$\mu_{k+1} = \mu_k \phi'(g_i(x)/\rho_i),$$

et la convergence a lieu pour $rho_i > 0$ constant. La difficulté principale de cette approche est l'obtention de règles de mise à jour de ρ efficaces en pratique. Les auteurs de LANCELOT [63] décrivent la supériorité d'un prototype de cette approche sur le logiciel dans [64]. Le solveur PENNON [103] utilise cet algorithme pour l'optimisation non linéaire.

Une fois les problèmes de robustesse et la géométrie des paysages d'optimisation entièrement contrôlés, la performance du solveur non linéaire sera homogénéisée sur celle obtenue actuellement sur les problèmes favorables. Améliorer les performances passera alors par l'utilisation de méthodes primales-duales, qui déterminent des itérations en les variables primales et duales à la fois. La méthode du lagrangien augmenté alterne les itérations dans les deux espaces, et sa version quad-log est similaire. Il existe cependant des versions primales-duales de cette dernière, par exemple l'approche *nonlinear rescaling method* décrite dans [136]. L'intérêt principal de cette approche par rapport à celles de points intérieurs primales-duales est que le démarrage à chaud ne demande que la modification du paramètre de barrière, et non pas celle des variables primales. Malheureusement, elle souffre du même problème que le lagrangien augmenté : une contrainte touchant k variables implique un bloc dense de taille $k \times k$ dans la hessienne. Plus généralement, les méthodes primales-duales permettent souvent de traiter explicitement certains types de contraintes, comme les contraintes de bornes ou les égalités linéaires. Enfin, nous pouvons illustrer l'intérêt d'une méthode primale-duale en comparant les performances de notre solveur non linéaire à MOSEK [30], un solveur de l'état de l'art en optimisation convexe différentiable. Dans [67], une description est faite de l'intégration des cônes exponentiels asymétriques au logiciel. Il est expliqué que de nombreuses instances de la MINLPLib comportant des opérateurs exp peuvent se formuler à l'aide de ce cône, et l'intérêt de sa prise en compte y est mesuré. Ceci nous permet de comparer, en termes d'itérations, et donc de systèmes linéaires résolus, une approche générique telle que la nôtre, et cette approche spécifique. Cette comparaison n'est pas

Instance	Mosek (sans cône exponentiel)	Mosek (avec cône exponentiel)	Solveur non linéaire
batch	69	29	124
batchs201210m	(fail)	44	249
enpro48pb	130	33	257
enpro56pb	158	37	349
ex1223	13	13	26
gams01	119	48	(fail)
ravem	153	41	149
rsyn0805h	30	19	107
syn30h	23	18	213
syn40h	25	18	276
synthes3	33	34	46

Tableau 5.9 – *Comparaison du nombre d'itérations nécessaires à la résolution de la relaxation continue de quelques problèmes de la MINLPLib.*

complète, mais donne déjà une idée des gains potentiels sur les instances où notre solveur fonctionne déjà bien. Le coût d'une itération dans MOSEK est la résolution d'un système linéaire de taille $n + m$, contre n pour notre solveur. En revanche, les systèmes primals-duals résolus par une méthode de points intérieurs sont aussi creux que le modèle factorisé, alors que ceux issus du lagrangien augmenté sont densifiés. La conclusion est que si l'on peut s'attendre à des gains presque systématiques, d'un facteur entre 2 et 10, en nombre de systèmes linéaires résolus, le coût de résolution de chaque système peut varier de façon importante.

Un autre point de vue permettant d'estimer les gains potentiels en termes de vitesse de résolution des relaxations non linéaires est d'analyser la proportion du temps passé dans le solveur. Ces proportions sont tracés, dans les cas linéaire et non linéaire, dans la figure 5.14 et le tableau 5.10 Les performances du solveur linéaire, en termes de temps de résolution des nœuds y est clairement supérieure à celle du solveur non linéaire. Si les deux axes, amélioration de la robustesse par suppression des problèmes de convexité et de discontinuité de la hessienne, ainsi qu'utilisation d'une méthode primale-duale dédiée, aboutissent aux conclusions attendues, on peut espérer que la situation s'équilibre.

Proportion du temps passé dans le solveur	Linéaire	Non linéaire
$> 99\%$	0.4%	19.8%
entre 90% et 99%	54.4%	77%
entre 50% et 90%	41.2%	22.9%
$< 50\%$	0.12%	0.12%

Tableau 5.10 – *Proportion du temps passé dans les solveurs pour les instances non résolues, selon la nature des relaxations utilisées.*

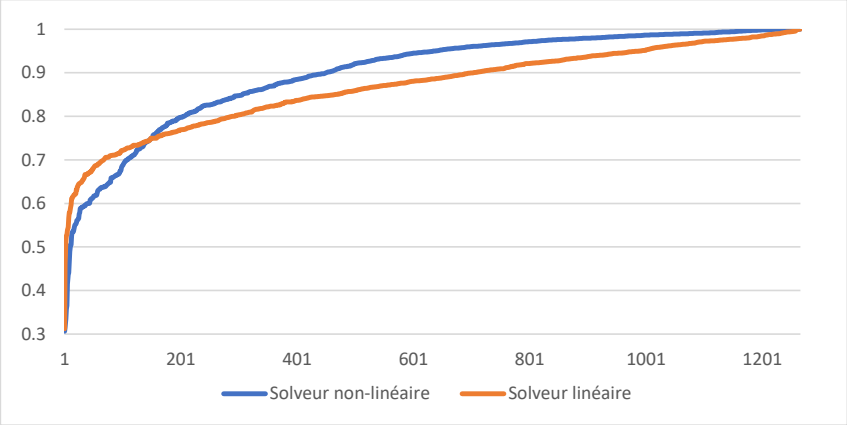

FIGURE 5.14 – *Proportion du temps passé dans les solveurs, pour les instances où ces temps sont mesurables, selon la nature des relaxations utilisées.*

