Programmation mathématique non convexe non linéaire en variables entières : un exemple d'application au problème de l'écoulement de larges blocs d'actifs

David Nizard

université
PARIS-SACLAY

Programmation mathématique non convexe non linéaire en variables entières : un exemple d'application au problème de l'écoulement de larges blocs d'actifs.

Non convex non linear integer mathematical programming : an example of an application to the problem of trading large block of assets.

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580,
sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique
Graduate School : Informatique et Sciences du Numérique
Référent : Faculté des sciences d'Orsay

Thèse préparée dans l'unité de recherche Laboratoire Interdisciplinaire des Sciences du Numérique, **LISN (Université Paris-Saclay, CNRS)**, sous la direction de **Dominique QUADRI**, Professeure des Universités

Thèse soutenue à Paris-Saclay, le 12 avril 2023, par

David Georges NIZARD

Composition du jury

Membres du jury avec voix délibérative

Sylvain CONCHON Professeur des universités, Université Paris-Saclay, Saclay, France	Président
Sonia CAFIERI Professeure, Ecole Nationale de l'Aviation Civile, Toulouse, France	Rapporteur & Examinatrice
Slim HAMMADI Professeur des universités , Ecole Centrale de Lille, Villeneuve-d'Ascq, France	Rapporteur & Examinateur
Houda LABIOD Expert Leader, Huawei R&D France, Boulogne-Billancourt, France	Examinatrice
Agnès PLATEAU Maîtresse de conférences, Conservatoire National des Arts et Métiers, Paris, France	Examinatrice

THESE DE DOCTORAT
NNT : 2023UPASG015

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Programmation mathématique non convexe non linéaire en variables entières : un exemple d'application au problème de l'écoulement de larges blocs d'actifs.

Mots clés : Programmation mathématique non linéaire en variables mixtes, optimisation non convexe, programmation dynamique, programme factorisable, liquidation optimale de portefeuille.

Résumé : La programmation mathématique fournit un cadre pour l'étude et la résolution des problèmes d'optimisation contraints ou non. Son domaine d'application est extrêmement vaste et riche, passant, sans être exhaustif, par la recherche opérationnelle, l'analyse numérique, les sciences de l'ingénieur, les réseaux, l'économie et la finance. Elle a fait l'objet d'une littérature abondante, depuis la deuxième moitié du XXème siècle et constitue une branche active des mathématiques appliquées.

L'objet de cette thèse est la résolution d'un programme mathématique non convexe non linéaire en variables entières, sous contrainte linéaire d'égalité. Le problème proposé, bien qu'abordé dans cette étude uniquement pour le cas déterministe, trouve son origine en finance, sous le nom d'écoulement de larges blocs d'actifs, ou de liquidation optimale de portefeuille.

Il consiste à vendre une (très large) quantité M donnée d'un actif financier en temps fini (discrétisé en N instants) en maximisant le produit de cette vente. A chaque instant, le prix de vente est modélisé par une fonction de pénalité qui reflète le comportement antagoniste du marché face à l'écoulement progressif. Du point de vue, de la programmation mathématique, cette classe de problèmes est difficile à résoudre et NP-difficile selon Garey et Johnson [51], car la non-convexité de la fonction objectif impose d'adapter les méthodes classiques de résolutions (*e.g Branch & Bound*, coupes) en variables entières. De plus, comme on ne connait pas de méthode de résolution générale pour cette classe de problèmes, les méthodes utilisées doivent être adaptées aux spécificités du problème.

La première partie de cette thèse est consacrée à la résolution exacte ou approchée utilisant la programmation dynamique. Nous montrons en effet, que l'équation de Bellman s'applique au problème proposé et permet ainsi de résoudre exactement et rapidement les petites instances.

Pour les moyennes et grandes instances, où la programmation dynamique n'est plus disponible et/ou performante, nous proposons des bornes inférieures *via* différentes heuristiques utilisant la programmation dynamique ainsi que des méthodes de recherche locale, dont nous étudions la qualité (optimalité, temps CPU) et la complexité.

La seconde partie de la thèse s'intéresse à la reformulation équivalente du problème de thèse sous forme factorisée et à sa relaxation convexe *via* les inégalités de McCormick. Nous proposons alors deux algorithmes de résolution exacte du type *Branch & Bound*, qui fournissent l'optimum global ou un encadrement en temps limité.

Dans une troisième partie, dédiée aux expérimentations numériques, nous comparons les méthodes de résolutions proposées entre elles et aux solvers de l'état de l'art. Nous observons notamment que les bornes obtenues sont souvent proches et mêmes parfois meilleures que celles des solvers libres ou commerciaux auxquels nous nous comparons (*LocalSolver*, *Scip*, *Baron*, *Couenne* et *Bonmin*). De plus, nous montrons que nos méthodes de résolutions peuvent s'appliquer à toute fonction de pénalité suffisamment régulière et croissante, ce qui comprend notamment des fonctions qui ne sont pas actuellement pas prises en charge par certains solvers, bien qu'elles aient un sens économique pour le problème, comme par exemple les fonctions trigonométriques ou la fonction arctangente.

Numériquement, la programmation dynamique permet de résoudre à l'optimum, sous la minute, des instances de taille $N < 100$ et $M < 10\ 000$. Les heuristiques proposées fournissent de très bonnes bornes inférieures, qui atteignent très souvent l'optimum, pour $N < 1\ 000$ et $M < 100\ 000$. Par contraste, la résolution du problème factorisé n'est efficace que pour $N \leq 10$, $M < 1\ 000$, mais nous obtenons des bornes supérieures relativement bonnes. Enfin, pour les grandes instances ($M > 1\ 000\ 000$), nos heuristiques à base de programmation dynamique, lorsqu'elles sont disponibles, fournissent les meilleures bornes inférieures, mais nous n'avons pas d'encadrement précis de l'optimum car nos bornes supérieures ne sont pas fines.

Title : Non convex non linear integer mathematical programming : an example of an application to the problem of trading large block of assets.

Keywords : Mixed integer non linear programming, non convex optimisation, dynamic programming, factorable programming, Optimal liquidation portfolio.

Abstract : Mathematical programming provides a framework to study and resolve optimization problems, constrained or not. Its field of application is vast and rich, ranging, non exhaustively, from Operational Research, Numerical Analysis, Engineering Sciences, Networks, Economy and Finance. It has been extensively studied in the literature, from the second half form the 20^{th} century and represents an active domain of Applied Mathematics.

The aim of this thesis is to solve an non convex, non linear, pure integer, mathematical program, under a linear constraint of equality. This problem, although studied in this dissertation only in the deterministic case, stems from a financial application, known as the large block sale problem, or optimal portfolio liquidation.

It consists in selling a (very large) known quantity M of a financial asset in finite time, discretized in N points in time, while maximizing the proceeds of the sale. At each point in time, the sell price is modeled by a penalty function, which reflects the antagonistic behavior of the market in response to our progressive selling flow.

From the standpoint of the mathematical programming, this class of problems is hard to solve and NP-hard according to Garey et Johnson [51], because the non convexity of the objective function imposes on us to adapt classical resolutions methods (*Branch & Bound*, cuts) for integer variables. In addition, as no general resolution method for this class of problems is known, the methods used for solving must be adapted to the problem specifics.

The first part of the thesis is devoted to solve the problem, either exactly or approximately, using Dynamic Programming. We indeed prove that Bellman's equation applies to the problem studied and thus enables to solve it exactly and quickly for small instances. For medium and large instances, for which Dynamic Programming is either not available and/or efficient, we provide lower bounds using different heuristics relying on Dynamic Programming, or local search methods, for which performance (tightness and CPU time) and complexity are studied.

The second part of this thesis focuses on the equivalent reformulation of the problem in a factored form, and on its convex relaxation using McCormick's inequalities. We introduce two exact resolution algorithms, which belongs to the *Branch & Bound* category. They return the global optimum or bound it in limited time.

In a third part, dedicated to numerical experiments, we compare our resolution methods between each other and to state of the art solvers. We notice in particular that our bounds are comparable and sometimes even better than solvers' bounds, both free and commercial (*LocalSolver*, *Scip*, *Baron*, *Couenne*, and *Bonmin*), which we use as benchmark. In addition, we show that our resolution methods may apply to sufficiently regular and increasing penalty functions, especially functions which are currently not handled by some solvers, even though they make economic sense for the problem, as does trigonometric functions or the arctangent function for instance.

Numerically, Dynamic Programming does optimally solve the problem, within a minute, for instances of size $N < 100$ and $M < 10\ 000$. Our heuristics provide very tight lower bounds, which often reach the optimum, for $N < 1\ 000$ and $M < 100\ 000$. By contrast, optimal resolution of the factored problem proves efficient for instances of size $N < 10, M < 1\ 000$, even though we obtain relatively good upper bounds. Lastly, for large instances ($M > 1\ 000\ 000$), our heuristics based on Dynamic Programming, when available, return the best lower bounds. However we are not able to bound the optimum tightly, since our upper bounds are not thin.

Université Paris-Saclay - Ecole Doctorale STIC
Bâtiment 650 - Ada Lovelace
91405 Orsay Cedex, France

université PARIS-SACLAY

Table des matières

Table des figures

Liste des tableaux

Glossaire

BARON : *Branch-And-Reduce Optimization Navigator.*

BONMIN : *Basic Open-source Nonlinear Mixed INteger programming.*

Branch & Bound : méthode exacte par séparation et évaluation, introduite initialement par Land et Doig [70].

COIN-OR : *COmputer INfracstructure for Operations Research.*

COUENNE : *Convex Over and Under ENvelopes for Nonlinear Estimation.*

DAG (*Directed Acyclic Graph*) : graphe acyclique orienté.

DP (*Dynamic Programming*) : programmation Dynamique. Théorie générale des processus de décisions multi-états (cf. définition de Bellman [15]).

Dark pool : marché ou bourse privée permettant d'échanger des instruments financiers entre institutionnels, de manière opaque, sans reporter les détails de la transaction au reste du marché avant son exécution.

ELBA : Ecoulement de Larges Blocs d'Actifs.

ILS (*Iterated Local Search*) : recherche locale par itérations.

MINLP (*mixed integer non linear programming*) : programmation non linéaire en variables mixtes.

OPL (ou LOP) (*Optimal Liquidation Portfolio*) : liquidation optimale de portefeuille.

PLNE/PLVE : Programmation (mathématique) linéaire en nombres entiers / variables entières.

PNL : Programmation mathématique non linéaire.

PNLNE/PNLVE : programmation (mathématique) non linéaire en nombres entiers / variables entières.

PNS : Problème non séparable. Acronyme du problème de thèse.

PQVE : Programmation quadratique en variables entières.

PRC : Problème relâché convexe. Relaxation convexe du problème PNS.

RLT (*Reformulation and linearization techniques*) : techniques de reformulations et de linéarisation (d'un programme mathématique).

rRLT (*reduced RLT*) : RLT réduite (réduction du nombre de contraintes).

SCIP : *Solving Constraint Integer Programs.*

Survey : (littéralement : sondage ou enquête) revue de l'état de l'art sur un sujet donné.

TSDP (*Two Stage Dynamic Programming*) : méthode en deux étapes fondée sur la programmation dynamique.

Upstairs market : (littéralement «marchés de l'étage au dessus»). Dans une banque, le bureau ou la salle de marché dédiée aux transactions de large blocs d'actifs est séparée physiquement de la salle de marché principale. L'appellation fait référence à sa location géographique, qui se trouve, communément, dans les étages supérieurs.

Remerciements

L'aventure d'une thèse est un long chemin initiatique porteur d'espoir. Elle incarne la promesse de nouveaux horizons ou d'un retour aux sources.
Dans mon cas, ce fut une renaissance, l'achèvement d'une œuvre démarrée il y a vingt ans, délaissée au profit de frivoles chimères, qui ont muté en une besogne cléricale, manquant souvent de stimulation intellectuelle.
Bien qu'il soit essentiellement solitaire, ce chemin a néanmoins été rythmé par de nombreuses rencontres, toutes enrichissantes, sans lesquelles je n'aurais jamais pu commencer ni terminer ma thèse.

Je tiens tout d'abord à remercier les professeurs Alain Sibille et Djamal Zeghlache, qui en février 2018 m'ont reçu en entretien, ont perçu ma motivation profonde et m'ont donné le droit de m'inscrire en doctorat, après 15 ans passés loin des bancs de la faculté.
Je remercie l'Ecole Doctorale STIC et plus particulièrement les professeurs Alain Denise, pour son accueil chaleureux et son écoute patiente et Nadjib AitSaadi, qui m'a fourni le cadre rigoureux dont j'avais besoin pour progresser et achever mes travaux.
Je remercie également le personnel informatique du LRI/LISN et plus particulièrement Laurent Darré, pour son aide amicale et ses réponses complètes, rapides et précises.
Je remercie les sœurs Anne et Marie-Pierre Armandou, ces douces âmes bienveillantes qui m'ont guidé et ont simplifié les démarches administratives et logistiques, parfois complexes, toujours sous la pression du temps. Elles m'ont permis, par des gestes simples et habiles (valider une inscription, faire intervenir un serrurier ou un plombier à mon bureau) de me consacrer sur l'essentiel et je leur en suis infiniment reconnaissant.
Je remercie les professeurs et maîtres de conférences du pôle C et parmi eux, je remercie Steven Martin, pour sa gentillesse et son aide au début de la thèse, qui ont contribué à me mettre sur de bons rails.
Je remercie Nicola Zema pour sa maîtrise sans faille des méandres de Linux et le temps qu'il m'a toujours gentiment consacré, pour installer les nombreux outils et solvers néces-

saires à ma recherche. Je remercie Nicolas Dupin pour ses cours de C++ et de calculs parallèles. Je remercie les professeurs Martin Schmidt, pour ses cours doctoraux sur les problèmes non convexes et Stefan Vigerske, pour son aide précieuse sur *Scip*.
Je remercie enfin tous les membres du jury pour leur lecture attentive de mon travail et pour leurs questions pertinentes, auxquelles j'espère avoir répondu de manière satisfaisante.

Hors de l'Académie, ma famille et mes proches m'ont encouragé et accompagné, lorsque je leur ai exposé, ce projet un peu fou de faire une thèse avec une famille et un travail en parallèle et je tiens à les en remercier.
Je tiens tout d'abord à avoir une pensée émue pour mes quatre grands-parents, Marcelle et Georges Nizard, Yvonne et Meyer Bellaiche, qui m'ont inculqué les valeurs du travail et des études. J'ai une pensée toute particulière pour ma grand-mère Yvonne, qui nous a quitté la semaine dernière. Elle aurait surement fait aujourd'hui une remarque pleine d'humour et elle nous manque.
Je remercie mes parents, Patricia et Jean-Claude, mes frères Niels et Alexis, pour leur support inconditionnel, quand moi-même je commençais à douter.
Je remercie également mes cousins Mandy et Mevyn, qui m'ont montré le chemin de la soutenance et pour qui la valeur n'a point attendu le nombre des années.
Je remercie également mes beaux-parents Carine et Bernard Cohen, et mes belles sœurs Leslie et Laura, pour toute l'aide qu'ils ont et qu'ils continuent de nous apporter.
Je vous remercie tous d'avoir gardé mes enfants et les avoir emmené dans différents parcs, activités, compétitions sportives, pour me permettre d'avancer et de terminer.

Je remercie mon oncle Guy-Pierre, mon éternel compagnon de flibuste, pour sa maîtrise de l'Arduino, du fer à souder et des imprimantes 3D, ainsi que pour ses conseils techniques et pratiques avisés, qui ont permis de canaliser mon énergie créatrice. Parrain, nous avons encore des projets qui nous attendent.

Je remercie ma femme Sandra. Chérie, tu as su voir que l'entreprise chronophage dans laquelle je t'ai un peu précipitée, était essentielle à mon équilibre et à mon accomplissement. Tu as fait tant d'efforts et de sacrifices, a supporté la majorité de la charge éducative que représente nos trois enfants, en particulier ces deux dernières années, depuis la fin du confinement. Tu as tenu bon dans le sprint final du manuscrit, dans les prolongations de

la soutenance, et en ce jour, je mesure à quel point cela a été dur pour toi.
Je ne te remercierai jamais assez, ma lumière, et j'espère vite te démontrer à quel point cela en valait la peine.
Je remercie mes enfants, Ethel, Benjamin et Leah. J'espère qu'ils me pardonneront et comprendront en grandissant que ce temps que je n'ai pas passé avec eux, je l'ai aussi passé pour eux.

Et enfin, je remercie celle avec qui j'ai préparé et mûri ce projet, celle qui m'a aidé à en définir le sujet, celle qui a insufflé vie à cette thèse, qui l'a encadrée, accompagnée, portée : Dominique, du fond du cœur MERCI.
Je n'y serai jamais arrivé sans toi et tu auras toujours une place particulière au sein de notre famille. Je te remercie de ton immense générosité et j'espère pouvoir encore beaucoup apprendre à tes cotés.

Introduction

La programmation mathématique fournit un cadre pour l'étude et la résolution des problèmes d'optimisation contraints ou non. Son domaine d'application est extrêmement vaste et riche, passant, sans être exhaustif, par la recherche opérationnelle, l'analyse numérique, les sciences de l'ingénieur, les réseaux, l'économie et la finance. Elle a fait l'objet d'une littérature abondante, depuis la deuxième moitié du XXème siècle et constitue une branche active des mathématiques appliquées.

L'objet de cette thèse est la résolution exacte d'un programme mathématique non convexe non linéaire en variables entières, sous contrainte linéaire d'égalité. Le problème proposé, que nous notons $(PNS_{N,M})$, bien qu'abordé dans cette étude uniquement pour le cas déterministe, trouve son origine en finance, sous le nom d'Ecoulement de Larges Blocs d'Actifs (ELBA), ou de Liquidation Optimale de Portefeuille (LOP).
Il consiste à vendre une (très large) quantité M donnée d'un actif financier en temps fini (discrétisé en N instants) en maximisant le produit de cette vente. A chaque instant, le prix de vente est modélisé par une fonction de pénalité qui reflète le comportement antagoniste du marché face à l'écoulement progressif.

Du point de vue de la programmation mathématique, cette classe de problèmes est difficile à résoudre et NP-difficile selon Garey et Johnson [51], car la non-convexité de la fonction objectif impose d'adapter les méthodes classiques de résolutions (*e.g Branch & Bound*, coupes) en variables entières. De plus, comme on ne connait pas de méthode de résolution générale, les techniques utilisées doivent être adaptées aux spécificités du problème. Nous explorons dans cette étude deux axes principaux de résolution exacte.

Le premier axe est inhérent à la structure du problème. En effet, nous montrons que $(PNS_{N,M})$ peut se décomposer en plusieurs problèmes plus simples et satisfait l'équation de Bellman, qui fournit une relation de récurrence entre les optima selon la taille de

l'instance. Dès lors, les techniques de Programmation Dynamique (DP), introduites par Bellman [14], vont s'appliquer et permettre de le résoudre exactement et efficacement pour les petites instances.
Pour les moyennes et grandes instances, où la DP n'est plus disponible et/ou performante, nous proposons des bornes inférieures *via* différentes heuristiques utilisant la DP, dont nous étudions la qualité (optimalité, temps CPU) et la complexité.

Le second axe de résolution consiste à convexifier le problème. Dans un contexte de maximisation, cela signifie que nous proposons des fonctions majorantes concaves, appelées surestimations concaves, de plus en plus proches de la fonction objectif initiale. Les problèmes convexes correspondants, appelés problèmes relâchés convexes, sont d'une part plus faciles à résoudre à l'optimum, et fournissent d'autre part des bornes supérieures, qui, sous certaines conditions, convergent vers l'optimum global du problème traité.
Pour pouvoir convexifier le problème, nous passons au préalable, par une étape de reformulation, c'est à dire une transformation du problème initial en un problème équivalent, qui permet d'en isoler les termes non convexes, afin de pouvoir les majorer. La reformulation proposée s'appuie sur la notion de programme *factorisable* introduite par McCormick [79]. La relaxation convexe de $(PNS_{N,M})$ peut alors nous servir de fonction d'évaluation pour un algorithme d'énumération implicite de type *Branch & Bound*. Il est important de noter que nous ne pourrons pas utiliser la relaxation continue de $(PNS_{N,M})$ directement, car elle est non convexe. Sa résolution globale, dont nous aurions besoin à chaque étape du *Branch & Bound*, est aussi difficile que celle du problème initial.

Notre démarche face à ce problème est une approche en ciseau. Nous essayons donc de le résoudre exactement (préférentiellement par la DP dont la complexité est quasi polynomiale). Cependant, lorsque la résolution exacte n'est pas efficace, nous proposons un encadrement de l'optimum, en s'appuyant «à gauche» (dans un contexte de maximisation) sur les heuristiques proposées issues de la DP et «à droite» sur les bornes supérieures fournies par les relaxations convexes.

Contributions : les principales contributions de cette thèse sont les suivantes :

— Nous proposons un modèle de pénalité à mémoire longue, avec une fonction de pénalité g en $g(\sum_{j=1}^{k} x_j)$, ce qui, pour des fonctions fortement non linéaires croissantes g est, à notre connaissance, nouveau pour la littérature financière reliée au

problème ELBA.

— Nous utilisons la DP pour résoudre un programme mathématique non convexe, non linéaire en variables entières, de manière exacte pour les instances de petites tailles et nous proposons une méthode en deux étapes (TSDP), basée sur la DP, qui renvoie des bornes inférieures très fines, voire optimales, pour les instances de tailles moyennes.
— Nous proposons différents algorithmes pour obtenir des bornes inférieures pour le problème $(PNS_{N,M})$ en variables entières.
— Nous établissons, *via* la factorisation du problème, une relaxation convexe fine du problème traité. Nous utilisons cette relaxation au sein d'algorithmes de type *Branch & Bound*, afin d'obtenir l'optimum global pour les très petites instances, ou une borne supérieure fine permettant d'encadrer l'optimum.
— Nous comparons les performances de nos méthodes à celles des solvers de l'état de l'art. Pour la résolution par DP, nous faisons mieux, sur de nombreuses instances, que le solver (fournissant des solutions admissibles) commercial *LocalSolver* (v9.5, *cf.* Benoist *et al.* [21]) et obtenons un ratio de couverture optimale beaucoup plus élevé.
— Nous obtenons des bornes inférieures très proches et parfois meilleures que celles de *Bonmin* (v1.8.8, *cf.* [31],[30]). Notre ratio de couverture optimale est plus élevé que celui du solver, pour les petites et moyennes instances.
— Pour la résolution par convexification, les bornes supérieures obtenues sont meilleures, sur la plupart des instances, que celles du solver libre *Scip* (v7.0.1, *cf.* [50]), que celles de *Baron* (v.2022.11.3 *cf.* [110],[98]).
Nous obtenons également une résolution exacte sur quelques instances de plus que *Scip* et *Baron*.
— Pour certaines fonctions de pénalité, les bornes supérieures obtenues sont meilleures que *Couenne* (v0.5.8, *cf.* [18],[16]), qui est notre solver global de référence.

Organisation du document :

Le chapitre 1 est consacré à la formulation du problème. Nous introduisons le problème à la section 1.1, puis nous présentons l'intérêt du problème traité et ses applications à la section 1.2.

Le chapitre 2 situe le problème traité dans l'état de l'art. Nous nous intéressons aux ap-

proches et aux reformulations classiques de la programmation mathématique non convexe dans la littérature. Nous commençons, à la section 2.1, par la programmation quadratique non convexe, puis, nous étendons au cas polynomial, à la section 2.2. Nous nous intéressons ensuite aux différentes approches dans le cas général à la section 2.3. Nous présentons enfin, à la section 2.4, l'état de l'art financier relatif à l'application ELBA .

Le chapitre 3 est consacré à la DP. Après des rappels de DP, à la section 3.1, nous démontrons l'équation de Bellman et en déduisons une résolution exacte à la section 3.2. Nous présentons ensuite à la section 3.3 différents algorithmes permettant de d'obtenir un maximum local de bonne qualité pour le problème traité. Nous introduisons notamment notre méthode en deux étapes TSDP. Nous proposons des résolutions approchées du problème continu à la section 3.4. Nous proposons enfin, à la section 3.5 une première borne supérieure du problème initial en utilisant un argument de monotonie.

Le chapitre 4 est consacré à la convexification du problème traité. Nous démarrons par des rappels de convexité et de factorisation à la section 4.1, puis nous appliquons cette technique au problème traité et en déduisons une factorisation dans le cas général à la section 4.2. Nous établissons la relaxation convexe du problème à la section 4.3 avec des contraintes de bornes indépendantes pour chaque variable de décision. Nous proposons une linéarisation du problème relâché convexe à la section 4.4. Nous présentons enfin les algorithmes de *Branch & Bound* utilisés à la section 4.5.

Le chapitre 5 est consacré aux expérimentations numériques. Nous définissons les outils, les données et les conventions utilisées, à la section 5.1. Nous discutons notamment la calibration des fonctions de pénalité. Nous présentons les résultats de la comparaison numérique des méthodes, à la section 5.2, pour les petites et moyennes instances, puis à la section 5.3 pour les (très) grandes instances.

Enfin, **le chapitre 6** offre une conclusion générale et présente les perspectives de notre étude.

Chapitre 1

Formulation du problème

Dans ce chapitre, nous présentons le problème étudié. Dans la section 1.1, nous en définissons le cadre formel, les hypothèses de travail et en établissons la complexité.
Puis, nous consacrons la section 1.2 aux motivations qui ont conduit à l'étude de ce problème. Nous nous intéressons en particulier à une application financière possible du problème d'Ecoulement de Larges Blocs d'Actifs (ELBA) dans un cadre déterministe.

1.1 Formulation et complexité

Soit $f : \mathbb{N}^N \to \mathbb{R}$, telle que $f(x) = \sum_{i=1}^{N} \Big[p_i - c_i \cdot g\Big(\sum_{k=1}^{i} x_k\Big)\Big] \cdot x_i$, où g est une fonction croissante de $\mathbb{N}$ dans $\mathbb{R}_+$ et les coefficients p_i et c_i sont des réels strictement positifs. Les variables $(x_1, \cdots, x_i, \cdots, x_N)$ sont à composantes entières. Nous considérons le programme mathématique non linéaire, en général non convexe, non séparable en variables entières, noté $(PNS_{N,M})$ qui s'énonce de la manière suivante :

$$(PNS_{N,M}) \begin{cases} \max f(x) & = \sum_{i=1}^{N} \Big[p_i - c_i \cdot g\Big(\sum_{k=1}^{i} x_k\Big)\Big] \cdot x_i \\ s.c & \left| \begin{array}{l} \sum_{i=1}^{N} x_i = M \\ \forall i,\ 0 \leq x_i \leq M \\ \forall i,\ x_i \in \mathbb{N} \end{array} \right. \end{cases} \tag{1.1}$$

Les x_i , pour i allant de 1 à N, sont les N variables de décisions du problème. Elles prennent leurs valeurs parmi les entiers naturels.
Le problème n'a qu'une contrainte linéaire d'égalité, de second membre M, où M est une constante positive. La fonction g est appelée fonction de pénalité.

Soit $\mathcal{D}$ l'ensemble des solutions admissibles :

$$\mathcal{D} = \left\{ (x_1, \ldots, x_N) \in \mathbb{N}^N, \quad \sum_{i=1}^{N} x_i = M \right\} \tag{1.2}$$

$(PNS_{N,M})$ est bien défini car $\mathcal{D}$ est fini. Il peut également s'écrire en notation réduite : $Opt_{N,M} = \max_{x \in \mathcal{D}} f(x)$

Nous nous intéressons également à sa relaxation continue $(\overline{PNS_{N,M}})$. Pour cela, nous étendons les définitions précédentes. Soit $\mathcal{C}$, l'ensemble des solutions admissibles à valeurs réelles :

$$\mathcal{C} = \left\{ (x_1, \ldots, x_N) \in \mathbb{R}_+^N, \forall i \; x_i \leq M, \quad \sum_{i=1}^{N} x_i = M \right\} \tag{1.3}$$

Soit $\overline{v_i}(x_i) = \left[p_i - c_i \cdot \overline{g}\Big(\sum_{k=1}^{i} x_k \Big) \right] x_i$ et $\overline{f}$, de $\mathbb{R}^N \to \mathbb{R}$, définie par : $\overline{f}(x) = \sum_{i=1}^{N} \overline{v_i}(x_i)$

où $\overline{g}$ est l'extension réelle de g, définie, continue et strictement croissante sur $\mathbb{R}_+$ et $\overline{f}$ celle de f de $\mathbb{R}^N \to \mathbb{R}$.

Par analogie au problème en variables entières, $(\overline{PNS_{N,M}})$ s'énonce :

$$(\overline{PNS_{N,M}}) \begin{cases} \max \overline{f}(x) & = \sum_{i=1}^{N} \Big[p_i - c_i \cdot \overline{g}\Big(\sum_{k=1}^{i} x_k \Big) \Big] x_i \\ s.c & \left| \begin{array}{l} \sum_{i=1}^{N} x_i = M \\ \forall i, \; 0 \leq x_i \leq M \\ \forall i, \; x_i \in \mathbb{R}_+^N \end{array} \right. \end{cases} \tag{1.4}$$

En notation réduite :

$$\overline{Opt}_{N,M} = \max_{x \in \mathcal{C}} \overline{f}(x)$$

Pour montrer que $(\overline{PNS_{N,M}})$ est bien défini, nous nous appuyons sur le résultat suivant :

Lemme 1. *$\mathcal{C}$ est compact.*

Démonstration. Soit N un entier (égal au nombre variables de décision), M un réel (égal au second membre) et u la fonction de $\mathbb{R}^N \to \mathbb{R}$ définie par $u(x) = \sum_{i=1}^{N} x_i$.

Le singleton $\{M\}$ est un fermé de $\mathbb{R}$ (car $\mathbb{R} \backslash \{M\}$ est ouvert) donc son image réciproque par une fonction continue est un fermé de $\mathbb{R}^N$ (caractérisation topologique de la continuité). De plus $[0; M]^N$ est un fermé borné de $\mathbb{R}^N$. Par conséquent $\mathcal{C} = u^{-1}(\{M\}) \cap [0; M]^N$ est un fermé borné de $\mathbb{R}^N$ ce qui prouve sa compacité. □

Nous en déduisons que lorsque $\overline{g}$ est continue alors le problème (1.4) est bien défini. Nous supposerons, dans toute notre étude que $\overline{g}$ est au moins continument différentiable et on confondra g et $\overline{g}$, f et $\overline{f}$, lorsqu'il n'y a pas d'ambiguité, pour simplifier les notations.

Remarque 1. *Lorsque $M \gg N$, le problème en variables entières fournit une bonne borne inférieure du problème en variables continues.*

Remarque 2. *La fonction objectif f est bornée.*
Nous notons $\overline{p} = \max_i p_i$ et nous obtenons alors, $0 \leq f(x) \leq M\,\overline{p}$. Cette majoration donne une borne supérieure naturelle de l'objectif, indépendante de la pénalité.

Remarque 3. *Dans l'application financière ELBA, nous supposerons également que $g(0) = 0$ et $\lim_{+\infty} g = 1$. Cette hypothèse supplémentaire n'est pas restrictive, car pour toute fonction h, $\mathcal{C}^1$ strictement croissante sur $\mathbb{R}_+$ (ou au moins sur $[0; M]$), nous pouvons définir g par : $g(x) = H \cdot \dfrac{h(x) - h(0)}{h(M) - h(0)}$, avec H une constante réelle de $]0; 1]$. Nous avons bien alors $g(0) = 0$ et $g(M) = H$, arbitrairement proche de 1.*

Complexité : Les problèmes $(PNS_{N,M})$ et $(\overline{PNS_{N,M}})$ sont NP-difficiles.
En effet, la programmation linéaire en variables binaires est un des problèmes de Karp dont il a montré la NP-complétude dans Karp [65] . La Programmation Linéaire en Variables Entières (PLVE ou PLNE) est NP-complet au sens fort (cf. Garey et Johnson [51] et Karp [65]). Selon Liberti [73], la Programmation Linéaire en Variables Mixtes (PLVM) est NP-difficile par réduction au problème SAT (*Boolean Statisfiability*), qui est lui même NP-difficile (selon l'article fondateur de Cook [38]).
Nous en concluons donc que la classe des Programmes Non Linéaires en Variables Mixtes (en abrégé PNLVM) (resp. PNLVE) est NP difficile car elle contient la sous-classe des PLVM (resp. PLVE).
Par ailleurs, nous montrons que $(PNS_{N,M})$ peut être résolu par programmation dynamique en temps pseudo polynomial. Nous en concluons donc qu'il n'est pas NP difficile au sens fort, comme observé dans Garey et Johnson [51] (p95, observation 4.2).

NP complétude : Nous considérons le problème de décision associé à $(PNS_{N,M})$ ou à sa relaxation continue $(\overline{PNS}_{N,M})$. Soit B un réel donné, existe-t-il un élément de $\mathcal{D}$ (resp. $\mathcal{C}$) tel que $f(x) \geq B$. Soit x^0 une solution fournie par un "oracle". Son appartenance à l'ensemble des solutions admissibles se vérifie en temps linéaire (donc polynomial), par

linéarité de la contrainte. Pour les fonctions de pénalisation considérées dans nos expérimentations numériques $f(x) \geq B$ se vérifie en temps polynomial. Nous en concluons que notre problème est NP-complet dans sa version continue et faiblement NP-complet dans sa version discrète.

1.2 Intérêt du problème

Il y a plusieurs raisons d'étudier le problème $(PNS_{N,M})$. Tout d'abord, il fait écho à des problèmes de recherche opérationnelle de l'histoire récente, comme la taille des convois de bateaux alliés durant la seconde guerre mondiale.
Le problème est alors d'acheminer des convois de bateaux à travers l'Atlantique, sillonnée par la marine ennemie, supérieure techniquement. En effet, comme le raconte Falconer [45], les équipes de recherche opérationnelles de la Royal Navy, menées par le Professeur Patrick Blackett, ont étudié le risque de détection, la taille des escortes et les pertes alliées correspondantes face aux sous-marins allemands *U-Boat* et ont convaincu l'état major britannique (et vraisemblablement Winston Churchill, qui jugeait que la «bataille de l'Atlantique» était d'une importance capitale dans l'issue de la guerre) de changer de politique en préférant, à partir du printemps 1943, aux petits convois, les plus larges convois (plus de 40 vaisseaux). Cette décision a joué, selon Falconer [45], un rôle prépondérant dans la préparation du débarquement des troupes alliées en Europe. L'impact de la recherche opérationnelle en temps de guerre est également rapporté par McCloskey [78].

Un autre parallèle à notre problème de thèse, se trouve également en sciences naturelles, dans l'étude du comportement de chasse des prédateurs en approche de leurs proies. Les variables clés, sont la distance à la proie, le risque de détection par celle-ci et par le troupeau entier en réaction à une tentative attaque et la vitesse relative de la proie par rapport au prédateur. Dans la littérature, le problème, abordé par l'optimisation combinatoire est désigné par *Hunter Prey Optimization* (HPO), ou *Predator Prey Optimization* (PPO). Un problème connexe est également *Grey Wolf Optimization* (GWO) qui propose des heuristiques inspirées des techniques de chasse en meutes des loups. Nous nous référons aux récents travaux de Naruei *et al.* [88] et Mirjalili *et al.* [85], Faris *et al.* [47] et rappelons néanmoins que nous ne sommes pas des spécialistes.

Nous nous intéressons, dans cette thèse, à une application financière du problème, l'Ecoulement de Larges Blocs d'Actifs (ELBA), dans un cadre déterministe.

Nous considérons un investisseur dans un marché financier. Il détient initialement un grand nombre M unités d'un actif et souhaite l'écouler (le vendre) entièrement durant un intervalle de temps de déterminé, discrétisé en N pas de temps. Nous supposons en général que M est large devant N.

Nous pouvons alors interpréter $(PNS_{N,M})$ dans le cadre déterministe suivant : x_i s'interprète comme la quantité d'actif écoulée au temps t_i. $x = (x_1, \cdots, x_i, \cdots, x_N)$ correspond à une solution, appelée stratégie d'écoulement.

Nous nous plaçons dans le cadre d'un écoulement unidirectionnel (uniquement composé de ventes), pour lequel nous supposons, à chaque pas de temps, que $x_i \geq 0$.

Par conséquent, l'ensemble $\mathcal{D}$ correspond à l'ensemble des stratégies réalisables (ou solutions admissibles) du problème en variables entières. Le coefficient p_i, strictement positif, correspond alors au prix de l'actif à l'instant t_i. Nous supposons également qu'il existe un prix plancher $q_i > 0$, pour les ordres de très larges blocs, avec $0 < q_i < p_i$.

Pour simplifier les notations, nous définissons également : $c_i = p_i - q_i > 0$.

La fonction g croissante de $\mathbb{N}$ dans $[0, 1[$, telle que $g(0) = 0$ et $\lim\limits_{+\infty} g = 1$, correspond à la fonction de pénalité de l'écoulement.

Au temps t_i, le prix d'exécution du bloc de x_i unités d'actif est donné par :

$$v_i(x_i) = p_i\, x_i - (p_i - q_i) \cdot x_i \cdot g\left(\sum_{k=1}^{i} x_k\right) = \left[p_i - c_i \cdot g\Big(\sum_{k=1}^{i} x_k\Big)\right] x_i \tag{1.5}$$

Soit f la fonction objectif de $\mathbb{N}^N \to \mathbb{R}$:

$$f(x) = \sum_{i=1}^{N} v_i(x_i) = \sum_{i=1}^{N} \Big[p_i - c_i \cdot g\Big(\sum_{k=1}^{i} x_k\Big)\Big] x_i \tag{1.6}$$

La fonction de pénalité g traduit la réponse du marché à l'action de vente de l'investisseur, prenant en compte la mémoire du marché. En effet, p_i n'est valide que pour un très faible volume comparé à taille M du bloc initial. Des volumes plus importants engendrent un prix d'exécution plus faible, cependant la vitesse de décroissance des prix reflète la connaissance des participants de marché vis à vis de nos intentions. Plus grande est cette connaissance, plus faible sera le prix. La fonction de pénalité g s'applique à $y_i = \sum_{k=1}^{i} x_k$ qui représente la quantité d'actif écoulé jusqu'au temps courant t. Le terme y_i mesure également l'information disponible dans le marché à mémoire parfaite. Ainsi q_i correspond à l'impact maximum sur le prix d'exécution, lorsque le marché réagit, tout en connaissant parfaitement nos intentions. Le terme y_i est croissant en i et la pénalité g est croissante en y_i. Plus la valeur de y_i est élevée, plus le prix d'exécution se rapproche de q_i, ce qui

justifie nos hypothèses (monotonie, limites) sur la fonction g. Par conséquent, le problème $(PNS_{N,M})$ modélise le problème de l'investisseur, qui cherche à maximiser le produit de la vente du bloc dans le marché, sous contrainte de temps et s'énonce de la façon suivante.

$$(PNS_{N,M}) \begin{cases} \max f(x) & = \sum_{i=1}^{N} \Big[p_i - c_i \cdot g\Big(\sum_{k=1}^{i} x_k \Big) \Big] x_i \\ s.c & \left| \begin{array}{l} \sum_{i=1}^{N} x_i = M \\ \forall i,\ 0 \leq x_i \leq M \\ \forall i,\ x_i \in \mathbb{N} \end{array} \right. \end{cases} \tag{1.7}$$

Dans ce cadre, l'étude théorique de sa version discrète est une étape dans la résolution du problème classique ELBA. Cependant, comme le montre la section 2.4 de l'état de l'art financier, les majorités des travaux font l'hypothèse d'un prix stochastique qui fait intervenir l'impact de la vente de blocs, tandis que notre maximisation suppose un prix déterministe et indépendant de l'écoulement.

Comme nous l'avons remarqué à la section précédente, bien que l'on ait particularisé la fonction de pénalisation à quelques cas, pour les expérimentations numériques, les techniques de résolutions de notre étude sont valables pour toute fonction $\mathcal{C}^1$ strictement croissante sur $\mathbb{R}_+$ (ou au moins sur $[0; M]$)
De ce fait, nous ne nous limitons pas aux fonctions de pénalité de type ingénieur (exponentiel, logarithme, puissance, etc.). Nous pouvons notamment inclure les fonctions trigonométriques et arc-trigonométriques, qui ne sont pas actuellement prises en charge par les solvers, bien qu'elles puissent avoir du sens dans le contexte du problème étudié, comme par exemple l'arctangente pour le problème ELBA.

Du point de vue empirique, pour les opérateurs financiers, le problème ELBA reste tout à fait pertinent. En effet, comme l'a rappelé l'étude empirique récente du *Wall Street Journal* (*cf.* Hoffman *et al.* [60], mars 2022) sur la vente de larges blocs, la dissémination de l'information (parfois même pré-écoulement) dans le marché, a des conséquences très néfastes sur les résultats de l'investisseur, ce qui tend à montrer l'importance cruciale de ne pas trahir ses intentions en adoptant une stratégie d'écoulement appropriée (qui minimise l'information transmise au marché).

Enfin, une dernière motivation pour étudier ce problème particulier tient à la diversité

des méthodes de résolutions, exactes ou approchées, qui lui sont applicables, ce que nous n'avions pas anticipé au début de la thèse (la DP est venue après l'étude du problème). En effet, la programmation dynamique, étudiée dans le chapitre 3, permet une résolution exacte et rapide pour les petites instances et fournit des bornes inférieures très fines, souvent optimales, pour les moyennes instances (une définition précise des caractéristiques des instances ainsi que la terminologie correspondante «petite, moyenne ou grande», instance est présentée au chapitre 5, en amont des expérimentations numériques).

La factorisation du problème et sa relaxation convexe, étudiées au chapitre 4, ne résolvent quant à elles que les très petites instances à l'optimum, mais fournissent de relativement bonnes bornes supérieures sur les petites et moyennes instances.

Chapitre 2

Etat de l'art

Souvent guidée par les besoins industriels, comme la modélisation de réseaux de transport de gaz (Koch *et al.* [68], chapitre 7), d'eau (Bragalli *et al.* [33]), d'électricité (Donde *et al.* [42]) ou le refroidissement d'un réacteur nucléaire (Leyffer *et al.* [72]) l'optimisation non convexe est l'objet d'intenses recherches. Sans chercher l'exhaustivité, qui n'entre pas dans le cadre de cette étude, nous renvoyons le lecteur vers les ouvrages de Horst et Tuy [61] et de Tawarmalani et Sahinidis [111], ainsi qu'aux nombreux *surveys*, dont Burer et Letchford [34] et Belotti *et al.* [17], pour une vue synthétique.

Nous nous intéressons, dans cette thèse, à un programme mathématique non convexe, non séparable, non linéaire en variables entières. Nous présentons, dans cette section, la démarche que nous avons suivie pour situer notre problème dans ce vaste état de l'art.
En général, ces problèmes sont NP-difficiles, comme discuté dans Garey et Johnson [51], et plus particulièrement pour notre problème, à la section 1.1. Cette classe de problèmes, qui trouve dans de nombreuses applications dans la vie réelle, est, en théorie comme en pratique, plus ardue à résoudre que les problèmes convexes non linéaires (la linéarité entraine la convexité), car elle pose deux nouvelles difficultés.

La première réside dans la réduction de la portée des techniques d'optimisation locale, qui ont également fait l'objet d'une littérature extensive. Nous nous référons notamment à l'ouvrage de Minoux [84]. Ces techniques fournissent toujours des extrema locaux, mais, contrairement au cas convexe, elles ne suffisent plus à assurer la résolution globale du problème à l'optimum.

La seconde est que, pour les problèmes en variables entières, la relaxation continue du

problème, reste non convexe, ce qui la rend, à priori, aussi difficile à résoudre que le problème discret. Cela a pour conséquence directe que notre meilleur outil, à ce jour, d'énumération implicite qu'est l'algorithme de *Branch & Bound*, ne peut plus s'appuyer sur cette relaxation comme fonction majorante, dans un contexte de maximisation, pour obtenir une borne supérieure fine.
Comme l'a souligné Rockafellar [94], dans ses travaux sur les multiplicateurs de Lagrange, «*le grand fossé en optimisation n'est pas entre la linéarité et la non linéarité, mais entre la convexité et la non convexité.*» [1].

Dans ce contexte, comme on ne connait pas d'algorithme générique, les méthodes de résolution dépendent fortement de l'origine de la non convexité et donc du type de problème étudié. Nous distinguons alors, pour la programmation en variables mixtes, trois catégories principales de recherche : le cas quadratique, que nous abordons à la section 2.1, le cas polynomial, dans la section 2.2 et le cas général, auquel appartient notre problème de thèse, abordé à la section 2.3. Nous présentons enfin l'état de l'art de l'application financière ELBA à la section 2.4.

2.1 Programmation quadratique non convexe

Lorsque nous avons démarré notre étude, nous nous sommes familiarisés avec la notion de convexité, et aux conséquences de son absence, à partir de la Programmation Quadratique en Variables Entières (PQVE). En effet, en dimension un, la parabole fournit une représentation graphique simple et directe de la notion de convexité.

$$(PQVE)\begin{cases} \max f(x) & = {}^t c\,x - \frac{1}{2}\,{}^t x\,Q\,x \\ s.c & A\,x \leq b \\ & \forall i,\ 0 \leq x_i \leq m \\ & \forall i,\ x_i \in \mathbb{N} \end{cases} \tag{2.1}$$

La première idée naturelle est de transformer ce problème en un problème équivalent plus simple. Pour le cas quadratique, sous contraintes linéaires, en variable binaires, les travaux de Billionnet *et al.* [27], en 2009, présentent une réformulation quadratique convexe (la méthode QCR), à l'aide de matrices semi-définies positives.
Ce problème convexe équivalent peut alors être résolu par un *Branch & Bound* classique dont la fonction d'évaluation correspond à la relaxation continue (convexe) du problème

1. *...,the great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity.*

équivalent. L'approche a été étendue aux variables entières dans Billionnet *et al.* [25] en 2012, puis aux variables entières sous contraintes quadratiques dans Billionnet *et al.* [26] et Elloumi et Lambert [44].

Les travaux d'habilitation d'Amélie Lambert [69] montrent comment construire une suite de problèmes, paramétrés par des matrices semi-définies positives, équivalents au problème initial et dont la relaxation (convexe) est de plus en plus fine. Elle montre la convergence de ces problème relâchés convexes vers l'optimum global, à ϵ près du problème quadratique (non convexe) initial.
Un autre intérêt pratique de ce type d'approche est que la résolution de ces sous-problèmes plus simples, peut être sous-traitée à un algorithme ou à un solver dédiés, afin d'améliorer la performance.

Une autre reformulation du problème quadratique non convexe sous contraintes linéaires est proposée par Quadri et Soutil [92], ainsi que Soutil *et al.* [107]. Ils transforment tout d'abord le problème en un problème séparable équivalent en variables mixtes.

$$(PQVE)\begin{cases} \max g(y) & = {}^t c_1\, y - \frac{1}{2}\, {}^t y\, D\, y \\ s.c & A\, x \leq b \\ & R\, x = y \\ & \forall i,\ 0 \leq x_i \leq m \\ & \forall i,\ x_i \in \mathbb{N} \\ & \forall i,\ y_i \in \mathbb{R}^n \end{cases} \tag{2.2}$$

Une méthode intuitive pour cela est de diagonaliser la matrice de la forme quadratique, qui est symétrique donc diagonalisable en base orthonormée. On peut donc, sous réserve d'être capable de trouver rapidement les matrices de passage, transformer le problème initial en un problème équivalent séparable. Cette transformation n'assure cependant pas la convexité. La discussion va alors porter sur le signe des valeurs propres.

Si elles sont toutes de même signe, mettons positives (resp. négatives), la forme quadratique sera semi-définie positive (resp. négative) et le problème convexe (resp. concave). C'est le cas particulier simple et rare en pratique pour lequel le problème initial est convexe dans un contexte de minimisation (resp. maximisation). Dans ce cas, nous pouvons lui appliquer les méthodes de l'optimisation convexe.

En pratique néanmoins, Quadri et Soutil [92] montrent que les valeurs propres, par leur irrationalité potentielle, et la complexité de l'algorithme de matrices de passage, posent des problèmes de précisions numériques. Quadri [91], dans ses travaux d'habilitation, leur préfère une similitude à une matrice diagonale obtenue une décomposition de Gauss, ce qui assure la rationalité des termes de la matrice diagonale et une complexité polynomiale pour obtenir les pivots de Gauss.

Le cas le plus fréquent et le plus intéressant est celui où termes diagonaux d_i (qui ne sont pas des valeurs propres) du problème équivalent séparable ne sont pas de même signe. Alors la matrice n'est pas semi-définie positive et le problème (initial ou séparable) n'est donc pas convexe.

La deuxième étape est alors de linéariser le problème séparé pour obtenir, dans un contexte de maximisation, un problème majorant linéaire fournissant une borne supérieure fine. Quadri et Soutil [92] utilisent des fonctions linéaires par morceaux obtenues à l'aide des cordes et/ou de tangentes afin de linéariser le problème séparé.

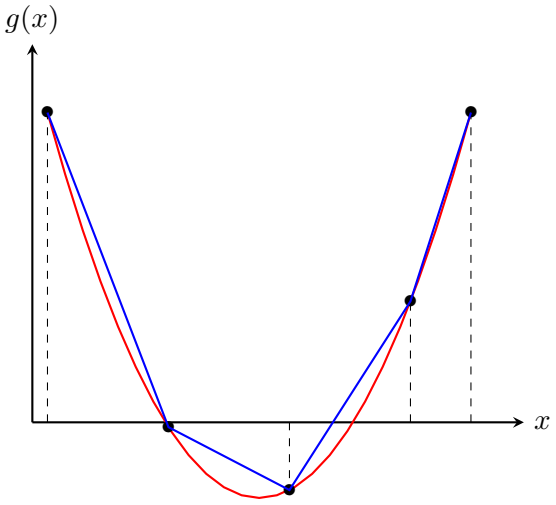


FIGURE 2.1 – Méthodes des cordes, lorsque le terme diagonal $d_i < 0$ (cf. [91])

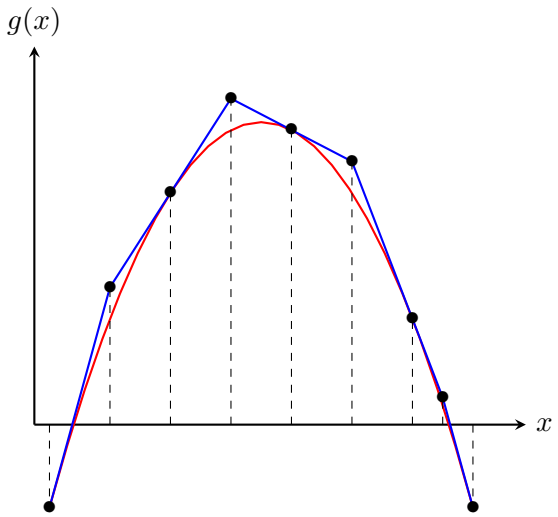


FIGURE 2.2 – Méthodes des tangentes, lorsque le terme diagonal $d_i > 0$ (cf. [91])

Quadri et Soutil [92] montrent pour un problème quadratique convexe en variables entières, la solution optimale du problème équivalent tend vers celle du problème initial, lorsque le pas de la subdivision est suffisamment faible. Des travaux sont en cours pour étendre ce résultat au cas non convexe en variables entières et mixtes. D'autres méthodes de linéarisation par morceaux sont proposées par Geißler *et al.* [55].

