Approximation linéaire par morceaux de fonctions de deux variables avec erreur bornée pour la résolution de problèmes d'optimisation non linéaire mixte en nombres entiers

Aloïs Duguet

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (Toulouse INP)

Discipline ou spécialité :

Informatique

Présentée et soutenue par :

M. ALOÏS DUGUET

le mercredi 22 novembre 2023

Titre :

Approximation linéaire par morceaux de fonctions de deux variables avec erreur bornée pour la résolution de problèmes d'optimisation non linéaire mixte en nombres entiers

---

Ecole doctorale :

Systèmes (EDSYS)

Unité de recherche :

Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS)

Directeur(s) de Thèse :

M. SANDRA ULRICH NGUEVEU

Rapporteurs :

MME AMELIE LAMBERT, CNAM PARIS

MME CLAUDIA D'AMBROSIO, ECOLE POLYTECHNIQUE PALAISEAU

Membre(s) du jury :

M. MARCEL MONGEAU, ECOLE NATIONALE DE L'AVIATION CIVILE, Président

M. MOURAD BAIOU, LIMOS, Membre

M. SANDRA ULRICH NGUEVEU, LAAS TOULOUSE, Membre

Résumé :

En optimisation, de nombreuses applications académiques et industrielles se modélisent sous la forme de problèmes de la classe appelée MINLP, pour mixed-integer nonlinear programming. Les problèmes de cette classe sont généralement difficiles à résoudre en l'absence de propriétés particulières sur les fonctions non linéaires, combinée à la présence de variables de décision binaires ou entières. Une méthode classique de résolution consiste à approximer le problème en remplaçant les fonctions non linéaires par des fonctions linéaires par morceaux pour obtenir un problème MILP, pour mixed-integer linear programming. Cela permet de tirer partie de la grande efficacité des solveurs et de toutes les avancées des trente dernières années sur la résolution de programmes linéaires en nombres entiers ou mixtes. Ces approches ont pour inconvénient majeur l'absence de garantie a priori sur la qualité des solutions obtenues. Cette thèse s'inscrit dans la lignée de travaux récents visant à éliminer ce défaut.

Nos travaux se focalisent sur l'approximation de fonctions non linéaires de deux variables par des fonctions linéaires par morceaux respectant une borne sur l'erreur d'approximation et minimisant le nombre de morceaux utilisés. Cela permet l'approximation de toute fonction décomposable en somme de termes dépendant de deux variables. Dans un premier temps nous avons résolu optimalement le cas particulier de la norme euclidienne de deux variables lorsque le domaine de définition est le plan tout entier, et étendu ce résultat aux normes dont les lignes de niveaux sont des ellipses. Dans un second temps nous nous sommes intéressés aux fonctions de deux variables continues définies sur un domaine polygonal. Pour ces dernières nous avons développé différentes méthodes permettant d'obtenir des solutions réalisables exploitant des propriétés d'approximation que nous avons identifiées, ainsi que des bornes inférieures basées sur une nouvelle famille d'inégalités valides dédiée au problème. Les expériences numériques ont montré que nos méthodes obtiennent de meilleures solutions que l'état de l'art, en particulier pour l'approximation de fonctions non linéairement séparables. Enfin, deux applications à la résolution de problèmes MINLP sont traitées pour illustrer le potentiel pratique de nos travaux, l'une sur un problème de placements de faisceaux de satellites de télécommunication et l'autre sur le calcul d'équilibres de Nash en théorie des jeux.

Mots clés : Fonction linéaire par morceaux, Approximation de fonction, MINLP, MILP

Abstract : In optimization, many academic and industrial applications can be modeled in the form of a problem of the mixed-integer nonlinear programming class (MINLP). The problems of this class are generally hard to solve without particular properties on the nonlinear functions and with the presence of binary or integer variables. A standard method to solve MINLP approximates the problem by replacing nonlinear functions by piecewise linear ones to obtain a mixed-integer linear programming problem (MILP). It enables to make use of the great efficiency of current solvers and the last 30 years of research on solving MILP problems. Those approaches have a major downside which is the complete lack of a priori guarantee on the quality of the solution obtained. This thesis follows recent work with the goal of eliminating this downside.

Our work focuses on the approximation of nonlinear functions of two variables by piecewise linear functions satisfying a bound on the approximation error and minimizing the number of pieces used. It allows to approximate any function that can be written as a sum of terms depending on one or two variables. First, we solved optimally the particular case of the euclidean norm of two variables when the definition domain is the whole plane, and extend this result to norms with level set in the shape of ellipses. Second, we deal with continuous two-variable functions defined on a polygonal domain. To that end, we developed different methods to obtain feasible solutions exploiting some approximation properties we identified, as well as lower bounds based on a new family of valid inequalities specific to the problem. The numerical experiences showed that our methods obtain better solutions than the state of the art algorithms, in particular for the approximation of nonlinearly separable functions. Finally, we show two numerical applications solving MINLP problems to illustrate the practical side of our works, one on the beam layout problem used in satellite communication, and the other on the computation of Nash equilibria in game theory.

Keywords : Piecewise linear function, Function approximation, MINLP, MILP

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Introduction

Nous nous intéressons à la résolution de problèmes d'optimisation. Ils consistent à minimiser ou maximiser un critère sur un ensemble. Ce critère, que l'on appelle aussi fonction objectif, est évalué avec les variables de décisions du problème. L'ensemble de valeurs que peuvent prendre les variables de décisions peut être défini par des contraintes sur ces variables. Les problèmes d'optimisation de la classe mixed-integer nonlinear programming (MINLP) sont décrits par des fonctions non linéaires et des variables de décisions continues et/ou entières. Ces deux caractéristiques combinées rendent les MINLP difficiles à résoudre en général. Cela motive l'approche qui nous intéresse. Elle consiste dans un premier temps à approximer le problème en remplaçant chaque fonction non linéaire par une fonction linéaire par morceaux afin d'obtenir un problème de mixed-integer linear programming (MILP), puis dans un second temps à résoudre le problème approximé. L'intérêt de cette approche est que les solveurs exacts de problèmes MILP sont très efficaces. En effet, il y a eu de nombreux travaux de recherche ces dernières décennies, motivés entre autre par l'immense quantité d'applications industrielles et académiques notamment issues de l'optimisation combinatoire.

Le principal défaut de cette approche est qu'il n'y a, a priori, aucune garantie ni sur la réalisabilité ni sur qualité de la solution obtenue par approximation. Par exemple, il est possible que la solution de l'approximation ne soit pas réalisable pour le problème original ou qu'elle soit très éloignée de la solution exacte du MINLP. Le travail de [Geißler et al., 2012] sur cette thématique est remarquable, puisqu'il propose plusieurs stratégies pour obtenir des propriétés d'approximation par une construction particulière du problème MILP approximant le problème original. Entre autres, il propose une formulation des fonctions linéaires par morceaux dans un problème MILP qui correspond à une relaxation des contraintes du problème original, et une construction des fonctions linéaires par morceaux telle qu'elles satisfont une borne sur l'erreur d'approximation des fonctions non linéaires correspondantes.