Quelques autres pistes à creuser

Accélérer la vitesse de résolution d'une relaxation convexe donnée n'est pas le seul axe d'amélioration du solveur non linéaire. La gestion des coupes représente un sujet ambitieux mais aussi des gains potentiels importants. La gestion des contraintes coniques, naturelle dans une méthode de points intérieurs primale-duale, est un second exemple. Pour les problèmes de grande taille, on doit appliquer notre méthode de Newton tronquée, car la factorisation directe est impossible. Comme l'a montré la figure 3.14, ceci induit une perte importante de robustesse. L'utilisation de méthodes itératives stabilisées (pour réduire les erreurs numériques), ainsi que des techniques de préconditionnement génériques (pour réduire le nombre d'itérations) permettrait de compenser cette dernière. Des expérimentations avec des préconditionneurs automatiques ont été menées, mais se sont finalement révélées délétères pour les préconditionneurs utilisés (basés sur la diagonale de la hessienne ou sur des approximations de quasi-newton à mémoire limitée).

Enfin, le théorème 3.1.3 a été exploité à de nombreuses reprises dans le code. Malgré son apparente simplicité, il s'est révélé, à lui seul, aussi bénéfique qu'une des techniques de réduction de borne les plus efficaces, a permis d'obtenir des certificats d'optimalité et d'inconsistance fiables pour les solveurs linéaires et non linéaires, de rendre fiables plusieurs techniques de réduction de bornes sujettes à des erreurs numériques, et, enfin, d'améliorer significativement la performance de notre solveur non linéaire. Nous savons déjà qu'il est sous-exploité, comme le montre le théorème suivant, malheureusement obtenu trop tard et non intégré à nos résultats. Celui-ci utilise le théorème 3.3 pour obtenir un point stationnaire du lagrangien en tout point primal-dual, puis applique les inégalités valides du théorème 4.3.1 aux contraintes de borne. Il constitue ainsi une technique de réduction de bornes, qui peut être assimilée à une application de la SBBT à chaque itération du solveur non linéaire.

Théorème 5.2.1. *Soient des fonctions* $f : \mathbb{R}^n \to \mathbb{R}$ *et* $g : \mathbb{R}^n \to \mathbb{R}^m$ *convexes et différentiables. Pour* $\mu \in \mathbb{R}^m_+$*, on note* $\mathcal{L}(x, \mu) = f(x) + \mu^T g(x)$*. Soient le problème*

de minimisation

$$\text{(P)} \min_{x \in \mathbb{R}^n} f(x)$$
$$s.c. \begin{cases} \ell \leq x \leq u, \\ g(x) \leq 0. \end{cases}$$

ainsi que son dual de Wolfe

$$\text{(D)} \max_{x,\mu} \theta(x, \mu),$$
$$s.c. \begin{cases} \mu \geq 0 \end{cases}$$

où

$$\theta(x, \mu) = \mathcal{L}(x, \mu) + (\ell - x)^T \nabla \mathcal{L}(x, \mu)^+ + (u - x)^T \nabla \mathcal{L}(x, \mu)^-.$$

Soit $\mathcal{U}$ une borne supérieure de (P). Alors quel que soit $x \in \mathbb{R}^n$ et $\mu \in \mathbb{R}^m_+$, le côté dans division par zéro de

$$u_i - \frac{\mathcal{U} - \theta(x, \mu)}{\nabla_i \mathcal{L}(x, \mu)^+} \leq x_i \leq \ell_i + \frac{\mathcal{U} - \theta(x, \mu)}{-\nabla_i \mathcal{L}(x, \mu)^-}$$

est une réduction de bornes valide pour (P).

D'autres questions se posent sur l'utilisation du théorème 3.1.4. Notre approche, consistant à se ramener au cas le plus facile de l'optimisation non linéaire, c'est-à-dire sans contraintes, n'est peut-être pas la bonne, et la gestion explicite des contraintes de borne, appelée par le théorème 3.1.3, est peut-être plus judicieuse. De plus, la possibilité de résoudre exactement le dual en les multiplicateurs de bornes pourrait être utilisé dans la méthode du lagrangien augmenté. Des mises à jour plus fréquentes de ces multiplicateurs, ou le remplacement des règles du premier ordre par la valeur optimale donnée par le dual de Wolfe, permettront peut être d'améliorer l'algorithme dans le cas des sous-problèmes non-contraints. Enfin, l'exploitation de ce résultat au sein d'algorithmes tels que les points intérieurs ou la programmation quadratique successive, voire de la programmation linéaire, est aussi possible. Dans ce cas, on peut espérer qu'il apporte les mêmes gains que ceux observés dans notre cas, au moins pour les problèmes convexes.