L'approche en deux étapes de Quadri et Soutil [92] repose donc sur une reformulation équivalente en un problème séparable (plus simple) puis une relaxation convexe du problème équivalent, obtenue ici par linéarisation. Ce type d'approche est très commun dans la littérature et on verra qu'il est également utilisé dans le cas général.

Une approche similaire est de reformuler le problème en un problème linéaire. Adams et Sherali [1] en 1986, pour des programmes quadratiques en variables binaires, linéarisent en rajoutant des variables auxiliaires dites de Fortet [48] pour les termes produits (cela revient aux inégalités de McCormick pour des variables binaires). Puis, ils construisent des inégalités linéaires valides de plus en plus fines pour le problème initial. Ils nomment cette technique *Reformulation and Linearization Techniques* (RLT en abrégé). Le lecteur peut se reporter à l'ouvrage de Sherali et Adams [103] sur les extensions de cette technique.

Enfin, RLT peut également être utilisée pour générer des coupes. Par exemple, Audet

et al. [10] propose un algorithme de *Branch & Cut* pour résoudre le cas non convexe avec des contraintes quadratiques qui s'appuie sur ces linéarisations.
Après avoir présenté les principaux travaux sur la programmation quadratique, l'étape suivante dans la compréhension des problèmes non convexe consiste à étudier les monômes de degrés plus élevés. C'est pourquoi nous nous intéressons maintenant à la programmation polynomiale.

2.2 Programmation polynomiale

Un programme mathématique est polynomial si les fonctions du problèmes (objectifs et contraintes) sont des polynômes de degré fini dans les variables de décision. Bien que nous n'ayons pas utilisé directement ces résultats dans notre étude, nous observons que certaines idées décrites dans le cas quadratiques s'appliquent au cas polynomial. Par exemple, comme dans le cas cas quadratique, nous cherchons à reformuler et à linéariser le problème. Dans le cas en variables binaires, les travaux de Glover et Woolsey [58] et la RLT de Sherali et Adams [103] fournissent des relaxations linéaires fines.

Ce cadre est étendu à la programmation polynomiale par Sherali et Adams [102]. Plus récemment, les travaux de Lazare [71] proposent plusieurs techniques de reformulation et/ou de linéarisation de programmes polynomiaux en variables binaires. Il étend notamment la méthode de relaxation convexe quadratique (QCR) au cas polynomial (méthode PQCR).

Cafieri *et al.* [35] cherchent à réduire le nombre de contraintes RLT. Ils montrent comment générer un nombre réduit de contraintes, permettant ainsi d'améliorer les performances de l'algorithme de *Branch & Bound*. Ils nomment la procédure *Reduced RLT* (ou rRLT).

Nous nous intéressons également aux relaxations convexes des monômes. La meilleure relaxation possible correspond, par définition, à l'enveloppe convexe de la fonction considérée, qui est cependant difficile à expliciter dans le cas général.

Cependant, pour les monômes de degré 2,3 et partiellement 4, l'enveloppe convexe des monômes est connue. Pour les expressions bilinéaires $x_i\, x_j$, les inégalités de McCormick [79] fournissent une relaxation convexe (resp. concave), très fine. Les travaux de Al-Khayyal

et Falk [4] ont montré qu'elles correspondaient avec l'enveloppe convexe (resp. concave) :

$$\begin{cases} x_i\,x_j & \leq & x_i^L\,x_j + x_j^U\,x_i - x_i^L\,x_j^U \\ x_i\,x_j & \leq & x_i^U\,x_j + x_j^L\,x_i - x_i^U\,x_j^L \\ x_i\,x_j & \geq & x_i^L\,x_j + x_j^L\,x_i - x_i^L\,x_j^L \\ x_i\,x_j & \geq & x_i^U\,x_j + x_j^U\,x_i - x_i^U\,x_j^U \end{cases}$$

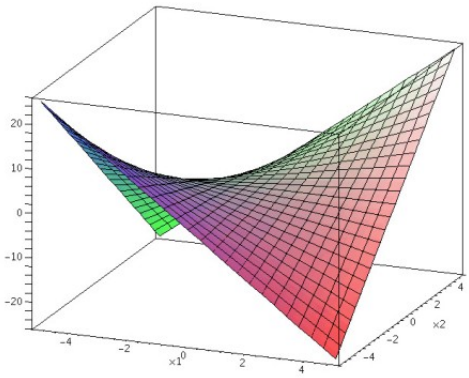


FIGURE 2.3 – Produit bilinéaire $x_1\,x_2$ (extrait de [39])

Les enveloppes convexes et concaves du produit correspondent aux sommets du polyèdre bleu. Ce résultat se généralise en dimension finie ([93, 109]), ce qui montre que les enveloppes peuvent s'exprimer à partir de contraintes linéaires.

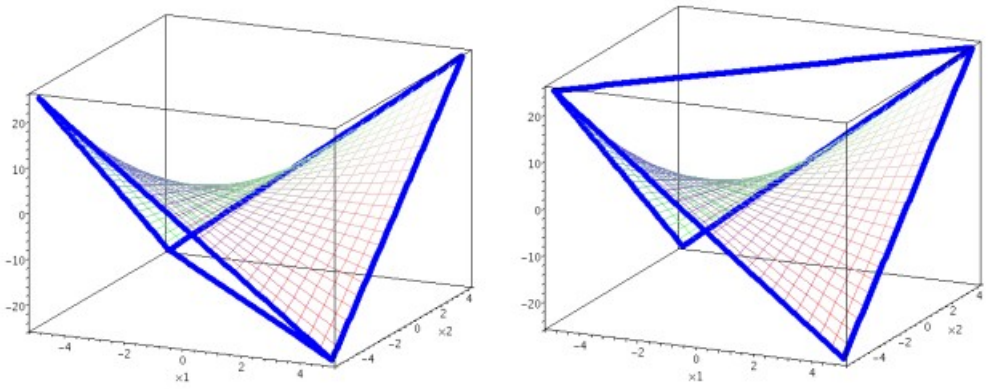


FIGURE 2.4 – Enveloppe concave, convexe du produit $x_1\,x_2$ (extrait de [39])

Pour les termes trilinéaires, les enveloppes sont établies dans les travaux de Meyer et Floudas [83],[82]. Enfin, pour les termes quadri-linéaires, Cafieri *et al.* [36] montrent comment regrouper en produits des monômes de plus faible degré, afin d'obtenir la relaxation la plus fine. Bien que l'enveloppe convexe ne soit pas explicitée, ils prouvent que la qualité de la relaxation obtenue est décroissante dans le nombre de compositions de relaxations de monômes de dégrés inférieurs.
Pour le degré 4, les relaxations du terme quadri-linéaire $(x_1\,x_2\,x_3\,x_4)$, obtenues par composition des relaxations $R_2\big(R_3(x_1\,x_2\,x_3), x_4\big)$ et $R_3\big(R_2(x_1\,x_2), x_3, x_4\big)$ sont meilleures que $R_2\Big(R_2\big(R_2(x_1\,x_2), x_3\big), x_4\Big)$, $R_2\big(R_2(x_1\,x_2), R_2(x_3, x_4)\big)$ et $R_2\Big(x_1\,R_2\big(x_2, R_2(x_3, x_4)\big)\Big)$,
avec R_i l'opérateur de relaxation bi- ou tri-linéaire qui est connu explicitement.

Même pour des monômes de faibles de degré, Costa et Liberti [39] remarquent que le nombre de termes croit exponentiellement en fonction du degré dans l'expression de la relaxation, ce qui peut être la source d'erreurs humaines ou numériques. Les travaux de Rikun [93] et Tardella [109] ont montré que pour les contraintes de boites $(\forall j,\ x_j^L \leq x_j \leq x_j^U)$, l'enveloppe convexe des termes produits $x_1\,x_2\,\cdots\,x_k$ correspond aux sommets d'un polyèdre. L'approche de Costa et Liberti [39] est de considérer la relaxations convexe du problème dual (Lagrangien), dont l'enveloppe convexe est simple à expliciter (i.e, le nombre de termes croit moins rapidement avec le degré, au moins jusqu'au degré 5).

Nous retenons de cette partie que les techniques de reformulation et linéarisation (RLT), ou reformulation et convexification, issues de la programmation quadratique, s'étendent au cas polynomial.

Nous allons, dans la prochaine section, nous intéresser au cas général, ou plutôt à certains autres cas et présenter quelques approches ciblées de résolution.

2.3 Cas général

Comme indiqué en introduction de la section 2, nous ne connaissons pas de traitement général des programmes mathématiques non convexes. Par conséquent, les méthodes de résolution sont nécessairement spécifiques à certains types de problème. Cependant les stratégies de reformulation et convexification présentées dans les sections 2.1 et 2.2 s'étendent, sous certaines conditions au cas général.

Dans ses travaux fondateurs, McCormick [79, 80] s'intéresse à une classe particulière de problèmes, composées de fonctions (objectifs et contraintes) qu'il nomme *factorisables*. Une fonction factorisable signifie implicitement qu'elle peut être décomposée en une somme de fonctions univariées, ou en produits de fonctions univariées d'expressions elles mêmes factorisables. Nous rappelons en détail ces notions et précisons leur application à notre problème à la section 4.1.

La factorisation, au sens de McCormick, est une transformation, qui n'est pas unique, en un problème équivalent au problème initial. L'objectif de cette factorisation est d'isoler les sources de non convexité du problème factorisé équivalent, afin de proposer leur relaxation convexe fine dans la seconde étape. En effet, lorsque le problème est factorisé, il ne comporte plus que des termes bilinéaires, pour lesquels McCormick [79] montre comment caractériser l'enveloppe convexe de fonctions univariées suffisamment régulières ($\mathcal{C}^2$ est déjà une hypothèse suffisante) en introduisant ses célèbres inégalités pour les termes bilinéaires, rappelées à la section précédente 2.1 et présentées dans les rappels de convexification de la section 4.3.2.

Enfin, il établit un algorithme de *Branch & Bound* spatial (sur des variables continues), dont il montre la convergence à ϵ près vers l'optimum global du problème initial. En pratique, la classe de programmes factorisables est très étendue et permet de modéliser la plupart des fonctions d'ingénieur et toutes celles qui s'expriment directement comme une combinaisons simple de fonctions régulières. A contrario, certaines fonctions qui s'expriment comme une série, une intégrale, ou les fonctions dites en «boîte noire » ne sont pas factorisables. L'exemple usuel donné par McCormick [79] d'une fonction non factorisable, est celui de la fonction $\Gamma\left(x \mapsto \int_0^{+\infty} e^{-u}\, u^{x-1} du\right)$.
Les travaux de Smith et Pantelides [104, 105, 106] étendent cette approche en deux étapes, aux problèmes factorisables en variables mixtes.

Plus spécifiquement pour la partie convexification, Androulakis *et al.* [9] proposent une approche alternative intéressante. Ils rajoutent une pénalité quadratique à la fonction objectif et étudient les conditions qui font de la somme un sous-estimateur convexe (fonction convexe minorante sur l'ensemble admissible).
En effet, si f est la fonction objectif, on définit : $f_\alpha(x) = f(x) + \prod_{i=1}^{n} \alpha(x_i^L - x_i)\,(x_i^U - x_i)$

Pour $\alpha > 0$, f_α minore f. De plus, une minoration directe de la matrice Hessienne de f_α permet d'obtenir que si $\alpha > -\frac{1}{2}\min_i \lambda_i$, alors f_α est convexe. Androulakis *et al.* [9] utilisent ces relaxations convexes α-paramétriques dans un algorithme *Branch & Bound* spatial qui permet une résolution globale, pour des fonctions $\mathcal{C}^2$. Ils nomment cette technique α-*Branch & Bound* (ou α-BB en abrégé). Adjiman et Floudas [3] [2] effectuent des expérimentations numériques et apportent des améliorations lorsque les termes non convexes ont une structure particulière.

Comme pour le cas quadratique, lorsque le problème est transformé en un problème équivalent séparable ou factorisé, la relaxation convexe peut simplement être linéaire. Beale et Tomlin [13] ont introduit dans les années 70, pour des programmes séparables, un maillage de l'espace en simplexes et remplacent les parties non convexes par des approximations linéaires par morceaux en introduisant des variables binaires sur ces ensembles ordonnés, appelés *Special Ordered Sets* (SOS).

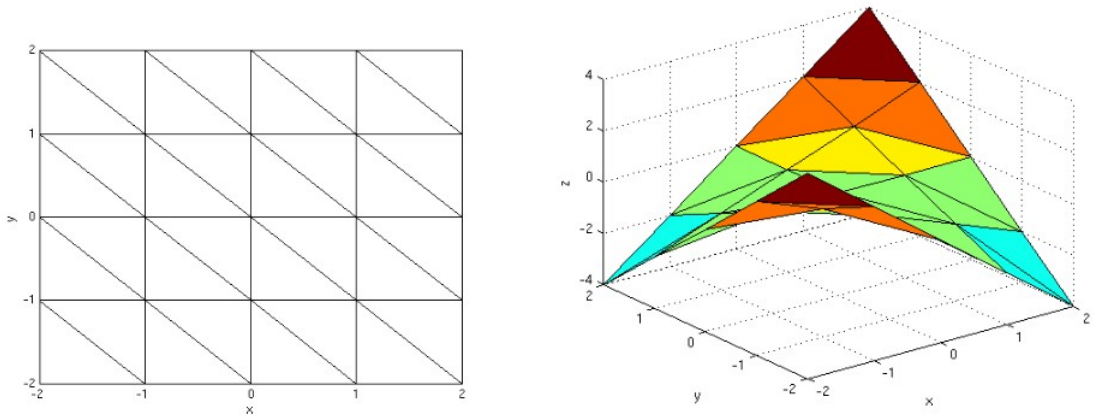


FIGURE 2.5 – Maillage en triangles du produit $x_1\, x_2$ et sous-estimation linéaire (cf. [72])

Cette approche a été étendue par Martin *et al.* [77], puis par Leyffer *et al.* [72] au cas factorisable et améliorée pour garantir la faisabilité des approximations successives et diminuer la cardinalité du maillage (qui est au départ exponentiel dans le nombre de variables de décision). Pour cela Leyffer *et al.* [72] cherchent également à réduire le domaine à chaque itération du *Branch & Bound*, en propageant les limites obtenues sur les variables dans les nœuds de l'arbre d'énumération. Ils nomment cette technique *Branch & Refine*. Plus récemment, les travaux de Huchette et Vielma [62] présentent de nouvelles formulations du SOS, via un maillage plus efficace, pour les problèmes en variables entières.

La réduction du domaine, au delà du simple branchement, à chaque itération de l'algorithme de *Branch & Bound*, a fait l'objet de plusieurs travaux de recherche. Ryoo et Sahinidis [97] introduisent une technique de *Branch & Reduce*. Ils cherchent à réduire le domaine des variables de décision, sans couper de solutions optimales. Cette réduction de domaine conduit à de meilleures relaxations convexes et ainsi de suite. L'objectif de cette boucle vertueuse est évidemment de réduire au maximum le nombre de nœuds visités dans l'arbre, pour déterminer l'optimum et conclure sur la résolution globale. Belotti *et al.* [18] s'intéressent aux différents ingrédients du *Branch & Bound* hors relaxation : réduction de domaine, choix de variables de branchement, point d'attachement.

Ces exemples suivent le schéma classique, de transformation équivalente en un problème, ou une suite de problèmes plus simples (séparés, factorisables, décomposables etc.), qui peuvent être convexifiés de manière fine et dont on peut extraire une suite de solutions qui convergent vers l'optimum global du problème initial.
Les améliorations portent alors sur les composants du *Branch & Bound* : la relaxation convexe, les techniques de branchement et de réductions de domaines. L'objectif poursuivi est, pour un problème donné d'en réduire la complexité, exponentielle à priori.

Une autre idée, proche, reste toujours de diviser pour régner, mais cette fois de manière récursive sur la structure du problème. Nous nous plaçons sur un domaine borné et considérons un problème comportant N variables de décisions. Supposons que la structure du problème permette, dans l'objectif et les contraintes, d'isoler un sous-ensemble de variables de décision, et, c'est fondamental, que cette structure nous permette également, si les solutions de tous les sous-problèmes sont connues, de résoudre le problème initial à l'optimum. Nous pourrions alors construire un algorithme de décomposition en sous problèmes et de résolution du problème initial. Évidemment ces hypothèses très générales, servant à illustrer la démarche doivent être précisées.

Dans ses travaux fondateurs de Programmation Dynamique, Bellman [14] établit les propriétés de *décomposabilité* que doivent satisfaire les fonctions (objectifs et contraintes) du problème. Il montre alors comment isoler la N^{ieme} de l'objectif et optimiser indépendamment le problème à $N-1$ lorsque l'état de x_N est fixée.
Il établit une propriété de récurrence (dite équation d'état de Bellman), entre les optima

des problèmes à N et $N-1$. Elle permet de décomposer le problème à N variables de décisions, en k_N problèmes à $N-1$ variables, où k_N correspond aux nombres d'*états* admissible que peut prendre x_N. Comme le domaine est fini par hypothèse, k_N est borné par une constante M indépendante de N. Alors récursivement, au bout de N étapes, nous aboutissons à des problèmes d'une variable simples à résoudre et l'équation d'état permet de remonter, de proche en proche, à l'optimum du problème initial. Nous rappelons ces notions et établissons l'équation de Bellman pour notre problème au chapitre 3.

Cette utilisation de la structure du problème se retrouve également dans les décompositions par blocs de problèmes linéaires. En effet, nous considérons les décompositions suivantes :

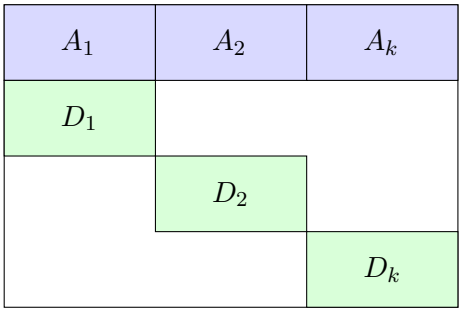


FIGURE 2.6 – Décomposition bloc-diagonale avec des contraintes couplantes (cf. [84] p478)

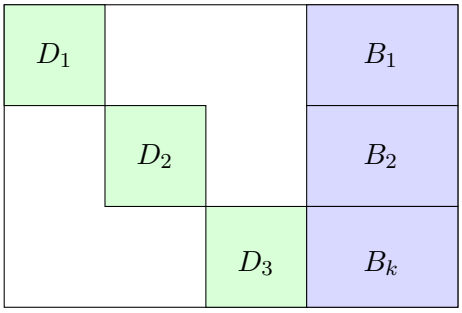


FIGURE 2.7 – Décomposition bloc-diagonale avec des variables couplantes (cf. [84] p478)

Dans les deux types de problèmes, les D_i sont des blocs diagonaux (représentés en vert) de variables séparées. Le premier cas, illustré à la figure 2.6, représente une décomposition par blocs avec des contraintes couplantes, les blocs A_i (en bleu). Une décomposition classique dans ce cas correspond à l'algorithme de Dantzig et Wolfe [41].

Le traitement du second cas, illustré à la figure 2.7, avec des variables couplantes (les blocs bleus B_i), fait l'objet de l'algorithme de Benders [19] pour des problèmes linéaires (généralisé par Geoffrion [57] aux problèmes non linéaires).

L'approche par décomposition récursive impose une structure rigoureuse sur les fonctions du problème. Elle est donc plus restrictive en termes de problèmes traités. L'intérêt sous-jacent des approches «structurelles» réside dans leur complexité. En effet, comme nous le détaillons à la section 3.2, la DP est pseudo-polynomiale en temps, au pire cubique, tandis que l'algorithme de *Branch & Bound* est exponentiel (au pire).

Une application de notre problème de thèse est l'écoulement de larges blocs d'actifs (ELBA) dans un cadre déterministe, dont nous abordons à la prochaine section, l'état de l'art financier.

2.4 Etat de l'art financier ELBA

Ma carrière dans l'industrie financière m'a, de longue date, sensibilisé aux problèmes d'optimisation dans ce secteur. De manière informelle, la question de l'allocation d'une ressource (fonds, quantité d'actif), soumise à des contraintes (de temps, coût, risque etc.) afin d'optimiser une fonction objectif (profit, rendement, utilité, risques) est au cœur des problématiques financières, scientifiques et réglementaires.

Un autre aspect qui est structurellement ancré dans la finance, et même dans la plupart des sciences humaines, est l'incertain. Le prix d'une action est modélisé par une distribution probabiliste, reflétant l'univers des possibles, mais nul ne sait ce que sera le prix exact au temps t, vu d'aujourd'hui.
Par conséquent pour modéliser aujourd'hui, le prix des contrats à terme (instrument financier dont la valeur est fonction du prix futur d'un actif ou plusieurs actifs), l'opérateur de marché a deux grandes avenues devant lui : la première est de répliquer le contrat, *via* une stratégie équivalente, ou bien simplement en le décomposant en d'autres actifs sous-jacent dont le prix est connu (ou répliqué) avec certitude. Il suppose alors que le prix est exactement (ou très proche en pratique) de celui de la stratégie de réplication équivalente, sinon un autre opérateur de marché, ne manquerait pas d'exploiter la différence à son profit, sans prendre aucun risque, et cet argument est fondamental, réduisant ainsi l'écart de prix. Cet argument est connu sous le nom *d'absence d'arbitrage*. Il est le pilier qui soutient

de nombreux prix et comportements relatifs à des produits dérivés (futures, forward, taux de change, réplication d'indice, *basis-trades* etc.) sur différentes classes d'actif.

La deuxième direction est de modéliser les actifs fondamentaux (actions, taux, devises) qui composent le contrat, par des variables aléatoires stochastiques, ce qui impose des hypothèses sur leur distribution de probabilité et, par calcul différentiel stochastique ou par simulation de Monte-Carlo, d'obtenir, le barycentre des prix possibles pondérés par leur probabilité qui n'est autre que l'espérance mathématique du prix du contrat à terme. En pratique, la force d'un modèle réside bien évidemment dans sa rigueur mathématique et dans la validité de ses hypothèses, mais aussi et surtout dans le consensus qu'il fait, c'est à dire le nombre d'opérateurs de marché qui l'utilisent.

Un exemple prégnant de ces cinquante dernières années est celui de la Formule Black et Scholes [28] et Merton [81]. Elle est la formule de calcul du prix d'une option européenne la plus utilisée à *Wall Street*. Pour rappel, une option est le droit (et non l'obligation) d'acheter (ou de vendre) un actif à un prix et à une date donnés, fixés aujourd'hui. Le prix de cette option d'achat (ou de vente) est l'objet de la formule.

Bien qu'elle fasse des hypothèses fausses en pratique sur la constance des variables qui la composent, notamment la volatilité, qui ont fait l'objet d'une littérature abondante (cf. travaux de Dupire *et al.* [43] sur la volatilité locale), elle a régné quasiment sans partage sur les marchés depuis son introduction. Et nous avons vécu dans le monde (ou au moins le paradigme de marché) de Black & Scholes & Merton. Ce qui suppose que les prix d'actif ont une distribution *lognormale* (leur logarithme est normal) et donc qu'*ils ne peuvent en aucun cas, devenir négatifs.*
Lorsque les taux d'intérêts sont entrés durablement dans le territoire négatif, entre 2014 et 2021, après certains ajustements temporaires, les opérateurs de marché ont été forcés de l'abandonner (temporairement ?) et se sont tournés en masse, vers les travaux fondateurs et vers la formule de Bachelier [11] datant de 1900. Une légitimation posthume pour le mathématicien dont les travaux fondateurs ont été redécouverts après la seconde guerre mondiale, et qui a connu une carrière académique difficile, racontée par Mandelbrot et Hudson [75].

En conclusion, l'incertain et sa modélisation sont au cœur des préoccupations des ins-

titutions financières.

Par conséquent, nous nous sommes intéressés, avec ma directrice de thèse, spécialisée en optimisation combinatoire et déterministe, aux applications financières possibles de mon sujet. Nous avons cherché parmi les problèmes que j'avais rencontrés et qui pouvaient être modélisés *via* la programmation mathématique, dans un cadre déterministe. Ce dernier point est beaucoup plus discriminant. Historiquement, l'optimisation quadratique est liée à l'allocation d'actifs suite aux travaux fondateurs de Markowitz [76]. Ils indiquent comment choisir la pondération des actifs et comment composer son portefeuille, afin de maximiser le rendement, pour un niveau donné de risques, mesuré par une fonction quadratique (la variance).

Nous nous sommes cependant intéressés, à un autre problème auquel j'ai été confronté personnellement, qui est celui de l'Ecoulements de Larges Blocs d'Actifs (ELBA), également appelé, Liquidation Optimale de Portefeuille (LOP). Il consiste à vendre ou à acheter, une très grande quantité d'actif (le bloc) dans un temps donné, discret (il y a nombre fini d'occasions de vente) ou continu.

Dans la littérature financière, ELBA a déjà été largement étudié, sous différents angles. Le premier sujet d'étude est l'impact des larges blocs sur le marché. Quel est la performance de l'actif durant et après la vente de blocs, à différents horizons de temps ? Quel est le temps de retour à l'équilibre d'un marché suite à l'écoulement d'un large bloc ? Guthmann et Bakay [59], Dann *et al.* [40], Gemmill [56] fournissent des études empiriques à ces questions. Par analogie avec un lac calme, ces approches étudient les remous provoqués par le lâché d'un ou plusieurs pavés ainsi que leurs évolutions temporelles. Dans ce contexte, notre objectif est donc poser le plus délicatement possible ce pavé dans l'eau, pour en minimiser les remous.

Une deuxième voie s'attache à l'étude de marchés spécifiques dédiés à ce type de transactions, connus sous le nom d'*upstairs market*[2], ainsi que leur récents alter ego électroniques, les *dark pools*[2]. Le lecteur peut se référer à Madhavan et Cheng [74] et plus récemment Mishra [86] ont abordé le problème ELBA sur ces marchés.

2. cf. Glossaire

Une troisième approche s'appuie sur la modélisation du comportement des différents participants (opérateur de marché, broker, spécialiste etc.) pour déterminer l'état d'équilibre et en étudier les conditions d'existence et d'unicité. Le lecteur peut se référer à Seppi [100] et Keim et Madhavan [66].

Une quatrième voie, s'attachant à la modélisation du carnet d'ordre (*Limit Order Book*), qui trie les meilleurs offres d'achat ou de vente avec leur volume respectif, a été introduit par Obizhaeva et Wang [89]. Des approfondissements dans cette direction sont proposés dans Alfonsi *et al.* [5, 6] et Obizhaeva et Wang [90].

Enfin, de nombreux articles abordent ELBA en temps continu, qui devient un problème de contrôle stochastique : Barles [12], Vath *et al.* [113], Seydel [101], Kharroubi et Pham [67], Gatheral [52], Gatheral et Schied [53], pour ne citer que quelques références.

Nous nous sommes placés dans cette étude, en temps discret et en variables entières. Dans ce cadre, la question de la décomposition optimale du blocs en plus petits ordres afin de maximiser le produit de la vente, ou de manière équivalente, de minimiser son impact sur les prix, a été traité dans des articles fondateurs de Bertsimas et Lo [23] et Almgren et Chriss [7], Almgren [8]. Ils introduisent un prix d'actif stochastique dont la dynamique dépend de la taille de l'ordre précédent sur le marché. La majorité des modèles qui font échos à ces deux articles, font la distinction entre l'impact temporaire sur le prix de l'actif, du aux déséquilibres temporaires entre l'offre et la demande, et l'impact permanent qui reflète l'effet sur le prix des informations transmises aux participants de marché par nos actions d'écoulement. Leur résolution est parfois fondée sur la programmation dynamique. Cependant, dans la plupart des travaux, les fonctions de pénalisation (impact sur les prix) sont linéaires ou suivent une loi de puissance.

La question de la «mémoire du marché», *via* la modélisation de l'information qui lui est transmise occupe également une place centrale. On rencontre principalement dans la littérature, soit des fonctions de pénalisations à mémoire courte, dans le sens où la pénalité au temps t_k est en $g(x_k)$ et ne dépend que des x_k unités d'actifs écoulées en t_k, ou bien avec une mémoire longue séparée (ex : Almgren et Chriss [7]) pour lesquelles la pénalité en t_k est de la forme $\sum_{j=1}^{k} g(x_j)$.

Comme indiqué précédemment, nous nous sommes placés, pour cette application, dans un cadre déterministe. Nous ne cherchons donc pas à prédire l'évolution du prix de l'actif, que l'on suppose connue (ou correctement estimée) avant toute optimisation. Ce cadre a été utilisé récemment par Boyd *et al.* [32] pour le problème d'allocation optimale de portefeuille.

Du point de vue du praticien financier, cette hypothèse n'est réaliste que pour des marchés très calmes (faible volatilité), pour lesquels aucune nouvelle économique ou autre, n'est susceptible d'influencer les cours de l'actif considéré. Bien qu'elle puisse paraître assez restrictive en première approche, elle fournit, dans notre expérience, un environnement très favorable à l'écoulement de larges blocs. En fait, la vente de blocs se pratique principalement, dans l'environnement de marché le plus calme possible, afin justement, de ne pas éveiller l'attention des autres opérateurs de marché.
A contrario, dans un marché volatil ou pour l'écoulement d'un bloc d'un actif peu liquide (qui n'est pas très fréquent), le résultat du produit de la vente du bloc dépend principalement de la dynamique des prix, quelque soit la stratégie d'exécution.

Une autre situation, bien plus rare mais plus confortable, est celle où nous sommes à contre-sens du marché (vente d'un bloc d'actions qui est très recherché). Dans cette situation, la pénalité à écouler un bloc est faible donc la stratégie de vente importe peu et le prix final est dominé par l'évolution des prix de l'actif. Ce cas, très rare en pratique, ne présente pas d'intérêt pour la programmation mathématique.

Enfin, cette hypothèse permet de découpler la réponse des participants de marché à la liquidation, qui constitue notre objet principal d'étude, de l'évolution des prix de l'actif. Par conséquent les méthodes de résolutions, les algorithmes et les techniques d'optimisation présentées dans cette thèse sont valables pour tout vecteur de prix.

Par conséquent, l'application financière de notre sujet est simplement de montrer comment optimiser la liquidation d'un portefeuille mono sous-jacent, sachant que les évolutions de prix sont correctement estimées, sans se soucier de comment ces estimations sont faites. Nous avons également inclus le cas des prix de l'actif constant durant la liquidation, à titre de benchmark dans les expérimentations numériques, afin de mesurer la dissémination de l'information dans le marché.

Hors de la finance, le vecteur de prix p peut être interprété comme le comportement standard de l'environnement, sans aucune influence de notre fait. Dans ce contexte, on suppose que le comportement standard est connu et on cherche à étudier, comment nos actions qui fournissent des informations à l'environnement à mémoire parfaite, influencent sa réponse. Notre objectif est alors de minimiser l'impact négatif de nos actions sur la métrique de sortie, ou de manière équivalente de maximiser cette métrique.

Chapitre 3

Programmation dynamique

Comme nous l'avons indiqué précédemment, la principale source de difficulté (et d'interêt) du problème traité ($PNS_{N,M}$) réside dans sa non convexité. Par conséquent, sa relaxation continue, est à priori non convexe et ne présente donc pas, *ceteris paribus*, d'intérêt évident (au contraire de la programmation convexe, par exemple) car elle est tout aussi difficile à résoudre. Nous étudierons dans le chapitre 4, la relaxation convexe de ($PNS_{N,M}$). Cependant, dans ce chapitre, nous nous intéressons à la structure du problème, afin de voir s'il peut être décomposé en sous-problèmes plus simples. C'est pour cela que nous introduisons et mettons en oeuvre la Programmation Dynamique (DP), qui s'avère être particulièrement adaptée à notre problème et permet de le résoudre exactement et efficacement sur les petites et moyennes instances.

Nous commençons, dans la section 3.1 par de brefs rappels sur la DP. Puis, dans la section 3.2, nous démontrons que l'équation de Bellman s'applique au problème ($PNS_{N,M}$) et en déduisons une méthode de résolution ainsi que sa complexité. Dans la section 3.3, nous présentons une méthode approchée de résolution en deux étapes fondée sur la DP. Nous introduisons également un algorithme de recherche local dans la section 3.3.3. Dans la section 3.4, nous recherchons des bornes inférieures pour le problème continu ($\overline{PNS_{N,M}}$). Enfin, dans la section 3.5, nous obtenons une majoration du problème ($\overline{PNS_{N,M}}$) par un problème séparé que nous résolvons pour en déduire une première borne supérieure du problème initial.

3.1 Rappels de Programmation Dynamique

La Programmation Dynamique (DP) a été présentée pour la première fois dans les travaux de Bellman [14] en 1957. Elle s'applique à certains programmes mathématiques

particuliers, qui possèdent une certaine propriété de décomposabilité.
Un tel programme satisfait le théorème d'optimalité de Bellman et peut être résolu, de manière exacte, par des algorithmes récursifs, qui exploitent cette structure particulière. Ces algorithmes permettent de construire une solution optimale, à partir des solutions optimales de sous-problèmes plus simples.
La relation de récurrence au cœur de ces algorithmes est appelée équation fonctionnelle de DP, ou équation de Bellman. Enfin, une solution optimale ainsi construite à partir de solution optimales de sous-problèmes satisfait le principe d'optimalité de Bellman.

A l'origine, la DP a été appliquée à l'optimisation de systèmes dynamiques et aux processus de Markov. Nous reprenons la définition d'un système dynamique établie par Minoux [84] p.559 : système dont l'évolution au cours du temps est régie par une équation d'état et dont le comportement peut être modifié par l'intermédiaire de variables de décision (appelées variables de contrôle ou de commande).
C'est dans ce contexte que le principe d'optimalité, que doit satisfaire une solution issue de la DP, a été énoncé par Bellman en 1957 :

> «Une politique optimale a la propriété que, quel que soient l'état initial et la décision initiale, les décisions restantes doivent constituer une politique optimale, vis à vis de l'état résultant de la première décision».

Une formulation résumée et parlante du principe est celle de Minoux [84] p.543 :

> «Toute solution optimale ne peut être formée que par des solutions partielles optimales».

Nous présentons ci-après de façon générale les concepts clés de DP et nous démontrerons, à la section 3.2 l'équation de Bellman, dans le cas particulier du problème $(PNS_{N,M})$.
La première notion importante est celle de *décomposabilité*, car elle caractérise la forme des problèmes auxquels la DP s'applique. Nous reprenons ici la définition et l'approche de Minoux [84] p.535 :

Définition 1. *Une fonction f est décomposable en f_1 et f_2, si elle peut se mettre sous la forme $f(x, y) = f_1(x, f_2(y))$ et si, de plus, à x fixé, la fonction $y \mapsto f_1(x, y)$ est monotone non décroissante.*

Nous pouvons alors énoncer **le théorème d'optimalité de Bellman** :

Théorème 1. *(Extrait de Minoux [84] p535)*
Soit f une fonction $\mathbb{R} \times \mathbb{R}^k \to \mathbb{R}$, qui à x réel et $(y_1, \cdots, y_k)$ associe $f(x, y)$.
Soit Ω l'ensemble solutions (x, y) admissibles et $\Omega_x = \{y/(x, y) \in \Omega\}$
Soit $Opt \equiv \min$ ou $\max$
On suppose :

- *f est décomposable avec $f(x, y) = f_1(x, f_2(y))$*
- *Il existe $\overline{y} \in \mathbb{R}^k$ tel que : $f_2(\overline{y}) = \underset{y \in \Omega_x}{Opt}\, f_2(y)$*
- *Il existe $\overline{x} \in \mathbb{R}$ tel que : $f_1(\overline{x}, \overline{y}) = \underset{x}{Opt} \left\{ f_1(x, \underset{y}{Opt} f_2(\overline{y})) \right\}$*

Alors on a :

$$\underset{(x,y)\in\Omega}{Opt}\ f(x, y) = \underset{x}{Opt} \left\{ f_1(x, \underset{y \in \Omega_x}{Opt}\, f_2(y)) \right\}$$

Ce théorème montre que si f est décomposable, elle peut être résolue d'abord en f_2, puis en f_1, en supposant l'optimal de $f_2(y)$ connu. Cette séparation dans la résolution est une idée structurelle de la DP. En effet, nous allons utiliser ce résultat pour décomposer le problème d'optimisation suivant à deux variables :

$$(P) \begin{cases} F = Opt f(x, y) \\ s.c\ \ g(x, y) \in E_t \end{cases}$$

Nous supposons f et g décomposables, avec $g(x, y) = g_2(y, g_1(x))$. Nous définissons alors les quantités :

$$\begin{cases} E_1 = \ \ g_1(x) \\ E_2 = \ \ g_2(y, E_1) \end{cases}$$

Nous employons ici la terminologie des systèmes dynamiques qui ont été parmi les premières applications de la DP. Les E_i sont appelés états du système et E_t l'ensemble des états terminaux. Les fonctions g_1 et g_2 sont appelées fonctions de transitions, qui traduisent l'évolution du système qui passe par les différents états E_i suivant la séquence des décisions (dans notre cas x, puis y). Nous considérons les sous-problèmes d'une variable suivants :

$$(P_2(E_1)) \begin{cases} F_2(E_1) = Opt f_2(y) \\ s.c\ \ g_2(y, E_1) \in E_t \end{cases}$$

Nous notons $\Omega_2(E_1) = \{y / g_2(y, E_1) \in E_t\}$ et remarquons que $\Omega_x = \Omega_2(g_1(x))$
Nous supposons que nous pouvons résoudre le problème $(P_2(E_1))$, Pour tout état $E_1 = g_1(x)$.
Alors, le théorème d'optimalité appliqué à f devient simplement :

$$F = \underset{(x,y)\in\Omega}{Opt} f(x,y) = \underset{x}{Opt} \left\{ f_1(x, \underset{y\in\Omega_2(g_1(x))}{Opt} f_2(y)) \right\}$$

$$F = \underset{x}{Opt} \left\{ f_1\big[x, F_2(g_1(x))\big] \right\}$$

P se résout alors comme un problème à une seule variable x.
Notons néanmoins les différents points suivants :

Remarque 4. *Lorsque la décision x conduit à un état $E_1 = g_1(x)$, pour lequel $g_1(x)$ n'est pas définie ou bien $\Omega_2(E_1)$ est vide, nous poserons $F_2(E_1) = \pm\infty$ selon le contexte (minimisation ou maximisation), ce qui exclut ces cas en pratique. L'équation ci-dessus préserve donc l'admissibilité.*

Remarque 5. *Nous avons supposé f décomposable, mais également g. Ce point est fondamental. L'application de la DP dépend de la decomposabilité de f mais également de la forme du domaine du domaine des solutions Ω, comme le remarque Minoux [84].*

L'approche précédente, avec f et g décomposables se généralise à n variables. Nous ne donnerons ici que le résultat extrait de Minoux [84]. La relation de récurrence à l'ordre k est appelée *Equation fonctionnelle de la programmation dynamique* ou *Equation de Bellman* :

$$\begin{aligned} F_k(E_{k-1}) &= \underset{x_k}{Opt} \left\{ f_{k-1}\big[x_k, F_{k+1}(g_k(x_k, E_{k-1}))\big] \right\},\ k < n \\ F_n(E_{n-1}) &= \underset{x_n / g_n(x_n, E_{n-1})\in E_t}{Opt} \{ f_n(x_n) \},\ k = n \end{aligned} \tag{3.1}$$

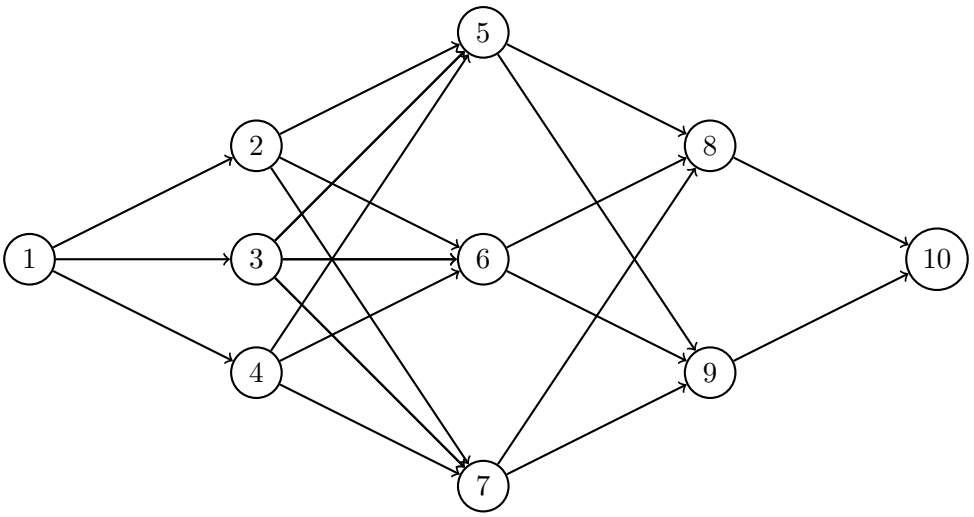
Où $f(x_1, \cdots, x_n) = f_1\Big(x_1, f_2(x_2, \cdots, f_i(x_i, f_{i+1}(\cdots, f_n(x_n)) \cdots\Big)$
Et $g(x_1, \cdots, x_n) = g_n\Big(x_n, g_{n-1}(x_{n-2}, \cdots, g_i(x_i, g_{i-1}(\cdots, g_2(x_2, g_1(x_1)) \cdots\Big)$
Et enfin les états $E_1 = g_1(x)$, $E_2 = g_2(x_2, E_1), \cdots$, $E_i = g_i(x_i, E_{i-1}), \cdots$, $E_n = g_n(x_n, E_{n-1})$

On peut donc résoudre le problème à n variables en résolvant à chaque étape de l'équation de Bellman un problème à une seule variable, pour tous les états admissibles. En effet, nous résolvons tout d'abord F_n puis l'ensemble des F_{n-1}, pour tous les état de E et ainsi

de suite. Nous déterminons enfin $F = \underset{x}{Opt}\{f_1[x, F_2(g_1(x))]\}$ C'est un algorithme par *retours arrière*, car les calculs sont effectués par ordre décroissant des indices.

Ce formalisme, qui provient de l'opérateur de composition pour les fonctions f et g, peut être considérablement simplifié pour des fonctions objectif simples, qui se rencontrent cependant souvent en pratique, comme les fonctions additives ($f(x) = \sum_i f_i(x_i)$) par exemple.

Nous en donnons un exemple avec le problème de la recherche du plus long chemin dans un graphe séquentiel. On suppose par exemple qu'un voyageur part du sommet 1 vers le point 10 avec 4 étapes, qui s'apparente au graphe suivant :



Le trajet (i)->(j) a un cout $c(i, j)$ qui est donné par le tableau ci-dessous :

départ/arrivée	2	3	4	5	6	7	8	9	10
1	3	6	8						
2				2	9	5			
3				1	8	7			
4				10	1	5			
5							2	1	
6							3	3	
7							8	4	
8									8
9									3

L'état du système E_k à l'étape k correspond au sommet atteint à l'étape k, noté x_k, sous la contrainte $x_k = 0$ et $x_4 = 10$. Nous cherchons le trajet qui maximise l'utilité c (gain ou utilité) du voyageur. Nous avons conservé la terminologie de c pour coût, bien que le principe fonctionne indifféremment pour une maximisation ou une minimisation. Nous notons $F_k(E_i)$ le trajet optimal du sommet (état) x_k à l'étape k. L'équation de Bellman (3.1) s'écrit :

$$F_k(x_k) = \max_{x_{k+1} \in E} \{c(x_k, x_{k+1}) + F_{k+1}(x_{k+1})\}$$

Nous procédons par étape dans l'ordre décroissant. Les étapes du calcul sont résumés dans les tableaux suivants :

k=3

x_3	$F_3(x_3)$	x_4
8	8	10
9	3	10

k=2

x_2	$F_2(x_2)$	x_3
5	10	8
6	11	8
7	16	8

k=1

x_1	$F_1(x_1)$	x_2
2	21	7
3	23	7
4	21	7

k=0

x_0	$F_0(x_0)$	x_1
1	29	3 ou 4

L'utilité maximale est de 29. Nous recomposons les trajets optimaux par disjonction de cas dans les tableaux en sens direct et obtenons 2 solutions distinctes (3,7,8,10), (4,7,8,10) d'utilité 29.

Pour une fonction objectif additive séparable, il existe également un algorithme *en avant*, où le calcul s'effectue par ordre croissant des indices. En effet, par application du principe du principe d'optimalité, une sous-trajectoire d'une solution optimale, reste optimale pour le sous problème. Nous pouvons aisément montrer que la solution optimale peut se construire à partir de cette sous-trajectoire.

Théorème 2. *Avec les notations précédentes, l'équation de Bellman s'écrit également :*

$$F_k(x_k) = \max_{x_{k-1} \in E} \{c(x_{k-1}, x_k) + F_{k-1}(x_{k-1})\} \tag{3.2}$$

Démonstration. Soit $(x_1, \cdots x_k)$ une solution optimale à l'étape k. La solution $(x_1, \cdots, x_{k-1})$ est admissible pour le problème $P_{k-1}(x_{k-1})$. Donc sa fonction objectif est bornée par $F_{k-1}(x_{k-1})$. Nous en déduisons :

$$\begin{aligned} \forall x_{k-1}, \quad F_k(x_k) &\leq F_{k-1}(x_{k-1}) + c(x_{k-1}, x_k) \\ \text{Donc } F_k(x_k) &\leq \max_{x_{k-1} \in E} \{c(x_{k-1}, x_k) + F_{k-1}(x_{k-1})\} \end{aligned}$$

Réciproquement, soit $(x_1, \cdots, x_{k-1})$ une solution optimale à l'état $k-1$. L'extension $(x_1 \cdots, x_{k-1}, x_k)$ où x_{k_1} est un sommet de E est une solution de $P_k(x_k)$. Elle est donc majorée par $F_k(x_k)$.
Nous en déduisons :

$$\begin{aligned} \forall x_{k-1}, \quad F_k(x_k) \geq & \quad (c(x_{k-1}, x_k) + F_{k-1}(x_{k-1}) \\ F_k(x_k) \geq & \quad \max_{x_{k-1} \in E} \{c(x_{k-1}, x_k + F_{k-1}(x_{k-1})\} \end{aligned}$$

□

On remplit le tableau par ordre croissant de l'application précédente. On en déduit les trajectoires optimales par disjonction de cas dans l'équation de Bellman, formulée *en avant* (3.2).

Etat/Sommet	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0
1	0	3	6	8	0	0	0	0	0	0
2	0	0	0	0	18	14	13	0	0	0
3	0	0	0	0	0	0	0	21	19	0
4	0	0	0	0	0	0	0	0	0	29

Nous terminons cette introduction sur la DP, par un aparté historique. Le terme de *Programmation Dynamique* est dû, à la fois au contexte des travaux de Bellman sur les systèmes dynamiques, mais également aux hasards de l'histoire. En effet, Bellman en expliqua l'origine, lors de discours de réception du Prix Norbert Wiener en mathématiques appliquées en 1970 :

> *«An interesting question is, Where did the name dynamic programming come from ? Now, as Garrett (Birkhoff) mentioned, the theory was developed in the 1950's which weren't good years for mathematical research. Some of you may remember that we had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred for the word research. I'm not using the term lightly ; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people use the term research,in his presence, You can imagine how he felt about the term, mathematical. At the time, I was employed by the Rand Corporation an it was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do someting to shield Wilson and the Airforce from the fact that I was really doing mathematics inside the Rand Corporation.*

What title, what name, could I choose ? Well, in the first place, I was interested in planning, in decision-making, in thinking. But planning, as you know is not a good word for various reasons. I decided therefore to use the word programming, I wanted to get across the idea that this was dynamic, this was multi-state, this was time-varying - I thought, let's kill two birds by one stone. Let's take a word which has an absolutely precise meaning, namely dynamic, in the classical sense. It also has a very interesting property as an adjective, and that it is impossible to use the word in the pejorative sense. Sometimes, when you want to amuse yourself, try thinking of some combination which will possible give it a pejorative meaning. It is impossible. Thus I thought dynamics programming was a good name. It was something not even a congressman could object to. So I used it as an umbrella for my research.».

3.2 Résolution exacte

Dans cette section, nous démontrons formellement l'équation de Bellman et décrivons l'algorithme de programmation dynamique, présenté dans Bellman [14], utilisé pour la résolution exacte de notre problème $(PNS_{N,M})$:

$$(P_{n,m}) \begin{cases} \max\limits_{(x_1,\dots,x_n)} f(x) = \sum\limits_{i=1}^{n} \Big[p_i - c_i\, g\Big(\sum\limits_{k=1}^{i} x_k\Big)\Big] x_i \\ s.t : h(x) = \sum\limits_{i=1}^{n} x_i - m = 0 \\ \forall i, x_i \in \mathbb{N} \\ \forall i, 0 \leq x_i \leq m \end{cases} \tag{3.3}$$

Les problèmes $(P_{n,m})$ et (1.1) sont équivalents, pour $n = N$ et $m = M$. Le problème $(P_{n,m})$ est un problème en variables entières, dont les variables de décisions sont bornées. Il admet donc des solutions pour $n \leq m$. Soit $O_{n,m}$ sa valeur optimale.

Théorème 3. ***Equation de Bellman :***

$$\forall n, m,\ 1 \leq n \leq m,\ O_{n,m} = \max_{i \in [\![0;m]\!]} \{O_{n-1,m-i} + [p_n - c_n\, g(m)]\, i\ \}$$

Sous les conditions initiales $O_{n,0} = O_{0,m} = 0$.