L'approximation par un MILP a pour but de fournir une solution approchée en un temps de calcul raisonnable. Des fonctions linéaires par morceaux avec un grand nombre de morceaux sont souvent nécessaires pour avoir une bonne approximation. Or, l'utilisation de telles fonctions ralentit la résolution du problème MILP. Sur la lancée de récents travaux sur les fonctions d'une seule variable, nous étudions ce point en nous concentrant sur le problème d'approximation d'une fonction de deux variables par une fonction linéaire par morceaux qui respecte une borne sur l'erreur d'approximation et qui a le nombre minimum de morceaux. Nous notons ce problème le $\mathbb{R}^2$-corridor fitting problem ($\mathbb{R}^2$-CFP).

Ce manuscrit est composé de 6 chapitres. Le Chapitre 1 présente une revue de littérature organisée de manière à se focaliser progressivement sur le cœur de notre sujet. Après une pré-

sentation des principales sous-classes des problèmes MINLP, la méthode d'approximation de MINLP par des problèmes MILP grâce aux fonctions linéaires par morceaux est introduite en détail. Finalement, nous présentons les méthodes existantes d'approximation de fonctions par des fonctions linéaires par morceaux respectant une borne sur l'erreur d'approximation et ayant pour but la minimisation du nombre de morceaux.

Le Chapitre 2 motive la construction de fonctions linéaires par morceaux les plus générales possibles dans le but d'en obtenir avec peu de pièces. Il présente aussi la notion de corridor pour représenter avec le même formalisme toute une classe d'erreurs d'approximation. Finalement, il introduit le $\mathbb{R}^2$-CFP et un vocabulaire spécifique pour le $\mathbb{R}^2$-CFP autour de la notion de corridor.

Le Chapitre 3 est le premier des chapitres de contribution. Il décrit une trame pour la génération d'heuristiques du $\mathbb{R}^2$-CFP basée sur des propriétés d'approximation et le respect de trois principes pour limiter le nombre de pièces utilisées. Des expériences numériques analysent les différents paramétrages de la trame et comparent nos meilleures heuristiques à l'état de l'art. Il en résulte que les solutions proposées par nos heuristiques utilisent en moyenne moins de morceaux que celles des algorithmes de la littérature, en particulier dans le cas des fonctions non linéairement séparables.

Les différences importantes de nombre de morceaux des solutions précédentes de l'état de l'art et de nos heuristiques du chapitre précédent motivent le calcul de bornes inférieures pour le $\mathbb{R}^2$-CFP du Chapitre 4. Nous signalons qu'aucune borne inférieure ou méthode exacte n'a été proposée précédemment dans la littérature sur l'approximation de fonctions de deux variables. Nous identifions une famille d'inégalités valides issue d'un problème de coloration d'hypergraphe à partir de laquelle nous proposons les quatre premières méthodes produisant des bornes inférieures sur le nombre de morceaux optimal. Les expériences numériques comparent ces quatre méthodes et prouvent pour la première fois des solutions optimales à un tiers du jeu d'instances.

Le Chapitre 5 traite de l'approximation linéaire par morceaux de la norme euclidienne de deux variables dans des contraintes issues de la méthode de linéarisation de [Camino et al., 2019]. Il est montré que cette linéarisation utilise le nombre minimum de contraintes linéaires pour respecter une borne sur une erreur d'approximation adaptée. Ensuite, deux extensions de cette méthode sont proposées. L'une adapte la linéarisation de [Camino et al., 2019] à des normes dont les lignes de niveaux sont des ellipses, et l'autre montre que l'on peut construire une solution optimale du $\mathbb{R}^2$-CFP dans le cas particulier de la norme euclidienne ou d'une norme dont les lignes de niveaux sont des ellipses lorsque le domaine de définition est l'ensemble du plan.

Deux applications numériques sont présentées dans le Chapitre 6. La première traite d'un problème de placement de faisceaux d'un satellite à l'aide de la linéarisation des contraintes utilisant les normes de lignes de niveaux elliptiques du chapitre précédent. Les résultats numériques

identifient pour quelles instances cette linéarisation est intéressante. La deuxième application établit une méthode de calcul d'équilibres de Nash approchés à partir de l'approximation de fonctions avec une borne sur l'erreur dans la classe de jeux appelée integer programming games (IPG), et applique cette méthode à un jeu d'investissement en cybersécurité. De la même manière que pour la première application, les caractéristiques des instances pour lesquelles notre méthode trouve une solution plus rapidement sont identifiées.

Le manuscrit se termine par une conclusion de nos travaux. Dans l'ensemble de cette thèse, les termes fonction linéaire et fonction linéaire par morceaux seront utilisés dans leurs acceptions anglaises, qui correspondent à fonction affine et fonction affine par morceaux en français.

Glossaire

Notations

Dans l'ensemble de ce travail, nous prenons les notations suivantes :

- A^T est la transposée de la matrice A ;
- $||x||_2 = \sqrt{\sum_{i=1,\dots,m} x_i^2}$ est la norme euclidienne de $\mathbb{R}^m$;
- $d_2(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$ est la distance euclidienne de $\mathbb{R}^m$.

Un certain nombre de termes utilisés par la suite sont issus de l'anglais, tandis que certains, moins courants ou spécifiques à ce travail, sont issus du français. Par exemple, une *fonction linéaire* L est de la forme $L(x) = ax + b$ car nous l'utilisons avec le sens de *polynôme de degré 1*, une définition issue du terme anglais de *linear function*.

Acronymes

CC (*convex combination formulation*) : formulation par combinaison convexe
cMINLP (*convex mixed-integer nonlinear programming*) : optimisation convexe non linéaire mixte en nombres entiers
DC CPWL (*difference of convex continuous piecewise linear*) : différence de fonctions linéaires par morceaux continues convexes
DLog (*disaggregated logarithmic formulation*) : formulation logarithmique désagrégée
GSIP (*generalized semi-infinite programming*) : optimisation semi-infinie généralisée
LP (*linear programming*) : optimisation linéaire
MILP (*mixed-integer linear programming*) : optimisation linéaire mixte en nombres entiers
MINLP (*mixed-integer nonlinear programming*) : optimisation non linéaire mixte en nombres entiers
MIQCQP (*mixed-integer quadratically constrained quadratic programming*) : optimisation quadratique mixte en nombres entiers avec contraintes quadratiques
NLP (*nonlinear programming*) : optimisation non linéaire
PWL (*piecewise linear*) : linéaire par morceaux (en parlant d'une fonction)

Définitions

Nous nous plaçons dans l'espace métrique $(\mathbb{R}^m, d_2)$, c'est-à-dire que nous travaillons avec l'ensemble $\mathbb{R}^m$ et que nous évaluons la distance entre deux points avec la distance euclidienne d_2. Nous notons $B(x_0, r) = \{x \in \mathbb{R}^m | d_2(x_0, x) < r\}$ la *boule ouverte de rayon r centrée en x_0*.