5.3 Conclusion

Les outils développés pendant cette thèse permettent de calculer des bornes inférieures aux problèmes d'optimisation définis par un graphe d'évaluation. Dans un tel graphe, les valeurs des nœuds sont asservies à celles de leurs enfants par les opérateurs qu'ils portent. Remplacer cet asservissement par une contrainte de couplage dans un espace de plus grande dimension permet de transformer le problème initial en un programme mathématique dans lequel l'expression analytique de l'objectif et de chaque contrainte est connue. Une bibliothèque de fonctions élémentaires, suffisamment expressive pour couvrir les opérateurs du graphe, permet ensuite de classifier les contraintes de ce nouveau programme, selon leurs propriétés structurelles et géométriques. Enfin, cette classification facilite l'implémentation d'un cœur d'optimisation globale, définissant la cinématique et les interactions des composants utiles au calcul de bornes : génération et résolution de relaxations convexes, techniques de réduction de bornes et recherche arborescente.

La complexité conceptuelle des algorithmes et des théories mathématiques utilisées pendant cette thèse ne constitue pas la difficulté principale du calcul de bornes inférieures. Non seulement la caractérisation et l'exploitation d'un maximum d'informations géométriques contenues dans le modèle factorisé sont une quête infinie, s'attaquant frontalement à la classe des problèmes NP-difficiles, mais, surtout, le diable se cache dans les détails. La validité d'une borne duale découle de certificats pouvant prendre des formes variées, comme l'utilisation d'arrondis directionnels dans la propagation et l'inférence de bornes, ou l'utilisation de théorèmes de dualité pour la génération de points stationnaires du lagrangien. Lorsque ces certificats se heurtent à la nature inexacte des calculs effectués par ordinateur, le travail d'implémentation et les réglages numériques nécessaires au maintien de leur validité sont conséquents. Le développement d'un solveur d'optimisation globale est ainsi long et fastidieux, puisque la traduction d'une propriété géométrique en code fonctionnel représente souvent plus de travail que sa dérivation théorique. Pourtant, de tels logiciels fleurissent depuis quelques dizaines d'années, poussés par la réalisation du fait que l'ambition universelle de l'optimisation globale est peut-être à portée de main. Toutefois, l'ampleur de la tâche à accomplir est telle que des sujets comme la gestion de la dette technique, la perte de connaissances, ou encore le partage des astuces d'implémentation deviendront problématiques avant son accomplissement. Le volume de code à produire, ainsi que la richesse de la géométrie des MINLP, appellent donc à une formalisation et à une standardisation des composants d'un tel solveur, dans le but d'en réduire les redondances et d'en faciliter le partage. Dans ce sens, nous pensons qu'un succès de l'optimisation globale, à la hauteur de celui des outils de programmation linéaire en nombres entiers, sera le fruit d'une plus grande collaboration entre les mondes académique et industriel, et entre les solveurs commerciaux aujourd'hui en concurrence.

La synthèse des techniques utilisées dans les différents solveurs d'optimisation globale disponibles au sein d'un unique logiciel représente une charge de travail titanesque, mais aussi une formidable opportunité. Beaucoup reste à faire, à découvrir, comme le montrent les quelques contributions de cette thèse : exploitation de propriétés géométriques des contraintes quadratiques pour améliorer les relaxations

convexes et techniques de réduction de bornes associées, implémentation soignée de techniques de réduction de bornes coûteuses comme le *probing*, généralisation de résultats de dualité assurant la validité des certificats d'optimalité et d'inconsistance, et, enfin, construction d'un solveur sur-mesure prouvant que l'optimisation non linéaire n'a pas dit son dernier mot face à la prédominance du simplexe dual. Le rapprochement de différents domaines de recherche, tels que l'optimisation linéaire, non linéaire, semi-définie, la programmation par contraintes, l'étude des problèmes de satisfiabilité, des problèmes combinatoires, de l'analyse convexe, des algorithmes d'approximation et bien d'autres encore, permettra de généraliser, d'améliorer et de fusionner certains outils, avec des répercussions positives pour chaque communauté concernée. La complémentarité de la recherche locale et de l'optimisation globale, déjà pressentie dans leurs dénominations, a justifié ce travail mais n'a été qu'effleurée. Si la philosophie pragmatique et la taille des problèmes que traite la première a eu une influence positive sur notre implémentation de la seconde, des interactions plus profondes entre les deux sont possibles, et laissent présager un futur riche d'innovations.

Bibliographie

[1] http ://people.brunel.ac.uk/ mastjjb/jeb/info.html.

[2] http ://plato.asu.edu/ftp/ampl-nlp.html.

[3] http ://qplib.zib.de.

[4] https ://miplib.zib.de/.

[5] https ://projects.coin-or.org/clp.

[6] https ://soplex.zib.de/.

[7] https ://www.artelys.com/docs/knitro/.

[8] https ://www.coin-or.org/.

[9] https ://www.gams.com/.

[10] www.cuter.rl.ac.uk.

[11] www.minlplib.org.

[12] www.netlib.org.

[13] www.roadef.org.

[14] Kumar Abhishek, Sven Leyffer, and Jeff Linderoth. Filmint : An outer approximation-based solver for convex mixed-integer nonlinear programs. *INFORMS Journal on computing*, 22(4) :555–567, 2010.