Démonstration : Par récurrence sur n.
Nous supposons m fixé et $(n < m)$. Pour $n = 1$, nous écoulons nécessairement tout le bloc à l'unique temps n autorisé.

$$\begin{aligned} O_{1,m} &= f(m) \\ &= [p_1 - c_1\, g(m)]\, m \\ &= \max_{i \in [\![0;m]\!]} \{[p_1 - c_1\, g(m)]i\}\,,\ car\ (p_1 - c_1\, g(m)) > 0 \\ &= \max_{i \in [\![0;m]\!]} \{O_{0,m-i} + [p_1 - c_1\, g(m)]i\}\,,\ car\ O_{0,m-i} = 0\ \forall i \end{aligned}$$

Ce qui prouve le résultat pour $n = 1$. Nous supposons le résultat vrai au rang n.
Nous définissons $g_i(x) \equiv \Big[p_i - c_i\, g\Big(\sum_{k=1}^{i} x_k\Big)\Big]$
Soit $(\widehat{x_1}, \cdots, \widehat{x}_{n+1})$ une solution optimale de $(P_{n+1,m})$. Par définition,

$$O_{n+1,m} = \sum_{i=1}^{n} g_i(\widehat{x})\, \widehat{x_i} + [p_{n+1} - c_{n+1}\, g(m)]\, \widehat{x}_{n+1}$$

Comme $\sum_{i=1}^{n} \widehat{x_i} = m - \widehat{x}_{n+1}$, les n premières coordonnées de $\widehat{x}$ forment une solution admissible de $(P_{n,m-\widehat{x}_{n+1}})$. Par hypothèse de récurrence, sa fonction objectif est bornée par $O_{n,m-\widehat{x}_{n+1}}$.
Nous en déduisons :

$$\begin{aligned} O_{n+1,m} &\leq O_{n,m-\widehat{x}_{n+1}} + [p_{n+1} - c_{n+1}\, g(m)]\, \widehat{x}_{n+1} \\ &\leq \max_{i \in [\![0;m]\!]} \{O_{n,m-i} + [p_{n+1} - c_{n+1}\, g(m)]i\ \} \end{aligned}$$

Réciproquement, pour tout $0 \leq i \leq m$, $(P_{n,m-i})$ admet une solution. Donc,

$$\exists (x_1, \cdots, x_n)\ s.t \begin{cases} O_{n,m-i} = \sum_{j=1}^{n} g_j(x)\, x_j \\ \sum_{j=1}^{n} x_j = m - i \end{cases}$$

L'extension de x par i $(x_1, \cdots, x_n, \mathbf{i})$ est admissible pour le problème $(P_{n+1,m})$. Par conséquent, sa fonction objectif est bornée par $O_{n+1,m}$:

$$\begin{aligned} f(x) &\leq O_{n+1,m} \\ \sum_{j=1}^{n} g_j(x)\, x_j + [p_{n+1} - c_{n+1}\, g(m)]\, i &\leq O_{n+1,m} \\ O_{n,m-i} + [p_{n+1} - c_{n+1}\, g(m)]\, i &\leq O_{n+1,m} \end{aligned}$$

Comme cette inégalité est vraie pour tout i de $[\![0; m]\!]$, nous prenons le maximum sur i et en concluons :

$$\max_{i \in [\![0;m]\!]} \{O_{n,m-i} + [p_{n+1} - c_{n+1}\, g(m)]\, i\} \leq O_{n+1,m}$$

Ce qui démontre le résultat pour $(P_{n+1,m})$. □

L'équation de Bellman du théorème 3 fournit un algorithme récursif de calcul de la valeur optimale $O_{n,m}$ du problème $(P_{n,m})$. Nous présentons ci-dessous l'algorithme de résolution du problème (1.1), ou de manière équivalente $(P_{N,M})$.

Nous construisons d'abord la matrice $(O_{n,m})_{n \leq N,\, m \leq M}$ qui contient les valeurs optimales des sous-problèmes. Cette matrice est de taille $N \times M$. Le dernier coefficient (ligne N, colonne M) correspond à la valeur optimale de $(P_{N,M})$.

Nous déterminons ensuite la stratégie d'écoulement optimale par retours en arrière dans l'équation de Bellman. Nous présentons l'algorithme de résolution exacte par DP, qui correspond à l'Algorithme 1.

Algorithm 1: Programmation dynamique exacte

```
||Construction de la matrice (O_{n,m})
for n = 1 to N do
    for m = 1 to M do
        for k = 0 to m do
            O_{n,m} = max( O_{n,m}, O_{n-1,m-k} + [p_n - c_n g(m)] k )
||Calcul de la stratégie optimale par retours arrière
m = M
for i = N to 1 do
    if O_{i,m} = O_{i-1,m} then
        x_i = 0
    else
        k = m
        while ( O_{i,m} ≠ O_{n-1,m-k} + [p_n - c_n g(m)] k ) ∧ (k > 0) do
            k = k - 1
        x_i = k
        m = m - k
```

Analyse de la complexité : la complexité en espace (mémoire) pour stocker la matrice $N \times M$ est $O(NM)$. En temps, pour n et m fixés, nous devons parcourir la troisième boucle de l'algorithme précédent, au pire M fois, pour déterminer $O_{n,m}$. Par conséquent, le nombre d'évaluations pour calculer $O_{N,M}$ est en $O(NM^2)$. Dans l'algorithme de backtracking (retours arrière), pour chaque $i \leq N$, nous effectuons, au pire, M comparaisons, donc sa complexité est en $O(NM)$. Par conséquent, la complexité temporelle de l'algorithme 1 reste en $O(NM^2)$.

Si nous stockons $(g(0), \cdots, g(M))$, en amont du calcul, les données en entrée se composent des vecteurs $p, c,$ *et* g et sont de taille $O(N + M)$. Nous en concluons que la complexité temporelle est polynomiale, cubique au pire car $O(NM^2) < O((N + M)^3)$. Par contre, si $g(i)$ est calculé, les données en entrée sont en $O(N)$. La complexité temporelle est alors dite pseudo-polynomiale (polynomiale seulement à M fixé).

Cependant, en pratique, la résolution exacte est bien trop couteuse en espace et en temps pour pouvoir traiter les grandes instances. Dans le prochain paragraphe, nous présentons différentes méthodes heuristiques permettant d'obtenir des bornes inférieures fines (proches de l'optimum).

3.3 Résolutions approchées du problème discret

Dans cette section, nous proposons différentes méthodes pour calculer rapidement des bornes inférieures fines. Nous commençons par deux heuristiques dites naïves, puis nous présentons deux algorithmes de DP et un algorithme de recherche locale par itération, que nous notons (ILS).

3.3.1 Heuristiques naïves

Il existe de nombreuses façons de générer un ensemble d'entiers naturels de somme donnée. Ils peuvent être par exemple générés aléatoirement. Dans cette section, nous avons choisi deux heuristiques basiques, mais intuitives pour le problème financier d'écoulement de large blocs d'actifs (ELBA), qui est une application possible de notre programme mathématique non convexe, non linéaire en variables entières. Elles nous serviront de benchmark, pour calibration de la fonction objectif (cf. section 5.1.5) ainsi que dans les expérimentations numériques du chapitre 5.

— La première est la vente «à tout prix» (*firesale*), où $x = (N, 0, \cdots, 0)$: nous liquidons l'intégralité du bloc en une seule fois au premier temps. Nous écopons logiquement de la pénalité maximale pour ce mouvement de panique. Ça n'est quasiment jamais une solution optimale et dans le cas des prix constants, elle s'avère être la pire solution quelque soit le temps auquel elle a lieu $x = (0, \cdots, 0, N, 0, \cdots, 0)$.

— La seconde est la vente uniforme dans laquelle $x = (M/N, \cdots, M/N)$: nous écoulons le bloc linéairement dans le temps. Nous montrons, dans la section 3.5, qu'elle peut s'avérer optimale sous certaines conditions sur les prix et la fonction de pénalité.

3.3.2 Méthode DP en deux étapes (TSDP)

La méthode que nous proposons consiste en deux étapes indépendantes de programmation dynamique. Dans une première étape, nous calculons très rapidement une solution approchée, en appliquant la DP à des "paquets" de taille P. P devient alors la nouvelle unité pour les variables de décision. Nous pouvons également en donner, par exemple une interprétation financière pour ELBA. Le paramètre P représente des *mini-blocs* plus faciles à écouler. Dans la deuxième étape, nous raffinons la solution obtenue en intensifiant la recherche dans le voisinage de la solution obtenue en utilisant un algorithme adapté de DP, décrit ci-après (Algorithme 2).

— **Première étape - DP à gros grain** : lorsque N est grand, la DP exacte vue précédemment est trop couteuse car sa complexité temporelle croit en $N\,M^2$. Une idée naturelle consiste alors à vendre des paquets de P unités d'actifs ($P \gg 1$), que l'on nomme le grain P. Nous appliquons l'algorithme de résolution exact présenté en section 3.2, avec le changement de variable suivant : $M' = \lfloor M/P \rfloor$. La complexité temporelle au grain P est en $O\left(\frac{N\,M^2}{P^2}\right)$, donc plus rapide par un facteur P^2. Plus le grain est gros, plus rapide est l'heuristique mais moins bonne est sa qualité (écart à l'optimum). Comme souvent, il y a donc un compromis à trouver, entre la qualité du résultat et le temps CPU, pour déterminer la taille du grain. Nous discutons ce point dans la section 3.3.5 pour en déduire la taille du grain qui minimise la complexité.

— **Deuxième étape - DP avec bornes** : nous partons d'une solution admissible obtenue à l'étape précédente, que l'on suppose proche de l'optimum. Nous ré-

appliquons alors l'algorithme de DP exact dans son voisinage. Nous restreignons la recherche en imposant des bornes sur les solutions admissibles. Nous introduisons tout d'abord les définitions suivantes : Soit $x^0 = (x_1^0, \cdots, x_N^0)$ la solution initiale. Soit l et u les bornes inférieures (resp. supérieures) de x telles que, pour tout i : $0 \leq l_i \leq x_i \leq u_i \leq M$

Soit : $L_i = \min\left(\sum_{k=1}^{i} l_k, M\right)$ et $U_i = \min\left(\sum_{k=1}^{i} u_k, M\right)$.

Donc, $\forall\, i\ L_i \leq y_i \leq U_i$, où $y_i = \sum_{k=1}^{i} x_k$

Nous réécrivons ensuite l'équation de Bellman restreinte (cf théorème 3) :

$O_{n,m} = \max_{l_n \leq k \leq u_n} \Big(O_{n-1,m-k} + \left[p_n - c_n\, g(m)\right] k\ \Big)$, subject to $L_n \leq m = y_n \leq U_n$

L'algorithme de DP avec bornes correspond à Algorithme 2, décrit ci-dessous :

Algorithm 2: Programmation dynamique avec bornes

```
||Construction de la matrice (O_{n,m})
for n = 1 to N do
    for m = L_n to U_n do
        for k = l_n to u_n do
            O_{n,m} = max(O_{n,m}, O_{n-1,m-k} + [p_t - c_t g(m)] k)
||Partie retours-arrière (backtracking) identique à celle de de l'algorithme (1)
||Trouve la solution par retours-arrière en O_{N,M} matrix
```

La complexité spatio-temporelle est obtenue directement en suivant l'algorithme précédent. En espace mémoire, nous devons stocker $O_{n,m}$, quand m varie de L_n à U_n et n varie de 1 à N. Nous en déduisons la complexité en espace en $O\left(\sum_{n=1}^{N} U_n - L_n\right)$.

Si nous supposons que $u_n - l_n$ est borné pour tout n, par un réel $0 \leq R < M$. C'est typiquement le cas pour le bandeau de taille R autour d'une heuristique donnée. Nous notons que R peut toujours être défini car les variables de décisions sont positives et bornées et $R \leq M$. Nous précisons la complexité dans le cas borné :

$$U_n - L_n = \sum_{i=1}^{n} u_i - l_i \leq n\, R$$

$$\text{Et, } \sum_{n=1}^{N} (U_n - L_n)(u_n - l_n) \leq R \sum_{n=1}^{N} (U_n - L_n)$$

$$\leq \sum_{n=1}^{N} n\, R^2 \ \leq\ \frac{N(N+1)R^2}{2}$$

En conclusion, lorsque le voisinage de recherche est borné par R, la complexité en espace est en $O\left(N^2\,R\right)$ et la complexité temporelle est en $O\left(N^2\,R^2\right)$.

— Les deux étapes peuvent être synchronisées. Nous calculons tout d'abord une heuristique au grain P dans une première étape. Puis, nous recherchons une solution plus fine en utilisant la DP avec bornes dans un bandeau de taille λP, autour de l'heuristique précédente, où λ est un scalaire. Dans cette approche en deux étapes, P et λ sont les seuls degrés de liberté. Le choix de ces paramètres permettant d'obtenir la meilleure heuristique, en terme de qualité et de temps CPU, est une question intéressante, que nous discutons au paragraphe 3.3.5.

— Les deux étapes peuvent aussi être effectuées de manière indépendante. Quelque soit l'heuristique obtenue dans la première étape, Nous pouvons lui appliquer l'algorithme de DP avec bornes. Cependant, afin d'améliorer l'heuristique, nous devons suspecter qu'un bon maximum local se trouve dans son voisinage. Typiquement, on ne peut pas s'attendre à un bon résultat si nous intensifions la recherche dans un petit bandeau autour la solution de vente «à tout prix». Bien qu'elles soient indépendantes, il faut maintenir une certaine cohérence entre les étapes.

— Il peut être également intéressant d'appliquer la DP avec bornes à une solution continue (admissible pour le problème $\overline{(PNS_{N,M})}$ de la section 1.4).
Par exemple, $l_n = \max\left(0, \lfloor x_n^0 \rfloor - \lambda P\right)$ et $u_n = \min\left(M, \lceil x_n^0 \rceil + \lambda P\right)$ définissent des bornes admissibles, lorsque P est bien choisi. Nous aurions tout aussi bien pu prendre $l_n = \max\left(0, \lfloor x_n^0 - \lambda P \rfloor\right)$ et $u_n = \min\left(M, \lceil x_n^0 + \lambda P \rceil\right)$. Les performances de cette approche hybride (relaxation continue couplée avec la DP bornée) sont abordées dans les expérimentations numériques au chapitre 5.

— Enfin, contrairement à la DP exacte, l'approche en deux étapes ne peut plus garantir l'optimalité, ce qui constitue son principal inconvénient.

3.3.3 Algorithme de recherche locale (ILS)

Dans cette partie, nous présentons, un algorithme de recherche locale, correspondant à l'Algorithme 3, qui part d'une solution admissible et retourne un maximum local.
En effet, soit x^0 une solution admissible. Dans sa plus simple forme, nous commençons par décaler la première coordonnée x_1^0 de $+P$ et la deuxième x_2^0 de $-P$, pour un entier P fixé. Nous effectuons ces changements $\left(x_1^1 = x_1^0 + P,\ x_2^1 = x_2^0 - P\right)$ si et seulement si les

variables de décisions résultantes restent dans le domaine contraint $[\![0; M]\!]$. Nous voyons directement que la somme $x_1^1 + x_2^1 + \sum_{i=3}^{N} x_i^0 = M$. Nous en déduisons que l'admissibilité d'une solution est stable par l'opérateur de décalage à l'ordre P (P-*shift*) ainsi défini, pour tout P et toute paire (x_i, x_{i+1}).

Si la fonction objectif s'améliore (croît) suite au décalage à l'ordre P, Nous stockons la différence entre les versions décalées et originales. Nous appliquons ensuite un décalage à l'ordre $-P$, sous les mêmes contraintes de bornes.

Nous procédons ensuite de même sur toutes les paires (x_i^0, x_{i+1}^0), pour i variant de 2 à N. Par conséquent, nous avons envisagé $2\,(N-1)$ décalages potentiels. Si aucun décalage, n'améliore la fonction objectif, alors l'algorithme s'arrête et renvoie x^0. Sinon, nous retenons la plus grande différence pour la fonction objectif et stockons la solution x^1 correspondante.

Nous effectuons à nouveau les étapes précédentes en partant de x^1 et ainsi de suite. l'algorithme s'arrête lorsqu'il trouve un maximum local, qui correspond à un point fixe pour l'opérateur de décalage à l'ordre $\pm P$. Sinon, il retourne la meilleure solution obtenue dans un temps ou un nombre d'itérations limités.

Enfin, afin améliorer ce maximum local, nous proposons d'appliquer plusieurs valeurs pour l'ordre P' du décalage. Par exemple, dans les expérimentations numériques, nous procédons par dichotomie sur P : $P_0 = 2^R$, $P_1 = 2^{R-1}, \cdots, P_R = 1$, et $R = \lfloor \frac{\ln(M)}{\ln(2)} \rfloor$ le plus grand entier pour lequel in décalage à l'ordre 2^R est possible.

Cet algorithme de recherche locale effectue un appel à la fonction objectif pour chaque décalage. Nous améliorons son efficacité en ne comparant que les termes de l'objectif affectés par le décalage.
L'algorithme 3 est décrit ci-après :

Algorithm 3: Recherche locale

$runSum = 0$
$auxSum = 0$
$\Delta = 0$
$f_{max} = 0$
$i_{opt} = -1$
for $i = 1$ **to** *M-1* **do**
  $d_+ = 0$
  $d_- = 0$
  $runSum = runSum + x_i$
  $auxSum = runSum + x_{i+1}$
  $d_0 = \left[p_i - c_i\, g(runSum)\right] x_i + \left[p_{i+1} - c_{i+1}\, g(auxSum)\right] x_{i+1}$
  if $(x_i + P \leq M) \wedge (x_{i+1} - P \geq 0)$ **then**
    $d_+ = \left[p_i - c_i\, g(runSum + P)\right](x_i + P) + \left[p_{i+1} - c_{i+1}\, g(auxSum)\right](x_{i+1} - P)$
  if $(x_{i+1} + P \leq M) \wedge (x_i - P \geq 0)$ **then**
    $d_- = \left[p_i - c_i\, g(runSum - P)\right](x_i - P) + \left[p_{i+1} - c_{i+1}\, g(auxSum)\right](x_{i+1} + P)$
  if $(d_+ \geq d_-)$ **then**
    if $(d_+ - d_0 > \Delta)$ **then**
      $\Delta = d_+ - d_0$
      $i_{opt} = i$
      $\epsilon = 1$
  else
    if $(d_- - d_0 > \Delta)$ **then**
      $\Delta = d_- - d_0$
      $i_{opt} = i$
      $\epsilon = -1$
  if $i_{opt} > -1$ **then**
    $x_{i_{opt}} = x_{i_{opt}} + \epsilon \cdot P$
    $x_{i_{opt}+1} = x_{i_{opt}} - \epsilon \cdot P$
    $f_{max} = f_{max} + \Delta$

Nous remarquons que, pour tout i, que les sommes courantes $\sum_{j=1}^{i} x_j$ et $\sum_{j=1}^{i+1} x_j$ sont incrémentées et que l'on ne compare que les termes modifiés de la fonction objectif f, que sont d_0 (pas de décalage), d_+ (décalage $(i, i+1, P)$) et d_- (décalage $(i, i+1, -P)$).

Concernant son champ d'application, cet algorithme peut être utilisé, soit pour générer une heuristique de première étape, au sens du paragraphe 3.3.2, à partir d'une solution naïve, ou bien comme un algorithme de deuxième étape pour affiner un maximum local déjà de bonne qualité.

3.3.4 LocalSolver

Enfin, dans ce dernier paragraphe, nous présentons un solver commercial que nous utlisons pour nous comparer. Ce solver est spécialisé dans la recherche des optima locaux. Nous avons sélectionné *LocalSolver*(*cf.* Benoist *et al.* [20, 22, 21]), qui est l'un des leaders du marché dans le domaine de la recherche locale.

Comme l'indiquent ses concepteurs dans Benoist *et al.* [20] : «*Le but de LocalSolver est d'obtenir en mode boîte noire de bonnes solutions en des temps d'exécution courts (comme le ferait des heuristiques standards de recherche locale), en particulier quand les solvers arborescents sont incapables d'en trouver une*». Les auteurs, dans [20], expliquent que la stratégie résolution de *Localsolver* repose sur trois piliers : (méta)heuristiques, mouvements autonomes et évaluation.

Le solver est en effet capable de générer des solutions admissibles, *via* différentes (méta) heuristiques. Il cherche également à permuter des variables, tout en préservant l'admissibité des solutions précédemment obtenues. Pour cela, il tente d'identifier un invariant inhérent à la structure du problème.

Pour notre problème $(PNS_{N,M})$ très peu contraint, le gradient discret ILS, introduit à la section 3.3.3, est un parfait exemple pour illustrer cette démarche. Nous supposons connue, une solution admissible $(x_1, \cdots x_N)$, telle que $\sum_{i=1}^{N} x_i = M$. L'heuristique de recherche locale de la section précédente, consiste alors à *shifter* $x_i \leftarrow x_i + 1, x_{i+1} \leftarrow x_i - 1$. Nous pourrions aisément changer les valeurs d'un sous-ensemble $\{x_{i_1}, x_{i_2}, \cdots, x_{i_k}\}$ de variables de décision, tout en préservant la valeur de la somme et donc l'admissibilité de la solution. L'opérateur somme offre de nombreuses combinaisons. En fait, toutes les solutions admissibles peuvent être atteintes. En cela, notre problème se prête très bien aux explorations des solutions admissibles dans le voisinage de la solution courante, que les auteurs nomment *mouvements autonomes*.
Enfin, le solver doit évaluer la qualité du mouvement proposé, l'accepter ou le rejeter et en cas d'acceptation le propager dans son arbre d'exploration.

A notre connaissance, *LocalSolver*, comprend également des étapes classiques de prétraitement (reformulation du problème, réduction du domaine, élimination de variables) et

pour le traitement des problèmes en variables continues, de l'utilisation des relaxations (linéaires et convexes), afin d'obtenir des bornes ou de prouver l'optimalité. L'architecture cible (v4.0 en 2014), qui devrait être en place dans la version utilisée (v9.5) est décrite dans Benoist *et al.* [22] :

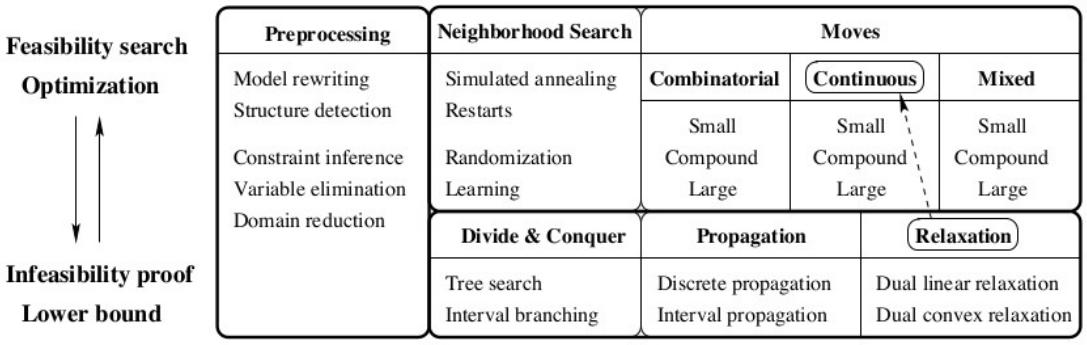


FIGURE 3.1 – Architecture cible de *LocalSolver* v4.0 (cf. [22] p64)

Nous n'avons pas mené une étude comparative exhaustive de tous les solvers du marché, permettant de résoudre localement le problème (1.1). Nous supposons donc que les résultats obtenus par *LocalSolver* sont représentatifs de l'état de l'art des solvers commerciaux, spécialisés dans la recherche locale, pertinents pour notre problème.

3.3.5 Taille du grain et complexité

Nous nous intéressons dans cette section à l'impact de la finesse du grain P (incrément minimum des variables de décision) sur la complexité de $TSDP$.
Nous supposons $N = 10^a, M = 10^b, P = 10^c$ avec $\lambda = 5$ et nous appliquons les résultats de complexité de la section 3.3.

La DP à gros grain est en $O(10^{a+2b-2c})$ et la DP avec bornes est en $O(10^{2(a+c+1)})$. La complexité temporelle s'avère donc être en $O(10^{\max(a+2b-2c,2a+2c+2)})$. Son minimum théorique est atteint pour $c = \frac{1}{4}(2b - a - 2)$, ce qui revient à une complexité temporelle minimale en $O(10^{\frac{3a}{2}+b+1})$ (ou $O(N^{\frac{3}{2}} \cdot M)$). Si nous restreignons c aux entiers naturels, nous devons faire un arrondi au plus proche entier, d'où $\overline{c} = \lfloor c + 0.5 \rfloor$. Par conséquent, la complexité temporelle réalisée est en $O\left(10^{\max(a+2b-2\overline{c},2a+2\overline{c}+2)}\right)$.

En comparaison de la DP exacte $O(10^{a+2b})$, nous gagnons théoriquement un facteur $10^{b-\frac{a}{2}-1}$ (ou $\frac{M}{10\sqrt{N}}$). De plus, le gain de temps réalisé grâce à $TSDP$ croit lorsque M croît relativement à N. Si nous supposons que, a, b et P sont entiers, afin de respecter le cadre du problème en variables entières $(PNS_{N,M})$, nous pouvons par exemple réaliser la quasi-totalité du gain théorique en supposant c rationnel et en posant $P = \lfloor 10^c \rfloor$.

3.4 Résolutions approchées du problème continu

Nous nous intéressons, dans cette section, au problème en variables continues, noté $(\overline{PNS_{N,M}})$, pour lequel nous cherchons à calculer des bornes inférieures (i.e solutions admissibles) de bonne qualité . En effet, sans autre hypothèse de convexité, les solutions obtenues, restent au mieux des extréma locaux.

Cependant, lorsque $M \gg N$ les solutions continues constituent de bonnes approximations pour le problème en variables entières. C'est pourquoi, nous espérons également trouver dans le voisinage de ces maxima locaux variables en continues, de (très) bonnes solutions entières de $(PNS_{N,M})$.

Le problème $(\overline{PNS_{N,M}})$ (*cf.* (1.4)) est muni d'une fonction objectif $\mathcal{C}^1$ définie sur un support compact $\mathcal{C}$. Il existe donc un maximum global, qui fournirait une borne supérieure pour le problème (1.1) en variables entières. Cependant, la résolution globale est à priori aussi difficile en variables continues qu'en variables entières.

C'est pourquoi nous considérons ici des techniques issues de la programmation mathématique non linéaire convexe, en réduisant nos ambitions à des maximas locaux. Avec une seule contrainte linéaire, nous nous sommes orientés vers des algorithmes de gradient. Nous avons sélectionné la méthode de gradient projeté proposée par Rosen [95], [96].

3.4.1 Gradient projeté

Nous commençons par expliciter l'opérateur de projection orthogonale sur l'hyperplan des solutions admissibles $\mathcal{C}$. Il se calcule directement, en raison de la linéarité de la contrainte :

— Le gradient projeté correspond à la projection orthogonale sur l'hyperplan tangent à une solution admissible. Pour une contrainte linéaire, il correspond simplement à l'hyperplan des contraintes $\mathcal{C}$.

— Soit x un vecteur de $\mathbb{R}^N$ et $P_{\mathcal{C}}(x)$ sa projection orthogonale sur $\mathcal{C}$. Par définition de la projection orthogonale, $x - P_{\mathcal{C}}(x)$ est orthogonal à $\mathcal{C}$.

— Le gradient de la contrainte $(\sum_{i=1}^{N} x_i - M)$ est $(1, \cdots, 1)$. Il est par définition orthogonal to $\mathcal{C}$, donc colinéaire à $x - P_{\mathcal{C}}(x)$. Nous en déduisons $\forall i,\ P_{\mathcal{C}}(x)_i = x_i + \mu \cdot 1$.

$P_{\mathcal{C}}(x)$ est une stratégie admissible. En sommant sur i, nous obtenons : $\mu = \frac{1}{T}(N - \sum_{i=1}^{T} x_i)$.

— Nous en concluons pour l'opérateur $P_{\mathcal{C}}$: $\forall i,\ P_{\mathcal{C}}(x)_i = x_i + \frac{1}{T}\Big(N - \sum_{j=1}^{T} x_j\Big)$.

Nous décrivons ensuite l'algorithme de gradient projeté :

— Nous partons d'une solution admissible x^0.

— Nous effectuons un déplacement depuis x^0 de la quantité $\eta \cdot \nabla f(x^0)$, où f correspond à la valeur de la function objectif du problème (1.4) et le pas du gradient η est constant.

— Pour simplifier les notations nous noterons : $\nabla f(x^0)_i \equiv \nabla f_i^0$.

— Nous projetons orthogonalement sur $\mathcal{C}$, $x^1 = \left(x^0 + \eta \cdot \nabla f^0\right)$: $P_{\mathcal{C}}(x^1)_i = x_i^0 + \eta\left(\nabla f_i^0 - \frac{1}{T}\sum_{j=1}^{T} \nabla f_j^0\right)$.

— Nous itérons sur sur x^i, jusqu'à la condition de sortie $\|\nabla f^i\| \leq \epsilon$, pour un seuil ϵ prédéfini.

3.4.2 Solver libre *NLopt*

Plutôt que d'implémenter directement différents algorithmes de gradient, nous nous sommes tournés vers un solver libre et ouvert, spécialisé en optimisation non linéaire continue : *NLopt* (*cf.* Johnson [64]). Nous avons testé tous les algorithmes de gradients disponibles et pertinents pour notre problème (1.4) et avons selectionné le *Conservative Convex Separable Approximation, Quadratic Version*, noté (CCSAQ) et proposé par Svanberg [108], qui est le plus efficace pour le problème continu.

Dans un contexte de maximisation, celui-ci génère et résout des sous-problèmes convexes séparés pourvus, à chaque itération, d'une fonction objectif et de contraintes approchées. Une propriété de ces sous-problèmes est qu'ils fournissent, sous certaines conditions, une suite de solutions réalisables qui converge dans l'ensemble des points qui satisfont les conditions de KKT. Cet ensemble est non vide car l'optimal existe. Mais, sans autre hypothèse, la solution limite satisfait une condition nécessaire non suffisante de convergence

globale. Elle ne fournit donc pas en général un maximum global et ne peut pas non plus être utilisée comme borne supérieure pour le problème en variables entières.

En pratique, cela se vérifie, car nous avons trouvé certaines instances pour lesquelles la borne inférieure fournie par *NLopt* était en dessous de l'optimal du problème en variables entières (par une très faible marge néanmoins).

3.5 Borne supérieure naïve : majoration par une fonction séparée

Les méthodes des sections précédentes ont permis de calculer des bornes inférieures, pour les problèmes en variables entières ou continues. Cependant, nous avons besoin de la résolution exacte par DP afin de mesurer la qualité des bornes inférieures obtenues. Lorsque la DP exacte n'est pas disponible (en raison de contrainte de mémoire) ou pas efficace (contrainte de temps), la distance à l'optimum ne peut être mesurée. Nous recherchons donc classiquement une borne supérieure, permettant d'encadrer l'optimum global, bien que, dans le cas non convexe, cela peut s'avérer une tâche ardue.

Nous proposons, dans cette section, de calculer première borne supérieure, en majorant par un problème séparé, qui peut être résolu analytiquement, sous certaines hypothèses de régularités sur la fonction de pénalité g. Pour cela, nous utilisons un argument de monotonie. La fonction g est strictement croissante et les variables de décision x_i sont positifs. Nous minorons la fonction de pénalité par sa dernière variable de décision, qui est positive, ce qui permet d'obtenir une fonction objectif séparée :

$$\text{On a } \overline{g}\Big(\sum_{k=1}^{i} x_k\Big) \geq \overline{g}(x_i)$$

$$\text{Donc } -\overline{g}\Big(\sum_{k=1}^{i} x_k\Big) \leq -\overline{g}(x_i)$$

$$\text{D'où } f(x) \leq \sum_{i=1}^{N} \Big[p_i - c_i \cdot \overline{g}(x_i)\Big] x_i$$

Nous nous intéressons au problème non linéaire, en variables continues suivant :

$$\mathrm{UB}_2 = \max_{x \in \mathcal{C}} U(x) \tag{3.4}$$

avec

$$U(x) = \sum_{n=1}^{N} u_i(x_i) = \sum_{n=1}^{N} \Big[p_i - c_i \cdot \overline{g}(x_i)\Big] x_i$$

Ce problème est bien défini. Il est de plus un majorant des problèmes (1.1) et (1.4). De plus, c'est un problème séparable (sa fonction objectif peut s'écrire comme une somme de de fonctions univariées).
Nous présentons maintenant une condition suffisante sur la fonction de pénalité g qui permet d'assurer que le problème (3.4) est convexe. [1]

Lemme 2. *Nous considérons la fonction $x \mapsto x\,\overline{g}(x)$, définie sur $\mathbb{R}_+$. Si celle-ci est strictement convexe, alors la fonction U is strictement concave et le problème (3.4) is convexe.*

Démonstration. Il vient directement de la définition de u_i :

$$u_i(x) = p_i\,x - c_i\,[x\,\overline{g}(x)]$$

Pour tout i, $c_i \geq 0$, donc $-c_i\,[x\,\overline{g}]$ est strictement concave ainsi que u_i, qui est la somme d'une fonction linéaire et d'une fonction concave. Par conséquent U, qui est la somme de fonction concaves est également concave. □

Nous avons choisi, dans les expérimentations numériques des fonctions pour lesquelles les résultats du lemme 2 s'appliquent.

Lemme 3. *Les fonctions concaves utilisées pour les expérimentations numériques (cf. section 5.1.3), $g(x) = \frac{x}{1+x}$, $g(x) = 1 - \frac{2}{1+\sqrt{1+x}}$, $g(x) = \frac{2}{\pi} arctan(x)$ sont concaves et satisfont les hypothèses du lemme 2.*

Démonstration. La convexité est obtenue par le calcul direct de $\big[x\,g(x)\big]''$ □

3.5.1 Résolution du problème séparé

Nous calculons le lagrangien du problème (3.4) et le recherchons ses points stationnaires. La condition de qualification de la contrainte est satisfaite par linéarité de cette

1. Dans ce document, nous parlerons uniquement de problème convexe, même dans un contexte de maximisation, étant bien entendu que nous cherchons à montrer la concavité des fonctions et/ou des contraintes, ou leur majoration par une fonction concave.

dernière et le problème (3.4) est convexe. Par conséquent les équations de Lagrange fournissent un maximum global.

$$L(x,\lambda) = \sum_{i=1}^{N} \Big[p_i - c_i \cdot \overline{g}(x_i)\Big] x_i - \lambda\Big(\sum_{i=1}^{N} x_i - M\Big)$$

$$\nabla L(x,\lambda) = 0 \iff \begin{cases} \forall i,\, \big[x\,\overline{g}\big]'(x_i) = \dfrac{p_i - \lambda}{c_i} \\ \displaystyle\sum_{i=1}^{N} x_i - M = 0 \end{cases}$$

Nous nous intéressons plus particulièrement à $x \mapsto x\,\overline{g}$:

$$\text{On a : } x\,\overline{g}(x) = \frac{p_i - \lambda}{c_i}\, x + b$$

Comme g est continue en 0, donc : $b = 0$

$$\text{D'où : } x\,\overline{g}(x) = \frac{p_i - \lambda}{c_i}\, x$$

Nous pouvons facilement résoudre les équations de Lagrange lorsque les vecteurs des prix p et c sont constants.

Lemme 4. *Sous les hypothèses du lemme 2, si les vecteurs de prix p et c sont constants, alors le maximum global du problème (3.4) est unique, la stratégie optimal d'écoulement est donnée par :* $\hat{x} = (\frac{M}{N}, \cdots, \frac{M}{N})$ *et* $UB_2 = M\left[p - c\,g(\frac{M}{N})\right]$

Démonstration. Les conditions de KKT ont permis d'obtenir $x\,\overline{g}(x) = \dfrac{p-\lambda}{c}\, x$.
La fonction g is strictement croissante et continue sur $\mathbb{R}_+$; elle est donc injective. Nous en déduisons à i fixé, soit $x_i = 0$ ou bien $x_i = \overline{g}^{-1}(\frac{p-\lambda}{c})$, qui est une constante indépendante de i.
De plus, sous les hypothèses du lemme 2, $x \cdot g$ est aussi strictement convexe, donc $[x \cdot g]'$ est strictement croissant sur $\mathbb{R}_+$ et par conséquent injective. Soit $x = (x_1, \cdots, x_N)$ satisfaisant aux conditions de KKT et $i < j$:

$$\big[x\,\overline{g}\big]'(x_i) = \big[x\,\overline{g}\big]'(x_j) = \frac{p-\lambda}{c}$$

Nous en concluons $x_i = x_j$. Puisque le vecteur nul ne satisfait pas la contrainte, il ne nous reste que l'écoulement linéaire $\forall, i,\ x_i = \frac{M}{N}, \lambda = p - c\,g(\frac{M}{N})$, et UB_2 est obtenu par calcul direct de U. □

Lorsque nous relâchons cette hypothèse et que p varie, nous pouvons encore obtenir une borne supérieure pour le problème (3.4). En effet, soit $\overline{p} = \max_i p_i$ et $\underline{c} = \min_i c_i$
Alors, pour tout t : $u_t(x_t) \leq \Big[\overline{p} - \underline{c}\,\overline{g}(x_t)\Big]x_t$ En appliquant le lemme 4 au membre de droite, nous obtenons :

Lemme 5. *sous les hypothèses du lemme 2, avec* $\overline{p} = \max_i p_i$ *et* $\underline{c} = \min_i c_i$*, nous établissons :*

$$UB_2 \leq \; M\left[\overline{p} - \underline{c}\, g\Big(\frac{M}{N}\Big)\right]$$

Pour simplifier les notations, en particulier expérimentations numériques, nous nous référerons à cette borne supérieure comme UB_2, sans séparer les différents cas (p constant ou non).

Une résolution analytique est également théoriquement possible, si l'on suppose que $\forall i,\ x_i > 0$: Nous repartons des conditions de KKT que nous sommons sur i. Nous en déduisons l'équation implicite en λ suivante :

$$\sum_{i=1}^{N} g^{-1}\Big(\frac{p_i - \lambda}{c_i}\Big) = M \tag{3.5}$$

Si l'on suppose cette équation résolue en utilisant les méthodes classiques de recherche de racines (méthode de Newton etc.) et définissons Λ l'ensemble de ses solutions, alors pour tout élément $\lambda \in \Lambda$, nous en déduisons une solution optimale x_λ, telle que :

$$\forall i,\ x_{\lambda,i} = g^{-1}\Big(\frac{p_i - \lambda}{c_i}\Big)$$

Nous n'avons cependant pas poursuivi dans cette direction lors des expérimentations numériques.

3.5.2 Autres résultats théoriques

Nous allons chercher à montrer, dans cette section, que, lorsque les vecteurs de prix p et c sont constants, la solution optimale du problème en variables entières ne contient pas d'éléments nuls. Nous supposerons cette hypothèse satisfaite dans toute cette section.

Pour cela, comme pour l'algorithme de gradient discret de la section 3.3.3, nous nous intéressons à l'opérateur de shift de taille P, au rang j, qui préserve l'admissibilité des stratégies d'écoulement. Soit $x = (x_1, \cdots, x_{j-1}, x_j, x_{j+1}, \cdots, x_N)$

et $z = (x_1, \cdots, x_{j-1}, x_j + P, x_{j+1} - P, x_{j+2}, \cdots, x_N)$
Nous calculons :

$$\begin{aligned} f(z) - f(x) &= \sum_{n=1}^{N} v_i(z_i) - u_i(x_i) \\ &= \sum_{n=1}^{N} p_i \left[z_i - x_i\right] + c_i \left[g\Big(\sum_{k=1}^{i} x_k \Big) \cdot x_i - g\Big(\sum_{k=1}^{i} z_k \Big) \cdot z_i \right] \end{aligned}$$

Sous l'hypothèse de prix constants, nous pouvons considérablement simplifier cette différence. En effet, dans ce cas, les termes u_i pour $i \leq j-1$, sont identiques entre x et z donc les sommes partielles de 1 à i en z_i et x_i le sont également. De même, pour $i \geq j+2,\ x_i = z_i$ et les termes $x_j + P$ et $x_j - P$ sont inclus dans les sommes partielles à i. Le shift P s'annule et les sommes partielles sont égales. Nous en déduisons donc :

$$\begin{aligned} f(z) - f(x) =& \sum_{i=j}^{j+1} v_i(z_i) - u_i(x_i) \\ =& p\left[x_j + P - x_j + x_{j+1} - P - x_{j+1}\right] \\ &+ c\Big[g(y_j)\, x_j - g(y_j + P)(x_j + P) + g(y_{j+1})\, x_{j+1} - g(y_{j+1})(x_j - P)\Big] \end{aligned}$$

Nous en déduisons :

$$f(z) - f(x) = c\left(x_j\left[g(y_j) - g(y_j + P)\right] + P\left[g(y_{j+1}) - g(y_j + P)\right]\right) \tag{3.6}$$

Où $y_j = \sum_{i=1}^{j} x_i$.

Théorème 4. *Soit x une solution admissible de $\mathcal{D}$. S'il existe un indice $j \leq N$ tel que $x_j = 0$, alors x n'est pas optimale pour le problème (1.1) en variables entières.*

Démonstration. Nous allons procéder par l'absurde.
Soit x la stratégie optimale de $\mathcal{D}$. Nous avons $x = (x_1, \cdots, x_{j-1}, 0, x_{j+1}, \cdots, x_N)$ où j est le premier indice de coordonnée nulle de x.
Etape 1 : Nous nous intéressons tout d'abord à $z_1 = (x_1, \cdots, x_{j-1}, x_{j+1}, 0, x_{j+2}, \cdots, x_N)$ et $z_2 = (x_1, \cdots, x_{j-2}, 0, x_{j-1}, x_{j+1}, \cdots, x_N)$
Nous faisons un shift de x, à l'ordre j de $P = +x_{j+1}$ pour obtenir z_1 :

$$f(z_1) - f(x) = c\left(0 \cdot \left[g(y_j) - g(y_j + x_{j+1})\right] + x_{j+1}\left[\underbrace{g(y_{j+1}) - g(y_j + x_{j+1})}_{=0}\right]\right)$$

Le second terme est nul car $y_{j+1} = y_j + x_{j+1}$. Nous en concluons $f(x) = f(z_1)$.
De même, nous effectuons un shift de x, à l'ordre $j-1$ de $P = -x_{j-1}$ pour obtenir z_2 :

$$\begin{aligned} f(z_2) - f(x) =& c\Big(x_{j-1} \cdot [g(y_{j-1}) - g(y_{j-1} - x_{j-1})] - x_{j-1}\left[g(y_j) - g(y_{j-1} - x_{j-1})\right]\Big) \\ =& c\Big(x_{j-1} \cdot [g(y_{j-1}) - g(y_{j-2})] - x_{j-1}\left[g(y_j) - g(y_{j-2})\right]\Big) \\ =& c \cdot x_{j-1}\Big(g(y_{j-1}) - g(y_j)\Big) \\ f(z_2) - f(x) =& 0, \textit{ parce que } y_j = y_{j-1} + \underbrace{x_j}_{=0} \\ \text{Donc } f(z_2) =& f(x) \end{aligned}$$

Etape 2 : Nous nous intéressons maintenant au shift de x d'une unité ($P = 1$),
à l'ordre j.

$$\begin{aligned} f(z) - f(x) =& c\Big(0 \cdot [g(y_j) - g(y_j + 1)] + [g(y_j + x_{j+1}) - g(y_j + 1)]\Big) \\ =& c\Big(g(y_j + x_{j+1}) - g(y_j + 1)\Big) \end{aligned}$$

Nous remarquons donc que si $x_{j+1} \geq 2$ alors cette différence est strictement positive, ce qui conduit à $f(z) > f(x)$ ce qui contredit x optimal.
Comme les x_k sont positifs, par hypothèse, cela impose donc $x_{j+1} = b_{j+1} \in \{0; 1\}$. Nous noterons b en référence à la variable binaire.
Donc $x = (x_1, \cdots, x_{j-1}, 0, b_{j+1}, x_{j+2}, \cdots, x_N)$.
Cependant, $z_1 = (x_1, \cdots, x_{j-1}, b_{j+1}, 0, x_{j+2}, \cdots, x_N)$ est également optimal car $f(z_1) = f(x)$. Nous procédons comme précédemment, en appliquant à z_1 un décalage (*i.e, shift*) d'une unité en $j+1$. Et nous concluons que $x_{j+2} = b_{j+2} \in \{0; 1\}$, sinon nous aboutissons à une contradiction.
En procédant ainsi sur tous les indices $i > j+1$, nous montrons que :
$x = (x_1, \cdots, x_{j-1}, 0, b_{j+1}, b_{j+2}, \cdots, b_N)$, où les b_i sont binaires.

Etape 3 : Symétriquement, $z_2 = (x_1, \cdots, x_{j-2}, 0, x_{j-1}, b_{j+1}, \cdots, b_N)$ est également optimal car $f(z_2) = f(x)$. Comme précédemment, le décalage d'une unité en $j-1$ permet de montrer que x_{j-1} est binaire. Nous procédons de même, de proche en proche pour $i < j$.
Nous avons donc établi que $x = (x_1, \cdots, x_{j-1}, 0, x_{j+1}, \cdots, x_N)$ optimal entraîne $x = (0, b_2, b_3, \cdots, b_N)$, où $b_i \in \{0; 1\}$.
Mais $\sum_{i=1}^{N} x_i < N < M$, ce qui contredit l'admissibilité de la solution x. □

3.6 Conclusion

Dans ce chapitre, nous avons présenté une méthode de résolution à l'optimum de $(PNS_{N,M})$, *via* la DP. Cependant, la résolution n'est efficace que pour les petites instances, en raison de sa complexité en $O(N\,M^2)$. C'est pourquoi nous avons introduit trois grands types d'heuristiques.

Tout d'abord les heuristiques naïves (écoulement à tout prix ou linéaire) qui servent de benchmark.

Puis, dans la continuité de la méthode exacte, la méthode TSDP, qui, comme son nom l'indique procède en deux étapes, à l'aide de la DP : une étape, dite à gros grain pour obtenir rapidement une solution admissible puis une recherche plus intensive dans un bandeau autour de cette solution.

Enfin, nous avons présenté un algorithme de recherche locale, qui part d'une solution admissible et l'améliore par décalages successifs, le long de la direction de plus grande variation (d'où son surnom de gradient discret). Nous notons, à titre de perspectives, que de nombreuses autres méthodes (meta)heuristiques, pourraient également être utilisées pour obtenir une bonne solution admissible (*e.g,* approfondissement de la méthode des voisinage, recherche tabou, algorithme génétique).

Concernant la méthode TSDP, nous avons tout d'abord discuté la complexité de cette heuristique en fonction de la taille du grain et du bandeau et estimé le gain de complexité, par rapport à la solution exacte, lorsque le grain minimise la complexité de l'heuristique TSDP.

Bien qu'elles puissent être couplées, nous avons étudié le cas où les deux étapes de la méthode TSDP sont indépendantes. En effet lorsque $M \gg N$, la relaxation continue de $(PNS_{N,M})$, fournit une bonne approximation du problème en variables entières.
Nous avons alors appliqué des méthodes d'optimisations continues, comme le gradient projeté, pour obtenir un maximum local du problème continu.
Nous avons ensuite utilisé ce maximum local (continu) comme point de départ de la deuxième étape, qui fournit des solutions entières de $(PNS_{N,M})$, par recherche dans le voisinage, en s'appuyant sur la DP avec bornes.

Nous avons donc proposé une méthode hybride où les solutions locales du problème continu, servent à approximer finement la solution globale du problème en variables entières.
Nous avons également présenté une variation de cette méthode hybride, dont la première étape, se compose des solutions continues obtenues à partir d'un solver dédié à l'optimisation continue non linéaire *NLopt*. Comme précédemment, nous notons, à titre de perspectives, l'existence de différentes méthodes d'optimisation continue, qui auraient pu être utilisées, comme première étape de notre méthode hybride.

L'objectif de la première partie de ce chapitre a été de proposer une résolution «à gauche» (dans un contexte de maximisation) de $(PNS_{N,M})$, c'est dire une résolution à l'optimum, ou une borne inférieure très proche de l'optimum global.

Dans la seconde partie du chapitre, nous avons, par un argument de monotonie, obtenu une première borne supérieure de $(PNS_{N,M})$, sous conditions supplémentaires sur g. De manière générale, l'obtention de bornes supérieures qui convergent vers l'optimum, fera l'objet du prochain chapitre.

Le dernier paragraphe établit un résultat théorique sur la nature de la solution optimale, qui, sous l'hypothèse de trajectoire constante, est dépourvue d'éléments nuls.

Le schéma suivant résume l'articulation des méthodes proposées dans ce chapitre :

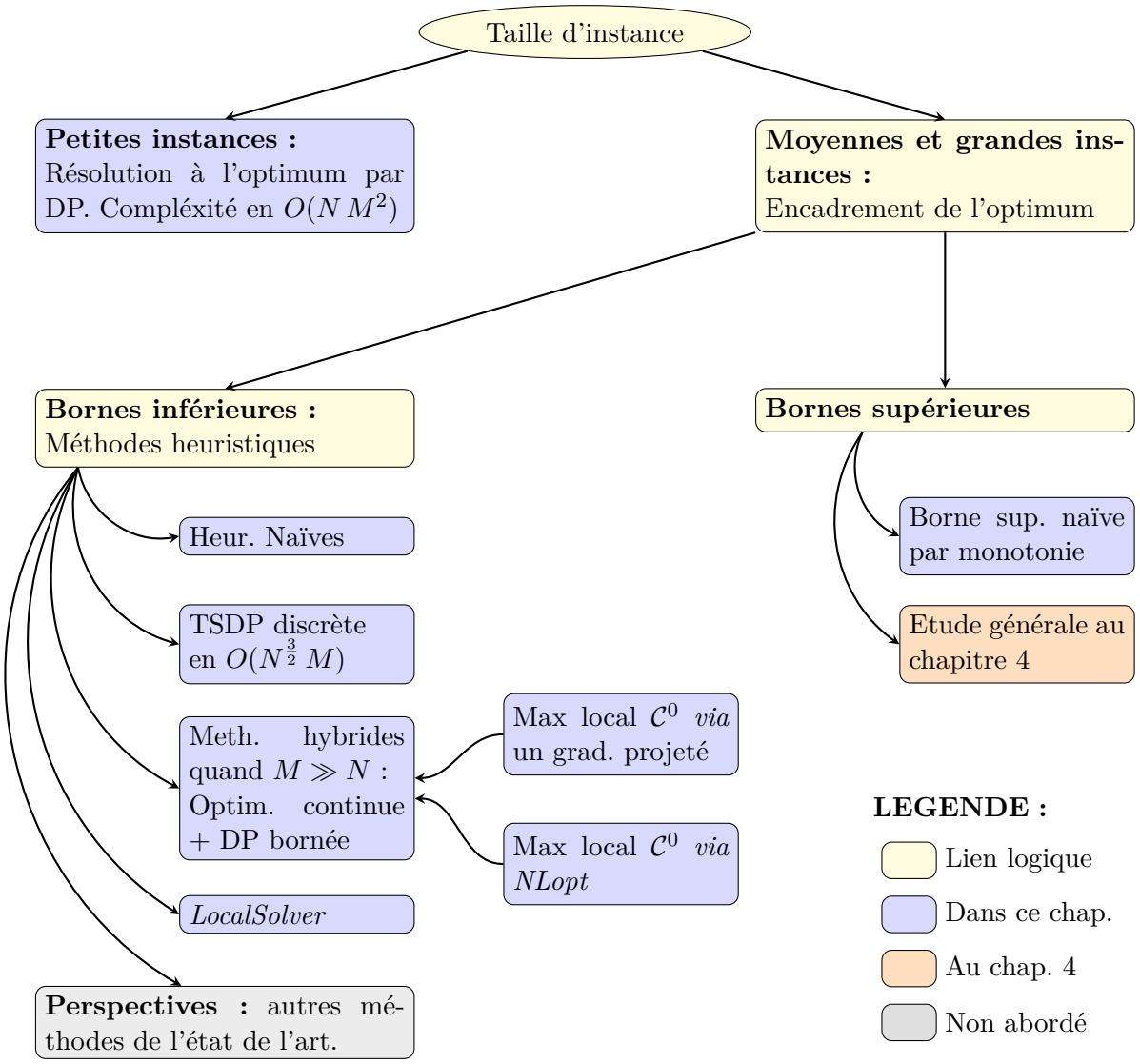


FIGURE 3.2 – Schéma récapitulatif des méthodes proposées dans ce chapitre

Chapitre 4

Convexification du problème

Nous avons introduit, dans le chapitre précédent, les méthodes exactes, pour résoudre le problème à l'optimum, ou approchées pour fournir de bonnes solutions admissibles (bornes inférieures de l'optimum) en nombres entiers. Cependant, lorsque les méthodes exactes ne sont pas disponibles ou efficaces, il est nécessaire d'avoir une borne supérieure afin d'obtenir un encadrement fin de l'optimal. Comme le problème est non convexe, les optima locaux ne sont pas globaux. La relaxation continue ne fournit donc pas directement une borne supérieure du problème discret. Elle doit pour cela être résolue globalement, ce qui est à priori aussi difficile que le problème en variables entières. La technique naturellement employée (*cf.* section 2.3 de l'état de l'art) dans cette situation, consiste à convexifier le problème, c'est à dire, l'approximer (le majorer dans un contexte de maximisation) par un problème convexe qui peut être résolu plus facilement et qui fournit une borne supérieure du problème initial. Nous pouvons itérer ce procédé, ce qui revient à déterminer une suite de problèmes convexes qui majorent le problème initial et en fournissent une borne supérieure de plus en plus fine (proche de l'optimum du problème initial). Nous nous intéressons à la convergence d'une telle suite vers l'optimum du problème initial, afin d'obtenir une résolution exacte (ou $\epsilon-$convergente dans le cas continu), ou bien fournir une borne supérieure fine en temps contraint.