Ouvert et fermé. Un ensemble U est un *ouvert* de $\mathbb{R}^m$ si et seulement si c'est une union de boules ouvertes de $\mathbb{R}^m$. Un ensemble F est un *fermé* de $\mathbb{R}^m$ si et seulement si c'est le complémentaire dans $\mathbb{R}^m$ d'un ouvert.

Intérieur, adhérence et frontière. Soit X un ensemble de $\mathbb{R}^m$. On appelle *intérieur de* X noté $Int(X)$ le plus grand ouvert U inclus dans X. On appelle *adhérence de* X le plus petit fermé F contenant X. On appelle *frontière de* X l'ensemble obtenu comme la différence ensembliste de l'adhérence de X et de l'intérieur de X. Intuitivement, la frontière d'un ensemble est sa bordure. Cependant, suivant s'il est ouvert ou fermé localement, la bordure fait ou non partie de l'ensemble.

Ensemble connexe. Soit un ensemble $X \in \mathbb{R}^m$. On dit que X est un *ensemble connexe* si ce n'est pas l'union de deux ouverts non vides disjoints de $\mathbb{R}^m$. Plus simplement, un ensemble est connexe s'il est "en un seul morceau".

Ensemble borné. Soit $X \subset \mathbb{R}^m$ un ensemble. On dit que X est un *ensemble borné* si la distance euclidienne entre deux points de X est bornée supérieurement par une valeur $v \in \mathbb{R}^+$.

Ensemble fini, ensemble infini dénombrable, ensemble infini indénombrable. Un ensemble S est dit un *ensemble fini* (respectivement *ensemble infini dénombrable*) lorsqu'il est possible d'indexer les éléments de S avec les entiers naturels inférieurs à une valeur donnée (resp. les entiers naturels) sans répéter ni omettre d'éléments. Si ce n'est pas possible, S est dit un *ensemble infini indénombrable*. L'ensemble $\mathbb{Z}$ est un ensemble infini dénombrable, tandis que $\mathbb{R}$ est un ensemble infini indénombrable.

Ensemble convexe. Soit $S \subset \mathbb{R}^m$ un ensemble. On dit que X est un *ensemble convexe* si pour tout couple de points $x_1, x_2 \in X$, le segment $[x_1, x_2]$ est inclus dans X.

Graphe d'une fonction. Soit f une fonction de $D \subset \mathbb{R}^m$ dans $\mathbb{R}$. On appelle *graphe d'une fonction* le sous-ensemble de $\mathbb{R}^m \times \mathbb{R}$ contenant les points $(x, f(x))$ pour tout $x \in D$.

Fonction convexe, fonction concave. Soit f une fonction de $\mathbb{R}^m$ dans $\mathbb{R}$. On dit que f est une *fonction convexe* si l'ensemble $\{(x, z) \in \mathbb{R}^m \times \mathbb{R} | f(x) \leq z\}$ des points au-dessus de la courbe de f est convexe. On dit que f est une *fonction concave* si $-f$ est convexe. Les fonctions linéaires sont à la fois convexes et concaves.

Fonction linéairement séparable. Soit f une fonction de $\mathbb{R}^m$ dans $\mathbb{R}$. On dit que la fonction f est *linéairement séparable* si elle peut s'écrire comme une somme de fonctions d'une variable : $f(x) = \sum_{i=1,\ldots,m} f_i(x_i)$. À l'inverse, si f n'est pas linéairement séparable elle est dite *non linéairement séparable*.

Fonction continue. Soient f une fonction de $\mathbb{R}^m$ dans $\mathbb{R}$ et $x_0 \in \mathbb{R}^m$. On dit que la fonction f est *continue en* x_0 si pour tout $\epsilon > 0$, il existe $\eta > 0$ tel que pour tout $x \in \mathbb{R}^m$, $d_2(x_0, x) < \eta \Rightarrow |f(x_0) - f(x)| < \epsilon$. On dit que la fonction f est *continue sur* $\mathbb{R}^m$ si elle est continue en tout point $x_0 \in \mathbb{R}^m$. Une fonction qui n'est pas continue est dite *discontinue*. Pour désigner une fonction qui peut être continue ou discontinue, nous utilisons le terme *non nécessairement*

continue. Intuitivement, cela revient à demander que le graphe de la fonction soit d'un seul morceau.

Fonction de classe $C^k(\mathbb{R}^m, \mathbb{R})$. Soit f une fonction de $\mathbb{R}^m$ dans $\mathbb{R}$. On dit que f *est de classe* $C^k(\mathbb{R}^m, \mathbb{R})$ si elle est différentiable k fois sur $\mathbb{R}^m$ et que sa k-ième différentielle est continue. Pour simplifier la notation, nous notons "f est C^k" pour dire "f est de classe $C^k(\mathbb{R}^m, \mathbb{R})$".

Polygone, polygone simple. Un *polygone* est un ensemble de points du plan délimité par une chaîne fermée de segments (ou *côtés*). Un polygone est dit *simple* si deux de ses côtés ne s'intersectent qu'en leurs extrémités et seulement s'ils sont consécutifs dans la chaîne. Dans ce manuscrit, à part mention explicite, le mot *polygone* est utilisé à la place de polygone simple. La Figure 1 montre un exemple de polygone non simple à quatre côtés car deux côtés non consécutifs s'intersectent au point rouge.

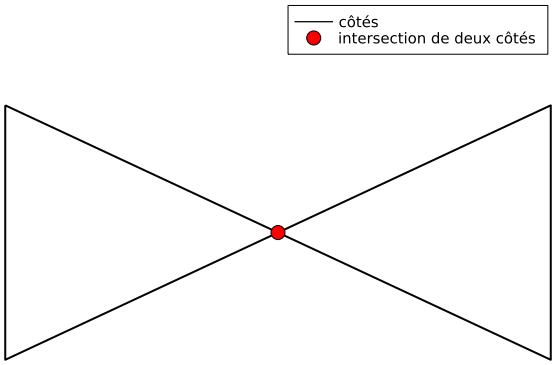


FIGURE 1: Exemple de polygone non simple à quatre côtés

Combinaison convexe. Soient des points x_i de $\mathbb{R}^m$ pour $i = 1, ..., n$ et des réels positifs λ_i pour $i = 1, ..., n$ respectant $\sum_{i=1}^{n} \lambda_i = 1$. On appelle *combinaison convexe* des x_i un point $x = \sum_{i=1}^{n} \lambda_i x_i$.

Point extrémal. Soient X un ensemble convexe et x, x_1 et x_2 des éléments de X. On appelle x un *point extrémal* de X si toute combinaison convexe de x_1 et x_2 donnant x implique que $x = x_1$ ou $x = x_2$.