[15] Tobias Achterberg. Scip : solving constraint integer programs. *Mathematical Programming Computation*, 1(1) :1–41, 2009.

[16] Tobias Achterberg and Timo Berthold. Hybrid branching. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 309–311. Springer, 2009.

[17] Tobias Achterberg, Timo Berthold, Thorsten Koch, and Kati Wolter. Constraint integer programming : A new approach to integrate cp and mip. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 6–20. Springer, 2008.

[18] Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. 2016.

[19] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1) :42–54, 2005.

[20] Claire S Adjiman, Ioannis P Androulakis, and Christodoulos A Floudas. A global optimization method, αbb, for general twice-differentiable constrained nlps—ii. implementation and computational results. *Computers & chemical engineering*, 22(9) :1159–1179, 1998.

[21] Claire S Adjiman, Stefan Dallwig, Christodoulos A Floudas, and Arnold Neumaier. A global optimization method, αbb, for general twice-differentiable constrained nlps—i. theoretical advances. *Computers & Chemical Engineering*, 22(9) :1137–1158, 1998.

[22] Ioannis G Akrotirianakis and Christodoulos A Floudas. A new class of improved convex underestimators for twice continuously differentiable constrained nlps. *Journal of Global Optimization*, 30(4) :367–390, 2004.

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

[24] Patrick R Amestoy, Ian S Duff, and J-Y L'Excellent. Mumps multifrontal massively parallel solver version 2.0. 1998.

[25] Edward Anderson, Zhaojun Bai, Christian Bischof, Susan Blackford, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, and D Sorensen. *LAPACK Users' guide*, volume 9. Siam, 1999.

[26] Roberto Andreani, Ernesto G Birgin, José Mario Martínez, and María Laura Schuverdt. On augmented lagrangian methods with general lower-level constraints. *SIAM Journal on Optimization*, 18(4) :1286–1309, 2007.

[27] Roberto Andreani, José Mario Martínez, and Benar Fux Svaiter. A new sequential optimality condition for constrained optimization and algorithmic consequences. *SIAM Journal on Optimization*, 20(6) :3533–3554, 2010.

[28] Ioannis P Androulakis, Costas D Maranas, and Christodoulos A Floudas. αbb : A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4) :337–363, 1995.

[29] Kurt M Anstreicher. On convex relaxations for quadratically constrained quadratic programming. *Mathematical programming*, 136(2) :233–251, 2012.

[30] MOSEK ApS. Mosek rmosek package, 2015.

[31] Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1) :1–3, 1966.

[32] Pietro Belotti. Couenne : a user's manual. Technical report, Technical report, Lehigh University, 2009.

[33] Pietro Belotti, Sonia Cafieri, Jon Lee, and Leo Liberti. Feasibility-based bounds tightening via fixed points. In *International Conference on Combinatorial Optimization and Applications*, pages 65–76. Springer, 2010.

[34] Pietro Belotti, Christian Kirches, Sven Leyffer, Jeff Linderoth, James Luedtke, and Ashutosh Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22 :1–131, 2013.

[35] Pietro Belotti, Christian Kirches, Sven Leyffer, Jeff Linderoth, James Luedtke, and Ashutosh Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22 :1–131, 2013.

[36] Pietro Belotti, Jon Lee, Leo Liberti, Francois Margot, and Andreas Wächter. Branching and bounds tightening techniques for non-convex minlp. *Optimization Methods & Software*, 24(4-5) :597–634, 2009.

[37] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization : analysis, algorithms, and engineering applications*, volume 2. Siam, 2001.

[38] Aharon Ben-Tal and Michael Zibulevsky. Penalty/barrier multiplier methods for convex programming problems. *SIAM Journal on Optimization*, 7(2) :347–366, 1997.

[39] T. Benoist. From local search to local search programming. *HDR, Université de Nantes*, 2014.

[40] Thierry Benoist. Soft car sequencing with colors : Lower bounds and optimality proofs. *European Journal of Operational Research*, 191(3) :957–971, 2008.

[41] Thierry Benoist, Bertrand Estellon, Frédéric Gardi, Romain Megel, and Karim Nouioua. Localsolver 1. x : a black-box local-search solver for 0-1 programming. *4OR*, 9(3) :299, 2011.

[42] Timo Berthold. Primal heuristics for mixed integer programs, 2006.

[43] Timo Berthold, Stefan Heinz, and Stefan Vigerske. Extending a cip framework to solve miqcp s. In *Mixed integer nonlinear programming*, pages 427–444. Springer, 2012.

[44] Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[45] Alain Billionnet, Sourour Elloumi, Amélie Lambert, and Angelika Wiegele. Using a conic bundle method to accelerate both phases of a quadratic convex reformulation. *INFORMS Journal on Computing*, 29(2) :318–331, 2017.

[46] Alain Billionnet, Sourour Elloumi, and Marie-Christine Plateau. Improving the performance of standard solvers for quadratic 0-1 programs by a tight convex reformulation : The qcr method. *Discrete Applied Mathematics*, 157(6) :1185–1197, 2009.

[47] Ernesto G Birgin and José Mario Martínez. Augmented lagrangian method with nonmonotone penalty parameters for constrained optimization. *Computational Optimization and Applications*, 51(3) :941–965, 2012.