Une première étape, préalable à sa convexification, consiste à transformer le problème en un problème équivalent, qui met en évidence les termes non convexes. En effet, le problème traité $(PNS_{N,M})$ a une seule contrainte linéaire d'égalité, donc sa non convexité réside dans la fonction objectif. Nous allons montrer qu'elle peut s'écrire comme une somme de termes linéaires dans les variables de décision, ou de produits bilinéaires, en introduisant des variables auxiliaires. Cette technique de transformation du problème en

un problème équivalent, en décomposant sa fonction objectif en sommes et produits bilinéaires est appelée la *factorisation* du problème. Cette technique, ainsi que l'étude des problèmes factorisables ont été introduites par McCormick [79].

Nous effectuons quelques rappels sur la factorisation à la section 4.1, puis nous appliquons cette technique à notre problème à la section 4.2. Nous proposons ensuite une relaxation convexe du problème factorisé à la section 4.3. Enfin, à la section 4.5, nous proposons deux algorithmes de *Branch & Bound*, en faisant varier les règles d'exploration. Nous cherchons à déterminer une solution exacte, ou bien une borne supérieure fine du problème factorisé.

4.1 Rappels de factorisation

Nous nous intéressons dans cette section à la *factorisation* d'un problème d'optimisation. Elle a été introduite dans les travaux fondateurs de McCormick [79] et [80]. Nous utilisons la factorisation pour atteindre deux objectifs. Nous cherchons tout d'abord à décomposer la fonction objectif et les contraintes du problème en éléments plus simples.

Comme au chapitre précédent, cela constitue d'ailleurs une démarche générale en optimisation, nous cherchons à nous ramener à des fonctions d'une seule variable ou à des fonctions séparables, qui s'écrivent comme une somme de fonctions univariées.

Puis, notre second objectif est d'isoler les sources de non convexité, afin de pouvoir les traiter séparément.
Prenons par exemple le problème (P_1) défini par :

$$(P_1)\begin{cases} \min & f(x) = \cos(x_1 + x_2) + {x_1}^2 + 7{x_2}^2 - 2x_1\,x_2 \\ s.c & \left|\begin{array}{l} a_1 \leq x_1 \leq b_1 \\ a_2 \leq x_2 \leq b_2 \end{array}\right. \end{cases}$$

Nous introduisons les variables auxiliaires suivantes :

$$\begin{array}{ll} x_3 = & x_1 + x_2 \\ x_4 = & \cos(x_3) - {x_3}^2 \\ x_5 = & x_4 + 2{x_1}^2 + 8{x_2}^2 \end{array}$$

et définissons le problème suivant :

$$(P_2)\begin{cases} \min & x_5 \\ s.c & x_1 + x_2 - x_3 = 0 \\ & \cos(x_3) - {x_3}^2 - x_4 = 0 \\ & x_5 = x_4 + 2{x_1}^2 + 8{x_2}^2 \\ & 0 \leq a_1 \leq x_1 \leq b_1 \\ & 0 \leq a_2 \leq x_2 \leq b_2 \\ & a_1 + a_2 \leq x_3 \leq b_1 + b_2 \\ & a_4 \leq x_4 \leq b_4 \end{cases}$$

Où les bornes a_4, b_4 se calculent directement par analyse réelle de la fonction univariée $x \mapsto \cos(x) - x^2$.

Les problèmes (P_1) et (P_2) sont clairement équivalents. Les deux problèmes sont non convexes. Cependant (P_2) est séparable. Il apparaît de plus clairement dans (P_2) que le seul terme non convexe dans l'expression de x_4 est : $\cos(x_3) - {x_3}^2$ (qui est en fait une fonction concave sur $\mathbb{R}$). Si nous connaissons une fonction minorante convexe proche de $x_3 \mapsto \cos(x_3) - {x_3}^2$ sur son domaine de définition, alors nous pouvons déterminer un problème minorant convexe de (P_2) qui fournira une borne inférieure fine.

Nous présentons maintenant la définition formelle d'une fonction factorisable, extraite de Schmidt [99] :

Définition 2. *Une fonction $f : \mathbb{R}^n \to \mathbb{R}$ est dite factorisable si elle peut être écrite comme la somme de produits de fonctions univariées continues issues d'un ensemble donné $\mathcal{O}$, dont les arguments sont des variables, des constantes ou d'autres fonctions factorisables.*

Un ensemble adapté à nos exemples est : $\mathcal{O} = \{+, \times, /, \wedge, \exp, \log, \sin, \cos, \}$.

Une interprétation graphique d'une fonction factorisable est celle d'un arbre dont les nœuds sont des opérateurs de $\mathcal{O}$ et les feuilles des variables ou des constantes.

Nous pouvons par exemple représenter la fonction objectif du problème précédent (P_1) :

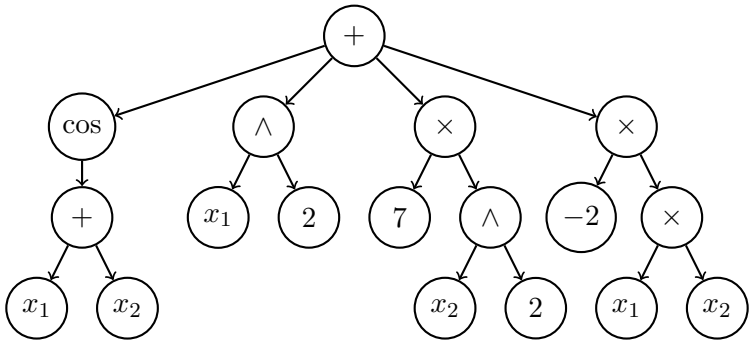


FIGURE 4.1 – Graphe de la fonction objectif du problème factorisable (P_1)

Définition 3. *Si la fonction objectif ainsi que toutes les contraintes sont factorisables, alors on dit que le problème est factorisable. Les graphes de fonctions ou de problèmes factorisables sont des graphes acycliques orientés (DAG).*

Remarque 6. *En général, la factorisation d'une fonction n'est pas unique.*

Comme nous l'avons indiqué précédemment, le second objectif est d'isoler les termes non convexes. Elle permet en fait de les ranger en deux catégories :

— La composée d'une variable ou d'une expression factorisable par une fonction univariée.

— Le produit de deux expressions de la première catégorie.

McCormick [79] propose une définition des fonctions factorisables, plus générale et implicite, qui correspond à une somme de produits de fonctions univariées continues, dont les arguments sont des variables de décision, des constantes ou d'autres fonctions factorisables. Il définit alors un programme factorisable, si nous pouvons poser $X_i = x_i,\ i \leq n$ (on conserve les variables de décisions initiales du problème) et définir des variables auxiliaires suivantes :

$$X_i(x) = \sum_{k=1}^{i-1} T^i[X_k(x)] + \sum_{k=1}^{i-1}\sum_{p=1}^{k} U^i_{p,k}[X_k(x)]\, V^i_{k,p}[X_p(x)],\ n < i \leq N$$

où T, U et V sont des fonctions univariées. Pour convexifier tous les problèmes factorisables, il faut et il suffit de savoir convexifier :

— Une fonction univariée réelle $\mathcal{C}^2$ composée avec une fonction continue de $\mathbb{R}^n$ dans $\mathbb{R}$.

— Un produit de deux fonctions de ce type.

Nous décrirons ces deux convexifications de manière générale, puis appliquées à notre problème $(PNS_{N,M})$, à la section 4.3. Pour résumer, la factorisation permet de transformer le problème initial en un problème équivalent dont les fonctions (objectifs et contraintes) sont des sommes de produits de fonctions univariées et dont les sources de non convexité se rangent en deux catégories, traitées de manière générique dans McCormick [79].

4.2 Factorisation du problème

Dans cette section, nous allons appliquer la technique de factorisation présentée à la section précédente au problème traité. Nous allons commencer par factoriser $(PNS_{N,M})$

dans le cas $N = 2$ à la section 4.2.1, puis le cas $N = 3$ à la section 4.2.2. Nous étendrons ensuite la factorisation au cas général à la section 4.2.3. Dans un contexte de maximisation, nous proposerons, dans chaque cas, un problème factorisé équivalent et identifierons les termes non concaves de la fonction objectif.

4.2.1 Factorisation pour le cas convexe à deux variables

Dans ce cas très simple, $x_1 + x_2 = M$ et la fonction objectif est naturellement univariée.

$$f(x_1) = [p_1 - c_1 \cdot g(x_1)]x_1 + [p_2 - c_2 \cdot g(M)](M - x_1)$$

La seule partie non linéaire est $-c_1 \cdot g(x_1)\, x_1$. Sous les hypothèses faîtes à la section 3.5 (g concave et $x \mapsto xg(x)$ convexe), la fonction objectif est une fonction concave et le problème $(PNS_{2,M})$ est convexe. Dans ce cas, il n'y a pas d'intérêt à factoriser et le problème se résout facilement par analyse réelle dans le cas continu et par DP dans le cas discret. Nous allons voir que ce cas fait figure d'exception, car à partir de $N \geq 3$, la fonction objectif comporte des termes non concaves.

4.2.2 Factorisation du produit bilinéaire pour le cas à trois variables

Le cas $(P_{3,M})$ offre plus d'intérêt car sa factorisation met en évidence un produit du type $X\, g(Y)$.

$$\begin{aligned} f(x) = \; & (p_1 - c_1\, g(x_1))\, x_1 \\ & + (p_2 - c_2\, g(x_1 + x_2))\, x_2 \\ & + (p_3 - c_3\, g(M))\, x_3 \end{aligned}$$

nous conservons les variables de décisions et introduisons 3 nouvelles variables auxiliaires :

$$\begin{aligned} & X_i = x_i,\ i = 1 \cdots 3 \\ & X_4 = X_1 + X_2 \\ & X_5 = X_1 + X_2 + X_3 = M \\ & X_6 = p_1\, X_1 - c_1\, g(X_1)\, X_1 + p_2\, X_2 - c_2\, g(X_4)\, X_2 + \Big(p_3 - c_3\, g(M)\Big)\, X_3 \end{aligned}$$

Pour simplifier les notations, nous poserons $g(M) \equiv H$.
La constante H correspond au seuil de calibration de la fonction de pénalité g. En pratique, nous prendrons $H = 0.99$ dans les expérimentations numériques. La justification et la méthode de calibration seront discutées à la section 5.1.5.

Le problème factorisé équivalent est le suivant :

$$\begin{cases} \max & & X_6 \\ X_4 & = & X_1 + X_2 \\ X_5 & = & X_1 + X_2 + X_3 = M \\ \forall i, & 0 \leq & X_i \leq M \\ \forall i, & & X_i \in \mathbb{N} \end{cases} \tag{4.1}$$

Comme nous l'avons remarqué à la section 4.1, cette factorisation n'est pas unique. L'équivalence des problèmes factorisés et initiaux se lit directement sur la définition de (4.1).
De plus, nous notons que les contraintes issues de la définition des variables auxiliaires sont linéaires (donc concaves).
Enfin, sous les hypothèses de la section 3.5, $x \mapsto -c_1\, g(x)\, x$ est une fonction concave, donc le seul terme non concave de f est le produit $-c_2\, X_2\, g(X_4)$, qui entre dans la seconde catégorie décrite à la section précédente 4.1. Il nous suffit de savoir majorer ce terme par une fonction concave pour en déduire un problème majorant de $(PNS_{3,M})$.

Par la suite, sauf mention contraire, on ne se référera plus qu'à la forme factorisée du problème. Nous pouvons maintenant passer à la factorisation dans le cas général.

4.2.3 Factorisation dans le cas général du problème $\left(PNS_{N,M}\right)$

La factorisation du problème $(PNS_{N,M})$ va simplement généraliser le traitement des termes linéaires et des termes produits, rencontrés à la section précédente pour le cas $N = 3$.
Nous posons, comme précédemment $X_i = x_i$ pour $1 \leq i \leq N$.
Le terme générique produit dans la fonction f s'écrit : $g(\sum_{k=1}^{i} x_k)\, x_i$
Nous définissons alors la variable auxiliaire qui correspond à la somme partielle de 1 à i, dans la fonction de pénalité g : $X_{N+i-1} = \sum_{k=1}^{i} x_k$
Nous procédons ainsi pour $2 \leq i \leq N-1$.
La dernière variable auxiliaire $X_{2N-1} = \sum_{k=1}^{N} x_k = M$ correspond à la somme totale de l'écoulement et on retrouve la contrainte initiale. Nous avons donc créé $N-1$ variables auxiliaires, soit $2N-1$ variables au total.
Nous avons également rajouté $N-2$ contraintes linéaires d'égalité, pour un total de $N-1$ contraintes.
Concernant les bornes, les variables initiales sont bornées par 0 et M. De plus, les variables

auxiliaires, qui sont des sommes partielles des variables initiales, sont évidemment positives et sont également majorées la somme les variables X_i de 1 à N qui est égale à M. Enfin, par convention, X_{2N} correspond à la fonction objectif.
Nous en déduisons finalement le problème factorisé suivant :

$$\begin{cases} \max\ X_{2N} = & \Big(\sum_{i=1}^{N} p_i\, X_i\Big) & -\Big(c_1\, g(X_1)\, X_1 + c_N\, g(M)\cdot X_N\Big) + \sum_{i=2}^{N-1} (-c_i)\cdot [X_i\, g(X_{N+i-1})] \\ X_i = & x_i & ,\ 1 \le i \le N \\ X_{N+i-1} = & \sum_{j=1}^{i} X_j, & 2 \le i \le N-1 \\ X_{2N-1} = & (\sum_{j=1}^{N} X_j) = & M \\ & 0 \le X_i \le M & ,1 \le i \le 2N-1 \\ & \forall i,\ X_i \in \mathbb{N} & \end{cases} \tag{4.2}$$

Nous avons plusieurs remarques utiles pour la suite sur le problème factorisé équivalent. Tout d'abord, dans le cas général, il y a $N-2$ facteurs non concaves de type produit.
De plus, comme nous l'avons mentionné à la section précédente, les $N-1$ contraintes d'égalité, sont linéaires.
Nous notons également les deux points suivants qui découlent des relations linéaires entre les variables initiales et auxiliaires :

Remarque 7. *Lorsque les N premières variables de décisions sont fixées, alors toutes les variables auxiliaires sont déterminées ainsi que la solution du problème factorisé.*

En effet, pour X_{N+i-1} est fixé lorsque les X_1 à X_i le sont et $X_{2N-1} = M$, pour tout $N \ge 3$.

Remarque 8. *Réciproquement, si les variables auxiliaires X_{N+i-1} sont fixées pour $i = 2$ à $N-2$, alors les variables initiales sont toutes déterminées, sauf X_1 et X_2 qui sont liées par $X_1 + X_2 = cst$ ($= X_4$ fixée). Il reste donc un unique degré de liberté au problème.*

Nous remarquons simplement que pour tout i, $2 \le i \le N-1$,

$$X_{N+i} - X_{N+i-1} = \sum_{k=1}^{i+1} X_k - \sum_{k=1}^{i} X_k = X_{i+1}$$

Donc X_3 à X_{2N-2} sont ainsi déterminées lorsque les variables auxiliaires sont fixées. $X_{2N-1} = M$, par définition, donc seules X_1 et X_2 ne sont pas déterminées ; mais elles

sont liées par $X_1 + X_2 = X_4$ (fixée), ce qui laisse un degré de liberté.
Ces remarques vont s'avérer utiles pour restreindre le domaine admissible après chaque branchement (cf. section 4.5).

Nous avons identifié les termes non concaves qui doivent être convexifiés pour obtenir une borne supérieure de $(PNS_{N,M})$. Nous présentons dans la prochaine section une méthode générale de convexification des deux seuls cas, discutés à la section 4.1, qui peuvent se présenter pour un problème factorisable.

4.3 Relaxation convexe

A la section précédente, nous avons introduit une méthode de transformation du problème, permettant d'en isoler les termes non convexes.

Dans cette section, nous nous intéressons à la relaxation convexe du problème $(PNS_{N,M})$.
Pour cela, à la section 4.3.1, nous commençons par des rappels généraux, qui font écho à la section 4.1, sur les problèmes convexes et les enveloppes convexes ou concaves.
Puis, nous rappelons à la section 4.3.2, les deux cas de non convexité pour les problèmes factorisables, également présentés à la section 4.1, que nous résolvons à l'aide des théorèmes et des inégalités de McCormick [79].
Nous proposons ensuite, à la section 4.3.3, une relaxation convexe dédiée au problème traité $(PNS_{N,M})$.
Enfin dans cette même section, nous proposons une relaxation convexe plus précise de $(PNS_{N,M})$ avec des contraintes de bornes indépendantes pour chaque variable de décision.
Cette dernière relaxation va jouer un rôle très important dans la résolution du problème via les algorithmes de *Branch & Bound* de la section 4.5.

4.3.1 Rappels de convexité

Dans cette section, nous rappelons ce qu'est un problème convexe et une enveloppe convexe (resp. concave) et nous nous intéressons au rôle de ces enveloppes dans le cadre de la relaxation convexe d'une fonction univariée.

Pour un PMVE, la notion de convexité n'est pas naturelle. Nous en donnons la définition suivante, extraite de Schmidt [99]. Nous considérons pour cela une fonction objectif

$f : \mathbb{R}^n \to \mathbb{R}$ et p fonctions de contraintes $c_j : \mathbb{R}^n \to \mathbb{R},\ j = 1 \cdots p$.

Définition 4. *Soit (P) un programme mathématique en variables entières défini par :*

$$(P) \begin{cases} \min\limits_{x \in \mathbb{R}^n} f(x) \\ c_j(x) \leq 0 \quad , \ j = 1 \cdots p \\ \forall i,\ x_i \in \mathbb{N} \end{cases}$$

P est un problème convexe si les fonctions f et c_j,pour tout j, sont convexes.

Un PMVE (P) est dit *convexe* si sa relaxation continue est un problème convexe d'optimisation, noté $(\overline{P})$.
En effet, l'ensemble des solutions admissibles de $(\overline{P})$ est $\mathcal{S} = \{x \in \mathbb{R}^n / \ \forall j, c_j(x) \leq 0\}$ est un ensemble convexe si c_j convexe pour tout j (la preuve vient directement de la définition de la convexité d'une fonction). De plus, f est une fonction convexe sur $\mathcal{S}$. Nous retrouvons donc bien la définition générale d'un problème convexe, pour un problème en variables continues.
Dans un contexte de maximisation, on propose, symétriquement, la définition suivante :

Définition 5. *Soit (P) un programme mathématique en variables entières défini par :*

$$(P) \begin{cases} \max\limits_{x \in \mathbb{R}^n} f(x) \\ c_j(x) \geq 0 \quad , \ j = 1 \cdots p \\ \forall i,\ x_i \in \mathbb{N} \end{cases}$$

Symétriquement, P est un problème convexe si les fonctions f et c_j, pour tout j, sont concaves.

Conformément à l'usage, nous ne parlerons, dans cette étude, que de problèmes et de relaxations convexes, pour simplifier le discours, sans perte de généralité.
Cependant, dans notre contexte de maximisation, nous chercherons une fonction majorante de l'objectif f qui soit concave. Une telle fonction sera appelée *surestimation concave* de f. Réciproquement dans un contexte de minimisation, on chercherait une sous-estimation convexe de f.
Nous introduisons donc les définitions suivantes, pour un ensemble convexe $\mathcal{S}$:

Définition 6. *Une fonction $\eta : \mathcal{S} \to \mathbb{R}$ est une surestimation concave de f sur $\mathcal{S}$, si :*

- *La fonction η est concave.*
- $\forall x \in \mathcal{S},\ f(x) \leq \eta(x)$

Définition 7. *Une fonction $\eta : \mathcal{S} \to \mathbb{R}$ est une sous-estimation convexe de f sur $\mathcal{S}$, si :*

- *La fonction η est convexe.*
- $\forall x \in \mathcal{S},\ f(x) \geq \eta(x)$

Nous considérons l'ensemble des surestimations concaves de f sur $\mathcal{S}$, noté usuellement $\mathcal{U}(f, \mathcal{S})$ (réciproquement, nous notons $\mathcal{L}(f, \mathcal{S})$ pour les sous-estimations convexes). Son infimum correspond à la plus petite fonction concave sur $\mathcal{S}$, qui majore f, donc à la meilleure (la plus fine) surestimation concave pour notre problème. Cette infimum est appelé *enveloppe concave* de f sur $\mathcal{S}$:

Définition 8. *La fonction définie pour tout x de $\mathcal{S}$ par : $E_f(\mathcal{S})(x) = \inf\{\eta(x), \eta \in \mathcal{U}(f, \mathcal{S})\}$ est appelée enveloppe concave de f sur $\mathcal{S}$.*

Symétriquement :

Définition 9. *La fonction définie pour tout x de $\mathcal{S}$ par : $e_f(\mathcal{S})(x) = \sup\{\eta(x), \eta \in \mathcal{L}(f, \mathcal{S})\}$ est appelée enveloppe convexe de f sur $\mathcal{S}$.*

En général, la caractérisation explicite de l'enveloppe (convexe ou concave) de f est une tache ardue. Remarquons tout d'abord que si f est convexe (resp. concave) sur $\mathcal{S}$, elle est sa propre enveloppe convexe (resp. concave).

De plus, la factorisation de la section 4.2 a simplifié notre tache, car nous n'aurons besoin d'utiliser que des fonctions d'une seule variable. Ceci découle directement de la définition des fonctions factorisables (cf. caractérisation section 4.1), qui s'écrivent comme somme ou produits de fonctions univariées réelles $\mathcal{C}^2$ composée avec des fonctions continues de $\mathbb{R}^n \to \mathbb{R}$. Enfin, la factorisation du problème traité (cf. section 4.2.3) a montré que les termes non convexes sont des produits du type $X \times g(Y)$, où g est une fonction univariée régulière (au moins $\mathcal{C}^2$) et X et Y sont directement des variables du problème.

Nous nous intéressons donc à la convexification d'une fonction univariée f réelle $\mathcal{C}^2$ qui est un problème beaucoup plus simple. Nous supposons f définie sur un segment réel $[L; U]$. Nous cherchons les intervalles de convexité monotone, sur lesquels f est strictement convexe ou concave. Ces intervalles se déterminent en analyse réelle en cherchant les points d'inflections de f, ce qui revient de manière équivalente à :

- Trouver les points où f'' s'annule en changeant de signe.
- Trouver les points où la tangente traverse la courbe :

  $\exists(a, x), \in [L; U],\ a < x, / \dfrac{f(x) - f(a)}{x - a} = f'(x)$

où a est un point donné de l'intervalle. La résolution analytique, suivant la forme de f, ou par des méthodes numériques, a fait l'objet d'une recherche extensive.
Nous proposons, comme McCormick [79], une résolution numérique simple à implémenter via la méthode de Newton :
Nous cherchons les zéros de la fonction $F(x) = f(x) - f(a) - (x-a)\,f'(x)$
On a : $F'(x) = f'(x) - (x-a)\,f''(x) - f'(x) = -(x-a)\,f''(x)$
La méthode de Newton introduit une suite $(x_k)_{k\in\mathbb{N}}$, par x_0 et $x_{k+1} = x_k - \dfrac{F(x_k)}{F'(x_k)}$
Dans notre cas cas la relation de récurrence revient à :

$$x_{k+1} = x_k + \frac{1}{f''(x_k)}\Big[\frac{f(x_k) - f(a)}{x_k - a} - f'(x_k)\Big]$$

Nous pouvons donc subdiviser $[L;U]$ en $[L = l_1; l_2; \cdots; l_i; \cdots l_k = L]$, tel que f soit de convexité monotone sur $[l_i; l_{i+1}]$ pour tout i.
Dans un contexte de maximisation, nous pouvons procéder de la manière suivante :

- Sur les intervalles concaves, f est sa propre enveloppe concave.
- Sur les intervalles convexes $[l_i, l_{i+1}]$, l'enveloppe concave correspond au segment qui relie les points $(l_i; f(l_i))$ et $(l_{i+1}; f(l_{i+1}))$.
  Ce point est discuté par Falk et Soland [46]. Horst et Tuy [61] donnent une interprétation géométrique de l'enveloppe convexe à l'aide de l'épigraphe de f en montrant que l'enveloppe convexe ϕ d'une fonction f à valeurs réelles, continue sur un compact de $\mathbb{R}^n$ vérifie : $epi(\phi) = conv(epi(f))$ (cf. figure 4.2).

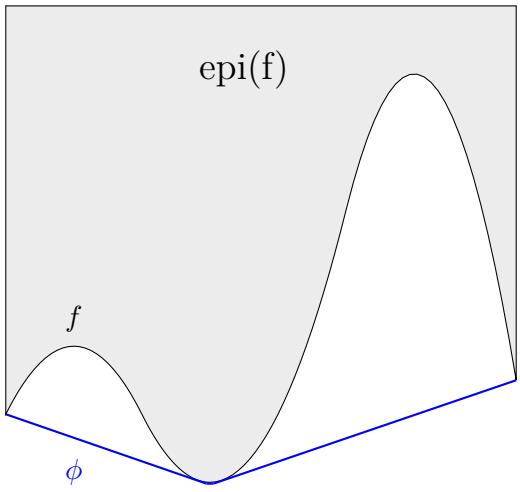


FIGURE 4.2 – Enveloppe convexe ϕ de f (cf. [61] p145)

Nous avons défini et caractérisé les enveloppes que nous allons utiliser pour convexifier le problème traité. Nous allons maintenant nous intéresser à la convexification des programmes factorisables en général et proposer une relaxation convexe spécifique pour notre problème.

4.3.2 Convexification des problèmes factorisables : cas général

Comme annoncé en début de chapitre, nous allons dans cette section nous intéresser à la convexification des programmes factorisables en général. Nous nous appuyons pour cela sur les travaux de McCormick [79]. Nous avons montré à la section 4.2.3 qu'il y avait deux sources de non convexité à traiter. Nous nous intéressons tout d'abord aux fonctions univariées, puis ensuite aux produits de fonctions univariées.
Nous rappelons tout d'abord quelques définitions, déjà évoquées dans les sections précédentes.

Définition 10. *Soit T une fonction de $\mathbb{R}$ dans $\mathbb{R}$ continue. On suppose que T est la fonction objectif d'un problème d'optimisation donné (P).*
Soit S un sous ensemble de $\mathbb{R}^n$ de l'ensemble des solutions admissibles de (P).
Soit t une fonction à n variables de $\mathbb{R}^n$ dans $\mathbb{R}$ continue.
Soit c une fonction à n variables de $\mathbb{R}^n$ dans $\mathbb{R}$ continue et convexe.
Soit C une fonction à n variables de $\mathbb{R}^n$ dans $\mathbb{R}$ continue et concave.
Soit $e_T(S)$ l'enveloppe convexe de T, la plus grande fonction convexe qui minore T sur S.
Soit $E_T(S)$ l'enveloppe concave de T, la plus petite fonction concave qui majore T sur S.
Enfin soit mid la fonction de $\mathbb{R}^3$ dans $\mathbb{R}$, qui à (x,y,z) renvoie la valeur intermédiaire du triplet.

Sous les hypothèses, suivantes :

$$\forall x \in S, c(x) \leq t(x) \leq C(x)$$
$$\exists\, , a_t, b_t \in \mathbb{R}, s.t \;\; a_t \leq t(x) \leq b_t$$
$$\exists\, z_T^{min}, A_T \in \mathbb{R}, s.t \inf_{a_t \leq z \leq b_t} T(z) = T(z_T^{min}) = A_T$$
$$\exists\, z_T^{max}, B_T \in \mathbb{R}, s.t \inf_{a_t \leq z \leq b_t} T(z) = T(z_T^{max}) = B_T$$

Une sous-estimation convexe (resp. surestimation concave) est donnée par le théorème de McCormick [79] :

Théorème 5. *Sous les hypothèses précédentes, pour tout* $x \in S \cap \{x \mid a_t \leq t(x) \leq b_t\}$, *nous avons :*

$$\begin{aligned} T\left[t(x)\right] \geq & \quad e_T\left[mid\Big(z_T^{min}, c(x), C(x)\Big)\right] \\ T\left[t(x)\right] \leq & \quad E_T\left[mid\Big(z_T^{max}, c(x), C(x)\Big)\right] \end{aligned}$$

Pour simplifier les notations :
Nous définissons la fonction g par : $g(x) = mid\Big(z_T^{min}, c(x), C(x)\Big)$.
Nous notons e_T (resp. E_T) au lieu de $e_T(S)$ (resp. $E_T(S)$) . La démonstration de ce théorème s'appuie sur la définition de la convexité d'une fonction et de l'enveloppe convexe.

Démonstration. On démontre la sous-estimation convexe. **1er cas :** $t(x) = z_T^{min}$
On a $c(x) \leq t(x) = z_T^{min} \leq C(x)$ donc $g(x) = t(x)$.
De plus par définition de l'enveloppe convexe :

$$\forall y, T(y) \geq e_T(y) \tag{4.3}$$

On applique (4.3) à $y = t(x)$. On obtient : $T \circ t(x) \geq e_T \circ t(x) = e_T \circ g(x)$.
2e cas : $c(x) \leq z_T^{min} < t(x)$
Dans ce cas $g(x) = z_T^{min}$. De plus $t(x) \neq z_T^{min}$ donc $T \circ t(x) \geq A_T = T(z_T^{min})$
Et on applique (4.3) à $y = z_T^{min}$. On obtient : $T \circ t(x) \geq e_T(z_T^{min}) = e_T \circ g(x)$.
3e cas : $z_T^{min} \leq c(x) \leq t(x)$
Dans ce cas, $g(x) = c(x)$. T est continue sur le segment $[a_t; b_t]$, donc $\exists\, y \in [a_t; b_t] \mid c(y) \leq t(y) = z_T^{min} \leq C(y)$
Donc $T \circ t(x) \geq T \circ t(y) \geq e_T \circ t(y)$
Et $T \circ t(x) \geq e_T \circ t(x)$
Soit λ un scalaire dans $[0; 1]$. On multiplie la première inégalité par λ et la seconde par $(1 - \lambda)$.
On en déduit : $T \circ t(x) \geq \lambda e_T \circ t(y) + (1 - \lambda)\, e_T \circ t(x)$
Par convexité de e_T :

$$T \circ t(x) \geq e_T\left[\lambda\, z_T^{min} + (1 - \lambda)\, t(x)\right] \tag{4.4}$$

Or par hypothèse $c(x)$ est dans le segment $[z_T^{min}; t(x)]$. Il existe donc un unique λ_0 tel que $c(x) = \lambda_0\, z_T^{min} + (1 - \lambda_0)\, t(x)$. On applique (4.4) en λ_0 et obtient $T \circ t(x) \geq e_T \circ c(x)$, ce qui prouve le résultat.
4e cas : $t(x) < z_T^{min} \leq C(x)$
Dans ce cas $g(x) = z_T^{min}$ et le traitement est analogue 2e cas. Enfin,

5e cas : $t(x) \leq C(x) \leq z_T^{min}$
Dans ce cas, $g(x) = C(x)$. Par continuité de T sur $[a_t; b_t]$, $\exists\ y \in [a_t; b_t] \mid t(y) = z_T^{min} \leq C(y)$
Par hypothèse, $C(x)$ appartient au segment $[t(x); z_T^{min}]$. De manière analogue au 3e cas, on retrouve (4.4) qu'on applique au scalaire λ_0 tel que $C(x) = \lambda_0\, z_T^{min} + (1 - \lambda_0)\, t(x)$. On en déduit $T \circ t(x) \geq e_T \circ C(x)$, ce qui achève la preuve.
Nous procéderons de la même façon pour prouver la sur-estimation concave.

□

Ce théorème fournit une sur-estimation concave du premier type de non convexité dans le cas général.
Cependant, pour le problème traité, la non convexité provient exclusivement des termes produits. Ce théorème, ne sera pas utilisé directement, mais représente une étape nécessaire pour la résolution dans le cas général.

Enfin, dans notre cas, les fonctions t, c, C correspondent à l'identité, car on travaille directement sur les variables de décisions ou leur composée par une fonction univariée. Donc il découle directement de la définition des enveloppes $\forall x \in [a; b]\ T(x) \leq E_T(x)$. Cependant, nous préférons rester dans le cas le plus général, afin de ne pas affaiblir le résultat.

Nous traitons maintenant le cas des produits de fonctions univariées bornées, grâce aux inégalités de McCormick [79], que nous rappelons :

Théorème 6. *Soit U et V deux fonctions réelles univariées sur un segment $[a; b]$, bornées par : $U_a \leq U(x) \leq U_b$ et $V_a \leq V(x) \leq V_b$. Alors les quatre inégalités suivantes sont vraies pour tout x de $[a; b]$:*

$$U.V \geq U\,V_b + U_b\,V - U_b\,V_b$$

$$U.V \geq U\,V_a + U_a\,V - U_a\,V_a$$

$$U.V \leq U_b\,V + U\,V_a - U_b\,V_a$$

$$U.V \leq U_a\,V + U\,V_b - U_a\,V_b$$

Et par suite :

$$U.V \geq \max\Big(U\,V_b + U_b\,V - U_b\,V_b\,,\, U\,V_a + U_a\,V - U_a\,V_a\Big)$$

$$U.V \leq \min\Big(U_b\,V + U\,V_a - U_b\,V_a\,,\, U_a\,V + U\,V_b - U_a\,V_b\Big)$$

La preuve est (de manière assez surprenante) très simple.

Démonstration. $\forall x \in [a;b], U_b - U(x) \geq 0$ et $V_b - V(x) \geq 0$
Donc $(U_b - U(x))\,(V_b - V(x)) \geq 0$
Pour simplifier les notations, on supprime l'indexation par x lorsqu'il n'y a pas d'ambiguité.
Et $U_b\,V_b - U\,V_b - U_b\,V + U\,V \geq 0$
On en déduit : $U.\,V \geq U\,V_b + U_b\,V - U_b\,V_b$
On procède de même : $(U_a - U)\,(V_a - V) \geq 0$, $(U_b - U)\,(V_a - V) \leq 0$, $(U_a - U)\,(V_b - V) \leq 0$, pour obtenir les trois autres inégalités. On prend le max des deux premières et le min des deux dernières.

□

Nous allons maintenant combiner ces deux résultats dans le corollaire suivant pour en déduire une sous-estimation convexe (resp. surestimation concave) du produit $U \circ u(x).\,V \circ v(x)$, U, V étant des fonctions univariées :

Corollaire 1. *Sous les hypothèses des deux théorèmes précédents :*

$$U(u(x)).\,V(v(x)) \geq \max\Big(e_U \circ g_u(x)\ V_b + U_b\,e_V \circ g_v(x) - U_b\,V_b,\ e_U \circ g_u(x)\,V_a + U_a\,e_V \circ g_v(x) - U_a\,V_a\Big)$$

Et

$$U(u(x)).\,V(v(x)) \leq \min\Big(U_b\,E_V \circ g_v(x) + E_U \circ g_u(x)\,V_a - U_b\,V_a\,,\ U_a\,E_V \circ g_v(x) + E_U \circ g_u(x)\,V_b - U_a\,V_b\Big)$$

Le corollaire ci-dessous fournit un résultat général. Nous rappelons que dans notre cas, u et v correspondent à l'identité, ce qui simplifie les expressions précédentes :

Corollaire 2. *Sous les hypothèses des deux théorèmes précédents :*

$$U(x).\,V(x) \geq \max\Big(e_U(x)\ V_b + U_b\,e_V(x) - U_b\,V_b,\ e_U(x)\,V_a + U_a\,e_V(x) - U_a\,V_a\Big)$$

Et

$$U(x).\,V(x) \leq \min\Big(U_b\,E_V(x) + E_U(x)\,V_a - U_b\,V_a\,,\ U_a\,E_V(x) + E_U(x)\,V_b - U_a\,V_b\Big)$$

Discussion sur le signe des bornes $U_a,\ U_b,\ V_a,\ V_b$ **:** Dans le cas d'une maximisation, nous cherchons à majorer $U\,V$ par une fonction concave séparée. Nous nous intéressons au membre de droite de l'inégalité correspondante dans le corollaire. Comme

le minimum de deux fonctions concaves est concave, il suffit que les fonctions sur lesquelles on prend le minimum soient toutes les deux concaves, pour que le membre de droite soit également une fonction concave. Si les bornes de U et de V sont toutes les quatre strictement positives, on conclut directement par le corollaire 2.
Le cas interéssant est lorsqu'une borne au moins est négative. Si l'on suppose, par exemple, $U_b < 0$, nous remarquons que comme $V(x) > e_V(x)$, par définition de l'enveloppe convexe, on a : $U_b\, V(x) < U_b\, e_V(x)$. Le membre de droite de cette inégalité est une fonction concave majorante de $U_b\, V$.
Dans le cas $U_b < 0$ *et* $U_a,\ U_b,\ V_a,\ V_b) \geq 0$, nous obtenons :
$U(x).\, V(x) \leq \min\left(\underline{U_b\, e_V(x)} + E_U(x)\, V_a - U_b\, V_a\ ,\ U_a\, E_V(x) + E_U(x)\, V_b - U_a\, V_b\right)$
qui est bien une surestimation concave du produit $U\, V$.

Par conséquent, nous pouvons, sans perte de généralité adapter le théorème 6 selon le signe des bornes $U_a,\ U_b,\ V_a,\ V_b$, afin d'utiliser soit l'enveloppe convexe (cas négatif) ou bien concave (cas positif) de la variable considérée, pour obtenir une surestimation concave du produit $U\, V$. Nous pouvons procéder de la même manière pour la sous-estimation convexe, dans le cadre d'une minimisation.

Nous avons rappelé les résultats de McCormick [79], dans le contexte de notre étude. Ils permettent de proposer une surestimation concave des problèmes factorisables. Nous avons donc désormais les ingrédients nécessaires à la relaxation convexe du problème traité $(PNS_{N,M})$ que nous abordons dans la section suivante.

4.3.3 Application au problème traité $\left(PNS_{N,M}\right)$

Nous proposons, dans cette section, une relaxation convexe dédiée au problème traité $(PNS_{N,M})$, en s'appuyant sur les résultats de convexification précédemment établis.
Nous étudions tout d'abord le cas $N = 3$, afin d'expliciter la convexification du terme produit.

Nous traitons ensuite le cas général, en intégrant des contraintes indépendantes de bornes. Cette formulation plus précise, va être utilisée comme fonction d'évaluation dans les algorithmes de *Branch & Bound* de la section 4.5.

Traitement du Cas $N = 3$

Considérons le problème $(PNS_{3,M})$:

$$(PNS_{3,M}) \begin{cases} \max & & X_6 \\ X_6 & = & p_1\,X_1 - c_1\,g(X_1)\,X_1 + p_2\,X_2 - c_2\,g(X_4)\,X_2 + \Big(p_3 - c_3\,g(M)\Big)\,X_3 \\ X_4 & = & X_1 + X_2 \\ X_5 & = & X_1 + X_2 + X_3 = M \\ \forall i, & 0 \leq & X_i \leq M \\ \forall i, & & X_i \in \mathbb{N} \end{cases} \tag{4.5}$$

Dans le cas général de la section 4.3.1, on décomposerait le segment $[0; M]$ en intervalles de convexité monotone pour la fonction g.

Nous en déduirions une subdivision $[l_0 = 0, a_1, \cdots, l_n = M]$, telle que g est strictement convexe, ou concave sur $[l_i; l_{i+1}]$ pour tout $i \leq n - 1$.

Sur les intervalles où g est concave, elle est sa propre enveloppe concave et sur les intervalles où elle est convexe, son enveloppe concave correspond à l'approximation affine $\alpha_i\,x + \beta_i$ qui passe par les points $A(l_i, g(l_i))$ et $B(l_{i+1}, g(l_{i+1}))$, ce qui permet de déterminer α et β de manière unique.

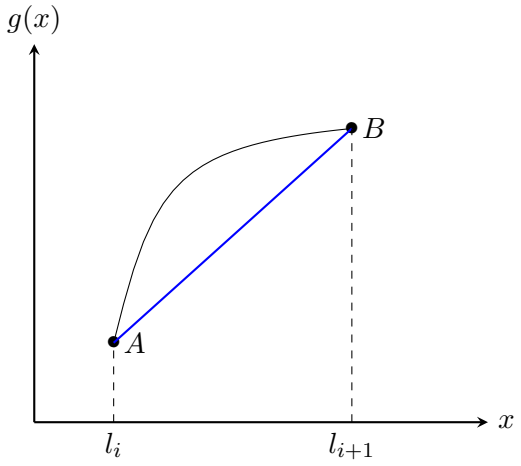


FIGURE 4.3 – Enveloppe convexe d'une fonction concave (cf. [46])

Dans notre cas, g est concave, donc de convexité monotone sur $[0; M]$ donc sur tous les intervalles $[a; b] \subset [0; M]$.

Nous rappelons $g(M) = H$, la constante de calibration ($H = 0.99$ dans les expérimenta-

tions numériques) Nous définissons la quantité :

$$A_2 \equiv -c_2\, X_2\, g(X_4)$$

où $X_4 = X_1 + X_2$ et $c_2 > 0$. Nous posons :

$$U = X_2 \;\text{ et }\; V = -g(X_4)$$

Nous sommes donc dans le cadre d'application du théorème 6 qui fournit une surestimation concave du produit $X_2\, g(X_4)$:
Sous les hypothèses de bornes suivantes :

$$\begin{array}{l} U_a = 0 \leq X_2 \leq U_b = M \\ V_a = -g(M) = -H \leq -g(X_4) \leq -g(0) = 0 = V_b \\ A_2 \leq c_2\, \min\Big(- M\, g(X_4) - X_2\, H + M\, H, -0 \cdot g(X_4) + X_2 \cdot 0 - 0 \cdot 0\Big) \\ A_2 \leq c_2\, \min\Big(0, -M\, g(X_4) - H\, X_2 + M\, H\Big) \end{array}$$

Par application du corollaire 2, nous obtenons la surestimation concave suivante :

$$A_2 \leq c_2\, \min\Big(- U_b\, g(X_4) + X_2\, V_a - U_b\, V_a, -U_a\, g(X_4) + X_2\, V_b - U_a\, V_b\Big)$$

Nous sommes dans le cas discuté à la section 4.3.2, avec une borne strictement négative $V_a < 0$. Cependant cette borne est multiplié par un terme linéaire dans l'inégalité de McCormick. Donc $V_a\, X_2$ reste linéaire.
Par ailleurs, $-g$ est convexe, donc son enveloppe concave correspond au segment passant par $(0, -g(0))$ et $(M, -H)$ Nous obtenons $-g(X_4) \leq \alpha \cdot X_4 + \beta$. Nous remarquons que (α, β) dépendent de X_4 et des bornes 0 et M.
Cependant, nous ne matérialiserons pas ces dépendances pour alléger les notations.
Par calcul direct, il vient $-g(0) = 0 = \beta$ et $\alpha = \dfrac{-g(M) + g(0)}{M} = -\dfrac{H}{M}$.

Nous en déduisons une surestimation concave du terme A_2. :

$$\begin{array}{l} A_2 \leq c_2\, \min\Big(0, -H\, X_4 - H\, X_2 + M\, H\Big) \\ A_2 \leq c_2\, H\, \min\Big(0, M - X_2 - X_4\Big) \end{array}$$

Par conséquent, nous obtenons la surestimation concave de f suivante :

$$f(x) \leq \sum_{i=1}^{3} p_i\, X_i - c_1\, X_1\, g(X_1) - c_3\, H\, X_3 + c_2\, H\, \min\Big(0, M - X_2 - X_4\Big) \tag{4.6}$$

Nous procédons de même pour le terme $A_1 \equiv -c_1\, X_1\, g(X_1)$. Nous en déduisons symétriquement :

$$A_1 \leq c_1\, H\, \min\Big(0, M - 2X_1\Big)$$

Nous obtenons finalement :

$$f(x) \leq \sum_{i=1}^{3} p_i\, X_i - c_3\, H\, X_3 + c_1\, H\, \min\Big(0, M - 2X_1\Big) + c_2\, H\, \min\Big(0, M - X_2 - X_4\Big) \quad (4.7)$$

Nous supposerons cependant, comme dans le chapitre 3, section 3.5 , que la fonction $x \mapsto x\, g(x)$ est convexe. Dans ce cas l'équation précédente 4.6 fournit une surestimation concave de f. Nous définissons ainsi $(PRC_{3,M})$ (problème relaché convexe) :

$$(PRC_{3,M}) \begin{cases} \max & & X_6 \\ X_6 & = & \displaystyle\sum_{i=1}^{3} p_i\, X_i - c_1\, X_1\, g(X_1) - c_3\, H\, X_3 + c_2\, H\, \min\Big(0, M - X_2 - X_4\Big) \\ X_4 & = & X_1 + X_2 \\ X_5 & = & X_1 + X_2 + X_3 = M \\ \forall i, & 0 \leq & X_i \leq M \\ \forall i, & & X_i \in \mathbb{N} \end{cases} \quad (4.8)$$

$(PRC_{3,M})$ est un problème non linéaire convexe en variables entières, sous contraintes linéaires d'égalité. Sa résolution fournit une borne supérieure de $(PNS_{3,M})$.
La relaxation convexe être résolue, soit directement par un solver, ou bien après une étape de linéarisation du minimum que nous abordons à la section 4.4.

Cependant, en pratique, cette borne supérieure n'est pas fine, car la majoration de $g(X_4)$ par un segment sur tout $[0; M]$, avec M grand, est trop brutale.
Il conviendra donc de restreindre l'intervalle de majoration de X_4 pour obtenir une borne plus fine. C'est pour cela que nous proposons de traiter le cas général, non pas directement sur $[0; M]^{2N-2}$, mais avec des contraintes de bornes indépendantes.
Comme discuté en introduction, cette flexibilité est requise, car la surestimation concave ainsi obtenue, va servir de fonction d'évaluation dans le cadre d'un algorithme de *Branch & Bound*.