Demi-espace, polyèdre. Soit $u \in \mathbb{R}^m$ un vecteur et b un réel. On appelle *demi-espace* un ensemble de la forme $\{x \in \mathbb{R}^m | u.x \leq b\}$. Soit $X \subset \mathbb{R}^m$. On dit que X est un *polyèdre* si c'est l'intersection d'un nombre fini de demi-espaces. Un demi-espace de $\mathbb{R}^2$ est appelé un *demi-plan*. Les points extrémaux des polyèdres s'appellent les *sommets*. Les termes polyèdre et *polytope* ont un sens différent :

Enveloppe convexe, polytope. L'*enveloppe convexe* $\mathcal{P}$ de points $(X_i)_{i \in I} \subset \mathbb{R}^m$ est l'ensemble

qui contient toutes les combinaisons convexes de points de $(X_i)_{i\in I}$:

$$\mathcal{P} = \{x \in \mathbb{R}^m | x = \sum_i \lambda_i X_i, \sum_i \lambda_i = 1, \lambda_i \geq 0 \ \forall i = 1, ..., n\}.$$

Un *polytope* est l'enveloppe convexe d'un ensemble fini de points. C'est la généralisation à $\mathbb{R}^m$ du polygone dans $\mathbb{R}^2$. Si un polyèdre est borné, alors c'est un polytope.

Partition. Soit X un ensemble. On appelle *partition* de X un ensemble de parties non vides de X tel que l'union des parties forme X et leurs intersections deux à deux sont vides.

Domaine polygonal, triangulation. Soit D un ensemble connexe de $\mathbb{R}^m$. On appelle D un *domaine polygonal* si la frontière de D est un polygone. Soit D un domaine polygonal de $\mathbb{R}^2$. On appelle *triangulation* une partition de D en triangles avec la propriété que si le sommet d'un triangle T est sur le côté d'un triangle T', alors c'est aussi un sommet du triangle T'.

Problème d'optimisation. Soient une fonction $f : \mathbb{R}^m \mapsto \mathbb{R}$ et un ensemble X de $\mathbb{R}^m$. On appelle *problème d'optimisation* le problème $(\mathcal{P})$ consistant à trouver une valeur $x^* \in X$ des *variables* x telle que $f(x^*) \leq f(x)$ pour tout x dans X, où X est appelé l'*ensemble réalisable* du problème $(\mathcal{P})$.

$$(\mathcal{P}) \begin{cases} \min\limits_x & f(x) & (1) \\ \textit{sous les contraintes} & x \in X & (2) \end{cases}$$

On appelle f la *fonction objectif* de $(\mathcal{P})$, x^* une *solution optimale* de $(\mathcal{P})$ et $f(x^*)$ la *valeur optimale* de $(\mathcal{P})$. L'ensemble X peut être décrit par des *contraintes* sur les variables x, comme par exemple $x \geq 0$ ou $g(x) \leq 0$ pour une fonction $g : \mathbb{R}^m \mapsto \mathbb{R}$. Le problème $(\mathcal{P})$ est plus particulièrement un problème de minimisation car le but est de trouver le minimum de la fonction objectif. Le terme min peut être remplacé par le terme max pour représenter un problème de maximisation, qui est aussi un problème d'optimisation.

Problème convexe. Soit $(\mathcal{P})$ un problème d'optimisation de la forme :

$$\begin{cases} \min\limits_x & f(x) & (3) \\ s.c. & x \in C. & (4) \end{cases}$$

Si C est un ensemble convexe et que f est une fonction convexe, alors on dit que $(\mathcal{P})$ est un *problème convexe*.

Relaxation d'un problème. Soit $(\mathcal{P})$ un problème d'optimisation de fonction objectif f_P. On appelle *relaxation* de $(\mathcal{P})$ un problème $(\mathcal{P}')$ de fonction objectif $f_{P'}$ tel que l'ensemble réalisable de $(\mathcal{P})$ est inclus dans celui de $(\mathcal{P}')$ et tel qu'il vérifie pour tout x dans l'ensemble réalisable de $(\mathcal{P})$: $f_{P'}(x) \leq f_P(x)$ dans le cas de la minimisation, et $f_{P'}(x) \geq f_P(x)$ dans le cas de la maximisation.

Contrainte d'intégrité, variable entière. Soit $(\mathcal{P})$ un problème d'optimisation sur les va-

riables $x \in \mathbb{R}^m$. On appelle *contrainte d'intégrité* une contrainte de la forme $x_i \in \mathbb{Z}$ pour $i \in \{1, ..., m\}$ car elle force la variable x_i à prendre des valeurs entières. Dans ce cas, on appelle la variable x_i une *variable entière*.

Relaxation continue, solution entière. On appelle *relaxation continue* d'un problème $(\mathcal{P})$ comportant des variables entières le problème obtenu en retirant les contraintes d'intégrité. Soit $(\mathcal{P})$ un problème. On dit qu'une solution optimale d'une relaxation continue de $(\mathcal{P})$ est une *solution entière* si elle respecte toutes les contraintes d'intégrité de $(\mathcal{P})$.

Inégalité valide. Soient $X \in \mathbb{R}^m$ un ensemble, $a \in \mathbb{R}^m$ un vecteur et b un réel. On dit que la contrainte linéaire $ax \leq b$ est une *inégalité valide* pour X si elle est vérifiée pour tout $x \in X$.

Coupe. Soient $(\mathcal{P})$ un problème d'optimisation avec des variables entières, $(\mathcal{P}')$ sa relaxation continue et $\hat{x}$ une solution réalisable de $(\mathcal{P}')$ mais qui n'est pas une solution réalisable de $(\mathcal{P})$. On appelle *coupe* une contrainte linéaire que $\hat{x}$ ne satisfait pas, mais que toutes les solutions réalisables de $(\mathcal{P})$ satisfont.

Borne inférieure d'un problème, borne supérieure d'un problème. Soit $(\mathcal{P})$ un problème d'optimisation de valeur optimale v^*. On appelle *borne inférieure* (respectivement *borne supérieure*) du problème $(\mathcal{P})$ toute valeur $v^- \leq v^*$ (resp. $v^+ \geq v^*$). On dit que v^- (resp. v^+) borne inférieurement (resp. supérieurement) la valeur optimale de $(\mathcal{P})$.

Application linéaire, application affine, base, changement de base. L'algèbre linéaire est la discipline qui étudie les propriétés des applications linéaires. On appelle *application linéaire* une fonction linéaire $L : \mathbb{R}^m \mapsto \mathbb{R}^n$, avec $m, n \in \mathbb{N}^*$. Soit $v \in \mathbb{R}^m$. Une *translation* de vecteur v est une fonction de $\mathbb{R}^m$ dans $\mathbb{R}^m$ qui à x associe $x + v$. Une *application affine* est la composition d'une application linéaire et d'une translation.