[48] Ernesto G Birgin, José Mario Martínez, and LF Prudente. Augmented lagrangians with possible infeasibility and finite termination for global nonlinear programming. *Journal of Global Optimization*, 58(2) :207–242, 2014.

[49] Ernesto G Birgin, José Mario Martínez, and Marcos Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, 10(4) :1196–1211, 2000.

[50] Ernesto G Birgin and Josâ Mario Mart_nez. *Practical augmented Lagrangian methods for constrained optimization*, volume 10. SIAM, 2014.

[51] Robert E Bixby, Mary Fenelon, Zonghao Gu, Ed Rothberg, and Roland Wunderling. Mixed-integer programming : A progress report. In *The sharpest cut : the impact of Manfred Padberg and his work*, pages 309–325. SIAM, 2004.

[52] Christian Bliek1ú, Pierre Bonami, and Andrea Lodi. Solving mixed-integer quadratic programming problems with ibm-cplex : a progress report. In *Proceedings of the twenty-sixth RAMP symposium*, pages 16–17, 2014.

[53] Pierre Bonami, Mustafa Kilinç, and Jeff Linderoth. Algorithms and software for convex mixed integer nonlinear programs. In *Mixed integer nonlinear programming*, pages 1–39. Springer, 2012.

[54] Pierre Bonami and Jon Lee. Bonmin user's manual. *Numer Math*, 4 :1–32, 2007.

[55] Glencora Borradaile and Pascal Van Hentenryck. Safe and tight linear estimators for global optimization. *Mathematical Programming*, 102(3) :495–517, 2005.

[56] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[57] Samuel Burer and Adam N Letchford. Non-convex mixed-integer nonlinear programming : A survey. *Surveys in Operations Research and Management Science*, 17(2) :97–106, 2012.

[58] Michael R Bussieck and Arne Drud. Sbb : A new solver for mixed integer nonlinear programming. *Talk, OR*, 2001.

[59] Michael R Bussieck and Stefan Vigerske. Minlp solver software. *Wiley encyclopedia of operations research and management science*, 2010.

[60] Richard H Byrd, Jorge Nocedal, and Richard A Waltz. K nitro : An integrated package for nonlinear optimization. In *Large-scale nonlinear optimization*, pages 35–59. Springer, 2006.

[61] Pedro M Castro. Tightening piecewise mccormick relaxations for bilinear problems. *Computers & Chemical Engineering*, 72 :300–311, 2015.

[62] Alice Chiche and Jean Charles Gilbert. How the augmented lagrangian algorithm can deal with an infeasible convex quadratic optimization problem. 2016.

[63] Andrew R Conn, GIM Gould, and Philippe L Toint. *LANCELOT : a Fortran package for large-scale nonlinear optimization (Release A)*, volume 17. Springer Science & Business Media, 2013.

[64] Andrew R Conn, Nicholas IM Gould, and Philippe Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2) :545–572, 1991.

[65] Andrew R Conn, Nicholas IM Gould, and Philippe L Toint. *Trust region methods*. SIAM, 2000.

[66] Frank E Curtis, Nicholas IM Gould, Hao Jiang, and Daniel P Robinson. Adaptive augmented lagrangian methods : algorithms and practical numerical experience. *Optimization Methods and Software*, 31(1) :157–186, 2016.

[67] Joachim Dahl and Erling D Andersen. A primal-dual interior-point algorithm for nonsymmetric exponential-cone optimization. 2019.

[68] Yu-hong Dai and Yaxiang Yuan. An efficient hybrid conjugate gradient method for unconstrained optimization. *Annals of Operations Research*, 103(1-4) :33–47, 2001.

[69] Claudia D'Ambrosio, Antonio Frangioni, Leo Liberti, and Andrea Lodi. A storm of feasibility pumps for nonconvex minlp. *Mathematical programming*, 136(2) :375–402, 2012.

[70] Frédéric Delbos and Jean Charles Gilbert. Global linear convergence of an augmented lagrangian algorithm for solving convex quadratic optimization problems. 2003.

[71] Ron S Dembo and Trond Steihaug. Truncated-newtono algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26(2) :190–212, 1983.

[72] Ferenc Domes and Arnold Neumaier. Constraint propagation on quadratic constraints. *Constraints*, 15(3) :404–429, 2010.

[73] Jonathan Eckstein and Paulo JS Silva. Proximal methods for nonlinear programming : double regularization and inexact subproblems. *Computational Optimization and Applications*, 46(2) :279–304, 2010.

[74] B Estellon, F Gardi, and K Nouioua. Ordonnancement de véhicules dans les usines renault : une approche par recherche localea voisinage large. In *Actes de ROADEF 2005, le 6eme Congres de la Société Française de Recherche Opérationnelle et d'Aidea la Décision*, pages 33–34, 2005.

[75] Giorgio Fasano and János D Pintér. Model development and optimization for space engineering : Concepts, tools, applications, and perspectives. In *Modeling and Optimization in Space Engineering*, pages 1–32. Springer, 2012.

[76] Dominique Feillet. A tutorial on column generation and branch-and-price for vehicle routing problems. *4or*, 8(4) :407–424, 2010.

[77] Marshall L Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations research*, 42(4) :626–642, 1994.