Traitement du cas général : nous nous intéressons donc à $(PNS_{N,M})$ avec des contraintes de bornes indépendantes pour chaque variable de décision :

$$\begin{cases} \max\ X_{2N} = & \Big(\sum_{i=1}^{N} p_i\, X_i\Big) - \Big(c_1\, g(X_1)\, X_1 + c_N\, g(M)\cdot X_N\Big) & + \sum_{i=2}^{N-1} (-c_i)\cdot [X_i\, g(X_{N+i-1})] \\ X_i = & x_i & ,\ 1 \le i \le N \\ X_{N+i-1} = & \sum_{j=1}^{i} X_j, & 2 \le i \le N-1 \\ X_{2N-1} = & (\sum_{j=1}^{N} X_j) = & M \\ & 0 \le L_i \le X_i \le U_i \le M & ,1 \le i \le N \\ & 0 \le a_i \le X_{N+i-1} \le b_i \le M & ,2 \le i \le N-1 \\ & \forall i,\ X_i \in \mathbb{N} & \end{cases}$$

Par rapport à $(PNS_{N,M})$ standard, nous ajoutons les contraintes de bornes suivantes :

$$\begin{cases} 0 \le L_i \le X_i \le U_i \le M \\ 0 \le a_i \le X_{N+i-1} \le b_i \le M,\ avec\ a_i < b_i \end{cases}$$

Nous allons établir une relaxation convexe de problème en généralisant l'approche du cas $N = 3$. Nous définissons les termes produits non concaves :

$$A_i = -c_i\, X_i\, g(X_{N+i-1}),\ \ pour\ \ 2 \le i \le N-1$$

Remarque 9. *Le cas $a_i = b_i$ correspond au cas où la variable X_{N+i-1} est fixée et le terme A_i devient linéaire. On supposera donc $a_i < b_i$, pour tout i dans la suite.*

Comme précédemment, les bornes dépendent du segment et de la variable considérés. Nous ré-appliquons le théorème 6 et le corollaire 2 de la section 4.3.2 : Sous les hypothèses de bornes suivantes :

$$\begin{cases} 0 \le L_i \le X_i \le U_i \le M \\ 0 \le a_i \le X_{N+i-1} \le b_i \le M,\ avec\ a_i < b_i \\ -H \le \underbrace{-g(b_i)}_{V_{a_i}} \le -g(X_{N+i-1}) \le \underbrace{-g(a_i)}_{V_{b_i}} \le 0 \\ V_{a_i} = -g(b_i) \\ V_{b_i} = -g(a_i) \end{cases}$$

Nous sommes dans le cas où deux bornes V_{a_i}, V_{b_i} sont négatives, mais elles s'appliquent au terme linéaire X_i.
Nous majorons à nouveau $-g(X_{N+i-1})$ par son enveloppe concave, qui correspondent au

segments passants par les bornes de l'intervalle considéré.

$$\begin{aligned} A_i &\leq c_i \min\Big(-U_i\, g(X_{N+i-1}) + X_i\, V_{a_i} - U_i\, V_{a_i}, -L_i\, g(X_{N+i-1}) + X_i\, V_{b_i} - L_i\, V_{b_i}\Big) \\ &\leq c_i \min\Big(U_i\,[\alpha_i\, X_{N+i-1} + \beta_i] + X_i\, V_{a_i} - U_i\, V_{a_i}, L_i\,[\alpha_i\, X_{N+i-1} + \beta_i] + X_i\, V_{b_i} - L_i\, V_{b_i}\Big) \\ &\leq c_i \min\Big(U_i\, \alpha_i\, X_{N+i-1} + X_i\, V_{a_i} + U_i\,(\beta_i - V_{a_i}), L_i\, \alpha_i\, X_{N+i-1} + X_i\, V_{b_i} + L_i\,(\beta_i - V_{b_i})\Big) \end{aligned}$$

où α_i et β_i sont déterminés par :

$$\begin{cases} \alpha_i = \dfrac{g(a_i) - g(b_i)}{b_i - a_i} < 0 \\ \\ \beta_i = \dfrac{a_i \cdot g(b_i) - b_i \cdot g(a_i)}{b_i - a_i} \end{cases}$$

Nous obtenons alors une surestimation concave de la fonction objectif f du problème factorisé :

$$\begin{aligned} f(x) \leq \quad & \Big[\Big(\sum_{i=1}^{N} p_i\, X_i\Big) \qquad -\Big(c_1\, g(X_1)\, X_1 + c_N\, g(M) \cdot X_N\Big) \\ & + \sum_{i=2}^{N-1} c_i \cdot \min\Big(\quad U_i\, \alpha_i\, X_{N+i-1} - g(b_i)\, X_i - U_i\, b_i\, \alpha_i\ , \\ & \qquad\qquad\qquad\qquad L_i\, \alpha_i\, X_{N+i-1} - g(a_i)\, X_i - L_i\, a_i\, \alpha_i\Big)\Big] \end{aligned}$$

Par conséquent, nous concluons avec le problème majorant suivant, noté $(PRC_{N,M})$:

$$\begin{cases} \max\ X_{2N} = & \Big[\Big(\sum_{i=1}^{N} p_i\, X_i\Big) - \Big(c_1\, g(X_1)\, X_1 + c_N\, g(M) \cdot X_N\Big) \\ + \sum_{i=2}^{N-1} c_i \cdot \min\Big(& U_i\, \dfrac{g(a_i) - g(b_i)}{b_i - a_i}\, X_{N+i-1} - g(b_i)\, X_i + U_i\, b_i \dfrac{g(b_i) - g(a_i)}{b_i - a_i}, \\ & L_i\, \dfrac{g(a_i) - g(b_i)}{b_i - a_i}\, X_{N+i-1} - g(a_i)\, X_i + L_i\, a_i \dfrac{g(b_i) - g(a_i)}{b_i - a_i}\Big)\Big] \\ X_i = & x_i,\ 1 \leq i \leq N \\ X_{N+i-1} = & \sum_{j=1}^{i} X_j,\ 2 \leq i \leq N-1 \\ X_{2N-1} = & (\sum_{j=1}^{N} X_j) = M \\ 0 \leq & L_i \leq X_i \leq U_i \leq M, 1 \leq i \leq N \\ 0 \leq & a_i \leq X_i \leq b_i \leq M, N+1 \leq i \leq 2N-2 \\ \forall i, & X_i \in \mathbb{N} \end{cases} \tag{4.9}$$

Nous avons donc obtenu une relaxation convexe du problème $(PNS_{N,M})$ dans le cas général, avec des contraintes de bornes indépendantes.
Nous remarquons également qu'une solution admissible de $(PRC_{N,M})$ est admissible pour $(PNS_{N,M})$ et en fournit également une borne inférieure.

Nous concluons cette section en appliquant cette relaxation convexe à $(PNS_{N,M})$, sans contrainte de borne pour $\forall i,\ X_i \in [0; M]$.
Cela revient à utiliser 4.9 avec les identifications suivantes :

$$\begin{cases} \forall i, & \\ a_i &= L_i = 0 \\ b_i &= U_i = M \\ V_{a_i} &= -H \\ V_{b_i} &= 0 \end{cases} \tag{4.10}$$

Nous en déduisons finalement :

$$\begin{cases} \max\ X_{2N} = & \Big(\sum_{i=1}^{N} p_i\, X_i\Big) - \Big(c_1\, g(X_1) \quad X_1 + c_N\, g(M) \cdot X_N\Big) + H \cdot \sum_{i=2}^{N-1} c_i\, \min\left(0, M - X_{N+i-1} - X_i\right) \\ X_i = & x_i,\ 1 \leq i \leq N \\ X_{N+i-1} = & \sum_{j=1}^{i} X_j,\ 2 \leq i \leq N-1 \\ X_{2N-1} = & (\sum_{j=1}^{N} X_j) = M \\ 0 \leq & \text{Ł}_i \leq X_i \leq U_i \leq M \qquad , 1 \leq i \leq N \\ 0 \leq & a_i \leq X_i \leq b_i \leq M \qquad , N+1 \leq i \leq 2N-2 \\ \forall i, & X_i \in \mathbb{N} \end{cases} \tag{4.11}$$

Comme pour le cas $N = 3$, cette relaxation convexe, ne fournit pas directement une bonne borne supérieure. Nous allons donc utiliser des algorithmes de *Branch & Bound* que nous décrivons à la section 4.5, afin de résoudre le problème à l'optimum.

4.4 Linéarisation du minimum

Dans cette section, nous allons chercher à linéariser l'expression $\min(A, B)$ dans la relaxation convexe, afin de simplifier le problème et améliorer les performances de résolution par un solver.
Nous nous appuyons sur la technique classique explicitée dans l'ouvrage de Billionnet [24] p202, appliquée au minimum :

Théorème 7. *Soit* $\Big(U_1(x),\cdots,U_k(x)\Big)$, *k fonctions bornées sur* $S \subset \mathbb{R}^N$ *par* $U_{A_i} \leq U_i(x) \leq U_{B_i}$.
Pour tout e réel, $e = \max_i \Big(U_1(x),\cdots,U_k(x)\Big)$, *si et seulement si il existe* $y \in \{0;1\}^k$, *tel que les contraintes suivantes soient vérifiées :*

$$\left\{\begin{array}{rl} e & \geq U_i(x), \quad \forall i\ 1 \leq i \leq k \\ e - \Big[\max_{i=1\cdots k} U_{B_i} - U_{A_i}\Big] y_i & \leq U_i(x), \quad \forall i\ 1 \leq i \leq k \\ \sum_{i=1}^k y_i & = k-1 \end{array}\right. \tag{4.12}$$

On peut facilement l'adapter au minimum de k fonctions :

Corollaire 3. *Soit* $\Big(U_1(x),\cdots,U_k(x)\Big)$, *k fonctions bornées sur* $S \subset \mathbb{R}^N$ *par* $U_{A_i} \leq U_i(x) \leq U_{B_i}$.
Pour tout e réel, $e = \min_i \Big(U_1(x),\cdots,U_k(x)\Big)$, *si et seulement si il existe* $y \in \{0;1\}^k$, *tel que les contraintes suivantes soient vérifiées :*

$$\left\{\begin{array}{rl} e & \leq U_i(x), \quad \forall i\ 1 \leq i \leq k \\ e - \Big[\min_{i=1\cdots k} U_{A_i} - U_{B_i}\Big] y_i & \geq U_i(x), \quad \forall i\ 1 \leq i \leq k \\ \sum_{i=1}^k y_i & = k-1 \end{array}\right. \tag{4.13}$$

Démonstration. On va prouver le corollaire. La preuve du théorème est complètement symétrique et peut être trouvée directement dans Billionnet [24].
On suppose que $e = \min_i \Big(U_1(x),\cdots,U_k(x)\Big)$. Par définition, $\forall i, e \leq U_i(x)$.
De plus, il existe un indice j tel que $e = U_j(x)$. On définit alors y par $y_j = 0$ et $\forall i \neq j$, $y_i = 1$. Par définition de y, $\sum_{i=1}^k y_i = k-1$
Enfin $e \geq U_j(x)$ et pour tout i différent de j :

$$\begin{aligned} E_i &= e - \Big[\min_{i=1\cdots k} U_{A_i} - U_{B_i}\Big] y_i - U_i(x) \\ &= \Big[\underbrace{e - \min_{i=1\cdots k} U_{A_i}}_{\geq 0}\Big] + \Big[\underbrace{U_{B_i} - U_i(x)}_{\geq 0}\Big] \\ &\geq 0 \end{aligned}$$

Réciproquement, on suppose qu'il existe y, vérifiant les inégalités du corollaire. Pour tout i, $e \leq U_i(x)$. Donc $e \leq \min_{i=1\cdots k} U_i(x)$ De plus, $\sum_i yi = k-1$, donc il existe un unique j tel que $y_j = 0$. En j, on obtient $e \geq U_j(x)$.

Donc $e \geq \min_{i=1\cdots k} U_i(x)$, ce qui achève la preuve.

□

Nous allons appliquer ce résultat au problème $(PRC_{N,M})$
Pour cela, nous fixons tout d'abord i et introduisons le terme E_i défini par :

$$E_i = \min\Big(U_i\,\alpha_i\,X_{N+i-1} - g(b_i)\,X_i - U_i\,b_i\,\alpha_i\ ,\ L_i\,\alpha_i\,X_{N+i-1} - g(a_i)\,X_i - L_i\,a_i\,\alpha_i\Big)$$

$\alpha_i < 0$, donc :

$$\Big(U_i\,\alpha_i\,b_i - g(b_i)\,U_i\Big) \leq \Big(U_i\,\alpha_i\,X_{N+i-1} - g(b_i)\,X_i\Big) \leq \Big(U_i\,\alpha_i\,a_i - g(b_i)\,L_i\Big)$$

Et par symétrie des indices, nous obtenons :

$$\Big(L_i\,\alpha_i\,b_i - g(a_i)\,U_i\Big) \leq \Big(L_i\,\alpha_i\,X_{N+i-1} - g(a_i)\,X_i\Big) \leq \Big(L_i\,\alpha_i\,a_i - g(a_i)\,L_i\Big)$$

Par application du corollaire 3, il existe des variables binaires $Y_{1,i}$ et $Y_{2,i}$ telles que :

$$\left\{\begin{array}{cll}
E_i & \leq U_i\,\alpha_i\,X_{N+i-1} - g(b_i)\,X_i - U_i\,b_i\,\alpha_i \\
E_i & \leq L_i\,\alpha_i\,X_{N+i-1} - g(a_i)\,X_i - L_i\,a_i\,\alpha_i \\
E_i - \Big[\gamma_i - \Big(U_i\,\alpha_i\,a_i - g(b_i)\,L_i - U_i\,b_i\,\alpha_i\Big)\Big]\,Y_{1,i} & \geq U_i\,\alpha_i\,X_{N+i-1} - g(b_i)\,X_i - U_i\,b_i\,\alpha_i \\
E_i - \Big[\gamma_i - \Big(L_i\,\alpha_i\,a_i - g(a_i)\,L_i - L_i\,a_i\,\alpha_i\Big)\Big]\,Y_{2,i} & \geq L_i\,\alpha_i\,X_{N+i-1} - g(a_i)\,X_i - L_i\,a_i\,\alpha_i \\
Y_{1,i} + Y_{2,i} & = 1 \\
\text{avec } \gamma_i = \min\Big(U_i\,\alpha_i\,b_i - g(b_i)\,U_i - U_i\,b_i\,\alpha_i\ ,\ L_i & ,\alpha_i\,b_i - g(a_i)\,U_i - L_i\,a_i\,\alpha_i\Big)
\end{array}\right. \tag{4.14}$$

Nous pouvons alors linéariser le problème $(PRC_{N,M})$ en appliquant le corollaire 3. Nous en déduisons le problème linéaire $(PL_{N,M})$ en variables mixtes suivant :

$$
\left\{
\begin{array}{rc}
\max\ X_{2N} = & \left[\Big(\sum_{i=1}^{N} p_i\, X_i\Big) - \Big(c_1\, g(X_1)\, X_1 + c_N\, g(M)\cdot X_N\Big) + \sum_{i=2}^{N-1} c_i\, E_i\right] \\
X_i = & x_i,\ 1 \leq i \leq N \\
X_{N+i-1} = & \sum_{j=1}^{i} X_j,\ 2 \leq i \leq N-1 \\
X_{2N-1} = & (\sum_{j=1}^{N} X_j) = M \\
\\
Pour & 2 \leq i \leq N-1 : \\
E_i \leq & U_i\, \alpha_i\, X_{N+i-1} - g(b_i)\, X_i - U_i\, b_i\, \alpha_i, \\
E_i \leq & L_i\, \alpha_i\, X_{N+i-1} - g(a_i)\, X_i - L_i\, a_i\, \alpha_i \\
E_i \geq & U_i\, \alpha_i\, X_{N+i-1} - g(b_i)\, X_i - U_i\, b_i\, \alpha_i + \lambda_{1,i}\, Y_{1,i} \\
E_i \geq & L_i\, \alpha_i\, X_{N+i-1} - g(a_i)\, X_i - L_i\, a_i\, \alpha_i + \lambda_{2,i}\, Y_{2,i} \\
& Y_{1,i} + Y_{2,i} = 1 \\
\\
Avec\ \alpha_i = & \dfrac{g(a_i) - g(b_i)}{b_i - a_i} < 0 \\
\gamma_i = & \min\Big(U_i\, \alpha_i\, b_i - g(b_i)\, U_i - U_i\, b_i\, \alpha_i\ ,\ L_i, \alpha_i\, b_i - g(a_i)\, U_i - L_i\, a_i\, \alpha_i\Big) \\
\lambda_{1,i} = & \gamma_i - \Big(U_i\, \alpha_i\, a_i - g(b_i)\, L_i - U_i\, b_i\, \alpha_i\Big) \\
\lambda_{2,i} = & \gamma_i - \Big(L_i\, \alpha_i\, a_i - g(a_i)\, L_i - L_i\, a_i\, \alpha_i\Big) \\
& 0 \leq L_i \leq X_i \leq U_i \leq M \quad , 1 \leq i \leq N \\
& 0 \leq a_i \leq X_i \leq b_i \leq M \quad , N+1 \leq i \leq 2N-2 \\
\forall i, & X_i \in \mathbb{N} \\
\forall i, & Y_{1,i}, Y_{2,i} \in \{0;1\}
\end{array}
\right. \tag{4.15}
$$

Le problème $(PL_{N,M})$ fournit une relaxation linéaire fine de $(PNS_{N,M})$ qui peut être utilisée comme fonction d'évaluation dans un algorithme de *Branch & Bound*.

Cependant, $(PL_{N,M})$ contient beaucoup plus de variables et de contraintes que le problème initial. Il se compose en effet, de $2N-1$ variables X_i, $N-2$ variables E_i et $2(N-2)$ variables $Y_{k,i}$, soit un total de $5\,N-7$ variables de décisions.
Nous dénombrons également $N-1$ contraintes sur les X_i, $4(N-2)$ sur les E_i et $(N-2)$ sur les Y_i, pour un total de $(6N-11)$ contraintes.

Par conséquent, lorsque N devient grand, $(PL_{N,M})$ est composé d'environ $5N$ variables

de décision et $6N$ contraintes, bien que le problème initial ne contienne que N variables et une seule contrainte. Le gain de complexité obtenu grâce à la linéarisation est partiellement compensé par l'augmentation de la dimension du problème relâché linéaire, à N fixé.

Cependant, dans les expérimentations numériques, nous nous sommes restreints à la formulation non linéaire de $(PRC_{N,M})$ (cf. 4.9). Nous laissons l'impact de la linéarisation comme une perspective de notre étude.

Nous avons maintenant établi une relaxation convexe fine de notre problème $(PNS_{N,M})$. Nous allons l'utiliser comme fonction d'évaluation, dans les algorithmes *Branch & Bound* que nous présentons à la section suivante.

4.5 Algorithmes de *Branch & Bound*

Les sections précédentes ont permis d'établir une relaxation convexe du problème initial. Cependant, cette relaxation sur tout l'ensemble admissible du problème ne renvoie pas une borne supérieure fine car la majoration affine des termes produit n'est pas assez précise.

Une idée naturelle et assez classique pour un PNVE non convexe est d'utiliser un algorithme de *Branch & Bound*, dans lequel la relaxation convexe du problème sert de fonction d'évaluation (jouant ainsi le rôle de la relaxation continue en optimisation convexe en variables entières). L'état de l'art sur ce type d'algorithme, depuis leur introduction par Land et Doig [70] en 1960, est extensive.

Nous en rappelons cependant le principe général dans un contexte de maximisation. Nous parcourons une liste de problèmes $\mathcal{L}$, qui est initialisée avec le problème de départ P_0. Nous initialisons également une variable de borne inférieure (qui peut , par exemple provenir d'une bonne solution admissible, fournie par une heuristique ou à défaut $-\infty$).

A chaque étape, nous en sélectionnons un problème P que nous supprimons de $\mathcal{L}$. Si la relaxation convexe de P est infaisable ou que sa valeur est inférieure à la borne inférieure connue par l'algorithme, alors il n'y a rien à faire car ce nœud ne contient pas l'optimum global et ne doit donc pas être exploré.

Sinon :

1. Nous stockons la borne supérieure et la solution correspondante.

2. Si nous disposons d'une solution admissible pour P, calculée à partir de celle de sa relaxation convexe, nous mettons à jour la borne inférieure de l'algorithme, si elle est moins bonne que celle correspondante à la valeur de la solution admissible de P.

3. Nous sélectionnons une variable x_i et un point d'attachement K entier et créons les deux problèmes fils P_1 et P_2 qui reprennent les contraintes de P avec une contrainte supplémentaire, $x_i \leq K$ pour P_1 et $x_i \geq K + 1$ pour P_2.

4. Nous rajoutons enfin P_1 et P_2 à $\mathcal{L}$.

L'algorithme s'arrête lorsque $\mathcal{L}$ est vide ou qu'une condition de sortie (e.g une limite de temps, d'itérations, ou un critère de précision sont atteints) est déclenchée.

— Si $\mathcal{L}$ est vide alors l'optimum global qui correspond à la meilleure borne inférieure, est retournée, ainsi que les différentes solutions admissibles qui l'atteigne.

— Si $\mathcal{L}$ n'est pas vide, l'algorithme fournit une une borne inférieure pour P_0. De plus, le maximum des valeurs de relaxation convexe des éléments $\mathcal{L}$ en fournit une borne supérieure. Nous obtenons alors un encadrement de l'optimum global.

La démarche précédente peut être résumée dans le schéma suivant :

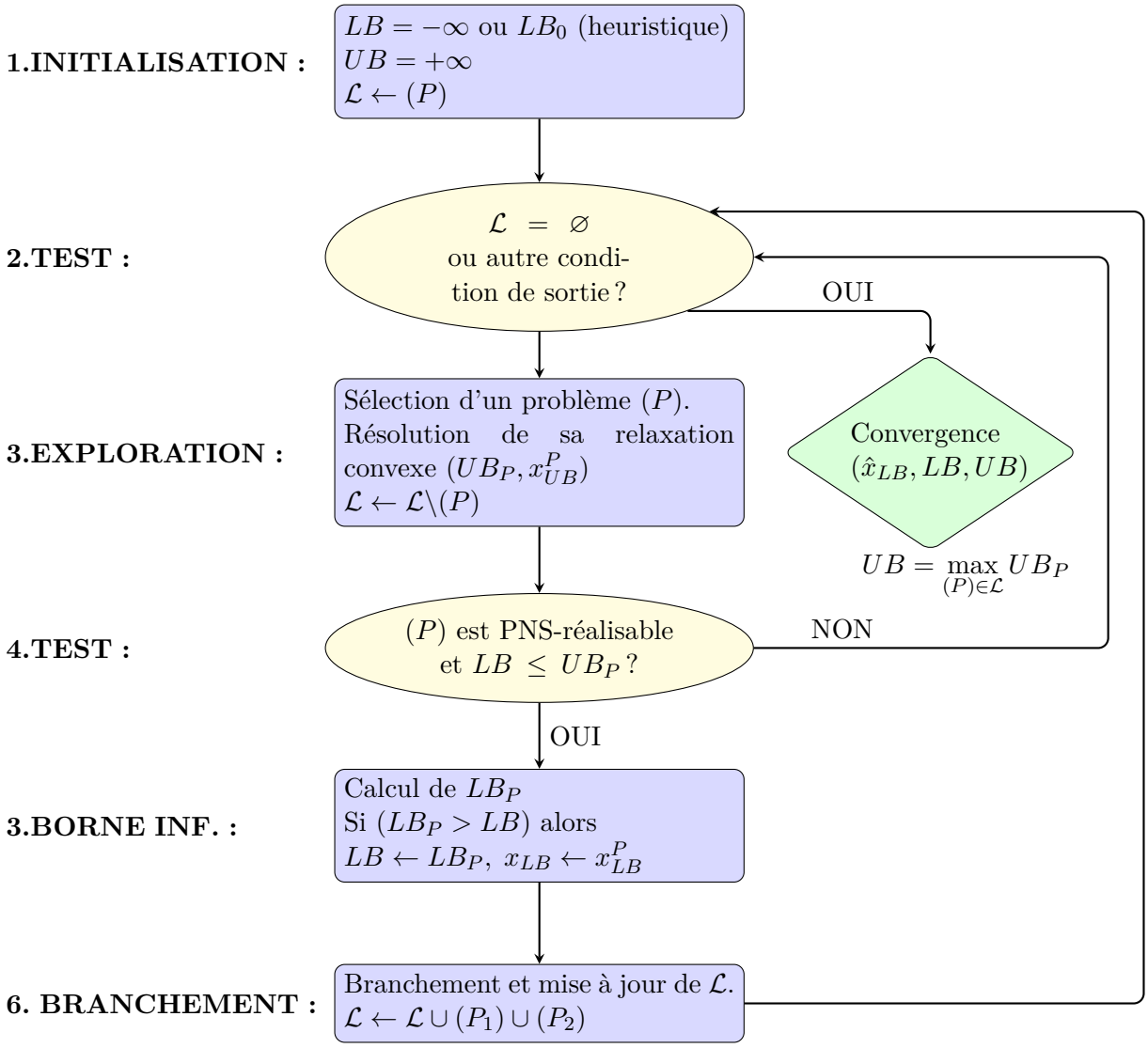


FIGURE 4.4 – Schéma de l'algorithme générique de *Branch & Bound*

Nous présentons également le pseudo code de l'algorithme générique :

Algorithm 4: algorithme de *Branch & Bound*

Initialisation $\mathcal{L} \leftarrow P_0$; $LB \leftarrow -\infty$; $UB \leftarrow +\infty$; $x_{LB} = \vec{0}$
while $\mathcal{L} \neq \emptyset$ *ou autre condition de sortie non déclenchée* **do**
  Exploration : sélection du problème $(P) \in \mathcal{L}$
  $\mathcal{L} \leftarrow \mathcal{L} \backslash (P)$
  Calcul de la relaxation convexe de (P) : couple (valeur, solution)= (UB_P, x^P_{UB})
  if (P) *faisable et* $LB < UB_P$ **then**
    Calcul d'une solution admissible (LB_P, x^P_{LB})
    if $LB_P > LB$ **then**
      $LB \leftarrow LB_P,\ x_{LB} \leftarrow x^P_{LB}$
    Branchement : création des descendants P_1 et P_2 tels que :
    Ω_P : ensemble des solutions admissibles de P
    $\Omega_{P_1} = \Omega_P \cap \{x / x_i \leq K\}$ et $\Omega_{P_2} = \Omega_P \cap \{x / x_i \geq K+1\}$
    $\mathcal{L} \leftarrow \mathcal{L} \cup P_1 \cup P_2$
if $\mathcal{L} = \emptyset$ **then**
  $OptP_0 = \{LB, x_{LB}\}$
else
  Calcul de $UB = \max_{P \in \mathcal{L}} UB_P$
  $OptP_0 \in [LB; UB]$

Les ingrédients de base d'un «bon» *Branch & Bound* sont :

— Une fonction d'évaluation, si possible rapide à évaluer, et proche du problème non relâché, en particulier sur les «petites» régions admissibles.

— Une stratégie d'exploration qui indique comment à chaque étape choisir (P).

— Une stratégie de branchement, qui indique sur quelle variable décomposer et comment choisir le point d'attachement K. Nous remarquons que le nombre de fils de (P) n'est pas nécessairement égal à deux. Nous pourrions choisir de doter (P) de i descendants, correspondants au partitionnement de l'ensemble admissible en i sous-régions.

— Une méthode pour calculer une solution admissible pour le problème initial, dans la sous-région traitée par la relaxation convexe.

— Éventuellement, une bonne heuristique pour initialiser LB à une valeur suffisamment élevée permettant d'élaguer de nombreux nœuds sous-optimaux de $\mathcal{L}$ sans exploration.

— Éventuellement, une technique de réduction de bornes, permettant à chaque itération de réduire le domaine admissible du problème sélectionné, sans éliminer de

solution optimale, afin d'accélérer la convergence de l'algorithme.
Ce sujet fait l'objet de nombreuses publications. Citons par exemple Belotti *et al.* [18].

Pour le problème traité, le problème relâché convexe $(PRC_{N,M})$ établi à la section 4.3.3 sera utilisé comme fonction d'évaluation.

De plus, comme le montre la factorisation de la section 4.2.3, les solutions du problème relâché convexe, sont toujours admissibles pour le problème initial et permettent donc d'en obtenir directement une borne inférieure (par évaluation de la fonction objectif f).

Enfin nous rappelons que la complexité, d'un *Branch & Bound* reste, au pire, exponentielle.
Il existe de nombreuses stratégies de d'exploration et de branchement et leur efficacité dépendent du problème traité. En effet, on ne connait pas de paramétrage général de l'algorithme qui fonctionne bien pour tous les problèmes ou en assurerait un niveau de complexité en moyenne sous exponentielle (i.e en $O(2^{\ln(n)})$) voire polynomiale.

Sans prétendre à une quelconque exhaustivité, nous présentons quelques stratégies de *Branch & Bound* dédiées au problème traité, dans les sections suivantes.
Le premier algorithme, dit principal, présenté à la section 4.5.1, explore à chaque étape le nœud de plus grande relaxation convexe et fournit donc une borne supérieure, de plus en plus fine. Il branche ensuite la variable dont le terme non convexe associé est le plus éloigné de son enveloppe concave. Un second algorithme, présenté à la section 4.5.2, effectue une exploration en largeur d'abord.

Nous appliquons à ces algorithmes les mêmes techniques de réductions de bornes simples fondées sur les relations (linéaires) entre les variables initiales et les variables auxiliaires du problème.

4.5.1 Algorithme Principal

Nous proposons dans cette section un algorithme de *Branch & Bound* dédié au problème $(PNS_{N,M})$ traité. Nous en présentons un schéma récapitulatif ci-dessous dont nous allons décrire ensuite les étapes.

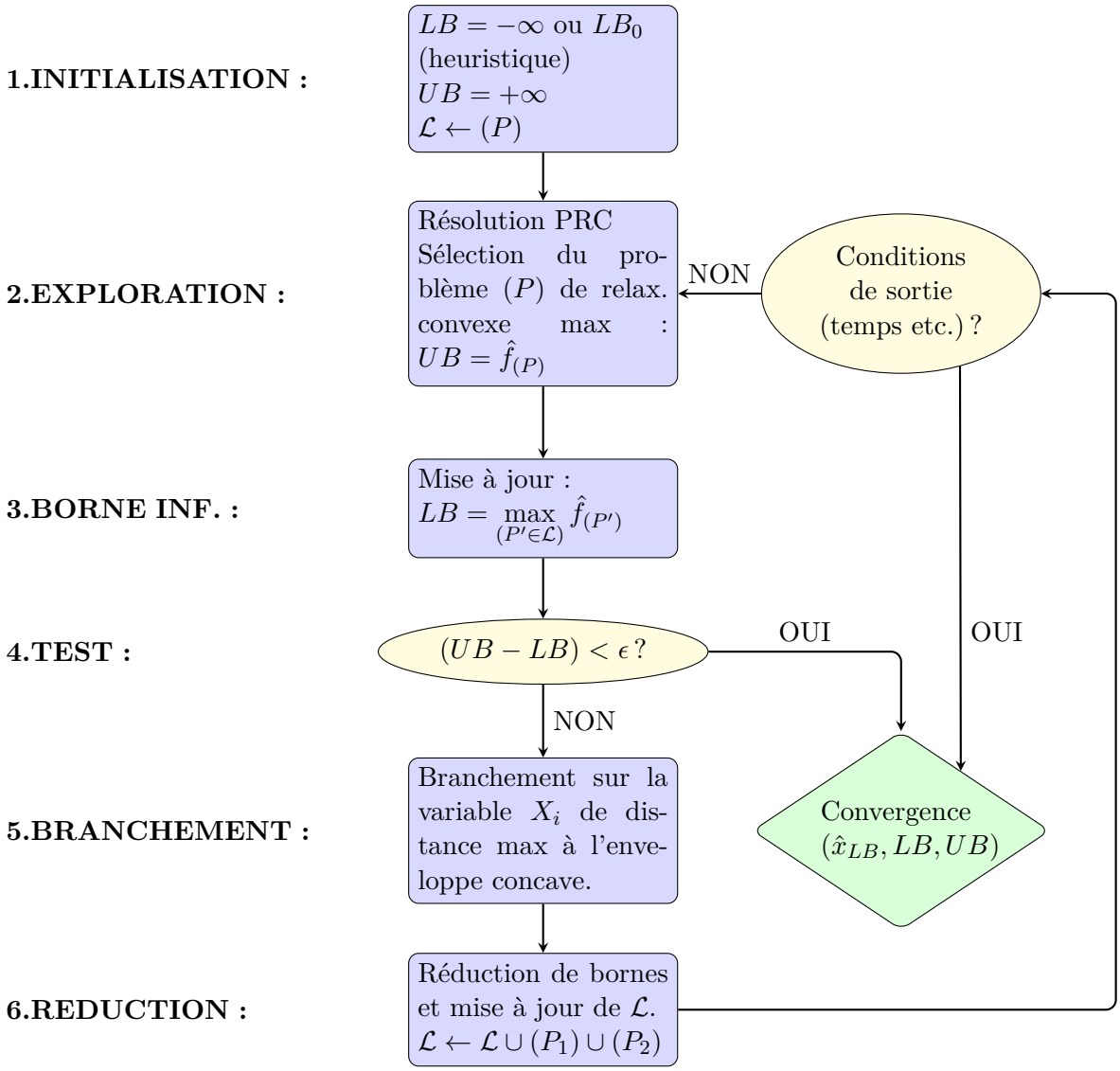


FIGURE 4.5 – Schéma de l'algorithme Principal de *Branch & Bound*

La fonction d'évaluation et le problème convexe majorant $(PRC_{N,M})$ sont déjà établis dans la section 4.3.3.
De plus les solutions $(PRC_{N,M})$ sont admissibles pour $(PNS_{N,M})$ en raison de la nature des contraintes. Il nous reste donc à choisir une stratégie de branchement et d'exploration.

Exploration : Nous choisissons d'explorer le nœud dont la relaxation convexe est la plus élevée, parmi la liste de problèmes $\mathcal{L}$.
La motivation derrière cette stratégie est liée à un critère de convergence. En effet, $\mathcal{L}$

contient une partition de l'ensemble des solutions admissibles de $(PNS_{N,M})$. Si, à une itération donnée, la relation convexe d'un nœud retourne une borne supérieure très proche (à distance ϵ donnée) de la solution admissible correspondante de $(PNS_{N,M})$, alors cette différence serait maximum sur toute la partition et cette solution admissible correspond donc à un maximum global à ϵ près.

Formellement, si la relaxation convexe d'un nœud retourne (UB, x_{UB}), tel que $f(x_{UB}) \geq UB - \epsilon$, alors pour toute solution admissible y de $(PNS_{N,M})$, il existe un problème $P_y \in \mathcal{L}$ dont y est solution. Cela entraîne $f(y) \leq UB_{P_y} \leq UB$. Donc UB est bien une borne supérieure de l'optimum.
De plus x_{UB} en fournit une borne inférieure. Nous avons donc encadré l'optimum global dans un intervalle de largeur ϵ donnée, ce qui satisfait notre critère de convergence. L'algorithme s'arrête et renvoie $(f(x_{UB}), x_{UB})$.

Nous supposons maintenant le nœud $\mathcal{N}$ faisable, non sous-optimal à priori et de relaxation convexe $(UB_{\mathcal{N}},\ x_{\mathcal{N}})$. Nous nous intéressons alors à la stratégie de branchement.

Branchement : Nous nous inspirons de la règle McCormick [79] et de Falk et Soland [46], appliquée au problème traité. Comme établi à la section 4.2, la source de non convexité pour notre problème (factorisable) provient des termes produit $U_i\, V_i = -c_i\, X_i\, g(X_{N+i-1})$. Pour chaque i, nous calculons la différence entre le terme produit de la fonction objectif f et son enveloppe concave (définie dans la section 4.3.3) dont la formulation correspond à celle de $(PRC_{N,M})$, à l'optimum $x_{\mathcal{N}}$.
Cette différence dépend bien évidemment de la région admissible considérée. Toute l'information nécessaire se trouve dans la définition du nœud/problème. Nous sélectionnons l'indice i qui correspond à la plus grande différence.
Formellement nous cherchons $\max_i E_{U_i \cdot V_i}(x_{\mathcal{N}}) - U_i(x_{\mathcal{N}})\, V_i(x_{\mathcal{N}})$
En pratique, nous stockons le vecteur de différences et les indices i correspondants, que nous réordonnons par ordre décroissant. Nous branchons alors sur la variable auxiliaire d'indice X_{N+i-1}.

Il nous reste à choisir le point d'attachement. La contrainte de la variable de décision X_{N+i-1} au nœud $\mathcal{N}$ du problème $(PRC_{N,M})$ est du type : $a_{i,\mathcal{N}} \leq X_{N+i-1} \leq b_{i,\mathcal{N}}$. La solution $x_{\mathcal{N}}$ est admissible, nous pouvons donc prendre $K = X_{N+i-1}(x_{\mathcal{N}}) = x_{\mathcal{N},N+i-1}$.

Nous prenons la $(N+i-1)^{ème}$ coordonnée de l'optimum comme point d'attachement. Nous créons donc les deux nouveaux problèmes $\mathcal{N}_1$ et $\mathcal{N}_2$ dans lesquels la contrainte $a_{i,\mathcal{N}} \leq X_{N+i-1} \leq b_{i,\mathcal{N}}$ est remplacée par $a_{i,\mathcal{N}} \leq X_{N+i-1} \leq x_{\mathcal{N},N+i-1}$
et $x_{\mathcal{N},N+i-1} + 1 \leq X_{N+i-1} \leq b_{i,\mathcal{N}}$ respectivement.
$\mathcal{N}$ est supprimé de la liste de problèmes $\mathcal{L}$. $\mathcal{N}_1$ et $\mathcal{N}_2$ sont rajoutés à $\mathcal{L}$.

Cette règle de branchement fonctionne dans le cas général, mais nous devons préciser la gestion de certaines exceptions. Lorsque $K = b_{i,\mathcal{N}}$, ou que $a_{i,\mathcal{N}} = b_{i,\mathcal{N}}$, le branchement X_{N+i-1} n'est pas possible. Nous branchons alors sur le second élément du vecteur des différences. Et ainsi de suite jusqu'au premier indice i_k tel qu'on puisse brancher sur X_{N+i_k-1}. Si le branchement n'est possible sur aucune variable auxiliaire, ce qui signifie, comme nous l'avons montré à la section 4.2.3, que toutes les variables sauf X_1 et X_2 sont fixées, nous branchons arbitrairement sur X_2. Enfin si le branchement sur X_2 n'est également pas possible, nous sommes sur une feuille de l'arbre qui n'a pas de descendant (aucun problème n'est rajouté à $\mathcal{L}$).

Cette stratégie de branchement vise, comme l'indique McCormick [79], à rapprocher la relaxation convexe de la fonction objectif à l'optimum $X_{N+i-1}(x_{\mathcal{N}})$ en séparant la variable dont le terme non convexe en est le plus éloigné. Le lecteur peut se référer à Falk et Soland [46] pour une justification dans le cas séparable.
Nous abordons maintenant le point de la réduction de bornes.

Réduction de bornes : elle se base sur deux implications qui découlent directement de la définition des variables auxiliaires :

— $X_{N+i-1} = (X_1 + \cdots + X_i) \leq K \Rightarrow X_1, X_2, \cdots X_i \leq K$
— Réciproquement, $X_{N+i-1} \geq K \Rightarrow X_{N+i}, \cdots X_{2N-2} \geq K$

Supposons, par exemple que l'on branche sur X_{N+i-1}. Le descendant $\mathcal{N}_1$ possède la contrainte $X_{N+i-1} \in [a_i; K]$. Pour toutes les variables initiales $X_1 \cdots, X_j, \cdots, X_i$, bornées respectivement par $[a_j; b_j]$, si $K < b_j$ alors nous remplaçons la contrainte par $X_i \in [a_j; K]$
Symétriquement $\mathcal{N}_2$ possède la contrainte $X_{N+i-1} \in [K+1; b_i]$. Pour toutes les variables auxiliaires, $X_{N+i-1} \cdots, X_{N+j-1}, \cdots, X_{2N-2}$, bornées respectivement par $[a_j; b_j]$, si $K + 1 > a_j$ alors nous remplaçons la contrainte par $X_{N+j-1} \in [K+1; b_j]$.

Heuristiques : Nous démarrons notre algorithme, en initialisant la borne inférieure

soit par défaut à $-\infty$, ou bien à partir de la meilleure solution obtenue par DP. Nous utilisons soit une résolution exacte présentée à la section 3.2, ou bien la méthode TSDP de la section 3.3. En pratique, pour les petites et moyennes instances, nous fournirons donc l'optimum comme borne inférieure. Nous comparons le gain obtenu dans les expérimentations numériques de la section 5.

Nous avons donc complètement décrit notre premier algorithme. Nous y référons comme *algorithme Principal* dans les prochaines sections. Il s'arrête lorsque la convergence est obtenue à ϵ près, ou si la liste de problèmes est vide, ou bien enfin si la limite en temps (3 heures) est atteinte. Dans les deux premiers cas, il renvoie l'optimum (à ϵ près au pire). Dans le dernier cas, il renvoie un encadrement $[LB; UB]$ de l'optimum.

Pseudo-code : Nous présentons le pseudo-code suivant de l'algorithme Principal :

Algorithm 5: algorithme Principal

Initialisation $\mathcal{L} \leftarrow P_0$; $LB \leftarrow LB_0$; $UB \leftarrow +\infty$; $x_{LB} = x_{LB_0}$
while *conditions de sortie (temps etc.)=false)* **do**
  Exploration : Résolution des $PRC_{(P)}$, pour $(P) \in \mathcal{L}$
  Sélection de $(P)/PRC_{(P)} = \max_{(P')\in\mathcal{L}} PRC_{(P')} = UB$
  Borne inférieure : $LB = \max\left(LB, \max_{(P')\in\mathcal{L}} PRC_{(P')}\right)$
  if $(UB - LB) \leq \epsilon$ **then**
    Fin de l'algorithme
  else
    Branchement : sélection de X_i tel que distance du terme produit $-c_i X\, ig(X_{N+i-1})$ à son enveloppe concave est maximale
    Si non branchable, on prend la 2ème plus grande différence, etc..
    Création des descendants : P_1 et P_2 tels que : $K = x^P_{LB}(i)$
    Ω_P : ensemble des solutions admissibles de (P)
    $\Omega_{P_1} = \Omega_P \cap \{x/x_i \leq K\}$ et $\Omega_{P_2} = \Omega_P \cap \{x/x_i \geq K+1\}$
    Réduction de bornes sur P_1 et P_2
    $\mathcal{L} \leftarrow \mathcal{L} \cup P_1 \cup P_2$
Résultat :
if $(UB - LB) \leq \epsilon$ **then**
  $OptP_0 = \{LB, x_{LB}\}$
else
  Calcul de $UB = UB_P$
  $OptP_0 \in [LB; UB]$

Nous concluons enfin par une discussion sur la convergence de l'algorithme.

Convergence : intuitivement, puisque les variables sont entières et que l'intervalle de définition de la variable est strictement réduit à chaque branchement, il y a donc un

nombre fini de branchements avant de fixer une variable donnée. Il y a un nombre fini de variables, donc un nombre fini d'étapes possibles, avant d'arriver à une feuille de l'arbre de décomposition, qui ne générera pas de descendant.
Nous pouvons donc explorer toutes les feuilles au pire, ce qui assure que l'algorithme converge en un nombre d'itérations fini, bien qu'exponentiel au pire. Plus formellement, cette intuition est confirmée par le théorème de McCormick [79] et Horst et Tuy [61], qui introduisent le critère de convergence suivant :

Théorème 8. *Une stratégie de branchement est dite cohérente finie (finitely consistent), si pour toute suite de branchements, la largeur des intervalles* $(b_{i,r} - a_{i,r}) \underset{r \to +\infty}{\to} 0$ *,pour tout* i*, en un nombre fini d'itérations* r*.*
Si la stratégie est cohérente finie, l'algorithme de Branch & Bound converge également vers l'optimum, en un nombre fini d'itérations.

Nous avons donc proposé un premier algorithme, dit algorithme Principal de *Branch & Bound*, dont la stratégie d'exploration est celle de la plus grande borne supérieure et la stratégie de branchement est celle de la plus grande distance à l'enveloppe concave. Nous en discuterons les performances au chapitre 5, qui est dédié aux expérimentations numériques.
Nous présentons un deuxième algorithme de *Branch & Bound*, à la section suivante, dont la stratégie d'exploration est différente.

4.5.2 Exploration en largeur d'abord

Nous proposons dans cette section un second algorithme de *Branch & Bound* dédié au problème $(PNS_{N,M})$ traité. Comme précédemment, nous commençons par un schéma de l'algorithme que nous décrivons ci-après.

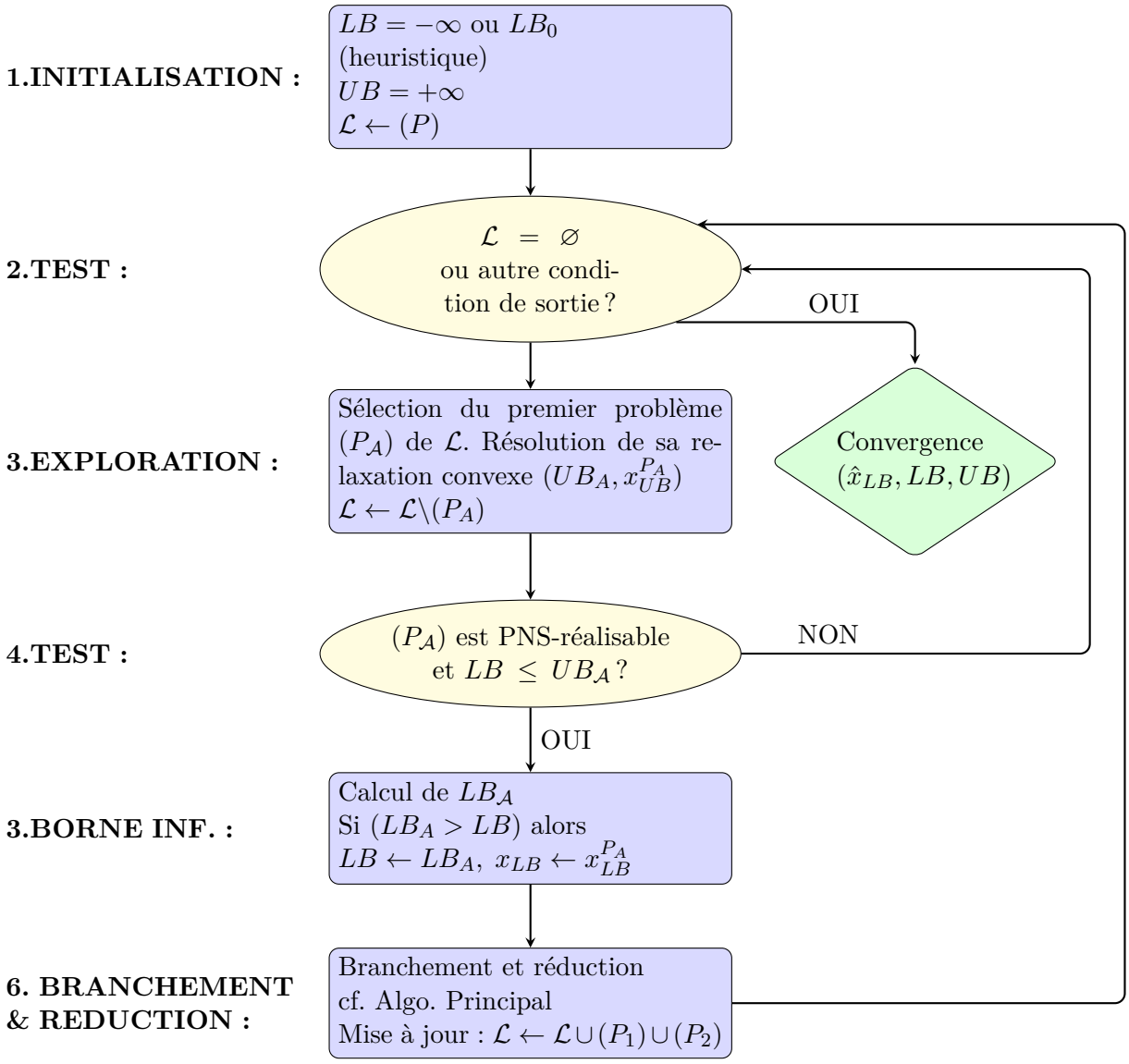


FIGURE 4.6 – Schéma de l'algorithme de *Branch & Bound* en largeur d'abord

La fonction d'évaluation et le problème convexe majorant $(PRC_{N,M})$ sont identiques à ceux de la section 4.3.3. Nous rappelons à nouveau que les solutions $(PRC_{N,M})$ sont admissibles pour $(PNS_{N,M})$.
Nous nous intéressons aux stratégies de branchement et d'exploration.

Exploration : La liste de problèmes $\mathcal{L}$ est initialisée avec le problème initial, noté (P_0). Nous explorons le premier nœud (P_0) de l'arbre qui renvoie en général deux fils (P_1) et

(P_2). A l'issue de la première itération $\mathcal{L} = \{(P_1),\ (P_2)\}$. Dans la deuxième itération, nous explorons le premier problème de la liste qui est (P_1) et qui renvoie deux fils, (P_3) et (P_4), que nous «poussons» dans $\mathcal{L}$ (*push* informatique, qui correspond à une insertion à la fin d'une liste). A la fin de la deuxième itération, $\mathcal{L} = \{(P_2),\ (P_3),\ (P_4)\}$. Et ainsi de suite. Nous explorons toujours le premier nœud de $\mathcal{L}$ et insérons ses descendants à la fin de la liste. Par conséquent, nous explorons le problème initial, ses enfants (P_1) et (P_2), puis ses petits-enfants $(P_3) \cdots (P_6)$ et nous explorons ainsi chaque génération avant de passer à la suivante. Une telle exploration, est dite en largeur d'abord. C'est une exploration classique de $\mathcal{L}$, qui va nous permettre de comparer les performances avec celles de l'algorithme principal.

Branchement : nous appliquons la même stratégie que pour l'algorithme principal, décrit à la section 4.5.1, qui maximise la distance du terme produit à son enveloppe concave. Nous appliquons également les mêmes traitements des cas aux bords.
Réduction de bornes : identique à celle de l'algorithme principal.

Bornes supérieures : L'algorithme principal, décrit à la section précédente, donne une borne supérieure à chaque itération en raison de sa stratégie d'exploration, qui visite justement ce problème de plus grande relaxation convexe.

Pour cet algorithme, nous pouvons premièrement, comme décrit dans l'algorithme général de la section 4.5, donner une borne supérieure à la fin de l'algorithme, en calculant la plus grande relaxation convexe, sur les nœuds restants de $\mathcal{L}$. Si $\mathcal{L}$ est vide l'optimum global est retourné.

Une seconde méthode consiste à remarquer que chaque génération de problème fournit une partition de l'ensemble des solutions admissibles. Par conséquent la relaxation convexe de (P_0) fournit une première borne. Le maximum de celle de ses enfants (P_1) et (P_2) également (à priori plus fine). Le maximum de celle de ses petits-enfants (P_3),(P_4),(P_5),(P_6) également. Et ainsi de suite.

De plus, les optimums fournissent des solutions admissibles, donc des bornes inférieures à chaque exploration de nœud. Nous avons donc un encadrement, à chaque génération de l'optimum, et nous pouvons éventuellement un appliquer un critère d'arrêt de l'algorithme du type $UB - LB < \epsilon$, à ϵ fixé. Ce critère est essentiel pour la version continue du pro-

blème, car la cohérence du branchement n'assure la convergence qu'à la limite dans ce cas.

Heuristiques : approche identique à celle de l'algorithme Principal : $LB_{init} = -\infty$ ou la meilleure solution obtenue par DP.

Convergence : Cet algorithme est cohérent fini, il converge donc en un nombre fini d'itération dans le cas discret. Dans le cas continu, il converge à ϵ prés.

Pseudo code : Nous avons décrit notre deuxième algorithme et en présentons le pseudo-code :

Algorithm 6: algorithme d'exploration en largeur d'abord

Initialisation $\mathcal{L} \leftarrow P_0$; $LB \leftarrow LB_0$; $UB \leftarrow +\infty$; $x_{LB} = x_{LB_0}$
while $\mathcal{L} \neq \emptyset$ *ou autre condition de sortie non déclenchée* **do**
  Exploration : sélection du premier problème (P_A) de $\mathcal{L}$
  $\mathcal{L} \leftarrow \mathcal{L} \backslash (P_A)$
  Calcul de la relaxation convexe de (P_A) : $(UB_A, x_{UB}^{P_A})$
  if $\left[(P_{\mathcal{A}})$ *faisable et* $LB < UB_A\right]$ **then**
    Calcul d'une solution admissible $(LB_A, x_{LB}^{P_A})$
    if $LB_A > LB$ **then**
      $LB \leftarrow LB_A,\ x_{LB} \leftarrow x_{LB}^{P_A}$
    Branchement & réduction de bornes :
    mêmes règles que pour l'algorithme Principal
    Branchement à droite des descendants P_1 **et** P_2 **:** $\mathcal{L} \leftarrow \mathcal{L} \cup P_1 \cup P_2$
if $\mathcal{L} = \varnothing$ **then**
  $OptP_0 = \{LB, x_{LB}\}$
else
  Calcul de $UB = \max_{P \in \mathcal{L}} UB_P$
  $OptP_0 \in [LB; UB]$

Nous avons donc proposé un second algorithme de *Branch & Bound*, qui diffère du précédent par la stratégie d'exploration, en largeur d'abord.
Nous en discutons les performances à la section 5 dédiée aux expérimentations numériques.

4.6 Conclusion

Dans ce chapitre, nous avons proposé, après des rappels de factorisation, une factorisation spécifique à notre problème de thèse. Cette factorisation nous a permis d'isoler les termes bilinéaires non concaves.

Nous avons ensuite traité ces termes à l'aide des inégalités de McCormick et ainsi proposé une surestimation concave de la fonction objectif du problème initial. Nous en avons également déduit une relaxation convexe fine du problème $(PNS_{N,M})$ avec des bornes indépendantes pour chaque variable de décision.