Soit une famille de réels $a_i \in \mathbb{R}$ pour tout $i = 1, ..., m$. Une famille $F = \{x_1, ..., x_m\}$ de vecteurs de $\mathbb{R}^n$ est *linéairement indépendante* si l'équation $\sum_{i=1,...,m} a_i x_i = 0$ a pour unique solution $a_i = 0$ pour tout i. Une famille $F = \{x_1, ..., x_m\}$ de vecteurs de $\mathbb{R}^m$ est dite *génératrice* de $\mathbb{R}^m$ si l'ensemble $\{\sum_{i=1,...,m} a_i x_i | a_i \in \mathbb{R}\ \forall i = 1, ..., m\}$ des combinaisons linéaires de F est $\mathbb{R}^m$. Une famille F de $\mathbb{R}^m$ qui est linéairement indépendante et génératrice de $\mathbb{R}^m$ est une *base* de $\mathbb{R}^m$. Une base $B = \{b_1, ..., b_m\}$ de $\mathbb{R}^m$ permet de représenter n'importe quel vecteur $x \in \mathbb{R}^m$ par un vecteur x_B de $\mathbb{R}^m$. x^B représente le vecteur de valeur $\sum_{i=1,...,m} x_i^B b_i$. La base canonique $E := \{e_1, ..., e_m\}$ est utilisée par défaut, où e_i représente le vecteur avec des 0 à chaque coordonnée sauf à la ième coordonnée où il vaut 1. La matrice de changement de base T de la base B de $\mathbb{R}^m$ vers une base B' de $\mathbb{R}^m$ permet de calculer la représentation de x^B dans la base B'. Cela s'appelle un *changement de base*.

Chapitre 1

État de l'art

Sommaire

Le travail de cette thèse a pour finalité la résolution d'une classe de problèmes d'optimisation appelée *optimisation non linéaire mixte en nombre entiers* (*mixed-integer nonlinear programming* en anglais) que l'on note MINLP. Nous décrivons cette classe dans la Section 1.1. Une méthodologie très étudiée pour approximer un problème MINLP par un problème MILP consiste à remplacer les fonctions non linéaires par des fonctions linéaires par morceaux. Elle est abordée étape par étape dans la Section 1.2. Ensuite, la Section 1.3 présente les méthodes de l'état de l'art pour approximer une fonction non linéaire par une fonction linéaire par morceaux en respectant une erreur d'approximation absolue et en minimisant le nombre de morceaux. Finalement, la Section 1.4 conclut sur l'état de l'art.

1.1 Les problèmes MINLP

Définition 1 (problème MINLP)**.** Soient n_1, n_2, m_e et m_c des entiers positifs. Soient f, g_i pour $i = 1, ..., n_1$ et h_j pour $j = 1, ..., n_2$ des fonctions de $\mathbb{R}^{m_e+m_c}$ dans $\mathbb{R}$. On appelle *problème de la*

classe MINLP un problème d'optimisation de la forme :

$$\begin{cases} \underset{x\in\mathbb{R}^{m_e+m_c}}{min} & f(x) & (1.1)\\ \textit{sous les contraintes} & g_i(x)\leq 0 \quad \forall i=1,...,n_1 & (1.2)\\ & h_j(x)=0 \quad \forall j=1,...,n_2 & (1.3)\\ & x_i\in\mathbb{Z} \quad \forall i=1,...,m_e & (1.4) \end{cases}$$

Les Contraintes (1.4) sont appelées *contraintes d'intégrité* car elles définissent le domaine des variables entières. Une revue de littérature sur la résolution de ces problèmes peut être trouvée dans l'article [Belotti et al., 2013].

De nombreuses situations se modélisent en problèmes MINLP [Maranas et Floudas, 1994, Borghetti et al., 2008, Misener et al., 2009, Boukouvala et al., 2016, Medeiros et Resendo, 2022, Hante et Schmidt, 2023]. Ils sont considérés difficiles à résoudre pour plusieurs raisons : la présence de fonctions non linéaires dans les contraintes et/ou dans la fonction objectif, la présence de variables entières qui rend l'ensemble réalisable du problème non connexe et la non convexité des problèmes. Il existe plusieurs sous-classes de problèmes MINLP très étudiées définies par l'absence de certaines de ces caractéristiques difficiles à traiter. Parmi les sous-classes de MINLP les plus connues, citons les problèmes d'optimisation non linéaire (NLP pour *nonlinear programming*) en l'absence de contraintes d'intégrité, les problèmes d'optimisation linéaire en nombres entiers (MILP pour *mixed-integer linear programming*) en l'absence de fonctions non linéaires, les problèmes d'optimisation linéaire (LP pour *linear programming*) en l'absence de contraintes d'intégrité et de fonctions non linéaires, et les problèmes MINLP convexes (cMINLP pour *convex MINLP*) si le problème privé des contraintes d'intégrité est un problème convexe.

Définition 2 (problème NLP)**.** Soient n_1, n_2 et m des entiers positifs, et soient f, g_i pour $i=1,...,n_1$ et h_j pour $j=1,...,n_2$ des fonctions de $\mathbb{R}^m$ dans $\mathbb{R}$. On appelle *problème de la classe NLP* un problème d'optimisation de la forme :

$$\begin{cases} \underset{x\in\mathbb{R}^{m}}{min} & f(x) & (1.5)\\ s.c. & g_i(x)\leq 0 \quad \forall i=1,...,n_1 & (1.6)\\ & h_j(x)=0 \quad \forall j=1,...,n_2 & (1.7) \end{cases}$$

Il est par exemple possible de modéliser un grand nombre de problèmes issus de processus chimiques en problème NLP [Biegler, 2010].

Définition 3 (problème MILP)**.** Soient n, m_e et m_c des entiers positifs, $c\in\mathbb{R}^{m_e+m_c}$, $b\in\mathbb{R}^n$ des vecteurs, et $A\in\mathbb{R}^{n\times(m_e+m_c)}$. On appelle *problème de la classe MILP* un problème

d'optimisation de la forme :

$$\begin{cases} \min\limits_{x\in\mathbb{R}^{m_e+m_c}} & c^T x & (1.8)\\ s.c. & Ax \leq b & (1.9)\\ & x_i \in \mathbb{Z} \quad \forall i = 1, ..., m_e & (1.10) \end{cases}$$

Pour correspondre à la formulation standard des problèmes MILP, ce sont des contraintes linéaires de la forme $Ax \leq b$ qui remplacent les contraintes d'inégalités et d'égalités $g_i(x) \leq 0$ et $h_j(x) = 0$ des problèmes MINLP et NLP. Les contraintes d'égalités de la forme $Ax = b$ peuvent se modéliser avec les contraintes d'inégalités $Ax \leq b$ et $(-A)x \leq -b$. Les problèmes MILP appartiennent à la classe de complexité NP-complet [Kannan et Monma, 1978] et sont donc considérés dur à résoudre.