[78] Christodoulos A Floudas and Chrysanthos E Gounaris. A review of recent advances in global optimization. *Journal of Global Optimization*, 45(1) :3–38, 2009.

[79] F. Gardi. Towards a mathematical programming solver based on local search. *UPMC Sorbonne Universités*, 2013.

[80] David M Gay. Using expression graphs in optimization algorithms. *Mixed Integer Nonlinear Programming*, pages 247–262, 2012.

[81] Semyon Aranovich Gershgorin. Uber die abgrenzung der eigenwerte einer matrix. . , (6) :749–754, 1931.

[82] Philip E Gill, Walter Murray, and Michael A Saunders. Snopt : An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1) :99–131, 2005.

[83] Philip E Gill, Michael A Saunders, and Elizabeth Wong. On the performance of sqp methods for nonlinear optimization. In *Modeling and Optimization : Theory and Applications*, pages 95–123. Springer, 2015.

[84] Ambros M Gleixner, Timo Berthold, Benjamin Müller, and Stefan Weltge. Three enhancements for optimization-based bound tightening. *Journal of Global Optimization*, 67(4) :731–757, 2017.

[85] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6) :1115–1145, 1995.

[86] Nick Gould and Philippe L Toint. Preprocessing for quadratic programming. *Mathematical Programming*, 100(1) :95–132, 2004.

[87] L Grippo, F Lampariello, and S Lucidi. A truncated newton method with nonmonotone line search for unconstrained optimization. *Journal of Optimization Theory and Applications*, 60(3) :401–419, 1989.

[88] Luigi Grippo, Francesco Lampariello, and Stephano Lucidi. A nonmonotone line search technique for newton's method. *SIAM Journal on Numerical Analysis*, 23(4) :707–716, 1986.

[89] Ignacio E Grossmann, Jagadisan Viswanathan, Aldo Vecchietti, Ramesh Raman, Erwin Kalvelagen, et al. Gams/dicopt : A discrete continuous optimization package. *GAMS Corporation Inc*, 37 :55, 2002.

[90] William W Hager and Hongchao Zhang. A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM Journal on optimization*, 16(1) :170–192, 2005.

[91] Peter L Hammer and Abraham A Rubin. Some remarks on quadratic programming with 0-1 variables. *Revue française d'informatique et de recherche opérationnelle. Série verte*, 4(V3) :67–79, 1970.

[92] Michael Held and Richard M Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6) :1138–1162, 1970.

[93] Magnus R Hestenes and William Karush. A method of gradients for the calculation of the characteristic roots and vectors of a real symmetric matrix. *Journal of Research of the National Bureau of Standards*, 47(1) :45–61, 1951.

[94] MJ Hopper. Harwell subroutine library : a catalogue of subroutines (1980). Technical report, CM-P00068607, 1980.

[95] Reiner Horst and Hoang Tuy. *Global optimization : Deterministic approaches*. Springer Science & Business Media, 2013.

[96] Christian Jansson. A rigorous lower bound for the optimal value of convex optimization problems. *journal of global optimization*, 28(1) :121–137, 2004.

[97] Antoine Jeanjean. *Recherche locale pour l'optimisation en variables mixtes : méthodologie et applications industrielles*. PhD thesis, Ecole Polytechnique X, 2011.

[98] W Kahan. Further remarks on reducing truncation errors, commun. *Assoc. Comput. Mach*, 8 :40, 1965.

[99] William Kahan. Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776) :11, 1996.

[100] R Baker Kearfott. Validated probing with linear relaxations. *submitted to Journal of Global Optimization*, 2005.

[101] Aida Khajavirad and Nikolaos V Sahinidis. A hybrid lp/nlp paradigm for global optimization relaxations. *Mathematical Programming Computation*, 10(3) :383–421, 2018.

[102] Mustafa R Kılınç and Nikolaos V Sahinidis. Exploiting integrality in the global optimization of mixed-integer nonlinear programming problems with baron. *Optimization Methods and Software*, 33(3) :540–562, 2018.

[103] Michal Kočvara and Michael Stingl. Pennon : A code for convex nonlinear and semidefinite programming. *Optimization methods and software*, 18(3) :317–333, 2003.

[104] Nathan Krislock, Jerome Malick, and Frédéric Roupin. Biqcrunch : a semidefinite branch-and-bound method for solving binary quadratic problems. *ACM Transactions on Mathematical Software (TOMS)*, 43(4) :32, 2017.

[105] Jan Kronqvist, David E Bernal, Andreas Lundell, and Ignacio E Grossmann. A review and comparison of solvers for convex minlp. *Optimization and Engineering*, 20(2) :397–455, 2019.

[106] Jan Kronqvist, Andreas Lundell, and Tapio Westerlund. The extended supporting hyperplane algorithm for convex mixed-integer nonlinear programming. *Journal of Global Optimization*, 64(2) :249–272, 2016.

[107] Jean Bernard Lasserre. On representations of the feasible set in convex optimization. *Optimization Letters*, 4(1) :1–5, 2010.

[108] Yahia Lebbah, Claude Michel, and Michel Rueher. Using constraint techniques for a safe and fast implementation of optimality-based reduction. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 326–331. ACM, 2007.

[109] Leo Liberti. Reduction constraints for the global optimization of nlps. *International Transactions in Operational Research*, 11(1) :33–41, 2004.