Nous avons utilisé cette relaxation convexe comme fonction d'évaluation d'un algorithme de *Branch & Bound*, pour lequel nous montrons la convergence en temps fini. Nous proposons enfin deux variations de l'algorithme, qui diffèrent par leur méthode d'exploration.

Par conséquent, nous avons proposé une méthode de résolution exacte en temps fini de notre problème en variables entières $(PNS_{N,M})$. Lorsque les tailles d'instances, ne permettent pas une résolution exacte dans le temps imparti, les algorithmes de résolutions proposés fournissent une borne supérieure.

De manière symétrique au chapitre 3, nous avons établi, dans ce chapitre, une méthode de résolution « à droite » (dans un contexte de maximisation), qui résout $(PNS_{N,M})$ à l'optimum, ou en fournit une borne supérieure.

Nous sommes donc désormais capables, à l'aide des techniques introduites dans les chapitres précédents, de proposer pour toute taille d'instance, soit une résolution exacte du problème $(PNS_{N,M})$, ou bien un encadrement de son optimum. La démarche est résumée dans le schéma de la page suivante.

Nous comparons dans le chapitre suivant, les performances numériques de nos méthodes.

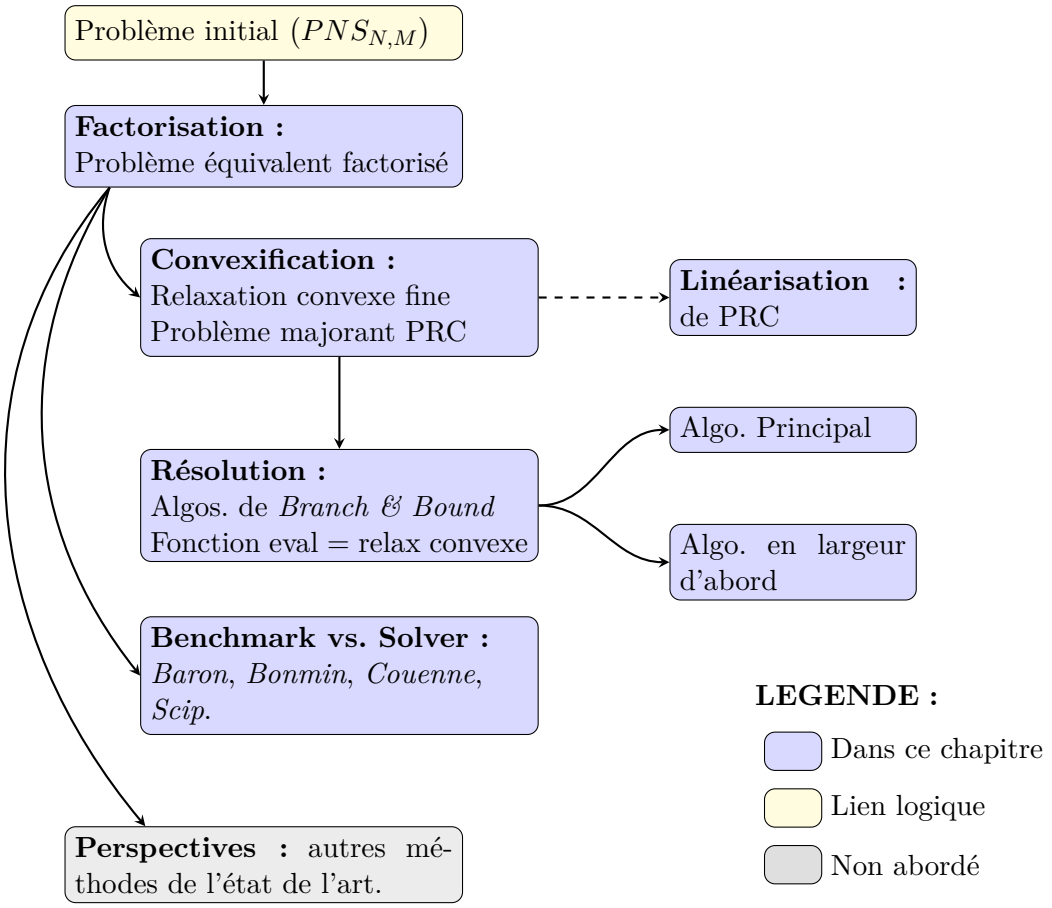


FIGURE 4.7 – Schéma récapitulatif de notre approche de convexification

Chapitre 5

Comparaison numériques des méthodes

Dans les chapitres précédents, nous avons proposé et décrit des méthodes de résolutions exactes ou approchées, qui permettent de résoudre le problème traité à l'optimum, ou d'en obtenir un encadrement.
Dans ce chapitre, nous comparons les performances des différentes méthodes utilisées, que nous mettons en œuvre dans le cadre de l'application financière ELBA.
Le chapitre est organisé de la manière suivante :

Nous décrivons tout d'abord le cadre des expérimentations numériques à la section 5.1. Nous définissons notamment la terminologie correspondante aux tailles d'instances. Nous présentons l'origine des données et justifions notre choix de solver.
Nous présentons également les fonctions de pénalités choisies, leurs propriétés et leur calibration.

Nous avons choisi de décomposer les expérimentations numériques par taille d'instance, puis par type de méthode de résolution (méthodes à base de DP, heuristiques ou bien convexification).

La section 5.2 est consacrée aux petites et moyennes instances.
Nous présentons tout d'abord, à la section 5.2.1, la résolution exacte par DP et discutons les temps CPU.
Puis, nous nous intéressons au paramétrage du grain et de la taille du bandeau dans la méthode TSDP discrète, à la section 5.2.2 et à l'approche hybride à la section 5.2.3. Nous

présentons les résultats agrégés des méthodes heuristiques à la section 5.2.4 et montrons également l'influence des différentes trajectoires lorsque le vecteur p varie.
Nous traitons ensuite la résolution par convexification à la section 5.2.5.

Nous passons ensuite à l'échelle à la section 5.3 et traitons les grandes instances, pour lesquelles la résolution exacte par DP n'est pas disponible. Nous étudions la limite en espace (mémoire) pour les grandes et très grandes instances, à la section 5.3.1 et proposons une amélioration directe en perspective.
Nous discutons, comme précédemment, la qualité et la rapidité des solutions obtenues. Nous présentons également des cas représentatifs pour montrer l'influence de la volatilité de p. Les résultats agrégés des méthodes de DP sont présentés à la section 5.3.2, et ceux de convexification à la section 5.3.3.

5.1 Cadre des expérimentations numériques

Dans cette section, nous définissons le cadre de nos expérimentations numériques.
Nous commençons par décrire les outils informatiques et justifions le choix des solvers utilisés à la section 5.1.1.
Puis nous définissons les conventions d'instance (taille et dénomination) à la section 5.1.2.
Nous expliquons l'origine des données et les valeurs des constantes utilisées dans les expérimentations numériques à la section 5.1.3.
Nous présentons, à la section 5.1.4, les fonctions de pénalité utilisées ainsi que leur propriétés.
Enfin, nous discutons enfin le choix des paramètres de calibration L et H à la section 5.1.5.

5.1.1 Outils informatiques et choix des solvers

Nous présentons tout d'abord les outils informatiques (machines, logiciels) utilisés.

Caractéristiques des machines : les expérimentations numériques ont tourné sur PC sous Intel Xeon(R) Silver 4114 at 2.20 Ghz , 2 sockets, 20 core, et 32 Gb de RAM. L'O/S est Linux Ubuntu 18.04 (Bionic Beaver) et le compilateur c++ gcc v9.3.0.
Puis nous abordons les solvers utilisés.

Solvers utilisés :

— LocalSolver v9.5 (Linux64, build 20201030) - Benoist *et al.* [20, 22, 21]

— NLopt v2.6.2 - Johnson [64]
— SCIP v7.0.1 - Gamrath *et al.* [50]
— BARON v2022.11.3 - Tawarmalani et Sahinidis [110],[98]
— COUENNE v0.5.8 - Belotti *et al.* [18],[16]
— BONMIN v1.8.8 - Bonami *et al.* [31],[30]
— Enfin la modélisation pour *Bonmin* et *Couenne* est effectuée à partir du langage *AMPL* v20221013 (Linux-5.4.0-1091-azure, 64-bit) - Fourer *et al.* [49], [54]

Justification du choix des solvers : Nos travaux de recherches, nous ont familiarisé avec l'écosysteme des solvers dédiés aux problèmes non linéaires en variables entières. Pour la partie heuristiques et recherche locale, dont le but est de trouver rapidement la meilleure solution admissible, notre référence est *LocalSolver*, dont les grands principes de résolution sont décrits à la section 3.3.4.

Puis, durant les cours doctoraux de Schmidt [99], nous nous sommes intéressés à *Scip*, qui est libre, très flexible, et constitue un bon point de référence, pour la programmation mathématique en variables entières.

Nous nous sommes également intéressés à des solvers qui reproduisaient nos méthodes de résolution, avec une relaxation convexe du problème, intégrée à un algorithme de *Branch & Bound*.
C'est pourquoi *Couenne*, nous a semblé une excellente référence, car, il traite les problèmes non convexes *via* une factorisation (au sens de McCormick), une relaxation convexe fondée sur la recherche des enveloppes convexes/concaves des fonctions du problème et enfin l'utilisation *Branch & Bound* spatial (pour les problèmes en variables mixtes).
De plus, il implémente les techniques avancées de linéarisation, de branchement, de réductions de domaine et de recherches d'heuristiques.

Nous nous sommes ensuite intéressés à *Baron* qui est également un solver de référence, pour l'optimisation globale des problèmes non convexes en variables mixtes.

Nos recherches sur les liens entre ces différents solvers, nous ont orienté vers le projet COIN-OR (*COmputer INfracstructure for Operations Research*).
Ce projet, qui a démarré en 2000, met en commun, les briques élémentaires de résolution (*e.g* solvers élémentaires linéaires et non linéaire, *Branch & Cut*, générateurs de coupes),

ainsi que les API (*Application Programing Interface*) permettant de les paramétrer et de les combiner.
Enfin, le projet fournit à la communauté, des dépôts de données, permettant de récupérer et d'enrichir les codes sources, ainsi qu'un suivi des bugs remontés et le *versioning* des solvers.
C'est dans ce contexte que nous avons découvert le solver *Bonmin*. Nous recommandons l'article de Bonami *et al.* [29] sur la genèse de *COIN-OR* et de *Bonmin*.
Enfin, nous notons un point important concernant *Bonmin*.

Remarque 10. *Bonmin ne garantit une résolution à l'optimum que pour les problèmes convexes (cf. avertissement sur le site du solver). Pour le non convexe, Bonmin ne retourne que des solutions admissibles (bornes inférieures). Il n'est donc pas, contrairement à Couenne et Baron, un solver global.*

Cependant, ces différents solvers, qui ont souvent leur *API* distincte en C++, et/ou leur langage propriétaire de modélisation, nécessitent un temps non négligeable d'intégration.
C'est notamment le cas de *Scip* dont le paradigme de programmation en expressions portées par un arbre, bien que parfaitement adapté aux fonctions factorisables, a nécessité le temps de prise en main le plus important.

Nous avons alors cherché des synergies et le langage *AMPL* (*A Mathematical Programming Language*), dont la première version date de 1990. *AMPL* nous a permis d'utiliser *Couenne*, *Baron* et *Bonmin*, que nous n'avions pas prévu initialement, au prix d'une seule modélisation (avec quelques variations mineures sur les options, selon les solvers).

Choix du solver de résolution du problème relaché convexe : comme nous l'avons discuté à la section 4.5, nos algorithmes de *Branch & Bound* reposent sur une fonction d'évaluation convexe. Nous avons choisi d'utiliser le solver *Scip*, pour la flexibilité de son intégration à notre code source C++, afin de résoudre le problème convexe $(PRC_{N,M})$, défini à la section 4.3.3, qui est appelé en chaque nœud de nos *Branch & Bound*.
En particulier, *Scip*, est appelé pour résoudre le problème relâché convexe (4.9), dans sa version non linéarisée.
Cependant, ces choix entrainent également des limitations.

Limitation des solvers : Outre le caractère commercial ou libre, les solvers que nous avons utilisés ont des limitations techniques qui restreignent leur application aux expérimentations numériques et que nous décrivons.

Tout d'abord aucun solver, à l'exception de *Bonmin*, ne prend en charge la fonction arctangente à ce jour. En particulier, *Baron* n'accepte pas les fonctions trigonométriques. Ces restrictions nous empêchent d'utiliser une des fonctions de pénalité, décrites à la section 5.1.4, pour les expérimentations numériques sur les méthodes de convexification.

De plus, comme la relaxation convexe fait appel à *Scip*, nous ne pouvons pas intégrer l'arctangente, dans nos algorithmes propriétaires de *Branch & Bound*.

Une autre limite concerne les restrictions imposées par les licences d'utilisation.
Pour notre licence académique, *BARON* est limité à 50 variables de décision.
Nous remarquons que, dans notre licence *Community Edition* d'AMPL, les solver *Couenne* et *Bonmin* sont annoncés bridés à 300 variables de décision mais nous avons néanmoins pu utiliser *Bonmin* pour 1000 variables.

Comme toujours, la taille des instances est cruciale dans un problème d'optimisation. Nous précisons, dans la section suivante, la notion de petite, moyenne ou grande instance.

5.1.2 Taille et nomenclature d'instances

Nous présentons les conventions de notations et de nomenclature d'instances.

Taille d'instances : Les instances sont contrôlées par le nombre de variables de décisions N et la taille de l'écoulement noté M (qui correspond également à la borne supérieure de chaque variable de décision). Elles sont présentées dans les tables de résultats, soit directement sous la forme (N, M), ou bien sous la forme $(N, M) = (10^a, 10^b),\ a < b$, avec $1 \leq a \leq 3$ et $2 \leq b \leq 9$.
En particulier, sauf pour les très petites instances (lorsque $N \leq 10$), N divise M, car il n'y a pas d'intérêt particulier à traiter les cas de blocs non subdivisables. Lorsqu'il n'y a pas ambiguïté, nous simplifions les notations dans le texte, en ignorant la base, et écrivons $(T, N) = (a, b)$, (au lieu de $(10^a, 10^b)$)

Dénomination d'instance : On se référera aux instances en utilisant la terminologie suivante :

- Très petites, lorsqu'en notation exacte $N \leq 10$ (moins de 10 variables).
- Petites, si $(N, M) \in \{(1,2), (1,3), (1,4), (2,3), (2,4)\}$.
- Moyennes, si $(N, M) \in \{(1,5), (1,6), (2,5), (3,5)\}$.
- Grandes, si $(N, M) \in \{(1,7), (1,8), (1,9), (2,6), (2,7), (2,8), (2,9), (3,6)\}$.
- Très grandes quand $(N, M) \in \{(3,7), (3,8), (3,9)\}$.

Nous passons à l'origine des données et des simulations utilisées.

5.1.3 Paramétrages statiques et origine des données

Nous passons à l'origine des données et des simulations utilisées.

Prix plancher q_i : Dans l'expérimentation financière ELBA, nous supposons que $q_i = (1-\beta)p_i$, avec $0 < \beta < 1$ constant. Cela signifie que le prix plancher pour de plus grands volumes d'écoulement (même dans le cas d'une vente à tout prix) est égal à une fraction constante du prix observé. Dans un contexte financier, le prix plancher peut représenter la valeur intrinsèque de la compagnie [1].

Pour les expérimentations numériques, nous utilisons la valeur $\beta = 0.9$. Comme $g(0) = 0$ et g tend asymptotiquement vers 1, cette valeur correspond à un prix plancher de 10% du prix observé. Plus β est proche de 1, plus sévère est la pénalité pour écouler plus d'actions, et plus large est l'intervalle de pénalité, car le prix plancher tend vers 0. En effet, le prix d'exécution est compris dans $[(1-\beta)\, p_i; p_i]$). Par conséquent, une valeur élevée de β se traduit par une différence plus marquée dans la fonction objectif, entre une bonne et une mauvaise stratégie d'écoulement.

Ce choix de β nous permet de préciser les bornes naturelles (cf. section 1.1) de la fonction objectif. Nous conservons la borne supérieure $\overline{p}\, M$. Mais dans ce cas, en notant $\underline{p} = \min_i p_i$, nous obtenons le prix d'exécution minimum $q_i = (1-\beta)p_i \geq (1-\beta)\, \underline{p}$
Nous en déduisons que $f \in [(1-\beta)\, \underline{p}\, M\, ;\, \overline{p}\, M]$
Dans le cas d'une trajectoire constante, nous obtenons l'encadrement $f \in [(1-\beta)\, p_0\, M\, ;\, p_0\, M]$

1. Une discussion économique sur la valeur intrinsèque d'une compagnie n'entre pas dans cadre de notre étude et n'est pas pertinente pour la programmation mathématique.

En effet, dans ce cas $\overline{p} = p_0$ donc $p_0\, M$ est une borne supérieure de l'objectif.
Enfin, si nous appliquons les paramètres numériques pour β et p_0, $f \in [10\, M; 100\, M]$, où M contrôle la taille de l'écoulement.

Nous nous intéressons ensuite aux trajectoires de p, ou de manière équivalente pour l'application ELBA, aux prix de l'actif à écouler. Nous décrivons la distribution des prix et la génération des trajectoires associées au vecteur de prix p.

Distribution du prix de l'action : Nous supposons que le prix de l'action suit un mouvement Brownien géométrique, de moments (μ, σ), démarrant à $p_0 = 100$. Cette dynamique classique est décrite dans Hull [63] :

$$\frac{dS_t}{S_t} = \mu\, dt + \sigma\, dW_t$$

Par application du lemme d'Ito, nous calculons $d\,(\ln S_t) = \Big(\mu - \dfrac{\sigma^2}{2}\Big) dt + \sigma\, dW_t$ En intégrant sur l'intervalle $[t; t + \Delta t]$, nous obtenons :

$$S_{t+\Delta t} = S_t \exp\left[\Big(\mu - \frac{\sigma^2}{2}\Big)\Delta t + \sigma\,(W_{t+\Delta t} - W_t)\right]$$

Nous utilisons la propriété de stationnarité du mouvement Brownien et en déduisons :

$$S_{t+\Delta t} = S_t \exp\left[\Big(\mu - \frac{\sigma^2}{2}\Big)\Delta t + \sigma\, W_{\Delta t}\right]$$

Ce qui revient à la formule suivante que nous utilisons dans les simulations :

$$S_{t+\Delta t} = S_t \exp\left[\Big(\mu - \frac{\sigma^2}{2}\Big)\Delta t + \sigma\,\sqrt{\Delta t}\, Z\right],\ \text{where } Z \sim \mathcal{N}(0,1) \tag{5.1}$$

Vecteurs de prix p : nous simulons les prix d'action pour ELBA en utilisant le processus stochastique décrit dans l'équation (5.1) ci-dessus, avec le point de départ $p_0 = 100$, et les moments (μ, σ). Nous nous intéressons à l'action conjuguée de ces deux moments, qui pour l'application financière ELBA peuvent s'interpréter comme la tendance et la volatilité de l'actif. Nous sélectionnons trois régimes pour chaque moment $\mu \in \{-0.05; 0; +0.05\}$ et $\sigma \in \{0.10; 0.25; 0.70\}$, ce qui entraine neuf combinaisons possibles. Enfin la dixième combinaison correspond à la trajectoire constante de prix $\forall i,\ p_i = 100$. Cette trajectoire sert de référence pour comparer les performances des algorithmes proposés.

Pour chacun des neuf couples de moments (μ_i, σ_i), nous lançons 10 simulations et nous prenons la moyenne sur S_t. Cela permet de rendre la trajectoire plus fluide et de mieux retranscrire les caractéristiques des moments.
Enfin, chacun des dix vecteurs de prix V_i a une longueur égale à $N_{max} = 10^3$, qui correspond au plus grand nombre de variables de décision considérés dans les expérimentations numériques. Quand N est plus petit que N_{max}, le vecteur de prix correspondant est extrait de V_i de façon uniforme. Par exemple, dans le cas $N = 10$, nous prenons $\hat{V} = \{V_{100}, V_{200}, \cdots, V_{1000}\}$. Nous remarquons qu'il peut y avoir un biais dans le choix de la longueur $\lfloor \frac{N_{max}}{N} \rfloor$. Nous pourrions en effet arbitrairement choisir $\hat{V} = \{V_j, V_{100+j}, \cdots, V_{900+j}\}$, où $1 \leq j \leq 100$. Cependant lorsque N croit vers N_{max}, ce biais disparait.

Nous pourrions légitimement nous interroger sur la pertinence d'une diffusion très volatile pour notre modèle de prix déterministes.
Comme mentionné aux sections 1.1 et 1.2, les techniques d'optimisation présentées dans cette étude s'appliquent à n'importe quel vecteur de prix strictement positif. Du point de vue de la programmation mathématique, l'étude de la finesse des bornes pour un large spectre de trajectoires est un sujet d'intérêt. En effet, nous cherchons pas ici à tirer des conclusions spécifiques pour l'application financière ELBA, fondées sur des instances irréalistes, mais plutôt à étendre nos résultats à un plus large champ d'applications, en dehors de la finance. A cet égard, il nous semble important de valider empiriquement nos résultats lorsque les trajectoires fluctuent significativement.

Nous terminons cette section par un rappel sur les conventions de lecture des tables de résultats.

Conventions dans les tables de résultats : comme indiqué précédemment nous avons simulé 10 vecteurs p avec différents couples de moments.
Pour la lisibilité des tables de résultats, nous affichons les métriques (*i.e* qualité et temps CPU) par défaut pour p constant et en moyenne sur les 9 autres processus simulés.

Dans les tables de résultats des prochaines sections, CST fait référence au vecteur p constant et AVG à la moyenne des neuf autres. Lorsque les résultats sont significativement différents entre les vecteurs p non constants, nous le mentionnons explicitement.

Dans les tables de résultats mesurant la qualité des bornes, les valeurs sont exprimées en pourcentage et mesure la différence relative à l'optimum (qui provient de la DP exacte) ou à la meilleure borne connue.
Pour les tables reflétant le temps CPU, les valeurs sont en secondes, ϵ correspond à la valeur numérique minimale dans les deux cas (qualité et temps) avec l'unité correspondante. Par conséquent, "$< \epsilon$ " signifie soit que la différence relative à l'optimum est inférieure à 0.01% (10^{-4}), ou bien que le temps de calcul est inférieur à $0.01s$.
Enfin, DNC (*Did Not Converge*) signifie que l'algorithme n'a pas convergé soit par manque de temps ou bien a retourné une erreur mémoire. Nous fixons la limite de temps CPU à $600s$, sauf cas mentionnés explicitement (*e.g* pour certaines instances, elle est de 3h, soit 10800s).

5.1.4 Fonctions de pénalité et calibration

Nous présentons dans cette section les fonctions de pénalité g employées dans les expérimentations numériques étudions leur propriétés et discutons leur calibration.
Nous définissons les trois fonctions prototypes G réelles univariées suivantes :

$$G \begin{cases} x \mapsto \dfrac{x}{1+x} \\ ou\ x \mapsto 1 - \dfrac{2}{1+\sqrt{1+x}} \\ ou\ x \mapsto \dfrac{2}{\pi}\arctan(x) \end{cases}$$

Les fonctions G ainsi définies sont $\mathcal{C}^2$ et bornées sur $\mathbb{R}^+$. De plus, $G(0) = 0$ *et* $\lim_{+\infty} G = 1$
Nous nous intéressons aux propriétés de convexité de G.

Théorème 9. *Les trois fonctions G, ainsi définies sont concaves sur $\mathbb{R}+$.*
De plus, chacune des fonctions $x \mapsto x \cdot G(x)$ est convexe $\mathbb{R}+$.

Démonstration. Par calcul direct des dérivées secondes de G et $x \cdot G$:

$$G'' \begin{cases} \left[\dfrac{x}{1+x}\right]'' = \dfrac{-2}{(1+x)^3} < 0\ sur\ \mathbb{R}_+ \\ \left[1 - \dfrac{2}{1+\sqrt{1+x}}\right]'' = \dfrac{-(1+3\sqrt{1+x})}{2\cdot(1+x)^{3/2}\cdot(1+\sqrt{1+x})^3} < 0\ sur\ \mathbb{R}_+ \\ \left[\dfrac{2}{\pi}\arctan(x)\right]'' = \dfrac{-4\,x}{\pi(1+x)^2} \leq 0\ sur\ \mathbb{R}_+ \end{cases}$$

Ce qui prouve la concavité.

$$(x \cdot G)'' \begin{cases} \Big[\frac{x^2}{1+x}\Big]'' = \frac{2}{(1+x)^3} > 0 \; sur \; \mathbb{R}_+ \\ \Big[x - \frac{2\,x}{1+\sqrt{1+x}}\Big]'' = \frac{4+3\,x+(x+4)\sqrt{1+x}}{2\cdot(1+x)^{3/2}\cdot(1+\sqrt{1+x})^3} > 0 \; sur \; \mathbb{R}_+ \\ \Big[\frac{2\,x}{\pi}\arctan(x)\Big]'' = \frac{4}{\pi\big(1+x^2\big)^2} > 0 \; sur \; \mathbb{R}_+ \end{cases}$$

Ce qui prouve la convexité.

□

Par conséquent, les prototypes associés aux fonctions de pénalité choisies sont concaves et vérifient les hypothèses du lemme 2 de la section 3.5, qui permet d'établir la majoration du problème $(PNS_{N,M})$ par monotonie.

Nous définissons ensuite les fonctions de pénalités $g : x \mapsto G(\eta\, x)$, où η est constant et dépend seulement de G. Soit L un niveau donné , nous définissons un seuil H, tel que $g(L) = H$. La constante η est donc un facteur d'échelle servant à projeter le segment $[0; L]$ (qui encadre les valeurs des variables de décisions) sur le segment $[0; H]$ (qui encadre la valeur de la pénalité sur le prix).

Les fonctions prototypes G choisies sont injectives sur $\mathbb{R}^+$, donc nous pouvons calculer leur inverse respectif G^{-1}, qui est donné par :

$$G^{-1} \begin{cases} x \mapsto \dfrac{x}{1-x} \\ o\grave{u}\; x \mapsto \tan\big(\dfrac{\pi}{2}\,x\big) \\ o\grave{u}\, x \mapsto \dfrac{4\,x}{(1-x)^2} \end{cases}$$

Nous en déduisons la valeur de $\eta = \dfrac{G^{-1}(H)}{L}$.

Nous passons aux choix des paramètres numériques L et H.
$L = M$ est un choix naturel car les variables de décision et leur sommes partielles sont bornées par M. Nous avons testé différentes valeurs pour $H \in]0; 1[$. Plus le seuil est proche de 1, plus grande est le pouvoir discriminant de la fonction de pénalité sous-jacente g.

Le pouvoir discriminant correspond à la distance entre une heuristique naïve comme la vente à tout prix ou la vente linéaire et l'optimum. Pour les expérimentations numériques, nous avons choisi $L = M$ et $H = 0.99$. La justification et le détail des tables de calibrations sont présentées à la section suivante.

Enfin, nous avons défini et calibré les fonctions de pénalité qui composent la fonction objectif de $(PNS_{N,M})$. Nous affichons les fonctions objectifs associées en dimension N=3 et M=100, avec p constant, pour chaque fonction de pénalité :

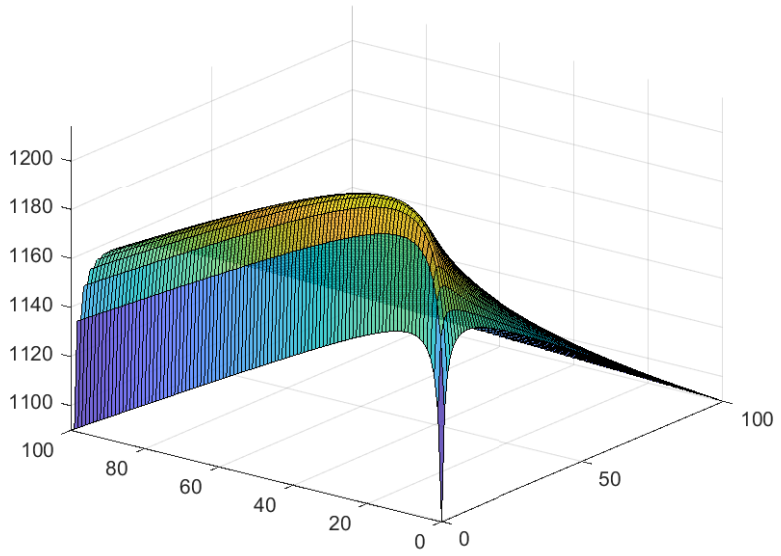


FIGURE 5.1 – Objectif $(PNS_{3,100})$, $G : x \mapsto \frac{x}{1+x}$

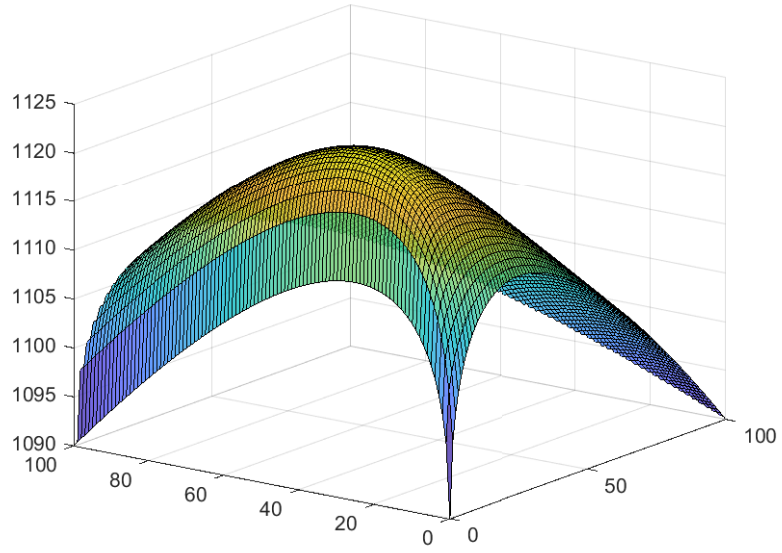


FIGURE 5.2 – Objectif $(PNS_{3,100})$, $G : x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$

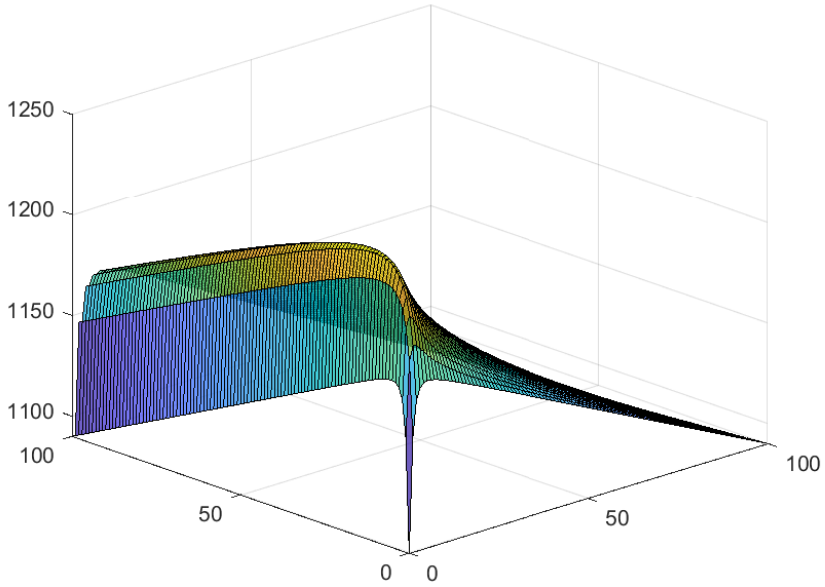


FIGURE 5.3 – Objectif $(PNS_{3,100})$, $G : x \mapsto \frac{2}{\pi}\arctan(x)$

Points importants :

— Les fonctions $x \mapsto \frac{x}{1+x}$ et $x \mapsto \frac{2}{\pi}\arctan(x)$ génèrent une fonction objectif assez abrupte avec une ligne de crête assez marquée. Cela est du aux dérivées très proches des deux fonctions ($\frac{1}{(x+1)^2}$ et $\frac{1}{1+x^2}$).

— L'objectif, sous la pénalité $x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$, a profil beaucoup plus arrondi, et qui semble concave en dimension 3.

— Nous faisons bien attention à ne pas tirer de conclusions générales, à partir de la dimension 3.

5.1.5 Justification du choix des paramètres L et H

Le facteur d'échelle η est complètement déterminé par les choix L et H dans l'équation $G(\eta\, L) = H$. La suite des sommes partielles y_k (avec $y_k = \sum_{i=1}^{k} x_i$) tend vers M, lorsque k tend vers N. Comme, $\lim_{+\infty} g = 1$, il est naturel pour la fonction g de se rapprocher de 1 lorsque y tend vers M. Le choix de prendre $L = M$ est donc logique.
La distance à laquelle passe la fonction g de sa limite est fixée par le seuil H. Comme discuté à la section précédente, nous avons testé différentes valeurs de $H \in]0; 1[$, avec pour objectif de préserver l'écart entre les heuristiques naïves et l'optimum. Un écart significatif nous permet de mettre en exergue plus facilement la qualité des heuristiques que nous proposons. L'absence d'écart, quant à lui, se caractérise par un paysage plat et uniforme, pour lequel il s'avère, numériquement, plus difficile de distinguer les meilleures solutions. Nous souhaitons aussi que cet écart reste stable, où au moins ne disparaisse pas, lors N, M augmentent.

Le vecteur p est supposé constant durant la calibration. Bien que nous ayons testé différentes valeurs pour H, nous présentons dans les tables $H = 0.75$, $H = 0.99$, qui sont représentatives. Enfin, nous présentons également le paramétrage $\eta = 1$, qui correspond au cas d'absence de calibration ($g \equiv G$), pour donner un autre point de référence.

Dans les tables de calibrations suivantes, FS correspond à la vente à tout prix (*Fire Sale*) et LIS (*Linear Sale*) à la vente linéaire. Les colonnes correspondent à la fonction prototype G. Les résultats sont exprimés en pourcentage et mesurent la différence relative à l'optimum, obtenu par DP exacte :

$\eta_{0.75}$		$\frac{x}{1+x}$		$\frac{2}{\pi}\arctan(x)$		$1-\frac{2}{1+\sqrt{1+x}}$	
N	M	FS	LIS	FS	LIS	FS	LIS
10^1	10^2	33.44	0.84	37.85	0.52	23.58	1.43
10^1	10^3	33.45	0.84	37.86	0.53	23.58	1.43
10^1	10^4	33.45	0.84	37.86	0.53	23.58	1.43
10^1	10^5	33.45	0.84	37.86	0.53	23.58	1.43
10^1	10^6	33.45	0.84	37.86	0.53	23.58	1.43
10^2	10^3	36.65	0.09	40.9	0.05	26.77	0.24
10^2	10^4	36.65	0.09	40.9	0.05	26.77	0.24
10^2	10^5	36.65	0.09	40.9	0.05	26.77	0.24
10^3	10^5	36.97	0.01	41.19	0.01	27.1	0.03

$\eta_{0.99}$		$\frac{x}{1+x}$		$\frac{2}{\pi}\arctan(x)$		$1-\frac{2}{1+\sqrt{1+x}}$	
N	M	FS	LIS	FS	LIS	FS	LIS
10^1	10^2	18.27	6.04	20.42	7.81	5.48	0.94
10^1	10^3	18.38	6.17	20.52	7.92	5.51	0.98
10^1	10^4	18.38	6.17	20.52	7.92	5.51	0.98
10^1	10^5	18.38	6.17	20.52	7.92	5.51	0.98
10^1	10^6	18.38	6.17	20.52	7.92	5.51	0.98
10^2	10^3	22.63	1.97	25.00	2.12	7.00	0.53
10^2	10^4	22.65	2.00	25.00	2.14	7.08	0.62
10^2	10^5	22.65	2.00	25.01	2.14	7.08	0.62
10^3	10^5	23.11	0.24	25.49	0.23	7.28	0.18

$\eta = 1$		$\frac{x}{1+x}$		$\frac{2}{\pi}\arctan(x)$		$1-\frac{2}{1+\sqrt{1+x}}$	
N	M	FS	LIS	FS	LIS	FS	LIS
10^1	10^2	18.16	6.03	15.25	6.41	23.91	1.98
10^1	10^3	3.44	1.79	2.53	1.46	18.43	2.59
10^1	10^4	0.45	0.28	0.32	0.21	9.47	1.6
10^1	10^5	0.05	0.04	0.04	0.02	3.68	0.66
10^1	10^6	0.01	$< \epsilon$	$< \epsilon$	$< \epsilon$	1.25	0.23
10^2	10^3	4.61	1.18	3.38	1.08	22.06	0.96
10^2	10^4	0.68	0.3	0.47	0.23	11.9	0.89
10^2	10^5	0.09	0.05	0.06	0.04	4.79	0.45
10^3	10^5	0.09	0.03	0.06	0.02	4.92	0.14

Conclusions :

— Pour les fonctions $x \mapsto \frac{x}{1+x}$ et $x \mapsto \frac{2}{\pi}\arctan(x)$, la calibration $\eta_{0.99}$ a le plus grand pouvoir discriminant.

— Pour la fonction $x \mapsto 1-\frac{2}{1+\sqrt{1+x}}$, la calibration $\eta_{0.75}$ est légèrement meilleure pour les petites instances, mais cet avantage disparait rapidement quand $N \geq 2$. La

calibration $\eta_{0.99}$ est plus stable et possède un plus grand pouvoir discriminant pour les petites et moyennes instances.

— Pour le facteur d'échelle $\eta_{0.99}$, la distance de FS à l'optimum reste stable lorsque N et M augmentent. Dans le cas de vente linéaire LIS, elle est stable en M, mais décroit, bien que plus lentement que pour les autres facteurs d'échelle, en fonction de N.

— Enfin, nous remarquons que les fonctions $x \mapsto \frac{x}{1+x}$ et $x \mapsto \frac{2}{\pi}\arctan(x)$ ont approximativement la même vitesse de convergence (ce qui est lié à la similitude entre les expressions de leur dérivée première), tandis que $x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$ renvoie des résultats plus faibles pour la distance de l'heuristique LIS à l'optimum, qui est inférieure à 1% environ, pour toutes les instances.

Après avoir défini et calibré les fonctions de pénalités, nous passons aux expérimentations numériques sur les petites et moyennes instances.

5.2 Expérimentations numériques sur les petites et moyennes instances

Dans cette section, nous présentons les résultats de nos expérimentations numériques-sur les petites et moyennes instances, selon la terminologie introduite à la section 5.1.3. Nous présentons les résultats de résolution exacte par DP à la section 5.2.1 et discutons les tailles limites d'instances efficaces.

La méthode TSDP nécessite un choix préalable de la taille du grain. Nous décrivons le paramétrage de la DP à gros grain à la section 5.2.2, dont nous calculons le ratio de couverture. Puis, nous rappelons les résultats de complexité de la méthode TSDP, établis à la section 3.3.5.

Nous appliquons une approche similaire aux méthodes TSDP hybrides (*NLopt* ou gradient projeté, suivis de la DP bornée) à la section 5.2.3.
Le paramétrage étant effectué, nous présentons les résultats agrégés dans les deux sous-sections suivantes.

Nous présentons tout d'abord, à la section 5.2.4, les résultats agrégés des méthodes heuristiques et/ou issues de la DP (cf. chapitre 3). Nous comparons les performances méthodes

et discutons également de l'influence de la volatilité de p à l'aide d'un cas représentatif.
Puis, à la section 5.2.5, nous présentons les résultats obtenus par convexification.
Nous attachons, à la section 5.2.5.1, une attention particulière aux très petites instances ($N \leq 10$), pour lesquelles nous comparons la performance des algorithmes de *Branch & Bound* et des différents solvers.
Enfin, nous présentons les résultats agrégés de convexification pour les petites et moyennes instances à la section 5.2.5.2.

5.2.1 Résolution exacte des petites et moyennes instances :

Nous résolvons le problème $(PNS_{N,M})$ à l'optimum pour les petites et moyennes instances en utilisant l'algorithme de DP exacte de la section 3.2. Le temps CPU apparait dans le tableau ci-dessous.

CPU		$\frac{x}{1+x}$		$\frac{2}{\pi}\arctan(x)$		$1-\frac{2}{1+\sqrt{1+x}}$	
N	M	CST	AVG	CST	AVG	CST	AVG
10^1	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
10^1	10^3	0.03	0.02	0.03	0.02	0.03	0.02
10^1	10^4	1.77	1.77	1.78	1.78	1.77	1.79
10^1	10^5	181	182	179	178	176	176
10^2	10^3	0.19	0.19	0.18	0.18	0.18	0.18
10^2	10^4	18.83	18.93	17.68	17.63	17.55	17.53
10^2	10^5	1829	1777	1786	1763	1752	1754
10^3	10^5	17 550	17 943	17 576	17 856	17 488	17 949
10^1	10^6	18 141	18 253	18 261	18 142	17 876	18 440

TABLE 5.1 – Temps CPU - DP exacte

Points importants :

- La complexité temporelle est $O(N\,M^2)$ comme attendue.
- Pour $(N, M) = (2, 6)$, Nous avons seulement calculé le temps CPU dans le cas CST (p constant). La durée du calcul est d'environ 180 000 s, soit environ 50 heures, pour chaque fonction de pénalité, en accord avec la complexité temporelle attendue.
- Le temps de calcul ne dépend pas de p ni de la fonction de pénalité. Ce résultat est attendu car $(p_1, \cdots, p_N)$ et $\big(g(1), \cdots, g(N)\big)$ sont calculés et stockés en amont de l'algorithme de DP.
- Lorsque $(N, M) \geq (2, 5)$ (dans un sens général), la résolution à l'optimum prend

de quelques heures à plusieurs jours. Elle n'est donc pas utilisable en pratique.

La limite de taille d'instances efficaces, pour l'algorithme de DP exacte, se situe autour de $(N, M) = (2, 5)$ (dans un sens général). Par conséquent, pour certaines instances moyennes et pour les (très) grandes instances, la DP exacte n'est plus applicable d'un point de vue opérationnel, bien qu'elle renvoie une solution exacte en temps fini.

C'est pourquoi, nous nous intéressons maintenant aux bornes inférieures fournies par les heuristiques introduites à la section 3.3.

Bien que les heuristiques naïves (FS et LIS), ainsi que l'algorithme de recherche locale ILS soient directement applicables, la méthode TSDP dépend du grain, qui contrôle à la fois la taille des paquets et du bandeau autour de l'heuristique. Le calcul théorique du grain optimal (minimisant la complexité temporelle de la méthode) a été établi à la section 3.3.5.

Nous présentons tout d'abord les résultats de la méthode TSDP pour différentes valeurs de grain, puis nous introduisons son ratio de couverture optimale (proportion d'instances où l'optimum est atteint).

5.2.2 DP à gros grain et recherche dans un bandeau

Comme nous l'avons discuté à la section 3.3, la méthode TSDP discrète consiste à lancer un algorithme de DP à gros grain P, dans une première étape, puis à raffiner la solution en intensifiant la recherche dans le bandeau de taille homothétique $\lambda.P$.
Nous avons effectué les calculs de $P = 10$ à $M/10$, et $\lambda \in \{1, 5\}$ (nous avons testé différentes valeurs pour $\lambda \in [0; 10]$ et avons retenu les valeurs les plus représentatives).

Nous calculons tout d'abord le **ratio de couverture optimale** (ratio du nombre d'instances où l'optimum est atteint par le nombre total d'instances) en fonction du grain P. Dans le tableau ci-dessous, la première valeur correspond au ratio, exprimé en pourcentage. La seconde, entre parenthèses, exprime le nombre total d'instances :

$\lambda P =$	10	50	100	500	1000	5000	10000	50000
CST	27 (22)	95 (22)	23 (13)	100 (13)	14 (7)	100 (7)	0 (2)	100 (2)
AVG	42 (177)	97 (177)	49 (104)	99 (104)	44 (50)	98 (50)	33 (18)	100 (18)
TOTAL	41 (199)	97 (199)	46 (117)	99 (117)	40 (57)	98 (57)	30 (20)	100 (20)

TABLE 5.2 – Ratio de couverture optimale - TSDP discrète

Points importants :

— Un bandeau de taille $5P$ (i.e $\lambda = 5, \lambda\ P = 50, 500, 5000$ etc.) génère un meilleur ratio de couverture pour tout P, que $\lambda = 1$.

— Le ratio de couverture maximum, pour un nombre significatif d'instances, est atteint pour $P = 100$ et un bandeau de taille $\lambda P = 500$. Dans ce cas, l'optimum est atteint pour presque toutes les petites et moyennes instances.

— Les résultats sont similaires pour les échantillons CST et AVG.

Nous affichons maintenant le temps CPU pour TSDP avec le paramétrage $\lambda P = 500$:

CPU		$\frac{x}{1+x}$		$\frac{2}{\pi}\arctan(x)$		$1 - \frac{2}{1+\sqrt{1+x}}$	
N	M	CST	AVG	CST	AVG	CST	AVG
10^1	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
10^1	10^3	0.02	0.017	0.017	0.016	0.016	0.015
10^1	10^4	0.127	0.108	0.118	0.102	0.143	0.076
10^1	10^5	0.206	0.199	0.211	0.189	0.204	0.133
10^2	10^3	0.15	0.15	0.15	0.15	0.151	0.15
10^2	10^4	2.122	1.795	2.122	1.797	2.121	1.608
10^2	10^5	13.939	6.817	12.898	6.771	13.997	6.771
10^3	10^5	213.399	137.701	213.357	138.243	213.378	124.176
10^1	10^6	2.103	2.003	2.104	2.002	2.035	1.914
10^2	10^6	39.585	27.914	39.581	28.097	37.319	25.135

TABLE 5.3 – Temps CPU - TSDP discrète

Points importants :

— Le temps CPU est comparable pour les différentes fonctions de pénalité.

— La résolution prend moins d'une minute pour $(N, M) = (2, 6)$.

— Elle prend marginalement plus longtemps pour le cas constant (CST) que l'échantillon moyenné (AVG). En effet, pour les trajectoires de p très fluctuantes, avec des pics, la stratégie optimale concentre l'essentiel de l'écoulement sur ces pics, laissant les autres variables de décisions, avec de très faibles (voires nulles) valeurs.

Pour l'application financière ELBA, nous dirons que l'essentiel de temps de transactions ont lieu durant ces pics de prix, délaissant les transactions pendant les périodes creuses, ce qui est conforme à l'intuition.
C'est pourquoi le nombre effectif d'étapes pour déterminer l'optimum à partir de l'équation Bellman (cf. section 3.2) est en pratique plus faible.

Complexité : Nous avons précédemment établi à la section 3.3.5, la taille de grain P qui minimise la complexité temporelle de TSDP. En effet, en notant $N = 10^a, M = 10^b, P = 10^c$, et en prenant $\lambda = 5$ et en appliquant les résultats de complexité de la 3.3, nous obtenons une complexité temporelle minimale $O(10^{\frac{3a}{2}+b+1})$ ($\sim O(N^{\frac{3}{2}} \cdot M)$).
Elle est obtenue pour $c = \frac{1}{4}(2b - a - 2)$ et en prenant $P = \lfloor 10^c \rfloor$.

Nous remarquons cependant que cela tend à introduire des blocs non subdivisables (lorsque P ne divise plus M), qui doit être géré par l'algorithme. C'est un effet de bord sans impact sur les résultats présentés dans cette section.

Enfin, nous rappelons le gain obtenu, d'un facteur $10^{b-\frac{a}{2}-1}$ ($\sim \frac{M}{10\sqrt{N}}$), par rapport à la DP exacte qui est en $O(10^{a+2b})$ ($\sim O(N\,M^2)$). De plus le gain augmente lorsque M croit par rapport à N.

Nous abordons maintenant la version hybride de la méthode TSDP qui s'appuie sur la relaxation continue du problème traité.

5.2.3 TSDP en utilisant la relaxation continue

Comme nous l'avons indiqué à la section 3.3, la méthode TSDP peut aussi fonctionner avec une solution continue en première étape, puis utiliser la DP avec bornes, dans la seconde étape, pour intensifier la recherche dans le voisinage de cette solution (cf. section 3.3.2).
Dans ce cas, nous déterminons, en première étape, un maximum local du problème $(\overline{PNS_{N,M}})$ obtenu en utilisant, soit un algorithme de gradient projeté, ou bien directement *via* le solver *NLopt*, qui implémente la méthode de gradient (CCSAQ), issue de Svanberg [108], décrite à la section 3.4.2.
Nous rappelons que les résultats obtenus sont des maxima locaux du problème $(\overline{PNS_{N,M}})$, qui est non convexe. Cela signifie, que non seulement, ils ne fournissent pas le maximum

global du problème continu, mais surtout qu'ils ne sont pas, à priori, des bornes supérieures du problème $(PNS_{N,M})$ en variables entières.
Nous avons vérifié ce dernier point expérimentalement et avons trouvé des instances où le maximum global entier était plus élevé que le maximum local continu (avec une très faible différence néanmoins).
Nous commençons par comparer la performance et la stabilité des deux algorithmes, à la section 5.2.3.1, puis nous présentons, à la section 5.2.3.2, comme pour la méthode TSDP discrète, la ratio de couverture optimale, afin de déterminer la taille du bandeau.

5.2.3.1 Comparaison du gradient projeté et de *NLopt*

Nous présentons tout d'abord, dans les tables de résultats ci-dessous, la qualité des solutions (locales), avec les conventions suivantes :
La méthode de gradient projeté à pas constant est notée **GP** et l'appel au solver *NLopt* est noté **NLO**. Le meilleur des deux optima locaux est coché d'une croix (cf. symbole $\times$). L'autre valeur correspond à la différence relative entre les deux optima locaux multipliée par un facteur 10^6.
Par exemple, dans la seconde ligne $(N, M) = (1, 3)$, *NLopt* fournit le plus grand maximum local et le gradient projeté fournit une solution très proche $\frac{NLO-GP}{NLO} = 0.02 \cdot 10^{-6}$, pour une trajectoire constante.
$(*^1)$: parmi les 9 instances dont la trajectoire varie, l'instance n°6 ($\mu = 0,\ \sigma = 0.7$) n'a pas convergé en 600s. Le lecteur peut se reporter à la section 5.1, pour la description des instances AVG.
$(*^2)$: trop peu d'instances ont convergé pour établir une moyenne représentative.