Définition 4 (problème LP)**.** Soient n et m des entiers positifs, $c \in \mathbb{R}^m$, $b \in \mathbb{R}^n$, et $A \in \mathbb{R}^{n\times m}$. On appelle *problème de la classe LP* un problème d'optimisation de la forme :

$$\begin{cases} \min\limits_{x\in\mathbb{R}^m} & c^T x & (1.11)\\ s.c. & Ax \leq b & (1.12) \end{cases}$$

Très étudiés, les problèmes LP font partie de la classe de complexité P car il existe des algorithmes polynomiaux pour les résoudre, notamment les méthodes de points intérieurs [Potra et Wright, 2000]. Des variantes de l'algorithme du simplexe sont couramment utilisées, malgré leur complexité algorithmique non polynomiale [Klee et Minty, 1972], du fait de leur performance en pratique.

La Figure 1.1 présente les différentes sous-classes de problème MINLP et leurs liens d'inclusion. Une flèche d'une classe A vers une classe B indique que la classe B est incluse dans la classe A, et la couleur et le style de la flèche indique la propriété nécessaire à un problème de la classe A pour qu'il soit aussi dans la classe B.

1.2 Méthodes d'approximation linéaire par morceaux d'un problème MINLP

Nous décrivons ici une méthode de résolution de MINLP qui approxime le problème en retirant toute sa non linéarité par l'intermédiaire des *fonctions linéaires par morceaux* (fonctions PWL) [Misener et Floudas, 2010, Geißler et al., 2012].

Définition 5 (fonction PWL)**.** Soit D un polytope de $\mathbb{R}^m$. Une fonction $g : D \mapsto \mathbb{R}$ est une *fonction PWL* à n pièces (ou morceaux) si et seulement s'il existe un ensemble de vecteurs $\{a_i\}_{i=1,...,n}$ de $\mathbb{R}^m$, un ensemble de réels $\{b_i\}_{i=1,...,n}$ et une famille de polytopes $\{D_i\}_{i=1,...,n} \subset \mathbb{R}^m$ d'intérieur non vide tels que :

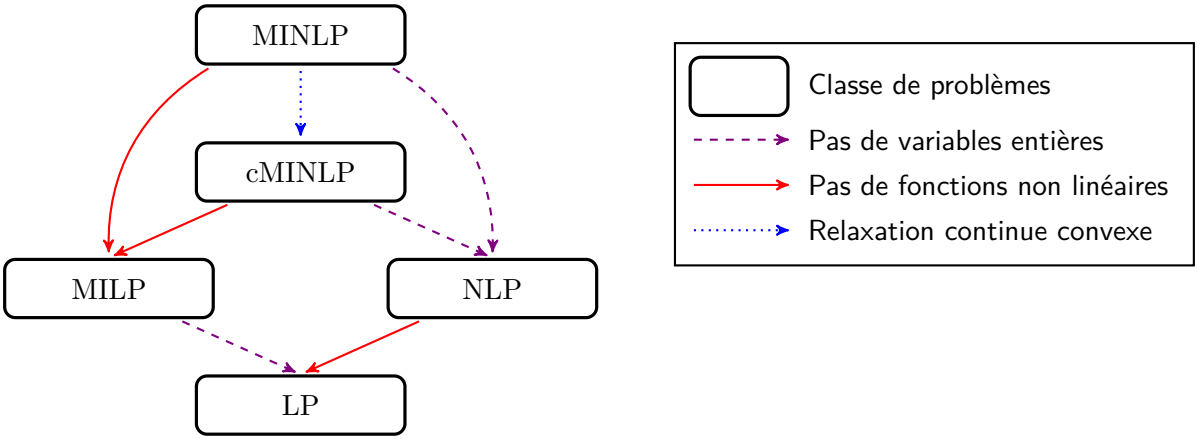


FIGURE 1.1: Sous-classes de problème MINLP

— $D = \cup_{i=1,...,n} D_i$,
— pour $i \neq j$ les polytopes D_i et D_j ont une intersection d'intérieur vide,
— g est définie sur l'intérieur de D_i par $g(x) = a_i^T \cdot x + b_i$, avec $\cdot$ la notation pour le produit scalaire usuel, et
— g est définie sur la frontière de D_i par $g(x) = \min\{a_k^T \cdot x + b_k | k = 1, ..., n \text{ et } x \in D_k\}$.

Soit $i = 1, ..., n$, on appelle *pièce* l'ensemble de points $(x, g(x))$ pour lesquels $g(x) = a_i^T \cdot x + b_i$ pour $x \in D_i$ et *domaine d'une pièce* l'ensemble de points x sur lequel $g(x) = a_i^T \cdot x + b_i$ pour $x \in D_i$.

La dernière condition assure la bonne définition de la fonction sur les frontières des domaines des pièces, la fonction PWL g pouvant être discontinue à la frontière des polytopes D_i, ce qui laisserait plusieurs évaluations possibles de $g(x)$ en ces points. $g(x)$ y est définie comme le minimum sur toutes les définitions possibles, donc g est semi-continue inférieurement. L'utilisation d'un maximum dans la dernière condition assurerait aussi la bonne définition de la fonction, et rendrait g semi-continue supérieurement. Ces propriétés de régularité de la fonction PWL ont un intérêt pour la formulation de fonctions PWL non continues dans un problème MILP [Jeroslow, 1985].

La Figure 1.2 montre une fonction PWL g d'une seule variable. la projection des points de cassure de la fonction PWL sur l'axe des abscisses permet de visualiser l'intervalle qui est le domaine de chaque pièce. La fonction g est discontinue au point d'abscisse 1.9, et vaut 0.6, parce que c'est le minimum parmi les deux fonctions linéaires candidates. La Figure 1.3 montre une fonction PWL de deux variables. La partie supérieure montre que le graphe de la fonction est formé de plusieurs pièces. La partie inférieure est la projection du graphe sur le domaine de la fonction, et montre le découpage du domaine en polygones, qui sont ici des triangles, sur lesquels la fonction est linéaire, c'est-à-dire de la forme $L(x) = ax + b$. Nous utilisons parfois

l'expression "linéariser par morceaux une fonction" au lieu de "construire une approximation linéaire par morceaux d'une fonction".