[110] Leo Liberti. Reformulations in mathematical programming : automatic symmetry detection and exploitation. *Mathematical Programming*, 131(1-2) :273–304, 2012.

[111] Leo Liberti. Mathematical programming bounds for kissing numbers. In *International Conference on Optimization and Decision Science*, pages 213–222. Springer, 2017.

[112] Leo Liberti, Sonia Cafieri, and Fabien Tarissan. Reformulations in mathematical programming : A computational approach. In *Foundations of Computational Intelligence Volume 3*, pages 153–234. Springer, 2009.

[113] Leo Liberti, Nenad Mladenović, and Giacomo Nannicini. A recipe for finding good solutions to minlps. *Mathematical Programming Computation*, 3(4) :349–390, 2011.

[114] Leo Liberti and Constantinos C Pantelides. Convex envelopes of monomials of odd degree. *Journal of Global Optimization*, 25(2) :157–168, 2003.

[115] Youdong Lin and Linus Schrage. The global solver in the lindo api. *Optimization Methods & Software*, 24(4-5) :657–668, 2009.

[116] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3) :503–528, 1989.

[117] Marco E Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations research*, 53(6) :1007–1023, 2005.

[118] Ladislav Lukšan and Jan Vlcek. Test problems for nonsmooth unconstrained and linearly constrained optimization. *Technická zpráva*, 798, 2000.

[119] Wing-Kin Ken Ma. Semidefinite relaxation of quadratic optimization problems and applications. *IEEE Signal Processing Magazine*, 1053(5888/10), 2010.

[120] Ashutosh Mahajan, Sven Leyffer, Jeff Linderoth, James Luedtke, and Todd Munson. Minotaur : A mixed-integer nonlinear optimization toolkit. *Optimization Online*, 6275, 2017.

[121] Jérôme Malick and Frédéric Roupin. Solving k-cluster problems to optimality with semidefinite programming. *Mathematical programming*, 136(2) :279–300, 2012.

[122] Joseé Mario Martínez and Benar Fux Svaiter. A practical optimality condition without constraint qualifications for nonlinear programming. *Journal of Optimization Theory and Applications*, 118(1) :117–133, 2003.

[123] S. Elloumi M.C. Plateau, A. Billionnet. Eigenvalue methods for linearly constrained quadratic 0-1 problems with application to the densest k-subgraph problem. *6éme congrés ROADEF*, pages 55–66, 2005.

[124] Frédéric Messine and Gilles Trombettoni. Reliable bounds for convex relaxation in interval global optimization codes. In *AIP Conference Proceedings*, volume 2070, page 020050. AIP Publishing, 2019.

[125] Ruth Misener and Christodoulos A Floudas. Glomiqo : Global mixed-integer quadratic optimizer. *Journal of Global Optimization*, 57(1) :3–50, 2013.

[126] Ruth Misener and Christodoulos A Floudas. Antigone : algorithms for continuous/integer global optimization of nonlinear equations. *Journal of Global Optimization*, 59(2-3) :503–526, 2014.

[127] Benjamin Müller, Renke Kuhlmann, and Stefan Vigerske. On the performance of nlp solvers within global minlp solvers. In *Operations Research Proceedings 2017*, pages 633–639. Springer, 2018.

[128] Bruce A Murtagh and Michael A Saunders. A projected lagrangian algorithm and its implementation for sparse nonlinear constraints. In *Algorithms for Constrained Minimization of Smooth Nonlinear Functions*, pages 84–117. Springer, 1982.

[129] Harsha Nagarajan, Mowen Lu, Emre Yamangil, and Russell Bent. Tightening mccormick relaxations for nonlinear programs via dynamic multivariate partitioning. In *International Conference on Principles and Practice of Constraint Programming*, pages 369–387. Springer, 2016.

[130] Stephen G Nash. A survey of truncated-newton methods. *Journal of computational and applied mathematics*, 124(1-2) :45–59, 2000.

[131] J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 1965.

[132] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*, volume 13. Siam, 1994.

[133] Arnold Neumaier and Oleg Shcherbina. Safe bounds in linear and mixed-integer linear programming. *Mathematical Programming*, 99(2) :283–296, 2004.

[134] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer, 2006.

[135] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 354–369. Springer, 2019.

[136] Roman A Polyak. Nonlinear rescaling vs. smoothing technique in convex optimization. *Mathematical Programming*, 92(2) :197–235, 2002.

[137] Michael JD Powell. Algorithms for nonlinear constraints that use lagrangian functions. *Mathematical programming*, 14(1) :224–248, 1978.

[138] Yash Puranik and Nikolaos V Sahinidis. Domain reduction techniques for global nlp and minlp optimization. *Constraints*, 22(3) :338–376, 2017.

[139] Liqun Qi and Jie Sun. A nonsmooth version of newton's method. *Mathematical programming*, 58(1-3) :353–367, 1993.

[140] R Tyrrell Rockafellar. Augmented lagrange multiplier functions and duality in nonconvex programming. *SIAM Journal on Control*, 12(2) :268–285, 1974.

[141] R Tyrrell Rockafellar. Augmented lagrangians and applications of the proximal point algorithm in convex programming. *Mathematics of operations research*, 1(2) :97–116, 1976.

[142] Tiphaine Rougerie. Hybridation de *Feasibility Pump* avec la recherche locale de localsolver. Congrès annuel de la ROADEF, 2019.