Qualité		CST, $\frac{2}{\pi}arctan(x)$		AVG, $\frac{2}{\pi}arctan(x)$	
N	M	GP	NLO	GP	NLO
10^1	10^2	0.03	$\times$	0.03 $(*^1)$	$\times$
10^1	10^3	0.02	$\times$	0.04 $(*^1)$	$\times$
10^1	10^4	<0.01	$\times$	0.05 $(*^1)$	$\times$
10^1	10^5	0.02	$\times$	0.04 $(*^1)$	$\times$
10^1	10^6	0.02	$\times$	<0.01 $(*^1)$	$\times$
10^2	10^3	0.02	$\times$	DNC $(*^2)$	$\times$
10^2	10^4	0.04	$\times$	DNC $(*^2)$	$\times$
10^2	10^5	0.11	$\times$	DNC $(*^2)$	$\times$
10^3	10^5	DNC	$\times$	DNC	$\times$

TABLE 5.4 – Qualité - comparaison entre le gradient projeté et *NLopt*

Points importants :

— Les deux méthodes renvoient des résultats quasiment identiques. Cependant *NLopt* fait mieux pour presque toutes les instances que le gradient projeté. Il y a néanmoins quelques instances, dans le lot AVG, où le gradient projeté fait mieux.

— Cependant, le gradient projeté n'est pas assez stable. Les instances avec des trajectoires en dents de scie, comme celle mentionnée en astérisque ($*^1$) ne convergent pas ou très lentement avec notre algorithme de *GP*. Cela est du au pas constant de l'algorithme que nous avons empiriquement défini, mais qui pour certaines trajectoires, provoque un saut de gradient qui ralentit l'algorithme ou le fait diverger. Un pas optimal aurait probablement été plus stable, mais il nécessite de résoudre à chaque itération un problème d'optimisation fortement non linéaire du type $\alpha_i = \arg\max_{\alpha} f(x^i + \alpha \cdot \nabla f^i)$, ce qui peut s'avérer coûteux. Ce point est abordé en perspective à la section 6.2.

Nous nous intéressons ensuite au temps de calcul, qui sont présentés en secondes, dans le tableau ci-dessous.

CPU		CST, $\frac{2}{\pi}arctan(x)$		AVG, $\frac{2}{\pi}arctan(x)$	
N	M	GP	NLO	GP	NLO
10^1	10^2	0.26	0.38	0.28	0.44
10^1	10^3	0.37	0.35	0.40	0.40
10^1	10^4	0.46	0.43	0.51	0.39
10^1	10^5	0.56	0.01	0.63	0.08
10^1	10^6	0.66	$< \epsilon$	0.74	$< \epsilon$
10^2	10^3	85.70	0.71	DNC	1.30
10^2	10^4	128.12	0.06	DNC	0.97
10^2	10^5	165.67	0.08	DNC	0.15
10^3	10^5	DNC	10.08	DNC	5.12

TABLE 5.5 – Temps CPU - comparaison entre le gradient projeté et *NLopt*

Points importants :

— Pour $N = 10$, les temps de calculs sont très proches de l'ordre de la seconde.

— Pour $N \geq 10^2$, *NLopt* reste stable entre 1 et 10 s, tandis que le gradient projeté, prend quelques minutes, lorsqu'il converge.

Conclusions : Les deux méthodes se valent en terme de performance. Cependant *NLopt* est plus rapide et surtout beaucoup plus stable que notre algorithme de gradient projeté.

Par conséquent, dans toute la suite des expérimentations numériques, nous n'utliserons plus que *NLopt* en première étape de l'approche hybride.
Nous allons maintenant, comme pour la version discrète, nous intéresser à la taille du bandeau de la méthode TSDP hybride.

5.2.3.2 Taille du bandeau pour la méthode hybride

Nous utilisons donc un maximum local du problème $(\overline{PNS_{N,M}})$ obtenu en utilisant le solver *NLopt*, avec les bornes l_t, u_t définies à la section 3.3.2.
Nous conservons le paramétrage précédent ($\lambda = 5$, $P = 100$) pour pouvoir comparer les résultats. Le tableau ci-dessous montre le ratio de couverture en fonction de P :

$\lambda P =$	10	50	100	500	1000	5000
CST	50 (20)	65 (20)	73 (11)	100 (11)	100 (5)	100 (5)
AVG	56 (172)	81 (172)	90 (99)	100 (99)	100 (45)	100 (45)
TOTAL	56 (192)	79 (192)	88 (110)	100 (110)	100 (50)	100 (50)

TABLE 5.6 – Ratio de couverture optimale - TSDP hybride

Points importants :

— La relaxation continue avec $\lambda P = 500$ atteint l'optimum pour toutes les petites et moyennes instances.

Nous affichons de même les temps CPU de ce paramétrage $\lambda P = 500$:

CPU		$\frac{x}{1+x}$		$\frac{2}{\pi}\arctan(x)$		$1 - \frac{2}{1+\sqrt{1+x}}$	
N	M	CST	AVG	CST	AVG	CST	AVG
10^1	10^2	0.578	0.417	0.38	0.444	0.333	0.503
10^1	10^3	0.337	0.433	0.368	0.413	0.252	0.469
10^1	10^4	0.403	0.511	0.565	0.492	0.336	0.49
10^1	10^5	0.198	0.267	0.201	0.251	0.192	0.206
10^2	10^3	0.88	1.525	0.864	1.45	0.446	1.916
10^2	10^4	2.196	2.775	2.135	2.703	2.167	2.842
10^2	10^5	12.531	6.484	12.463	6.517	12.73	5.04
10^3	10^5	219.582	132.47	221.32	127.751	224.953	87.648
10^1	10^6	0.198	0.182	0.198	0.182	0.206	0.141
10^2	10^6	19.348	10.015	19.405	10.175	17.889	7.15

TABLE 5.7 – Temps CPU - TSDP hybride

Points importants :

— La méthode TSDP hybride (couplée à la relaxation continue) engendre des temps de calculs comparables à sa version à gros grain. Les différences résident dans la première étape (heuristique *NLopt* vs. DP à gros grain), tandis que la seconde étape de DP avec bornes est relativement stable entre les deux versions, car la taille du bandeau est identique.
Nous notons cependant que pour $(N, M) = (1, 6)$ or $(2, 6)$, La TSDP hybride est nettement plus rapide que la version à gros grain.

Les méthodes TSDP, discrète ou hybride, sont désormais clairement définies.
Dans la prochaine section, nous présentons les résultats agrégés de nos algorithmes (TSDP à gros grain et hybride, ILS et DP exacte) en plus des heuristiques, appliqués aux petites et moyennes instances.

5.2.4 Résultats agrégés des méthodes de DP

Les méthodes TSDP étant bien paramétrées, nous pouvons maintenant comparer nos bornes inférieures et supérieures. Les deux tables ci-dessous sont dédiées à la qualité et au temps CPU. Nous commençons par définir les notations suivantes, pour faciliter la lecture.

Notations dans les tables de résultats : FS (*firesale*) fait référence à la solution admissible de vente à tout prix, **LIS** à la vente linéaire (qui écoule $x_i = \frac{M}{N}$, à chaque pas de temps), **TS1** représente TSDP avec une DP gros grain et **TS2** sa version hybride avec une solution continue issue d'*NLopt*. Enfin **ILS** correspond au gradient discret décrit à la section 3.3.3 et initialisé avec la vente linéaire. **LS** fait référence à *LocalSolver* qui tourne avec un temps CPU proche de celui de l'algorithme qui fournit la meilleure borne inférieure (sous la contrainte du cap à 600s énoncé à la section 5.1.3). Enfin $\mathbf{UB}_2$ correspond à la borne supérieure obtenue par la majoration monotone de la section 3.5.

Les profils de résultats sont similaires pour les différentes fonctions de pénalité. Nous n'indiquons dans les deux tables suivantes, que la fonction de pénalité associée au prototype $G = \frac{2}{\pi} arctan(x)$, pour les échantillons CST et AVG :

Qualité		CST, $\frac{2}{\pi}arctan(x)$						
N	M	FS	LIS	ILS	LS	TS1	TS2	UB_2
10^1	10^2	20.42	7.81	0.41	0	0	0	38.19
10^1	10^3	20.52	7.92	0.02	$< \epsilon$	0	0	38.02
10^1	10^4	20.52	7.92	$< \epsilon$	$< \epsilon$	0	0	38.02
10^1	10^5	20.52	7.92	$< \epsilon$	$< \epsilon$	0	0	38.02
10^1	10^6	20.52	7.92	$< \epsilon$	$< \epsilon$	0	0	38.02
10^2	10^3	25	2.12	1.04	$< \epsilon$	0	0	364.61
10^2	10^4	25	2.14	0.13	0.01	0	0	364.56
10^2	10^5	25.01	2.14	$< \epsilon$	0.01	0	0	364.56
10^3	10^5	25.49	0.23	0.12	0.10	0	0	558.72

TABLE 5.8 – Qualité - Petites et moyennes instances (CST)

Qualité		AVG, $\frac{2}{\pi}arctan(x)$						
N	M	FS	LIS	ILS	LS	TS1	TS2	UB_2
10^1	10^2	21.66	10.47	0.96	0	0	0	139.4
10^1	10^3	21.77	10.58	0.72	$< \epsilon$	0	0	139.07
10^1	10^4	21.77	10.59	0.71	$< \epsilon$	0	0	139.07
10^1	10^5	21.77	10.59	0.71	$< \epsilon$	0	0	139.07
10^1	10^6	21.77	10.59	0.71	$< \epsilon$	0	0	139.07
10^2	10^3	28.48	7.77	5.26	$< \epsilon$	0	0	419.26
10^2	10^4	28.49	7.8	4.04	0.01	0	0	419.13
10^2	10^5	28.50	7.8	3.97	0.01	0	0	419.13
10^3	10^5	30.49	5.98	7.39	0.14	0	0	576.05

TABLE 5.9 – Qualité - Petites et moyennes instances (AVG)

Points importants :

— Comme attendu, les heuristiques naïves FS et LIS sont les moins performantes. l'heuristique FS fournit une valeur objectif environ $20\% - 25\%$ plus faible que l'optimum, tandis que LIS devient plus fine lorsque N augmente. Lorsque N et M sont du même ordre de grandeur, il y a suffisamment de variables de décision en comparaison du total M à atteindre. Nous pouvons donc écouler linéairement et une stratégie de vente est moins pertinente. C'est pourquoi l'heuristique LIS se rapproche alors de l'optimum.

— *LocalSolver* fournit une très bonne borne inférieure pour toutes les instance, mais rarement l'optimum. Son ratio de couverture optimale est de l'ordre de 10% pour l'échantillon CST et AVG.

— TS1 et TS2 atteignent l'optimum pour presque toutes les petites et moyennes instances.

— UB_2 obtenue par monotonie n'est pas fine, même pour les petites instances. La qualité est stable en M, mais se dégrade considérablement lorsque N augmente. Elle se rapproche de la borne naturelle $M\,\overline{p}$. Dans le cas constant, pour l'instance (3,5) l'écart relatif de cette borne à l'optimum est de 584%.

— L'algorithme de recherche local ILS renvoie de meilleurs résultats pour l'échantillon CST que pour AVG, a fortiori lorsque les tailles d'instances croissent. Dans l'échantillons AVG, ILS fait seulement marginalement mieux que la vente linéaire LIS. De plus, pour AVG les résultats sont hétérogènes selon le type de trajectoire pour p. Nous allons approfondir ce point ci-dessous.

Nous avons choisi un exemple représentatif pour l'instance de taille $(T, N) = (3, 5)$. Nous affichons les performances pour chacune des neufs diffusions de p (guidées par leurs moments μ, σ) dans le tableau ci-dessous :

Qualité	$\frac{2}{\pi}arctan(x)$	
N=10^3, M=10^5		
μ	σ	ILS
-0.05	0.10	2.13
-0.05	0.25	3.46
-0.05	0.70	8.76
0.00	0.10	2.06
0.00	0.25	5.02
0.00	0.70	13.67
+0.05	0.10	3.66
+0.05	0.25	5.33
+0.05	0.70	22.42

TABLE 5.10 – Qualité - Instance (3,5) - Trajectoires de p

Points importants :

— Les performances de l'algorithme ILS sont décroissantes en σ. Plus les trajectoires sont fluctuantes, ce qui peut s'interpréter comme une volatilité élevée dans le marché pour ELBA, moins bonne est la valeur objectif du maximum local. Plus les variations sont douces, meilleure est la performance d'ILS. En effet, les instances très volatiles concentrent l'essentiel de l'écoulement autour des pics. Nous en concluons que l'algorithme ILS est moins efficace pour ce type de profil en dent de scie.

Nous concluons cette section par l'étude des temps CPU :

CPU		CST, $\frac{2}{\pi}arctan(x)$						
N	M	FS	LIS	ILS	LS	TS1	TS2	UB_2
10^1	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	10	$< \epsilon$	0.38	$< \epsilon$
10^1	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	10	0.02	0.37	$< \epsilon$
10^1	10^4	$< \epsilon$	$< \epsilon$	$< \epsilon$	10	0.12	0.57	$< \epsilon$
10^1	10^5	$< \epsilon$	$< \epsilon$	$< \epsilon$	10	0.21	0.20	$< \epsilon$
10^1	10^6	$< \epsilon$	$< \epsilon$	$< \epsilon$	10	2.10	0.20	$< \epsilon$
10^2	10^3	$< \epsilon$	$< \epsilon$	0.01	10	0.15	0.86	$< \epsilon$
10^2	10^4	$< \epsilon$	$< \epsilon$	0.22	10	2.12	2.14	$< \epsilon$
10^2	10^5	$< \epsilon$	$< \epsilon$	0.73	600	12.90	12.46	$< \epsilon$
10^3	10^5	$< \epsilon$	$< \epsilon$	35.42	600	213.36	221.32	$< \epsilon$

TABLE 5.11 – Temps CPU - Petites et moyennes instances (CST)

CPU		AVG, $\frac{2}{\pi}arctan(x)$						
N	M	FS	LIS	ILS	LS	TS1	TS2	UB_2
10^1	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	10	$< \epsilon$	0.44	$< \epsilon$
10^1	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	10	0.02	0.41	$< \epsilon$
10^1	10^4	$< \epsilon$	$< \epsilon$	$< \epsilon$	10	0.11	0.49	$< \epsilon$
10^1	10^5	$< \epsilon$	$< \epsilon$	$< \epsilon$	10	0.19	0.25	$< \epsilon$
10^1	10^6	$< \epsilon$	$< \epsilon$	$< \epsilon$	10	2.02	0.18	$< \epsilon$
10^2	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	10	0.15	1.45	$< \epsilon$
10^2	10^4	$< \epsilon$	$< \epsilon$	0.07	10	1.8	2.7	$< \epsilon$
10^2	10^5	$< \epsilon$	$< \epsilon$	0.14	600	6.77	6.52	$< \epsilon$
10^3	10^5	$< \epsilon$	$< \epsilon$	2.92	600	138.24	127.75	$< \epsilon$

TABLE 5.12 – Temps CPU - Petites et moyennes instances (AVG)

Points importants :

— Pour les petites et moyennes instances, tous les algorithmes proposés convergent relativement rapidement, en quelques minutes.

— Si l'on excepte les heuristiques naïves, ILS est le plus rapide des algorithmes, bien qu'il ne ne renvoie pas nécessairement un bon maximum local, suivi par les méthodes TSDP.

— FS, LIS et UB_2 consistent en une formule directe qui se calcule presque instantanément pour pour les couples (N, M) considérés.

Nous abordons maintenant les expérimentations numériques des méthodes de convexification. Elles fournissent des solutions admissibles, mais aussi des bornes supérieures et permettent donc d'encadrer l'optimum.

5.2.5 Résolution par convexification

Nous présentons, dans cette section, les performances des méthodes de *Branch & Bound*, introduites à la section 4.5, qui s'appuient sur la relaxation convexe du problème factorisé équivalent, établie à la section 4.3.3. Nous comparons notamment ces résultats à ceux des différents solvers dédiés, présentés à la section 5.1.

Nous étudions tout d'abord les très petites instances ($3 \leq N < 10$, $10^2 \leq M \leq 10^4$, en notation directe), la section 5.2.5.1.
En effet, le comportement des algorithmes de *Branch & Bound* et des solvers sont déjà représentatifs pour ces instances et leur comportement relatif reste similaire lorsqu'on passe à l'échelle.

Puis, à la section 5.2.5.2, nous présentons les résultats agrégés de convexification pour les petites et moyennes instances.

5.2.5.1 Très petites instances

Nous nous intéressons dans cette section aux très petites instances et commençons par quelques remarques préliminaires.

Les solvers *Baron*, *Couenne* et *Scip* ne prennent pas en charge l'arctangente, au moins dans les versions mentionnées à la section 5.1. Seul le solver *Bonmin* en est actuellement capable.

Nous travaillons donc, dans cette section sur les autres fonctions de pénalité dont les prototypes sont $x \mapsto \dfrac{x}{1+x}$ et $x \mapsto 1 - \dfrac{2}{1+\sqrt{1+x}}$

Les algorithmes de *Branch & Bound* de la section 4.5, ainsi que les solvers sont appelés dans les mêmes conditions, sans initialisation et sans pré-traitement.

Enfin, nous relevons la limite de temps de calcul à 10 800 secondes (3 heures).

Nous introduisons les notations suivantes dans les tables de résultats ci-dessous, pour en faciliter la lecture :
Les colonnes portant le nom du solver (cf. section 5.1.3), indique une résolution directe par celui-ci, sans aucun pré-traitement, sur le problème $(PNS_{N,M})$ en variables entières. *Principal* fait référence à l'algorithme de *Branch & Bound* principal, défini à la section 4.5.1. De manière analogue, *Explo Large* fait référence à l'exploration en largeur d'abord, défini à la section 4.5.2.

Dans le tableau suivante, nous présentons la qualité des bornes inférieures obtenues par les différents algorithmes.
Les valeurs représentent l'écart relatif à l'optimum global, obtenu par DP exacte. La valeur «$< \epsilon$» signifie, que l'écart est inférieur à 0.01%, bien que l'optimum ne soit pas nécessairement atteint. Nous précisons dans les commentaires et par l'analyse des bornes supérieures, lorsqu'il est atteint.

Nous commençons par la fonction de pénalité $x \mapsto \frac{x}{1+x}$:

Qualité - **Bornes Inférieures**				CST, $G : x \mapsto \frac{x}{1+x}$			
N	M	Principal	Explo Large	*Baron*	*Bonmin*	*Couenne*	*Scip*
3	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
4	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
5	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
6	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	0.07
7	10^2	$< \epsilon$	$< \epsilon$	0.03	$< \epsilon$	$< \epsilon$	0.21
8	10^2	$< \epsilon$	$< \epsilon$	0.33	$< \epsilon$	0.02	1.95
9	10^2	$< \epsilon$	0.46	0.22	$< \epsilon$	0.01	3.45
10	10^2	0.03	1.03	0.02	$< \epsilon$	0.03	1.16
3	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
4	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
5	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	0.38
6	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	0.55
7	10^3	$< \epsilon$	0.03	$< \epsilon$	$< \epsilon$	$< \epsilon$	0.6
8	10^3	0.02	0.62	$< \epsilon$	$< \epsilon$	$< \epsilon$	1.91
9	10^3	0.04	1.41	$< \epsilon$	$< \epsilon$	$< \epsilon$	1.3
10	10^3	0.19	2.02	$< \epsilon$	$< \epsilon$	$< \epsilon$	3.47
3	10^4	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
4	10^4	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	1.92
5	10^4	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	0.62
6	10^4	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	2.66
7	10^4	$< \epsilon$	0.03	$< \epsilon$	$< \epsilon$	$< \epsilon$	4.63
8	10^4	0.01	0.64	$< \epsilon$	$< \epsilon$	$< \epsilon$	17.32
9	10^4	0.11	1.46	$< \epsilon$	$< \epsilon$	$< \epsilon$	17.9
10	10^4	0.28	2.16	$< \epsilon$	$< \epsilon$	$< \epsilon$	5.06

TABLE 5.13 – Bornes Inf. (Convexification) - Très petites instances - $x \mapsto \frac{x}{1+x}$

Points importants :

— Les bornes inférieures sont égales ou très proches de l'optimum pour les algorithmes de *Branch & Bound* ainsi que pour les différents solvers, à l'exception de *Scip*.

— L'algorithme Principal fait à peu prés aussi bien que *Baron*, *Bonmin* et *Couenne*, pour toutes les très petites instances.

— Bien qu'il ne garantisse pas l'optimalité, *Bonmin* atteint en pratique l'optimum pour toutes les très petites instances, pour cette fonction de pénalité.

— L'algorithme d'exploration en largeur, fait aussi bien que les solvers pour $N \leq 6$, puis donne de moins bons résultats pour $N \geq 7$ et $M = 10^3$ ou $M = 10^4$.

— Nous notons cependant que l'algorithme Principal est uniformément meilleur que l'exploration en largeur d'abord, pour les très petites instances.
— Enfin, le solver *Scip* retourne les moins bons résultats avec des bornes inférieures significativement plus faibles que les autres algorithmes.

Nous nous intéressons ensuite aux bornes supérieures. Lorsque l'optimalité est prouvée, la borne supérieure est égale à la borne inférieure (au seuil de tolérance près). Cependant, dans les cas où les algorithmes ne convergent pas en 3h, les bornes supérieures permettent d'encadrer l'optimum.
Comme précédemment, dans le tableau de résultats suivants, les valeurs représentent l'écart relatif à l'optimum.

Qualité - **Bornes Supérieures**				CST, $G : x \mapsto \frac{x}{1+x}$		
N	M	Principal	Explo Large	*Baron*	*Couenne*	*Scip*
3	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
4	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
5	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
6	10^2	$< \epsilon$	$< \epsilon$	39.04	0.65	132.57
7	10^2	$< \epsilon$	$< \epsilon$	353.33	40.01	187.67
8	10^2	$< \epsilon$	$< \epsilon$	468.85	82.32	370.19
9	10^2	0.19	183.68	409.88	134.94	423.27
10	10^2	1.23	242.47	499.63	168.65	468.99
3	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
4	10^3	$< \epsilon$	$< \epsilon$	5.26	$< \epsilon$	$< \epsilon$
5	10^3	$< \epsilon$	$< \epsilon$	182.74	2.27	157.66
6	10^3	0.02	10.66	511.09	48.9	283.96
7	10^3	0.3	65.92	627.4	93.5	486.21
8	10^3	1.01	171.25	638.12	170.31	545.56
9	10^3	2.69	310.93	636.84	166.73	626.17
10	10^3	10.55	359.91	642.51	207.56	629.58
3	10^4	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
4	10^4	$< \epsilon$	$< \epsilon$	6.69	$< \epsilon$	1.92
5	10^4	$< \epsilon$	0.05	568.61	8.11	302.81
6	10^4	0.09	14.48	665.92	68.73	607.2
7	10^4	0.44	80.3	663.29	106.38	631.84
8	10^4	1.28	183.15	655.42	169.17	655.31
9	10^4	3.4	315.36	652.46	167.95	653.12
10	10^4	13.58	364.24	648.3	211.86	644.87

TABLE 5.14 – Bornes Sup. (Convexification) - Très petites instances - $x \mapsto \frac{x}{1+x}$

Points importants :

- Pour $N \leq 8, M = 10^2$ les algorithmes de *Branch & Bound* résolvent le problème à l'optimum.
- Pour $M = 10^3$, 10^4 leur résolution à l'optimum s'arrête très vite des que $N \geq 6$.
- L'algorithme Principal renvoie de très bonnes bornes supérieures pour les très petites instances ($< 5\%$). Pour $N = 10, M = \{10^3, 10^4\}$, l'encadrement est à 10-15% près.
- Comme pour les bornes inférieures, l'algorithme Principal fait uniformément mieux que l'exploration en largeur, mais la différence de qualité est bien plus marquée sur les bornes supérieures. En effet l'exploration en largeur renvoie un écart à plus de 100% pour $N \geq 8$.
- L'étude des bornes supérieures des solvers révèle des résultats très surprenant, par leur hétérogénéité :
- Les solvers *Baron*, *Couenne* et *Scip* ne résolvent à l'optimum que quelques instances : $N = \{3, 4, 5\}$, $M = 10^2$ et $N = \{3, 4\}$, $M = \{10^3, 10^4\}$. Par contre, leurs bornes supérieures sont uniformément beaucoup moins fines que celles des *Branch & Bound*.
- Les solvers *Baron* et *Scip* notamment donnent de très mauvaises bornes supérieures. Elles tendent très vite vers la borne naturelle $M\,\overline{p}$, qui se trouve numériquement entre 650 et 700% (en écart relatif à l'optimum) pour les instances considérées. En 3h de calcul, les solvers renvoient alors une borne supérieure peu pertinente.
- Nous remarquons que *Scip* renvoie les plus mauvais résultats, lorsqu'il résout par lui-même, cependant, lorsqu'il est appelé pour résoudre la relaxation convexe, sur les multiples sous-domaines, dans le cadre d'un *Branch & Bound*, ses performances sont meilleures que le solver seul, mais également meilleures que *Baron* et *Couenne* qui sont nos principaux benchmark.
- Enfin, comme indiqué à la section 5.1.1, *Bonmin* ne renvoie pas de bornes supérieures.

Nous présentons enfin les temps CPU des différentes méthodes. «$> 3h$» signifie que l'algorithme n'a pas convergé en 3h.

CPU				CST, $G : x \mapsto \frac{x}{1+x}$			
N	M	Principal	Explo Large	*Baron*	*Bonmin*	*Couenne*	*Scip*
3	10^2	0.67	1.56	0.78	0.1	1.01	0.27
4	10^2	5.17	7.83	28	0.13	24	14
5	10^2	45	62	649	0.18	513	514
6	10^2	226	357	$> 3h$	0.2	$> 3h$	$> 3h$
7	10^2	925	1619	$> 3h$	0.22	$> 3h$	$> 3h$
8	10^2	4271	7476	$> 3h$	0.26	$> 3h$	$> 3h$
9	10^2	$> 3h$	$> 3h$	$> 3h$	0.25	$> 3h$	$> 3h$
10	10^2	$> 3h$	$> 3h$	$> 3h$	0.37	$> 3h$	$> 3h$
3	10^3	2.15	3.64	16	0.11	4.33	0.9
4	10^3	48	62	$> 3h$	0.15	269	1427
5	10^3	969	1231	$> 3h$	0.22	$> 3h$	$> 3h$
6	10^3	$> 3h$	$> 3h$	$> 3h$	0.25	$> 3h$	$> 3h$
7	10^3	$> 3h$	$> 3h$	$> 3h$	0.32	$> 3h$	$> 3h$
8	10^3	$> 3h$	$> 3h$	$> 3h$	0.34	$> 3h$	$> 3h$
9	10^3	$> 3h$	$> 3h$	$> 3h$	0.37	$> 3h$	$> 3h$
10	10^3	$> 3h$	$> 3h$	$> 3h$	0.38	$> 3h$	$> 3h$
3	10^4	7.15	9.64	101	0.15	10	3.53
4	10^4	512	645	$> 3h$	0.23	842	158
5	10^4	$> 3h$	$> 3h$	$> 3h$	0.3	$> 3h$	$> 3h$
6	10^4	$> 3h$	$> 3h$	$> 3h$	0.36	$> 3h$	$> 3h$
7	10^4	$> 3h$	$> 3h$	$> 3h$	0.45	$> 3h$	$> 3h$
8	10^4	$> 3h$	$> 3h$	$> 3h$	0.59	$> 3h$	$> 3h$
9	10^4	$> 3h$	$> 3h$	$> 3h$	1.04	$> 3h$	$> 3h$
10	10^4	$> 3h$	$> 3h$	$> 3h$	0.74	$> 3h$	$> 3h$

TABLE 5.15 – Temps CPU (Convexification) - Très petites instances - $x \mapsto \frac{x}{1+x}$

Points importants :

- Les algorithmes de *Branch & Bound* que nous proposons sont significativement plus rapides que les solvers *Baron*, *Couenne* et *Scip*.
- Cependant leur complexité empirique semble exponentielle en N et polynomiale en M. Cependant elle reste difficile à mesurer en raison du faible nombre de résolutions à l'optimum.
- Comme précédemment, notre algorithme Principal est plus rapide que l'exploration en largeur.
- Comme pour la qualité, le solver *Bonmin* est extrêmement rapide comparé aux autres méthodes, et converge très rapidement pour toutes les très petites instances, en quelques secondes.

Nous présentons également les tables de résultats analogues pour la fonction de pénalité $x \mapsto 1 - \dfrac{2}{1+\sqrt{1+x}}$:

Qualité - **Bornes Inférieures**				CST, $G : 1 - \frac{2}{1+\sqrt{1+x}}$			
N	M	Principal	Explo Large	*Baron*	*Bonmin*	*Couenne*	*Scip*
3	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon(*)$	$< \epsilon$	$< \epsilon$
4	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
5	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
6	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon(*)$	$< \epsilon$
7	10^2	$< \epsilon$	$< \epsilon$	0.01	$< \epsilon$	$< \epsilon$	$< \epsilon$
8	10^2	$< \epsilon$	0.03	0.01	$< \epsilon$	$< \epsilon$	0.08
9	10^2	0.01	0.28	$< \epsilon$	$< \epsilon$	$< \epsilon$	0.13
10	10^2	0.04(*)	0.36	$< \epsilon$	$< \epsilon$	$< \epsilon$	0.33
3	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
4	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
5	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	0.03
6	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon(*)$	0.03
7	10^3	$< \epsilon$	0.03	$< \epsilon$	(NO) $< \epsilon$	$< \epsilon$	0.09
8	10^3	0.03	0.32	$< \epsilon$	$< \epsilon$	$< \epsilon$	0.35
9	10^3	0.18	0.48	$< \epsilon$	$< \epsilon$	$< \epsilon$	0.57
10	10^3	0.19	0.86	$< \epsilon$	$< \epsilon$	$< \epsilon$	1.08
3	10^4	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
4	10^4	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
5	10^4	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	0.04
6	10^4	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon(*)$	0.34
7	10^4	0.04	0.27	$< \epsilon$	$< \epsilon$	$< \epsilon$	1.2
8	10^4	0.12	0.7	$< \epsilon$	$< \epsilon$	$< \epsilon$	0.78
9	10^4	0.21	0.83	$< \epsilon$	$< \epsilon$	$< \epsilon$	2.29
10	10^4	0.22	1.15	$< \epsilon$	$< \epsilon$	$< \epsilon$	1.62

TABLE 5.16 – Bornes Inf. (Convexification) - Très petites instances - $x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$

Qualité - **Bornes Supérieures**				CST, $G : x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$		
N	M	Principal	Explo Large	*Baron*	*Couenne*	*Scip*
3	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
4	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
5	10^2	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
6	10^2	$< \epsilon$	$< \epsilon$	5.26	DNC	$< \epsilon$
7	10^2	$< \epsilon$	$< \epsilon$	63.65	$< \epsilon$	$< \epsilon$
8	10^2	0.09	227.47	116.69	$< \epsilon$	6.7
9	10^2	0.48	366.11	170.71	$< \epsilon$	16.16
10	10^2	13.88(∗)	535.38	354.8	0.03	25.61
3	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
4	10^3	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
5	10^3	$< \epsilon$	$< \epsilon$	17.3	$< \epsilon$	8.62
6	10^3	0.04	66.28	94.61	DNC	12.5
7	10^3	0.22	178.46	299.3	1.51	16.2
8	10^3	4.59	357.64	380.37	9.46	32.19
9	10^3	64.96	545.5	486.7	19.54	98.64
10	10^3	139.99	657.97	521.47	24.04	105.9
3	10^4	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$	$< \epsilon$
4	10^4	$< \epsilon$	$< \epsilon$	5.26	$< \epsilon$	0.65
5	10^4	$< \epsilon$	11.37	5.26	$< \epsilon$	51.63
6	10^4	0.1	124.74	391.77	DNC	24.98
7	10^4	1.97	310.18	772.16	5.33	207.89
8	10^4	37.61	517.72	363.63	19.85	176.95
9	10^4	118.16	642.37	503.74	36.01	624.87
10	10^4	178.72	730.03	700.74	48.3	407.78

TABLE 5.17 – Bornes Sup. (Convexification) - Très petites instances - $x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$

CPU				CST, $G : 1 - \frac{2}{1+\sqrt{1+x}}$			
N	M	Principal	Explo Large	*Baron*	*Bonmin*	*Couenne*	*Scip*
3	10^2	0.73	1.94	0.52	DNC	0.27	0.21
4	10^2	14	18	3.24	0.12	0.75	1.91
5	10^2	152	184	1911	0.15	2.87	40
6	10^2	923	1124	$> 3h$	0.22	DNC	1356
7	10^2	4647	6032	$> 3h$	0.22	98	9501
8	10^2	$> 3h$	$> 3h$	$> 3h$	0.3	639	$> 3h$
9	10^2	$> 3h$	$> 3h$	$> 3h$	0.35	1917	$> 3h$
10	10^2	DNC	$> 3h$	$> 3h$	0.39	$> 3h$	$> 3h$
3	10^3	3.67	5.54	4.38	0.12	0.61	3.7
4	10^3	163	183	123	0.17	7.74	182
5	10^3	5924	6605	$> 3h$	0.24	184	$> 3h$
6	10^3	$> 3h$	$> 3h$	$> 3h$	0.31	DNC	$> 3h$
7	10^3	$> 3h$	$> 3h$	$> 3h$	0.31	$> 3h$	$> 3h$
8	10^3	$> 3h$	$> 3h$	$> 3h$	0.32	$> 3h$	$> 3h$
9	10^3	$> 3h$	$> 3h$	$> 3h$	0.4	$> 3h$	$> 3h$
10	10^3	$> 3h$	$> 3h$	$> 3h$	0.54	$> 3h$	$> 3h$
3	10^4	35	42	38	0.13	0.63	51
4	10^4	3186	3493	$> 3h$	0.33	14	$> 3h$
5	10^4	$> 3h$	$> 3h$	$> 3h$	0.28	1450	$> 3h$
6	10^4	$> 3h$	$> 3h$	$> 3h$	0.39	DNC	$> 3h$
7	10^4	$> 3h$	$> 3h$	$> 3h$	0.33	$> 3h$	$> 3h$
8	10^4	$> 3h$	$> 3h$	$> 3h$	0.54	$> 3h$	$> 3h$
9	10^4	$> 3h$	$> 3h$	$> 3h$	0.4	$> 3h$	$> 3h$
10	10^4	$> 3h$	$> 3h$	$> 3h$	0.53	$> 3h$	$> 3h$

TABLE 5.18 – Temps CPU (Convexification) - Très petites instances - $x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$

Points importants :

— Pour cette fonction de pénalité, les résultats sont plus disparates. Globalement, les *Branch & Bound* régressent au profit des solvers.

— L'algorithme Principal fournit encore des bornes fines lorsque $N \leq 8$. Cependant les bornes de l'exploration en largeur se détériorent significativement, au point de tendre vers la borne naturelle dans les plus «grandes» instances.

Nous notons que l'algorithme Principal a renvoyé une erreur, avant la fin du temps imparti, lors de la résolution de l'instance $(N, M) = (10, 100)$. Il retourne cependant un encadrement de l'optimum. Nous notons $(*)$ les valeurs concernées des tableaux de résultats.

— Les bornes des solvers sont plus fines, ils résolvent un plus grand nombre d'instances à l'optimum. Leurs temps de résolutions sont également un peu plus faibles.

— Le solver *Bonmin* obtient des performances quasi similaires pour cette fonction de pénalité : ses solutions admissibles sont optimales pour toutes les très petites instances, sauf pour l'instance $(7, 10^3)$. Nous la notons NO (non optimal) dans le tableau.

On note également qu'il retourne une erreur interne pour $N = 3$, $M = 10^2$. Nous récupérons sa meilleure solution admissible qui est très proche de l'optimum. Nous le notons également $(*)$ dans le tableau.

— *Couenne* et *Scip* sont les solvers dont les performances s'améliorent le plus.

— *Couenne* résout à l'optimum jusque $N = 9$, lorsque $M = 100$ (contre $N = 5$ précédemment). Les bornes supérieures de *Couenne* sont bien meilleures (environ 4 à 5 fois plus fines) pour toutes les très petites instances, avec une borne supérieure maximale de l'ordre de 50% (contre 200% auparavant).

— Les bornes supérieures de *Baron* s'améliorent également, mais elles restent moins fines que celles de l'algorithme principal.

— Les bornes supérieures de *Couenne* sont désormais proches, voire un peu meilleures, pour certaines instances $N \geq 8$ que celles de l'algorithme principal.

— Nous remarquons également que, *Couenne*, s'arrête, avant la fin du temps imparti et renvoie un code sortie d'infaisabilité du problème, spécifiquement lorsque $N = 6$, et ce, pour tout $M = 10^2,\ 10^3,\ 10^4$. Il ne renvoie dans ce cas que la meilleure solution admissible trouvée, qui est déjà très proche de l'optimum. Nous notons également $(*)$ dans les tableaux.

— Les performances du *Scip* s'améliorent significativement, bien qu'il reste le solver le moins performant pour notre problème.

— A la différence de la fonction de pénalité précédente, nous remarquons que *Scip*, utilisé seul s'améliore, mais que l'algorithme Principal qui l'utilise pour résoudre le problème relâché convexe se dégrade, bien qu'il reste au niveau de *Couenne* et meilleur que *Baron*.

Conclusions sur les très petites instances :

Pour les deux fonctions de pénalité étudiées, les différentes méthodes retournent toutes de très bonnes bornes inférieures.

Cependant les résultats sont très hétérogènes sur la qualité des bornes supérieures et les temps de résolutions associés, selon les solvers et les fonctions de pénalité.

Notre algorithme le plus performant est l'algorithme principal.
Il renvoie d'aussi bonnes inférieures, (solutions admissibles) que les solvers de l'état de l'art *Baron* et *Couenne*, qui sont dédiés à la résolution par relaxation convexe.
Par contre, de manière assez surprenante, ses bornes supérieures sont bien meilleures que celles de *Baron*, pour les deux fonctions.
Pour la première fonction de pénalité $x \mapsto \frac{x}{1+x}$, l'algorithme Principal fait beaucoup mieux que *Couenne*.
Par contre, pour la seconde, $x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$, les deux méthodes se valent lorsque $N \leq 7$ et *Couenne* renvoie des bornes supérieures plus fines pour $N \geq 8$. Nous supposons que cela provient de la forme beaucoup plus abrupte de la fonction objectif, associée à la première fonction de pénalité, par rapport la seconde, d'apparence concave en dimension 3, comme l'indique les graphes de la section 5.1.4. Il aurait été intéressant de comparer également l'arctangente, dont le graphe objectif est proche de celui de $x \mapsto \frac{x}{1+x}$. Cependant, elle n'est pas, à ce jour, prise en charge par les solvers *Couenne* et *Scip*.

Le solver *Scip*, qui est également un solver de l'état de l'art est le moins performant des solvers dans les deux cas.
Par contre *Scip* utilisé comme solver «convexe» dans l'algorithme principal, renvoie les très bons résultats discutés précédemment.

La complexité empirique de nos méthodes et des solvers (sauf *Bonmin*) semble exponentielle en N et polynomiale en M. Cela restreint le nombre d'instances résolues à l'optimum, mais nous fournissons une borne supérieure souvent fine.

Enfin, le solver *Bonmin* est étonnamment performant. Comme discuté à la section 5.1.1, nous avons profité d'une synergie de modélisation, grâce à *AMPL*, pour le rajouter et enrichir nos comparaisons.
Bien qu'il ne garantisse pas l'optimalité, *Bonmin* atteint en pratique sur la quasi-totalité des très petites instances, pour les deux fonctions de pénalité, très rapidement (quelques secondes). De plus, il est le seul solver qui prenne en charge la fonction arctangente.

Nous allons maintenant agréger ces résultats, sur les petites et moyennes instances, à la section suivante.

5.2.5.2 Résultats agrégés de convexification

Nous présentons, dans cette section, les résultats agrégés de convexification pour les petites et moyennes instances.
Afin de préserver la lisibilité des tableaux, nous ne gardons que les algorithmes les plus performants.
Nous conservons l'algorithme principal, qui est notre meilleur algorithme de *Branch & Bound*, ainsi que les solvers *Couenne*, qui est notre solver de référence et *Bonmin*, qui affiche incontestablement les meilleurs performances.
Nous présentons les résultats pour la trajectoire constante (CST), pour les deux fonctions de pénalité mentionnées précédemment.

Bornes Inf.		CST, $G : x \mapsto \frac{x}{1+x}$			CST, $G : x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$		
N	M	Principal	*Bonmin*	*Couenne*	Principal	*Bonmin*	*Couenne*
10^1	10^2	0.03	*Opt*	0.03	0.04	*Opt*	$< \epsilon$
10^1	10^3	0.19	*Opt*	$< \epsilon$	0.19	*Opt*	$< \epsilon$
10^1	10^4	0.28	*Opt*	$< \epsilon$	0.22	*Opt*	$< \epsilon$
10^1	10^5	0.58	*Opt*	$< \epsilon$	0.3	*Opt*	$< \epsilon$
10^1	10^6	0.59	(NO) $< \epsilon$	$< \epsilon$	0.36	(NO) $< \epsilon$	$< \epsilon$
10^2	10^3	7.21	*Opt*	0.06	1.54	*Opt*	0.03
10^2	10^4	6.62	(NO) $< \epsilon$	$< \epsilon$	1.6	(NO) $< \epsilon$	$< \epsilon$
10^2	10^5	6.52	*Opt*	$< \epsilon$	1.13	(NO) $< \epsilon$	$< \epsilon$
10^3	10^5	11.84	(NO) $< \epsilon$	$< \epsilon$	1.3	(NO) $< \epsilon$	$< \epsilon$

TABLE 5.19 – Bornes Inf. (Convexification) - Petites et moyennes instances

Points importants :

- Comme pour les très petites instances, *Bonmin* et *Couenne* renvoient des solutions admissibles (bornes inférieures), très proches de l'optimum.
- Le ratio de couverture optimale de *Bonmin* est de 61% environ. Il fournit l'optimum pour 11 instances sur 18. C'est moins bon que les méthodes TSDP qui l'atteignent pour presque toutes les petites et moyennes instances. Comme précédemment, nous notons (NO) les instances non résolues de manière globale, dans le tableau.
- Les bornes inférieures de l'algorithme Principal sont moins fines que celles obtenues

pour les très petites instances, mais elles restent correctes pour la fonction $x \mapsto \frac{x}{1+x}$ entre 0 et 12% d'écart à l'optimum selon les instances, et très bonnes pour la fonction $x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$, avec moins de 2% d'écart relatif à l'optimum.

Cependant, les résultats des sections précédentes (cf. section 5.2.4) ont montré que les méthodes à base de DP atteignaient l'optimum pour les petites et moyennes instances, dans quasiment tous les cas.

L'intérêt fondamental des méthodes de convexification, lorsqu'elles ne fournissent pas l'optimum, est de fournir une borne supérieure permettant de l'encadrer plus précisément.

Nous présentons dans les deux tables ci-dessous (une par fonction de pénalité), les bornes supérieures et les comparons à la borne naïve UB_2 établie par monotonie à la section 3.5.

Bornes Sup.		CST, $G : x \mapsto \frac{x}{1+x}$		
N	M	Principal	*Couenne*	UB_2
10^1	10^2	1.23	168.65	36.89
10^1	10^3	10.55	207.56	36.71
10^1	10^4	13.58	211.86	36.71
10^1	10^5	15.1	219.01	36.71
10^1	10^6	14.78	215.39	36.71
10^2	10^3	607.33	609.83	292.02
10^2	10^4	609.52	609.64	291.91
10^2	10^5	609.56	607.22	291.91
10^3	10^5	605.34	max	548.19

TABLE 5.20 – Bornes Sup. (Convex.) - Petites et moyennes instances - $x \mapsto \frac{x}{1+x}$

Bornes Sup.		CST, $G : x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$		
N	M	Principal	*Couenne*	UB_2
10^1	10^2	13.88	0.03	11.13
10^1	10^3	139.99	24.04	11.09
10^1	10^4	178.72	48.3	11.09
10^1	10^5	195.14	110.25	11.09
10^1	10^6	254.51	110.86	11.09
10^2	10^3	716.12	753.22	58.72
10^2	10^4	743.93	746.61	58.58
10^2	10^5	722.08	741.91	58.57
10^3	10^5	750.64	max	192.78

TABLE 5.21 – Bornes Sup. (Convex.) - Petites et moyennes instances - $x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$

Points importants :

- Pour $N = 10$ et pour la fonction de pénalité $x \mapsto \frac{x}{1+x}$, l'algorithme Principal donne de très bonnes bornes supérieures ($\sim 15\%$ d'écart), tandis que celles de *Couenne* sont en multiples ($> 100\%$).
- Pour la fonction de pénalité, $x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$, c'est le contraire. *Couenne* retourne des bornes bien plus fines que celles de l'algorithme Principal.
- Pour $N \geq 100$, l'algorithme Principal et *Couenne* retournent des bornes qui ne sont pas pertinentes, car elles sont assez proches de la borne naturelle. Pour l'instance $(3, 5)$, *Couenne* renvoie une borne supérieure infinie (notée «max» dans le tableau).
- UB_2 est donc la meilleure borne supérieure à partir de $(N, M) \geq (2, 3)$ (dans le tableau) pour la première fonction de pénalité et $(N, M) \geq (1, 5)$, pour la seconde.

Nous présentons enfin les tableaux de temps CPU :

CPU		CST, $G : x \mapsto \frac{x}{1+x}$			
N	M	Principal	*Bonmin*	*Couenne*	DP Exacte
10^1	10^2	$> 3h$	0.37	$> 3h$	$< \epsilon$
10^1	10^3	$> 3h$	0.38	$> 3h$	0.03
10^1	10^4	$> 3h$	0.74	$> 3h$	1.77
10^1	10^5	$> 3h$	0.58	$> 3h$	181
10^1	10^6	$> 3h$	0.56	$> 3h$	18 141
10^2	10^3	$> 3h$	*Opt* 500	$> 3h$	0.19
10^2	10^4	$> 3h$	423	$> 3h$	18.83
10^2	10^5	$> 3h$	*Opt* 324	$> 3h$	1829
10^3	10^5	$> 3h$	$> 3h$	$> 3h$	17 550

TABLE 5.22 – Temps CPU (Convex.) - Petites et moyennes instances - $x \mapsto \frac{x}{1+x}$

CPU		CST, $G : x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$			
N	M	Principal	*Bonmin*	*Couenne*	DP Exacte
10^1	10^2	6740	0.39	$> 3h$	$< \epsilon$
10^1	10^3	$> 3h$	0.54	$> 3h$	0.02
10^1	10^4	$> 3h$	0.53	$> 3h$	1.77
10^1	10^5	$> 3h$	0.85	$> 3h$	176
10^1	10^6	$> 3h$	(NO) 0.26	$> 3h$	17 876
10^2	10^3	$> 3h$	*Opt* 4516	$> 3h$	0.18
10^2	10^4	$> 3h$	3146	$> 3h$	17.55
10^2	10^5	$> 3h$	2649	$> 3h$	1752
10^3	10^5	$> 3h$	$> 3h$	$> 3h$	17 949

TABLE 5.23 – Temps CPU (Convex.) - Petites et moyennes instances - $x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$

Points importants :

- Aucune instance de l'algorithme Principal, ni de *Couenne* ne convergent en 3h, ce qui est cohérent avec les résultats obtenus sur les très petites instances.
- La complexité empirique de *Bonmin* semble polynomiale en N, mais décroissante en M, ce qui est contraire à l'intuition.
- *Bonmin* ne converge plus en 3h pour $N = 10^3$.
- La comparaison avec la DP exacte qui est en $O(NM^2)$ est particulièrement intéressante :
  - Les deux méthodes résolvent instantanément l'instance $(1, 2)$. Puis lorsque N

augmente (M étant fixé), la DP devient plus rapide sur *Bonmin*, qui semble être au moins en N^2.

— Par contre, lorsque M augmente (N étant fixé), *Bonmin* converge alors beaucoup plus rapidement que la DP, mais pas nécessairement vers l'optimum. Nous remarquons deux instances emblématiques de ces comportements :

• L'instance (1,6) pour laquelle *Bonmin* converge instantanément contre près de 6h pour la DP. Bien que *Bonmin* ne trouve pas l'optimum, il renvoie rapidement une très borne inférieure avec un écart relatif inférieur à $\epsilon = 10^{-4}$.

• L'instance (2,3) qui est résolue instantanément par DP, contre environ 1h15mn à l'aide de *Bonmin*.

• Les différences les plus marquantes sont colorées dans le tableau.

— Nous rappelons enfin que *Bonmin* ne garantit pas l'optimalité. Dans les tableaux, nous notons *Opt* lorsqu'il est atteint et (NO) sinon.

— Ces différences montrent l'hétérogéneité des résultats des algorithmes et des solvers d'où l'importance opérationnelle cruciale, d'estimer (même empiriquement) la complexité au préalable, afin de choisir la méthode appropriée.

Avoir avoir étudié les performances des différents algorithmes pour les petites et moyennes instances, nous passons à l'échelle à section suivante et présentons les résultats des expérimentations numériques pour les grandes et très grandes instances.

5.3 Expérimentations numériques sur les grandes instances

A cette échelle, la DP exacte n'est plus disponible car elle est trop couteuse en temps et en espace mémoire. De plus, plusieurs des algorithmes proposés dans les chapitres précédents ne convergent plus pour les (très) larges instances, dans la limite ou d'espace mémoire fournis. Nous commençons par aborder les limites en espace, à la section 5.3.1, puis nous présentons les résultats des expérimentations numériques, suivant la même approche qu'à la section 5.2.

Nous présentons les résultats agrégés pour les méthodes à base de DP à la section 5.3.2. Puis, nous intéressons à la convexification pour les grandes instances à la section 5.3.3.

5.3.1 Limitations en espace (mémoire)

La plupart des algorithmes ont renvoyé des erreurs mémoire pour les très grandes instances. Nous nous sommes penchés sur la complexité spatiale des méthodes de résolution

pour déterminer les tailles limites d'instances tractables avec notre machine.

Un double prend 8 bytes dans la mémoire «tas» (*heap memory*) et la complexité en espace des algorithmes de DP est en $O(N\,M)$. Un calcul direct montre que, pour notre machine possédant 32Gb RAM et avec les notations, $N = 10^a$ et $M = 10^b$
Nous savons que 1Gb=$1024^3 = 2^{30}$ bytes. Nous en déduisons :

$$\begin{aligned} 8\,N\,M &= 8 \cdot 10^{a+b} \leq 32 \cdot 2^{30} \\ \text{Donc, } a+b &= \frac{\ln(N\,M)}{\ln(10)} \leq 32 \cdot \frac{\ln(2)}{\ln(10)} \approx 9.63 \\ \text{D'où, } a+b &\leq 9.63 < 10 \end{aligned}$$

La limite est donc de 9 pour la somme des exposants entiers $a + b$.
Par conséquent, dans nos expérimentations numériques, nous sommes limités aux plus grandes instances tractables $(N, M) = (1, 8),\ (2, 7)$ en résolution exacte par DP ou $(3, 6)$ pour la méthode TSDP.

Cependant, dans l'algorithme de DP avec bornes, la recherche de la meilleure solution est restreinte à un bandeau de taille $R = 2\lambda\,P$ dans les expérimentations numériques (où P correspond à la taille du grain), pour lequel nous avons montré, à la section 3.3.5 que la complexité en espace est en $O(N^2\,P)$. Donc une matrice N fois M gaspille trop d'espace mémoire, pour la méthode TSDP tandis qu'une structure de donnée plus adaptée pourrait économiser de l'espace mémoire.