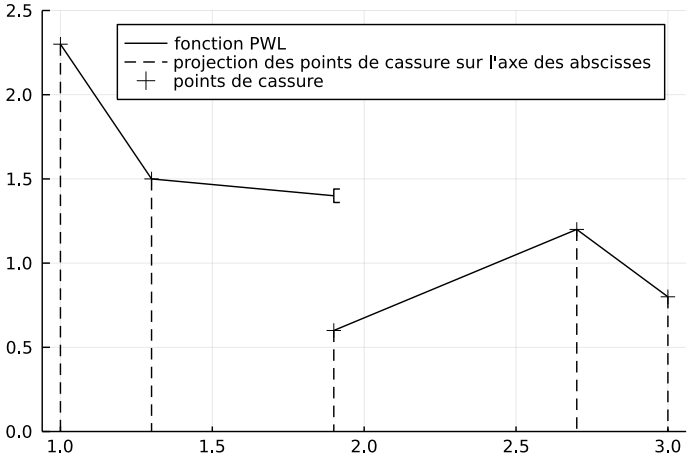


FIGURE 1.2: Fonction PWL d'une variable

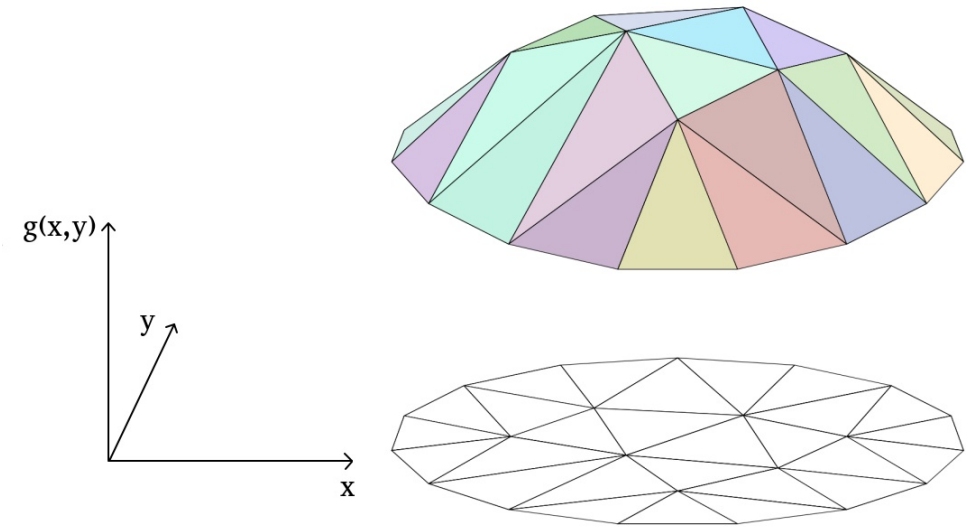


FIGURE 1.3: Fonction PWL de deux variables

Soit $(\mathcal{P})$ un problème MINLP. Le principe de la résolution de problème MINLP via une approximation linéaire par morceaux des fonctions non linéaires est le suivant. Il faut calculer pour chaque fonction non linéaire dans la modélisation de $(\mathcal{P})$ une fonction PWL qui l'approxime (Sections 1.2.1 et 1.3), puis remplacer dans le modèle mathématique chaque fonction non linéaire par sa linéarisation par morceaux, ce qui permet d'obtenir un MILP (Section 1.2.2). La résolution de ce MILP donnera une valeur qui n'est pas forcément une solution réalisable de $(\mathcal{P})$ (Section 1.2.3). En revanche, cette valeur peut vérifier des propriétés sous certaines

conditions, décrites dans la Section 1.2.4. Finalement, la Section 1.2.5 présente des méthodes basées sur l'itération de la procédure décrite ci-dessus en approximant localement de plus en plus précisément les fonctions PWL du problème MILP pour réduire le temps de calcul.

1.2.1 Construction classique d'une fonction PWL par interpolation

Dans cette section nous nous focalisons sur la construction de fonctions PWL par interpolation, qui a longtemps été la méthode par défaut. D'autres méthodes de construction, plus récentes, sont décrites en Section 1.3.

Commençons par définir l'interpolation pour une fonction d'une variable. Soit $f : [c, d] \mapsto \mathbb{R}$ une fonction non linéaire d'une seule variable. L'idée la plus simple pour l'approximer par une fonction PWL g est de l'interpoler en un certain nombre de points $(x_i)_{1\leq i\leq n+1}$ vérifiant $c = x_1 < x_2 < ... < x_{n+1} = d$ que l'on appelle les *points de cassure*. Nous définissons g entre deux points de cassure consécutifs de la manière suivante : $g(x_i) = f(x_i)$ pour $1 \leq i \leq n+1$, et $g(x) = f(x_i) + \frac{x-x_i}{x_{i+1}-x_i}(f(x_{i+1}) - f(x_i))$ si $x \in [x_i, x_{i+1}]$ pour $1 \leq i \leq n$. La fonction g est donc continue et linéaire sur chacun des n intervalles $[x_i, x_{i+1}]$. Il est possible de calculer les coefficients de la fonction linéaire $g_i(x) = a_i x + b_i$ définissant g sur l'intervalle $[x_i, x_{i+1}]$ grâce aux équations : $a_i = \frac{f(x_{i+1})-f(x_i)}{x_{i+1}-x_i}$ et $b_i = f(x_i) - a_i x_i$.

Pour les interpolations de fonctions de plusieurs variables, nous ne donnons que quelques exemples qui illustrent des idées clefs. L'article [Zhang et Wang, 2008] décrit une manière d'interpoler une fonction non linéaire de plusieurs variables $f : [c^1, d^1] \times ... \times [c^m, d^m] \mapsto \mathbb{R}$ à partir d'une division régulière de son domaine. Supposons que chaque intervalle $[c^i, d^i]$ soit divisé en n intervalles de même longueur. Le domaine de la fonction f est alors l'union de n^m hyperrectangles $[c^1_{k_1}, d^1_{k_1}] \times ... \times [c^m_{k_m}, d^m_{k_m}]$, $k_i \in \{1, ..., n\}$. Chacun de ces hyperrectangles a 2^m sommets utilisables pour l'interpolation, mais il suffit d'utiliser $m + 1$ points représentant m vecteurs qui forment une base pour définir de manière unique une fonction linéaire sur un sous-domaine de la fonction f. Pour ce choix de points, [Zhang et Wang, 2008] propose de découper chaque hyperrectangle en $m!$ *simplexes*, la généralisation du triangle à $m \geq 2$ dimensions, qui possèdent exactement les $m+1$ points requis. Plus précisément, le nombre minimum de simplexes nécessaires pour une partition de l'hypercube $[0, 1]^m$ de dimension m augmente très rapidement, comme calculé dans [Hughes et Anderson, 1996] : 5 simplexes pour la dimension 3, 16 pour la dimension 4, 67 pour la dimension 5 et au moins 270 pour la dimension 6. Cette croissance rapide du nombre de simplexes en fonction de la dimension limite l'intérêt de construire des fonctions PWL par interpolation pour un grand nombre m de dimensions : en effet, le nombre de variables et contraintes à introduire dans un modèle mathématique pour y formuler une fonction PWL augmente avec son nombre de pièces.