[143] Frédéric Roupin et al. Semidefinite relaxations of the quadratic assignment problem in a lagrangian framework. *IJMOR*, 1(1/2) :144–162, 2009.

[144] Hong S Ryoo and Nikolaos V Sahinidis. A branch-and-reduce approach to global optimization. *Journal of global optimization*, 8(2) :107–138, 1996.

[145] Yousef Saad. *Numerical methods for large eigenvalue problems : revised edition*, volume 66. Siam, 2011.

[146] Nikolaos Sahinidis. The baron software for minlp. International symposium on mathematical programming, 2018.

[147] Nikolaos Sahinidis. Spectral relaxations and branching strategies for global optimization of mixed-integer quadratic programs. CRM/DIMACS Workshop on Mixed-Integer Nonlinear Programming, 2019.

[148] Asteroide Santana and Santanu S Dey. The convex hull of a quadratic constraint over a polytope. *arXiv preprint arXiv :1812.10160*, 2018.

[149] Martin WP Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4) :445–454, 1994.

[150] Meinolf Sellmann. Theoretical foundations of cp-based lagrangian relaxation. In *International Conference on Principles and Practice of Constraint Programming*, pages 634–647. Springer, 2004.

[151] Hanif D Sherali and Warren P Adams. *A reformulation-linearization technique for solving discrete and continuous nonconvex problems*, volume 31. Springer Science & Business Media, 2013.

[152] Anders Skjäl and Tapio Westerlund. New methods for calculating alpha bb-type underestimators. *Journal of Global Optimization*, 58(3) :411–427, 2014.

[153] Edward MB Smith and Constantinos C Pantelides. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex minlps. *Computers & Chemical Engineering*, 23(4-5) :457–478, 1999.

[154] Daniel A Spielman and Shang-Hua Teng. Smoothed analysis of termination of linear programming algorithms. *Mathematical Programming*, 97(1-2) :375–404, 2003.

[155] Mohit Tawarmalani and Nikolaos V Sahinidis. *Convexification and global optimization in continuous and mixed-integer nonlinear programming : theory, algorithms, software, and applications*, volume 65. Springer Science & Business Media, 2002.

[156] Mohit Tawarmalani and Nikolaos V Sahinidis. Product disaggregation in global optimization and relaxations of rational programs. In *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming*, pages 71–123. Springer, 2002.

[157] Mohit Tawarmalani and Nikolaos V Sahinidis. Global optimization of mixed-integer nonlinear programs : A theoretical and computational study. *Mathematical programming*, 99(3) :563–591, 2004.

[158] Paul Tseng and Dimitri P Bertsekas. On the convergence of the exponential multiplier method for convex programming. *Mathematical Programming*, 60(1-3) :1–19, 1993.

[159] Stefan Vigerske. Decomposition in multistage stochastic programming and a constraint integer programming approach to mixed-integer nonlinear programming. 2013.

[160] Stefan Vigerske and Ambros Gleixner. Scip : Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. *Optimization Methods and Software*, 33(3) :563–593, 2018.

[161] Xuan-Ha Vu, Hermann Schichl, and Djamila Sam-Haroud. Interval propagation and search on directed acyclic graphs for numerical constraint solving. *Journal of Global Optimization*, 45(4) :499, 2009.

[162] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1) :25–57, 2006.

[163] Tapio Westerlund and Kurt Lundqvist. *Alpha-ECP, version 5.01 : An interactive MINLP-solver based on the extended cutting plane method*. Åbo Akademi Turku, Finland, 2001.

[164] Hermann Weyl. Das asymptotische verteilungsgesetz der eigenwerte linearer partieller differentialgleichungen (mit einer anwendung auf die theorie der hohlraumstrahlung). *Mathematische Annalen*, 71(4) :441–479, 1912.

[165] Jakob Witzig, Timo Berthold, and Stefan Heinz. A status report on conflict analysis in mixed integer nonlinear programming. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 84–94. Springer, 2019.

[166] Philip Wolfe. A duality theorem for non-linear programming. *Quarterly of applied mathematics*, 19(3) :239–244, 1961.

[167] Philip Wolfe. Convergence conditions for ascent methods. *SIAM review*, 11(2) :226–235, 1969.

[168] Philip Wolfe. Convergence conditions for ascent methods. ii : Some corrections. *SIAM review*, 13(2) :185–188, 1971.

[169] Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*. John Wiley & Sons, 2014.

[170] Dexuan Xie and Tamar Schlick. Efficient implementation of the truncated-newton algorithm for large-scale chemistry applications. *SIAM Journal on Optimization*, 10(1) :132–154, 1999.

[171] Dexuan Xie and Tamar Schlick. Remark on algorithm 702—the updated truncated newton minimization package. *ACM Transactions on Mathematical Software (TOMS)*, 25(1) :108–122, 1999.

[172] Luze Xu, Jon Lee, and Daphne Skipper. More virtuous smoothing. *SIAM Journal on Optimization*, 29(2) :1240–1259, 2019.

[173] Qing Zhao, Stefan E Karisch, Franz Rendl, and Henry Wolkowicz. Semidefinite programming relaxations for the quadratic assignment problem. *Journal of Combinatorial Optimization*, 2(1) :71–109, 1998.