En effet, la matrice de DP avec bornes est creuse et la taille de chaque ligne est contrôlée par les bornes du bandeau. Une idée simple est alors de concaténer les données dans un grand vecteur ligne (cf. figure 5.4), ce qui permet d'éviter la perte mémoire due au stockage des 0, qui est trop onéreuse sur les grandes instances.

Le gain en espace correspondant est de l'ordre de $\dfrac{N\,M}{N^2\,R} = \dfrac{M}{2\lambda\,P\,N}$. Dans nos expérimentations numériques sur de (très) larges instances, $M \gg N$, par un facteur 10^3 au moins (et jusqu'à 10^6). Nous en concluons que cette structure de donnée améliorée aurait beaucoup de sens et nous permettrait de gagner au moins un ordre de grandeur. C'est une des perspectives de notre étude que nous reprenons à la section 6.

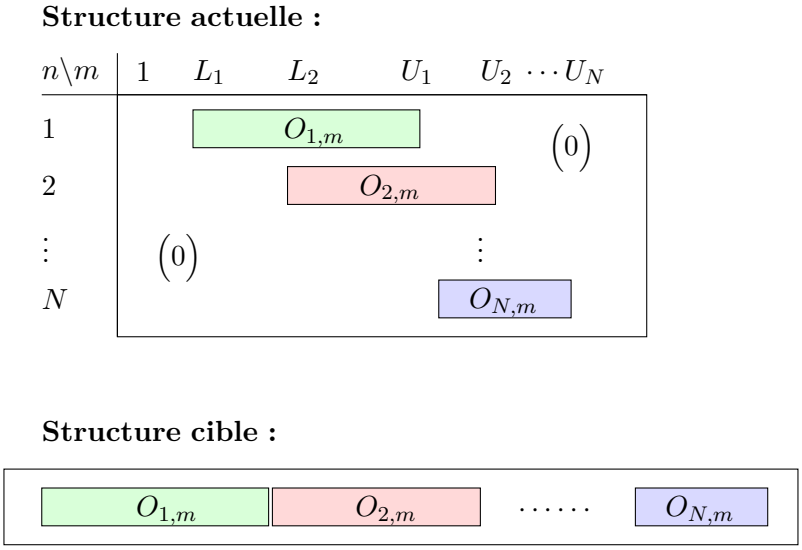


FIGURE 5.4 – Comparaison des structures de résultats de DP

Nous présentons maintenant nos résultats agrégés pour les (très) grandes instances avec les méthodes du chapitre 3 à base de DP.

5.3.2 Résultats agrégés des méthodes de DP

Nous précisons tout d'abord quelques conventions pour faciliter la lecture des tableaux de résultats.

Conventions dans les tableaux de résultats : dans le tableau de qualité des bornes présentée ci-dessous, la meilleure borne inférieure pour chaque instance est spécifiée dans la troisième colonne. Elle sert de référence pour mesurer la qualité, qui est définie comme la distance relative à la meilleure borne inférieure connue.

La borne supérieure UB_2 est séparée par les $\|$. La valeur correspond également à l'écart relatif à la meilleure borne inférieure connue.

Qualité			CST, $G : x \mapsto \frac{2}{\pi} arctan(x)$						
N	M	BEST LB	FS	LIS	ILS	LS	TS1	TS2	UB_2
10^1	10^7	TS1 (5000)	20.52	7.92	$< \epsilon$	$< \epsilon$	0	$< \epsilon$	38.02
10^1	10^8	TS1 (5000)	20.52	7.92	$< \epsilon$	$< \epsilon$	0	$< \epsilon$	38.02
10^1	10^9	ILS	20.52	7.92	0	$< \epsilon$	DNC	DNC	38.02
10^2	10^6	TS1 (5000)	25.01	2.14	$< \epsilon$	0.01	0	$< \epsilon$	364.56
10^2	10^7	TS1 (5000)	25.01	2.14	$< \epsilon$	0.01	0	$< \epsilon$	364.56
10^2	10^8	ILS	25.01	2.14	0	$< \epsilon$	DNC	DNC	364.56
10^2	10^9	ILS	25.01	2.14	0	$< \epsilon$	DNC	DNC	364.56
10^3	10^6	TS1 (500)	25.49	0.23	0.01	0.07	0	$< \epsilon$	558.72
10^3	10^7	ILS	25.49	0.23	0	0.07	DNC	DNC	558.72
10^3	10^8	ILS	25.49	0.23	0	0.07	DNC	DNC	558.72
10^3	10^9	ILS	25.49	0.23	0	0.07	DNC	DNC	558.72

TABLE 5.24 – Qualité - Grandes instances (CST)

Qualité			AVG, $G : x \mapsto \frac{2}{\pi} arctan(x)$						
N	M	BEST LB	FS	LIS	ILS	LS	TS1	TS2	UB_2
10^1	10^7	TS1 (5000)	21.77	10.59	0.71	$< \epsilon$	0	$< \epsilon$	139.07
10^1	10^8	TS1 (5000)	21.77	10.59	0.71	$< \epsilon$	0	$< \epsilon$	139.07
10^1	10^9	LS	21.77	10.59	0.71	0	DNC	DNC	139.07
10^2	10^6	TS1 (500)	28.49	7.8	3.97	$< \epsilon$	0	$< \epsilon$	419.13
10^2	10^7	TS1 (5000)	28.49	7.8	3.97	0.01	0	$< \epsilon$	419.13
10^2	10^8	LS	28.49	7.79	3.97	0	DNC	DNC	419.16
10^2	10^9	LS	28.43	7.72	4.33	0	DNC	DNC	443.43
10^3	10^6	TS1 (500)	30.35	7.87	7.19	0.29	0	$< \epsilon$	576.06
10^3	10^7	LS	30.31	7.82	7.11	0	DNC	DNC	576.49
10^3	10^8	LS	30.29	7.79	7.08	0	DNC	DNC	576.71
10^3	10^9	LS	30.31	7.81	7.11	0	DNC	DNC	576.52

TABLE 5.25 – Qualité - Grandes instances (AVG)

Points importants :

— Pour les deux échantillons CST et AVG :

- TS1 fournit la meilleure borne inférieure, partout où elle est disponible. TS2 est proche à ϵ prés de TS1, mais pas meilleure. Ainsi, contrairement à l'intuition, un maximum local du problème relâché ne fournit pas forcément un meilleur point de départ pour la recherche locale intensive par DP.

- La vente à tout prix (FS) sous-performe l'optimum d'environ 20-30% pour tout (N, M).
- La borne supérieure UB_2, obtenue par majoration monotone, est pertinente seulement dans le cas CST pour $N = 10$.
  Dans les autres, UB_2 n'est pas fine et tend vers la borne naturelle lorsque $N = 10^3$.

— Plus spécifiquement, lorsque p est constant (échantillon CST) :
- L'algorithme ILS fournit la meilleure borne inférieure lorsque les méthodes TSDP ne sont pas disponibles. En particulier, ILS fait très légèrement mieux que *LocalSolver* dans ce cas.
- Cependant, toutes les bornes inférieures sont proches. Elles sont situées dans un rayon de 0.1%.
- L'heuristique LIS (vente linéaire) a une qualité d'environ 7% quand $N = 10$. Cet écart est stable en M mais diminue quand N augmente et s'élève à moins de 1% pour $N = 10^3$.

— Enfin, dans l'échantillon en moyenne (AVG) :
- Lorsque la méthode TSDP est disponible, elle fournit la meilleure borne inférieure. LS est très proche de TS1. L'écart est de moins de 1%, mais il semble augmenter avec N.
- Lorsque TSDP n'est plus disponible, LS devient la meilleure borne inférieure. Dans ce cas, *LocalSolver* fait mieux que l'algorithme de recherche locale ILS.
- L'algorithme de gradient discret ILS suit la meilleure borne à environ 1-10% près. Cependant l'écart augmente également avec N.
- Dans le cas AVG, comme nous l'avons observé pour les petites et moyennes instances de la section 5.2, les résultats sont hétérogènes selon les trajectoires de p, entre LS et ILS.
  Nous prenons l'exemple de l'instance $(T, N) = (3, 6)$. Les résultats de chaque trajectoire apparaissent dans le tableau ci-dessous :

Qualité	$G : x \mapsto \frac{2}{\pi} arctan(x)$			
N=10^3, M=10^6				
μ	σ	BEST LB	ILS	LS
constant prices		TS1(500)	0.01	0.07
-0.05	0.10	TS1(500)	2.08	0.04
-0.05	0.25	TS1(500)	3.45	1.23
-0.05	0.70	TS1(500)	8.75	0.31
0	0.10	TS1(500)	1.74	0.05
0	0.25	TS1(500)	5.01	0.08
0	0.70	TS1(500)	13.65	0.35
0.05	0.10	TS1(500)	2.26	0.09
0.05	0.25	TS1(500)	5.32	0.26
0.05	0.70	TS1(500)	22.42	0.25

TABLE 5.26 – Qualité - Instance (3,6) - Trajectoires de p

Points importants :

— En cohérence avec les résultats précédents, la qualité de l'algorithme ILS décroit significativement avec σ, et l'effet est plus prononcé lorsque le moment μ est élevé. Dans le langage financier de l'application ELBA, ILS est moins performant lorsque la volatilité de marché est élevée et ce d'autant plus que la tendance est marquée.

Comme nous l'avons remarqué précédemment, l'algorithme ILS est moins efficace sur les profils p en dent de scie, car le gradient discret reste bloqué dans un maximum local créé par les pics de p. Nous proposons des améliorations en perspectives à la section 6.

— Nous observons le même schéma de résultat pour *LocalSolver*, sauf pour le cas $(\mu, \sigma) = (-0.05, 0.25)$.

— ILS fait légèrement mieux que *LocalSolver* dans le cas constant seulement, mais nettement moins bien que le solver dans l'échantillon AVG.

Enfin, nous nous intéressons aux temps CPU correspondants dans les tables ci-dessous :

CPU		CST, $G : x \mapsto \frac{2}{\pi} arctan(x)$						
N	M	FS	LIS	ILS	LS	TS1	TS2	UB_2
10^1	10^7	$< \epsilon$	$< \epsilon$	$< \epsilon$	600	20.92	19.11	$< \epsilon$
10^1	10^8	$< \epsilon$	$< \epsilon$	4	600	197.31	19.61	$< \epsilon$
10^1	10^9	$< \epsilon$	$< \epsilon$	36	600	DNC	DNC	$< \epsilon$
10^2	10^6	$< \epsilon$	$< \epsilon$	1.31	600	39.58	19.40	$< \epsilon$
10^2	10^7	$< \epsilon$	$< \epsilon$	1.99	600	1948.18	1918.30	$< \epsilon$
10^2	10^8	$< \epsilon$	$< \epsilon$	6	600	DNC	DNC	$< \epsilon$
10^2	10^9	$< \epsilon$	$< \epsilon$	38	600	DNC	DNC	$< \epsilon$
10^3	10^6	$< \epsilon$	$< \epsilon$	2220.02	3600	1466.14	1269.08	$< \epsilon$
10^3	10^7	$< \epsilon$	$< \epsilon$	7930.42	3600	DNC	DNC	$< \epsilon$
10^3	10^8	$< \epsilon$	$< \epsilon$	14719.71	10800	DNC	DNC	$< \epsilon$
10^3	10^9	$< \epsilon$	$< \epsilon$	25646.61	10800	DNC	DNC	$< \epsilon$

TABLE 5.27 – Temps CPU - Grandes instances (CST)

CPU		AVG, $G : x \mapsto \frac{2}{\pi} arctan(x)$						
N	M	FS	LIS	ILS	LS	TS1	TS2	UB_2
10^1	10^7	$< \epsilon$	$< \epsilon$	$< \epsilon$	600	19.21	5.7	$< \epsilon$
10^1	10^8	$< \epsilon$	$< \epsilon$	3.34	600	195.45	18.01	$< \epsilon$
10^1	10^9	$< \epsilon$	$< \epsilon$	38.33	60	DNC	DNC	$< \epsilon$
10^2	10^6	$< \epsilon$	$< \epsilon$	0.21	60	28.1	10.18	$< \epsilon$
10^2	10^7	$< \epsilon$	$< \epsilon$	0.29	600	1041.73	1011.34	$< \epsilon$
10^2	10^8	$< \epsilon$	$< \epsilon$	4.89	60	DNC	DNC	$< \epsilon$
10^2	10^9	$< \epsilon$	$< \epsilon$	39.34	60	DNC	DNC	$< \epsilon$
10^3	10^6	$< \epsilon$	$< \epsilon$	154.12	600	557.35	396.7	$< \epsilon$
10^3	10^7	$< \epsilon$	$< \epsilon$	257.41	600	DNC	DNC	$< \epsilon$
10^3	10^8	$< \epsilon$	$< \epsilon$	366.52	600	DNC	DNC	$< \epsilon$
10^3	10^9	$< \epsilon$	$< \epsilon$	648.6	600	DNC	DNC	$< \epsilon$

TABLE 5.28 – Temps CPU - Grandes instances (AVG)

Points importants :

— Comme précédemment, nous adaptons les limites de temps de *LocalSolver* à celle de la meilleure borne inférieure afin de pouvoir comparer les résultats.

— L'algorithme ILS est rapide pour les grandes instances. Cependant le temps de calcul augmente faiblement avec M mais croit fortement entre $N = 10^2$ et $N = 10^3$, pour la trajectoire constante.

Pour les trajectoires fluctuantes, ILS est beaucoup plus rapide, mais moins performant (cf. analyse des trajectoires ci-après).

— La complexité de TS1 est conforme à l'attendu en $O(N^{\frac{3}{2}} \cdot M)$. Le temps CPU de TS1 reste tractable pour les grandes instances. Malheureusement, en raison des contraintes de mémoire exposées au début de cette section, TS1 et TS2 ne sont plus disponibles pour les très larges instances.

— TS2 s'avère légèrement plus rapide que TS1, sauf pour les cas $(N, M) = (1, 8), (2, 6)$ où la différence est plus marquée.

— La borne supérieure UB_2 est calculée directement en temps presque constant.

Nous présentons enfin l'instance $(N, M) = (3, 6)$, pour montrer l'influence de la volatilité des trajectoires :

CPU	$G : x \mapsto \frac{2}{\pi} arctan(x)$			
T=10^3, N=10^6				
μ	σ	TS1(500)	ILS	LS
constant prices		1466.14	2220.02	3600
-0.05	0.1	408.06	52.62	600
-0.05	0.25	448.05	3.15	600
-0.05	0.7	340.44	1.04	600
0	0.1	729.33	229.31	600
0	0.25	657.45	8.58	600
0	0.7	294.03	1.59	600
0.05	0.1	882.91	1082.77	600
0.05	0.25	656.1	6.99	600
0.05	0.7	599.75	1.07	600

TABLE 5.29 – Temps CPU - Instance (3,6) - Trajectoires de p

Points importants :

— L'algorithme ILS converge beaucoup plus vite pour les trajectoires de p très fluctuantes, car il est très vite bloqué au premier maximum local qu'il retourne.

— Le temps CPU de TS1 décroit également avec σ. Mais contrairement à l'algorithme ILS, la méthode TSDP nous semble bien plus adaptée aux profils en dent de scie, pour lesquels la masse de l'écoulement est concentrée autour des pics de p.

Nous nous intéressons, à la section suivante, aux résultats de convexification afin d'encadrer l'optimum.

5.3.3 Résultats de convexification

Comme nous l'avons indiqué à la section 5.1.1, seul *Bonmin* prend en charge l'arctangente, contrairement à *Baron*, *Couenne* ou *Scip*.
C'est pourquoi, nous comparons les résultats de *Bonmin*, aux meilleures bornes (inférieures et supérieures) obtenues, pour la fonction de pénalité $x \mapsto \frac{2}{\pi}\arctan(x)$.

Dans le tableau suivant, pour les bornes inférieures, la meilleure des deux bornes est cochée ($\times$) et l'autre valeur fournit leur écart relatif.
Pour la partie bornes supérieures, les valeurs mesurent l'écart relatif de chaque borne par rapport à la meilleure borne inférieure.

Qualité		CST, $G : x \mapsto \frac{2}{\pi} arctan(x)$			
		Bornes Inf.			**Bornes Sup.**
N	M	Méthode	Best LB	*Bonmin*	UB_2
10^1	10^7	TS1(5000)	$\times$	DNC	38.02
10^1	10^8	TS1(5000)	$\times$	DNC	38.02
10^1	10^9	ILS	$\times$	DNC	38.02
10^2	10^6	TS1(5000)	$\times$	$< \epsilon$	364.56
10^2	10^7	TS1(5000)	$\times$	$< \epsilon$	364.56
10^2	10^8	ILS	$< \epsilon$	$\times$	364.56
10^2	10^9	ILS	$< \epsilon$	$\times$	364.56
10^3	10^6	TS1(500)	$\times$	DNC	558.72
10^3	10^7	ILS	$\times$	DNC	558.72
10^3	10^8	ILS	$\times$	DNC	558.72
10^3	10^9	ILS	$\times$	DNC	558.72

TABLE 5.30 – Qualité (Convexification) - Grandes Instances

Points importants :

- *Bonmin* renvoie une erreur «problème infaisable» pour les instances (1,7) (1,8) et (1,9). De plus, il ne converge pas et renvoie pas de bornes à partir de (3,6).
- Sur les instances (2,6) (2,7), TS1 (TSDP discrète) fait mieux que *Bonmin*, mais par une très faible marge.
- Pour les instances (2,8) et (2,9), *Bonmin* fournit la meilleure borne inférieure avec, une nouvelle fois, un écart très faible avec ILS.
- Nous remarquons que les différents écarts entre *Bonmin*, ILS et TS1 sont faibles ($\sim 10^{-5}$). Par conséquents, ces différentes approches se valent en termes de bornes

inférieures.

— Comme indiqué précédemment, la borne monotone UB_2 n'est pertinente que pour $N = 10$.

— Cependant, sans garantie d'optimalité, nous n'avons pas d'encadrement précis de l'optimum, sur les grandes et très grandes instances.

Nous présentons enfin le tableau des temps CPU correspondants :

CPU		CST, $G : x \mapsto \frac{2}{\pi} arctan(x)$		
N	M	Méthode	Best LB	*Bonmin*
10^1	10^7	TS1(5000)	20.92	DNC
10^1	10^8	TS1(5000)	197.31	DNC
10^1	10^9	ILS	36	DNC
10^2	10^6	TS1(5000)	39.58	9615.50
10^2	10^7	TS1(5000)	1948.18	121.58
10^2	10^8	ILS	6	$> 3h$
10^2	10^9	ILS	38	11.22
10^3	10^6	TS1(500)	2220.02	DNC
10^3	10^7	ILS	7930.42	DNC
10^3	10^8	ILS	14719.71	DNC
10^3	10^9	ILS	25646.61	DNC

TABLE 5.31 – Temps CPU (Convexification) - Grandes Instances

Points importants :

— Peu d'instances convergent pour les deux méthodes, ce qui limite la comparaison.

— Pour les instances (2,6) (2,7) et (2,9), pour lesquelles *Bonmin* converge, le temps de calcul décroit en M, ce qui est contraire à l'intuition. Le résultat est similaire pour les petites et moyennes instances (cf. section 5.2.5.2).

— Par contre l'instance (2,8), *Bonmin* ne converge pas en 3h.

— ILS est rapide pour $N = 100$, car il s'arrête au premier maximum local trouvé (qui est néanmoins proche de la solution fournie par *Bonmin*), le temps dépasse les 3h pour $N = 1000$.

Enfin, nous nous intéressons aux performances de la méthode TSDP et de l'algorithme Principal sur deux instances, (1,7) et (1,8) que *Bonmin* ne parvient pas à traiter. Nous comparons alors les performances de nos méthodes au solver *Couenne*.

Dans le tableau de qualité des résultats, les valeurs correspondent à l'écart relatif avec

la meilleure solution admissible (ou meilleure borne inférieure), qui est donnée par la colonne *BestLB* :

Qualité		CST, $G : x \mapsto \frac{x}{1+x}$						
		Bornes Inf.				**Bornes Sup.**		
N	M	BestLB	Principal	*Couenne*	TS1(5000)	Principal	*Couenne*	UB_2
10	10^7	TS1(5000)	2.07(∗)	0.53	0	136.75(∗)	215.0	36.71
10	10^8	TS1/Couenne	7.41(∗)	0	0	617.03(∗)	212.88	36.71
CST, $G : x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$								
10	10^7	TS1(5000)	0.81	0.49	0	105.53	142.84	11.09
10	10^8	TS1/Couenne	0.49(∗)	0	0	212.78(∗)	41.35	11.09

Table 5.32 – Qualité (Convexification) - Instances (1,7) et (1,8)

CPU		CST, $G : x \mapsto \frac{x}{1+x}$			
N	M	Principal	*Couenne*	TS1(5000)	UB_2
10	10^7	354.28(∗)	$> 3h$	20.95	$< \epsilon$
10	10^8	665.23(∗)	$> 3h$	197.36	$< \epsilon$
CST, $G : x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$					
10	10^7	$> 3h$	$> 3h$	21	$< \epsilon$
10	10^8	8735(∗)	$> 3h$	197.07	$< \epsilon$

Table 5.33 – Temps CPU (Convexification) - Instances (1,7) et (1,8)

Points importants :

- L'algorithme Principal ne s'est pas terminé car *Scip* renvoie une erreur (de limite de profondeur d'arbre) dans la résolution de la relaxation convexe, pour trois instances sur quatre, ce que nous notons (*).
- Les bornes inférieures de la méthode TSDP discrète et de *Couenne* sont très proches (moins de 1% d'écart).
- Les bornes supérieures de l'algorithme Principal ne sont pas bonnes, en comparaison des résultats de la section 5.2.4 sur les petites et moyennes instances, car il s'arrête assez vite.

  L'algorithme Principal fait mieux que *Couenne* pour l'instance (1,7), mais il est largement battu par le solver pour (1,8).
- *Couenne* retourne des bornes supérieures relativement mauvaises, sauf pour l'instance (1,8) pour la fonction de pénalité $x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$.
- Compte tenu des performances du solver et des problèmes de convergence de l'al-

gorithme principal, UB_2 est la meilleure borne supérieure pour les quatre grandes instances, ce qui est contraire à l'intuition.

— Concernant les temps de calculs, comme précédemment, la meilleure borne inférieure est obtenue en quelques minutes par la méthode TSDP discrète. Le calcul d'UB_2 est instantané.

— Par conséquent, pour ces quatre grandes instances, le meilleur encadrement de l'optimum est obtenu à l'aide des méthodes à base de DP et la borne monotone, qui font mieux que les méthodes de convexification ainsi que les solvers de l'état de l'art.

— Nous rappelons que l'existence d'UB_2 est soumise à des conditions sur la fonction de pénalité g, notamment que $x \mapsto x\,g$ soit convexe (cf. section 3.5, lemme 2).

Chapitre 6

Conclusions et perspectives

6.1 Conclusion générale

Nous avons résolu le problème traité $(PNS_{N,M})$, non convexe, non linéaire en variables entières, sous contrainte linéaire d'égalité, de manière exacte et approchée en poursuivant deux voies :

La première voie consiste à décomposer récursivement le problème en problèmes plus simples (avec moins de variables de décision) afin de se ramener à un problème univarié. Nous avons établi l'équation de Bellman et déduit un algorithme de résolution exacte par Programmation Dynamique (DP). Cette résolution exacte se fait en temps polynomial. Elle n'est efficace en pratique que pour les petites instances.

Nous avons proposé une méthode approchée en deux étapes (TSDP), dont nous avons établi la meilleure complexité, par une taille optimale de grain. Numériquement, elle fournit de très bonnes solutions admissibles, atteignant très souvent l'optimum pour les petites et moyennes instances. Nous obtenons un meilleur résultat que *LocalSolver* (solver de référence en recherche locale), sur de nombreuses instances, par une très faible marge cependant.

Nous avons également proposé une version hybride de la méthode TSDP qui utilise les maxima locaux de la relaxation continue du problème $(\overline{PNS_{N,M}})$ et effectue une recherche locale utilisant la DP, dans le bandeau autour de l'heuristique. Nous avons appliqué une méthode de gradient projeté pour déterminer les maximas locaux du problème continu, que nous avons comparés au solver libre *NLopt* (solver libre spécialisé dans l'optimisa-

tion non linéaire en variables continues). Numériquement, les résultats obtenus sont très proches de *NLopt*, cependant le solver est plus stable et plus rapide.

Nous avons également proposé un algorithme de recherche local (ILS), qui est très performant lorsque p est constant. Cependant sa performance se dégrade rapidement lorsque les trajectoires sont très fluctuantes, car il reste bloqué dans un maximum local.

Pour les grandes et très grandes instances, pour lesquelles la DP n'est pas disponible, notre méthode TSDP fournit la meilleure solution, lorsqu'elle est calculable, en un temps raisonnable. Elle fait notamment mieux que *LocalSolver*. Lorsque la méthode TSDP ne peut être appliquée, nous sommes battus par le solver dans les cas non constants et ne faisons mieux que dans le cas constant, *via* l'algorithme ILS.
Enfin, par un argument de monotonie, nous avons proposé un problème majorant convexe que nous avons résolu analytiquement. Nous en avons déduit une première borne supérieure du problème $(PNS_{N,M})$, qui n'est cependant numériquement pas très fine.

La seconde voie que nous avons explorée consiste à convexifier le problème en le majorant par des problèmes convexes de plus en plus fins qui convergent vers l'optimum. Nous avons établi que le problème pouvait être transformé en un problème *factorisé* équivalent, ce qui nous a permis d'en isoler plus facilement les termes non concaves. Nous avons alors établi une relaxation convexe fine du problème initial. Puis nous avons proposé deux algorithmes de *Branch & Bound*, utilisant cette relaxation comme fonction d'évaluation.

Numériquement, seules les très petites instances convergent vers l'optimum dans le temps imparti de 3h. Pour les petites et moyennes instances, nous obtenons de meilleures bornes supérieures *via* notre algorithme Principal de *Branch & Bound*, que les solvers *Scip* et *Baron*. Ce point est intéressant, car nous utilisons *Scip* au sein de notre algorithme, pour résoudre le problème relâché convexe.
Comparé à *Couenne*, qui est notre solver de référence, les performances dépendent de la fonction de pénalité. Nous obtenons en général de meilleures bornes supérieures pour la fonction de pénalité $x \mapsto \frac{x}{1+x}$, mais moins bonnes pour $x \mapsto 1 - \frac{2}{1+\sqrt{1+x}}$.

Enfin, bien qu'il ne soit pas un solver global pour les problèmes non convexes, *Bonmin*, atteint en pratique l'optimum pour la quasi-totalité des très petites instances, et ce avec

des temps de calculs inférieurs à tous les autres solvers employés.
Pour les petites et moyennes instances, *Bonmin* n'atteint l'optimum que pour 61% des instances proposées, ce qui est moins performant que nos méthodes TSDP (qui l'atteignent dans 99% des cas testés).

Lorsque nous passons à l'échelle, sans surprise, les performances de toutes les méthodes et solvers se dégradent.
Les bornes supérieures fournies par l'algorithme Principal et par *Couenne* ne sont plus fines et tendent vers les bornes naturelles lorsque $N \geq 100$.
Bonmin ne converge que sur un nombre limité d'instances. Sur ces instances, nous montrons que le solver, la méthodes TSDP discrète et la méthode ILS se valent en qualité.
Cependant, nous ne pouvons comparer que peu d'instances, car nous atteignons pour ces (très) grandes instances les limites en variables ou en espace mémoire et de nombreuses expérimentations numériques ne convergent plus ou renvoient des erreurs.
Pour les (très) grandes instances, seule la seule borne supérieure UB_2 disponible, bien qu'elle ne soit plus pertinente dés que $N \geq 10^2$.
Par conséquent, nous n'avons pas d'encadrement précis de l'optimum sur ces instances.

La comparaison des temps de calcul révèle une forte hétérogénéité des performances relatives entre les méthodes, selon la fonction de pénalité et la taille d'instance.
Nous interprétons cette diversité dans la qualité des résultats comme une des richesses du problème, et en concluons que l'estimation de la complexité (théorique ou empirique dans le cas des solvers) est un pré-requis au choix de la méthode de résolution.

Nous rappelons enfin que, bien que nous ayons particularisé les fonctions de pénalité, les principales techniques présentées dans cette thèse sont applicables à toute fonction $\mathcal{C}^1$ réelle, croissante dans $\mathbb{R}_+$.

6.2 Perspectives

Nous abordons maintenant les limites des approches et des méthodes présentées dans cette thèse et proposons différents axes d'amélioration, techniques ou méthodologiques, qui peuvent faire l'objet de futures recherches.
Nous commençons par les perspectives techniques. Nous avons démarré des travaux qui ne sont pas achevés sur les deux points ci-dessous.

Gestion de la mémoire : Dans l'algorithme de DP avec bornes, présenté à la section 3.3, la recherche de la meilleure solution s'effectue dans le bandeau de taille R, avec $R \sim 2\lambda\, P \ll M$. Nous avons établi sa complexité spatiale, en $O(N^2\, R)$. Une matrice N par M est donc très gourmande en mémoire, tandis qu'une structure de donnée adaptée permettrait une allocation de donnée plus efficace. L'espace mémoire économisé par ce type de structure est de l'ordre de $\frac{N\,M}{N^2\,R} = \frac{M}{2\lambda\, P\, N}$. Dans les expérimentations numériques sur les (très) grandes instances, où $M \gg N$ par un facteur de 10^3 à 10^6, cette structure de donnée serait très utile.

Calculs parallèles : Nous pouvons utilise notre processeur multi-cœurs pour lancer des calculs parallèles, par exemple pour les algorithmes de DP de la section 3.3. L'API OpenMP (standard de programmation parallèle, cf. Chapman *et al.* [37]), nous semble un bon point de départ. Cela permettra d'améliorer matériellement la vitesse de convergence des algorithmes. De plus, l'étude des stratégies de parallélisation et du gain de vitesse correspondants est un sujet d'intérêt.

Nous passons ensuite aux perspectives algorithmiques et méthodologiques :

Algorithme de gradient projeté : la méthode de recherche de maximum locaux en variables continues peut être amélioré, soit en utilisant un gradient projeté avec un pas adapté, ou bien en utilisant d'autre méthodes primales (*e.g* le gradient réduit, gradient réduit généralisé décrit dans [84] p234-240). Nous notons cependant que la détermination de ce pas optimal peut entrainer la résolution de sous-problèmes, donc le gain de performance global doit être étudié.

Algorithme de recherche local ILS : comme nous l'avons montré, ILS n'est pas performant quand les trajectoires de p sont très fluctuantes. Il en est ainsi car les pics de p génère des maximum locaux qui piègent rapidement ILS. Nous n'avons perturbé que les variables de décisions adjacentes (d'où l'interprétation de gradient discret). Nous pourrions également perturber x_i, x_{i+k} pour un k arbitraire. Sur quels critère choisir le k, à chaque itération pour sortir du voisinage de maximum local et améliorer les performances. Bref, comment déterminer une bonne stratégie de perturbation pour l'algorithme ILS est une question intéressante pour notre problème.

Amélioration du *Branch & Bound* : Bien qu'une étude approfondie d'algorithmes de *Branch & Bound* soit trop large pour le cadre de notre étude, nous pensons que les techniques plus avancées de réduction de bornes et de branchement, comme celles proposées par Belotti *et al.* [18] pourraient s'appliquer à notre problème, afin d'améliorer la convergence de l'algorithme et réduire sa complexité.
De plus, la relaxation convexe s'appuie sur les inégalité de McCormick [79]. Les travaux de Tsoukalas et Mitsos [112],[87] s'intéressent aux relaxations convexes multivariées au moins aussi fines que celle de McCormick, qui peuvent être appliquées à notre problème.

Dynamique de p : nous avons supposé que p suit une trajectoire déterministe indépendante des variables de décision. La relaxation de cette hypothèse au profit d'une dynamique stochastique, faisant intervenir les variables de décision précédentes nous semble une direction particulièrement intéressante pour de futurs travaux.

Bibliographie

[1] Warren P Adams et Hanif D Sherali : A tight linearization and an algorithm for zero-one quadratic programming problems. *Management Science*, 32(10):1274–1290, 1986.

[2] Claire S Adjiman, Ioannis P Androulakis et Christodoulos A Floudas : Global optimization of mixed-integer nonlinear problems. *AIChE Journal*, 46(9):1769–1797, 2000.

[3] Claire S Adjiman et Christodoulos A Floudas : Rigorous convex underestimators for general twice-differentiable problems. *Journal of Global Optimization*, 9(1):23–40, 1996.

[4] Faiz A Al-Khayyal et James E Falk : Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8(2):273–286, 1983.

[5] Aurélien Alfonsi, Antje Fruth et Alexander Schied : Constrained portfolio liquidation in a limit order book model. *Banach Center Publ*, 83:9–25, 2008.

[6] Aurélien Alfonsi, Antje Fruth et Alexander Schied : Optimal execution strategies in limit order books with general shape functions. *Quantitative Finance*, 10(2):143–157, 2010. URL `https://doi.org/10.1080/14697680802595700`.

[7] Robert Almgren et Neil Chriss : Optimal execution of portfolio transactions. *Journal of Risk*, 3, 2000. URL `https://www.math.nyu.edu/faculty/chriss/optliq_f.pdf`.

[8] Robert F. Almgren : Optimal execution with nonlinear impact functions and trading-enhanced risk. *Applied Mathematical Finance*, 10(1):1–18, 2003.

[9] Ioannis P Androulakis, Costas D Maranas et Christodoulos A Floudas : αbb :

A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, 1995.

[10] Charles Audet, Pierre Hansen, Brigitte Jaumard et Gilles Savard : A branch and cut algorithm for nonconvex quadratically constrained quadratic programming. *Mathematical Programming*, 87(1):131–152, 2000.

[11] Louis Bachelier : Théorie de la spéculation. *In Annales scientifiques de l'École normale supérieure*, volume 17, pages 21–86, 1900.

[12] Guy Barles : *Solutions de viscosité des équations de Hamilton-Jacobi*. Springer, 1994. ISBN 3-540-58422-6.

[13] Evelyn Martin Lansdowne Beale et John A Tomlin : Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. *OR*, 69(447-454):99, 1970.

[14] Richard Bellman : *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 édition, 1957.

[15] Richard Bellman : Norbert wiener prize acceptance speech. Rapport technique, University of Southern California, Los Angeles (USA), 1970.

[16] Pietro Belotti : *COUENNE(Convex Over and Under ENvelopes for Nonlinear Estimation) : a user's manual*.

[17] Pietro Belotti, Christian Kirches, Sven Leyffer, Jeff Linderoth, James Luedtke et Ashutosh Mahajan : Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013.

[18] Pietro Belotti, Jon Lee, Leo Liberti, François Margot et Andreas Wächter : Branching and bounds tighteningtechniques for non-convex minlp. *Optimization Methods and Software*, 24(4-5):597–634, 2009.

[19] Jacques F. Benders : Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, december 1962.

[20] Thierry Benoist, Bertrand Estellon, Frédéric Gardi, Romain Megel et Karim Nouioua : Localsolver 1. x : a black-box local-search solver for 0-1 programming. *4or*, 9(3):299–316, 2011.

[21] Thierry Benoist, Frédéric Gardi, Julien Darlay et Romain Megel : Localsolver, 2020. https ://www.localsolver.com/home.html.

[22] Thierry Benoist, Frédéric Gardi, Julien Darlay, Bertrand Estellon et Romain Megel : *Mathematical programming solver based on local search*. John Wiley & Sons, 2014.

[23] D. Bertsimas et A. Lo : Optimal control of execution costs. *Journal of Financial Markets*, 1, 1998.

[24] Alain Billionnet : *Optimisation discrète, De la modélisation à la résolution par des logiciels de programmation mathématique*. Dunod, 2007. ISBN 978-2-10-049687-7.

[25] Alain Billionnet, Sourour Elloumi et Amélie Lambert : Extending the QCR method to general mixed-integer programs. *Mathematical programming*, 131(1):381–401, 2012.

[26] Alain Billionnet, Sourour Elloumi et Amélie Lambert : Exact quadratic convex reformulations of mixed-integer quadratically constrained problems. *Math. Program.*, 158(1-2):235–266, 2016. URL `https://doi.org/10.1007/s10107-015-0921-2`.

[27] Alain Billionnet, Sourour Elloumi et Marie-Christine Plateau : Improving the performance of standard solvers for quadratic 0-1 programs by a tight convex reformulation : The QCR method. *Discret. Appl. Math.*, 157(6):1185–1197, 2009. URL `https://doi.org/10.1016/j.dam.2007.12.007`.

[28] Fischer Black et Myron Scholes : The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654, 1973.

[29] P Bonami, JJ Forrest, J Lee et A Wächter : Rapid development of an open-source mlnlp solver with coin-or. *Optima Newsletter*, 75, 2007.

[30] Pierre Bonami : *BONMIN(Basic Open-source Nonlinear Mixed INteger programming)*.

[31] Pierre Bonami, Lorenz T Biegler, Andrew R Conn, Gérard Cornuéjols, Ignacio E Grossmann, Carl D Laird, Jon Lee, Andrea Lodi, François Margot, Nicolas Sawaya *et al.* : An algorithmic framework for convex mixed integer nonlinear programs. 2005.

[32] Stephen Boyd, Enzo Busseti, Steven Diamond, Ronald N Kahn, Kwangmoo Koh, Peter Nystrup et Jan Speth : Multi-period trading via convex optimization. *arXiv preprint arXiv :1705.00109*, 2017.

[33] Cristiana Bragalli, Claudia D'Ambrosio, Jon Lee, Andrea Lodi et Paolo Toth : An minlp solution method for a water network problem. *In European Symposium on Algorithms*, pages 696–707. Springer, 2006.

[34] Samuel Burer et Adam N. Letchford : Non-convex mixed-integer nonlinear programming : A survey. *Surveys in Operations Research and Management Science*, 17(2):97–106, 2012. ISSN 1876-7354. URL `https://www.sciencedirect.com/science/article/pii/S1876735412000037`.

[35] Sonia Cafieri, Pierre Hansen, Lucas Létocart, Leo Liberti et Frédéric Messine : Compact relaxations for polynomial programming problems. *In International Symposium on Experimental Algorithms*, pages 75–86. Springer, 2012.

[36] Sonia Cafieri, Jon Lee et Leo Liberti : On convex relaxations of quadrilinear terms. *Journal of Global Optimization*, 47(4):661–685, 2010.

[37] Barbara Chapman, Gabriele Jost et Ruud Van Der Pas : Using openmp, 2008.

[38] Stephen A Cook : The complexity of theorem-proving procedures. pages 151–158, 1971.

[39] Alberto Costa et Leo Liberti : Relaxations of multilinear convex envelopes : dual is better than primal. *In International Symposium on Experimental Algorithms*, pages 87–98. Springer, 2012.

[40] L. Dann, D. Mayers et R. Raab : Trading rules, large blocks and the speed of price adjustment. *Journal of Financial Economics*, 4, 1977.

[41] George B Dantzig et Philip Wolfe : The decomposition algorithm for linear programs. *Econometrica : Journal of the Econometric Society*, pages 767–778, 1961.

[42] V. Donde, V. Lopez, B. Lesieutre, A. Pinar, Chao Yang et J. Meza : Identification of severe multiple contingencies in electric power networks. *In Proceedings of the 37th Annual North American Power Symposium, 2005.*, pages 59–66, 2005.

[43] Bruno Dupire *et al.* : Pricing with a smile. *Risk*, 7(1):18–20, 1994.

[44] Sourour Elloumi et Amélie Lambert : Global solution of non-convex quadratically constrained quadratic programs. *Optimization Methods and Software*, 34(1):98–114, 2019. URL `https://doi.org/10.1080/10556788.2017.1350675`.

[45] Noel Falconer : On the size of convoys : an example of the methodology of leading wartime or scientists. *Journal of the Operational Research Society*, 27(2):315–327, 1976.

[46] James E Falk et Richard M Soland : An algorithm for separable nonconvex programming problems. *Management science*, 15(9):550–569, 1969.

[47] Hossam Faris, Ibrahim Aljarah, Mohammed Azmi Al-Betar et Seyedali Mirjalili : Grey wolf optimizer : a review of recent variants and applications. *Neural computing and applications*, 30:413–435, 2018.

[48] Robert Fortet : L'algèbre de boole et ses applications en recherche opérationnelle. *Cahiers du Centre d'Etudes de Recheche Operationnelle*, 1(4):5–36, 1959.

[49] Robert Fourer, David M Gay et Brian W Kernighan : A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990.

[50] Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger et Jakob Witzig : The SCIP Optimization Suite 7.0. ZIB-Report 20-10, Zuse Institute Berlin, March 2020. URL `http://nbn-resolving.de/urn:nbn:de:0297-zib-78023`.

[51] M.R. Garey et D.S. Johnson : *Computers and Intractability : A guide to the theory of NP-Completeness*. Freeman, New-York, 1979.

[52] Jim Gatheral : No-dynamic-arbitrage and market impact. *Quantitative Finance*, 10, 08 2010.

[53] Jim Gatheral et Alexander Schied : Dynamical models of market impact and algorithms for order execution. *SSRN Electronic Journal*, 2012.

[54] David M Gay : The ampl modeling language : An aid to formulating and solving optimization problems. *In Numerical analysis and optimization*, pages 95–116. Springer, 2015.

[55] Björn Geissler, Alexander Martin, Antonio Morsi et Lars Schewe : Using piecewise linear functions for solving minlp s. *In Mixed integer nonlinear programming*, pages 287–314. Springer, 2012.

[56] Gordon Gemmill : Transparency and liquidity : A study of block trades on the london stock exchange under different publication rules. *The Journal of Finance*, 51 (5):1765–1790, 1996. URL `http://www.jstor.org/stable/2329537`.

[57] Arthur M Geoffrion : Generalized benders decomposition. *Journal of optimization theory and applications*, 10(4):237–260, 1972.

[58] Fred Glover et Eugene Woolsey : Converting the 0-1 polynomial programming problem to a 0-1 linear program. *Operations research*, 22(1):180–182, 1974.

[59] H. G. Guthmann et A. J. Bakay : The market impact of the sale of large blocks of stock. *The Journal of Finance*, 20, 12 1965.

[60] Liz Hoffman, Corrie Driebusch, Tom McGinty, Susan Pulliam et Chung Juliet : Big stock sales are supposed to be secret. the numbers indicate they aren't., March 30th 2022. The Wall Street Journal - Online Edition.

[61] Reiner Horst et Hoang Tuy : *Global optimization : Deterministic approaches*. Springer Verlag, 1990.

[62] Joey Huchette et Juan Pablo Vielma : Nonconvex piecewise linear functions : Advanced formulations and simple modeling tools. *Operations Research*, 2022.

[63] John C. Hull : *Options, futures and other derivatives*. Pearson Prentice-Hall, Upper Saddle River, NJ, USA, 5th édition, 2002.

[64] Stephen G Johnson : The nlopt nonlinear-optimization package, 2021. http ://github.com/stevengj/nlopt.

[65] Richard M Karp : Reducibility among combinatorial problems. pages 85–103, 1972.

[66] Donald B. Keim et Ananth Madhavan : The upstairs market for large-block transactions : Analysis and measurement of price effects. *Review of Financial Studies*, 9, SPR 1996.

[67] Idris Kharroubi et Huyen Pham : Optimal portfolio liquidation with execution cost and risk. *SIAM Journal on Financial Mathematics*, 1(1):897–931, 2010.

[68] Thorsten Koch, Benjamin Hiller, Marc E Pfetsch et Lars Schewe : *Evaluating gas network capacities*. SIAM, 2015.

[69] Amélie Lambert : *Exact solutions of polynomial programs through quadratic convex reformulations : theory and applications*. Habilitation à diriger des recherches, Conservatoire National des Arts et Métiers, novembre 2021. URL `https://hal.archives-ouvertes.fr/tel-03443199`.

[70] Ailsa H Land et Alison G Doig : An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497, 1960.

[71] Arnaud Lazare : *Optimisation globale de polynômes en variables mixte-entières*. Theses, École doctorale de mathématiques Hadamard (EDMH, ED 574), décembre 2019. URL `https://hal.archives-ouvertes.fr/tel-02462537`.

[72] Sven Leyffer, Jeff Linderoth, James Luedtke, Andrew Miller et Todd Munson : Applications and algorithms for mixed integer nonlinear programming. *Journal of Physics : Conference Series*, 180(1):012014, jul 2009. URL `https://dx.doi.org/10.1088/1742-6596/180/1/012014`.

[73] Leo Liberti : Undecidability and hardness in mixed-integer nonlinear programming. *RAIRO-Oper. Res.*, 53(1):81–109, 2019. URL `https://doi.org/10.1051/ro/2018036`.

[74] Ananth Madhavan et Minder Cheng : In search of liquidity : Block trades in the upstairs and downstairs markets. *Review of Financial Studies*, 10, SPR 1997.

[75] Benoît B Mandelbrot et Richard L Hudson : *Une approche fractale des marchés : risquer, perdre et gagner*. Odile Jacob, 2009.

[76] Harry M Markowitz : Portfolio selection. *Journal of finance*, 7(1):71–91, 1952.

[77] Alexander Martin, Markus Möller et Susanne Moritz : Mixed integer models for the stationary case of gas network optimization. *Mathematical programming*, 105 (2):563–582, 2006.

[78] Joseph F McCloskey : Or forum—british operational research in world war ii. *Operations Research*, 35(3):453–470, 1987.

[79] Garth P McCormick : Computability of global solutions to factorable nonconvex programs : Part i—convex underestimating problems. *Mathematical programming*, 10(1):147–175, 1976.

[80] Garth P McCormick : *Nonlinear programming : Theory, algorithms, and applications*. John Wiley & Sons, Inc., 1983.

[81] Robert C Merton : Theory of rational option pricing. *The Bell Journal of economics and management science*, pages 141–183, 1973.

[82] Clifford A Meyer et Christodoulos A Floudas : Trilinear monomials with mixed sign domains : Facets of the convex and concave envelopes. *Journal of Global Optimization*, 29(2):125–155, 2004.

[83] Clifford A Meyer et Christodoulos A Floudas : Trilinear monomials with positive or negative domains : Facets of the convex and concave envelopes. *In Frontiers in global optimization*, pages 327–352. Springer, 2004.

[84] Michel Minoux : *Programmation Mathématique. Théorie et Algorithmes*. Lavoisier, 2008. URL `https://hal.archives-ouvertes.fr/hal-01303928`.

[85] Seyedali Mirjalili, Seyed Mohammad Mirjalili et Andrew Lewis : Grey wolf optimizer. *Advances in engineering software*, 69:46–61, 2014.

[86] Ranjan Mishra : Optimal portfolio liquidation in dark pool. Mémoire de D.E.A., Indian Statistical Institute, Kolkata, India, 07 2017. URL `http://library.isical.ac.in:8080/jspui/handle/10263/6821`.

[87] Jaromił Najman, Dominik Bongartz, Angelos Tsoukalas et Alexander Mitsos : Erratum to : multivariate mccormick relaxations. *Journal of Global Optimization*, 68(1):219–225, 2017.

[88] Iraj Naruei, Farshid Keynia et Amir Sabbagh Molahosseini : Hunter–prey optimization : Algorithm and applications. *Soft Computing*, 26(3):1279–1314, 2022.

[89] Anna A. Obizhaeva et Jiang Wang : Optimal trading strategy and supply/demand dynamics, working paper. *SSRN Electronic Journal*, 2005.

[90] Anna A Obizhaeva et Jiang Wang : Optimal trading strategy and supply/demand dynamics. *Journal of Financial Markets*, 16(1):1–32, 2013.

[91] Dominique Quadri : Programmation mathématique en variables entières : théorie et applications. *Habilitation à diriger des recherches*, 2015. URL `https://hal.inria.fr/tel-01688213`.

[92] Dominique Quadri et Eric Soutil : Reformulation and solution approach for non-separable integer quadratic programs. *Journal of the Operational Research Society*, 66(8):1270–1280, 2015.

[93] Anatoliy D Rikun : A convex envelope formula for multilinear functions. *Journal of Global Optimization*, 10(4):425–437, 1997.

[94] R Tyrrell Rockafellar : Lagrange multipliers and optimality. *SIAM review*, 35 (2):183–238, 1993.

[95] J. Ben Rosen : The gradient projection method for nonlinear programming. part i. linear constraints. *Journal of the society for industrial and applied mathematics*, 8 (1):181–217, 1960.

[96] J. Ben Rosen : The gradient projection method for nonlinear programming. part ii. nonlinear constraints. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):514–532, 1961.

[97] Hong S Ryoo et Nikolaos V Sahinidis : A branch-and-reduce approach to global optimization. *Journal of global optimization*, 8(2):107–138, 1996.

[98] N. V. Sahinidis : *BARON 22.11.3 : Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, November 2022.

[99] Martin Schmidt : Mixed-integer nonlinear optimization, June 2021. JPOC Spring School on MINLPs and Bilevel Problems "in Paris".

[100] D. Seppi : Equilibrium block trading and asymmetric information. *The Journal of Finance*, 45(1):73–94, 1990. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.1990.tb05081.x`.

[101] Roland C. Seydel : Existence and uniqueness of viscosity solutions for qvi associated with impulse control of jump-diffusions. *Stochastic Processes and their Applications*, 119, 2009.

[102] Hanif D Sherali et Warren P Adams : A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990.

[103] Hanif D Sherali et Warren P Adams : *A reformulation-linearization technique for solving discrete and continuous nonconvex problems*, volume 31. Springer Science & Business Media, 2013.

[104] Edward Smith et Constantinos C Pantelides : Global optimisation of general process models. *In Global optimization in engineering design*, pages 355–386. Springer, 1996.

[105] Edward MB Smith et Constantinos C Pantelides : Global optimisation of nonconvex minlps. *Computers & Chemical Engineering*, 21:S791–S796, 1997.

[106] Edward MB Smith et 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.

[107] Eric Soutil, Dominique Quadri et David Nizard : Non-convex Quadratic Integer Programming : a piecewise linearization. *In ISMP - International Symposium on Mathematical Programming*, Bordeaux, France, juillet 2018. URL `https://hal.archives-ouvertes.fr/hal-02465339`.

[108] Krister Svanberg : A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM journal on optimization*, 12 (2):555–573, 2002.

[109] Fabio Tardella : Existence and sum decomposition of vertex polyhedral convex envelopes. *Optimization Letters*, 2(3):363–375, 2008.

[110] M. Tawarmalani et N. V. Sahinidis : A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103:225–249, 2005.

[111] Mohit Tawarmalani et 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, 2013.

[112] Angelos Tsoukalas et Alexander Mitsos : Multivariate mccormick relaxations. *Journal of Global Optimization*, 59(2):633–662, 2014.

[113] Vathana Ly Vath, Mohamed Mnif et Huyên Pham : A model of optimal portfolio selection under liquidity risk and price impact. *Finance and Stochastics*, 11, 01 2007.