En complément, pour le cas des fonctions de deux variables qui nous intéresse dans cette

thèse, l'article [D'Ambrosio et al., 2010] propose l'analyse de trois modélisations de fonctions non linéaires de deux variables par interpolation. Une méthode exploite l'interpolation de fonctions d'une seule variable, une autre produit des pièces dont les domaines sont des triangles tandis que pour la dernière ce sont des rectangles. Il est montré que c'est la méthode à base de rectangles qui permet à la fois de résoudre suffisamment vite le modèle MILP et d'obtenir des solutions qui approximent avec une qualité satisfaisante la solution du problème original. La vitesse de résolution de cette méthode vient du nombre de variables binaires nécessaires dans le modèle, qui est significativement plus faible que dans la méthode à base de triangles. Il existe aussi des méthodes d'interpolations avec des points de cassure répartis non uniformément, comme [Bärmann et al., 2023] pour l'approximation de la fonction de deux variables $f(x, y) = xy$.

Mentionnons enfin que dans le cadre applicatif qui nous intéresse, il n'est pas nécessaire d'imposer la propriété $g(x, y) = f(x, y)$ pour tout (x, y) sommet d'une pièce de g, ceci afin de mieux approximer f globalement. Ce n'est donc pas forcément une interpolation de la fonction f que nous recherchons. En revanche, nous avons besoin de méthodes garantissant que la différence en n'importe quel point du domaine entre la fonction non linéaire et la fonction PWL est plus petite qu'une valeur prédéfinie. Ces méthodes dédiées à ce cas sont décrites dans la Section 1.3.

1.2.2 Formulation d'une fonction PWL dans un problème MILP

Une fois qu'une fonction PWL approximant une fonction non linéaire a été construite, il faut la modéliser au sein d'un problème MILP. La façon de produire des variables et des contraintes décrivant cette fonction PWL dans un problème MILP est ce que l'on appelle une *formulation* de la fonction PWL.

Les formulations de fonctions PWL dans un problème MILP ont fait l'objet de nombreuses études [Padberg, 2000, Keha et al., 2004, Vielma et Nemhauser, 2011, Sridhar et al., 2013, Vielma, 2015, Huchette et Vielma, 2022]. Nous discutons ici des caractéristiques de ces formulations. Considérons tout d'abord l'important cas particulier de la modélisation d'une fonction PWL convexe de plusieurs variables dans un problème de minimisation. Cette formulation ne requiert qu'une variable continue additionnelle et des contraintes linéaires : une fonction PWL convexe g peut s'écrire comme $g(x) = \max_{k=1,\dots,n} m_k x + c_k$ qui se traduit en contraintes linéaires par $z \geq m_k x + c_k$ pour tout $k = 1, \dots, n$ et par l'ajout du terme $+z$ dans la fonction objectif à minimiser. La Figure 1.4 montre une fonction PWL convexe à trois pièces formulée grâce aux trois contraintes linéaires dont la fonction linéaire correspondante est en pointillé.

Nous désignons par *taille d'une formulation* le nombre de variables continues, de variables binaires et de contraintes linéaires qu'il faut ajouter à un modèle MILP pour formuler une fonction PWL en fonction de son nombre de pièces et parfois d'autres informations spécifiques [Vielma et al., 2010]. Il est possible de séparer les formulations en deux catégories de tailles :

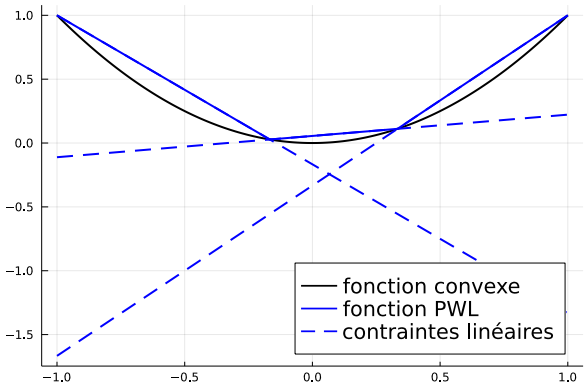


FIGURE 1.4: Exemple de formulation de fonction PWL convexe à trois pièces

celles qui introduisent un nombre de variables continues, de variables binaires et de contraintes qui augmente au moins linéairement avec le nombre de pièces ; et celles qui en introduisent un nombre linéaire pour les variables continues et logarithmique pour les variables binaires et les contraintes. Nous appelons la deuxième catégorie les *formulations logarithmiques*. Ces formulations logarithmiques sont possibles en exploitant le fait qu'un vecteur binaire de k coordonnées peut prendre 2^k valeurs différentes.

Comme exemple de formulation non logarithmique, considérons la formulation dite par *combinaison convexe* (CC). Elle consiste à représenter le graphe $\mathcal{P}$ de la fonction PWL continue g d'une ou de plusieurs variables à partir de combinaisons convexes des sommets $v \in \mathcal{V}(\mathcal{P})$ des pièces $P \in \mathcal{P}$ de g. Notons $\mathcal{P}(v) = \{P \in \mathcal{P} | v \in P\}$ l'ensemble de pièces de g auquel appartient le sommet v. Les variables intervenant dans le modèle sont la variable x qui représente le point en lequel la fonction g est évaluée, la variable z qui représente $g(x)$, les variables continues λ_v pour $v \in \mathcal{V}(\mathcal{P})$ qui permettent d'exprimer la combinaison convexe des sommets de pièces de g et les variables binaires y_P qui sélectionnent la pièce P sur laquelle se trouve la variable z. Finalement, sur la pièce P, g est la fonction linéaire $g(x) = m_P x + c_P$, avec m_P et c_P des paramètres. Les contraintes à intégrer au modèle sont :

$$\sum_{v \in \mathcal{V}(\mathcal{P})} \lambda_v v = x, \tag{1.13}$$

$$\sum_{v \in \mathcal{V}(\mathcal{P})} \lambda_v (m_P v + c_P) = z, \tag{1.14}$$

$$\lambda_v \geq 0 \quad \forall v \in \mathcal{V}(\mathcal{P}), \tag{1.15}$$

$$\sum_{v \in \mathcal{V}(\mathcal{P})} \lambda_v = 1, \tag{1.16}$$

$$\lambda_v \leq \sum_{P \in \mathcal{P}(v)} y_P \qquad \forall v \in \mathcal{V}(\mathcal{P}), \tag{1.17}$$

$$\sum_{P\in\mathcal{P}(v)} y_P = 1, \tag{1.18}$$

$$y_P \in \{0,1\} \quad \forall P \in \mathcal{P}. \tag{1.19}$$

Les contraintes (1.13) et (1.14) sont les combinaisons convexes des variables λ_v pour donner x et z respectivement. Les contraintes (1.15) et (1.16) proviennent de l'expression de la combinaison convexe des variables continues λ_v, tandis que les contraintes (1.17) forcent la variable λ_v à ne pas participer à la combinaison convexe si z n'est pas sur une pièce de l'ensemble $\mathcal{P}(v)$. Finalement, la contrainte (1.18) limite l'appartenance de z à une seule pièce de g.

Comme exemple de formulation logarithmique, nous présentons la formulation *logarithmique par combinaison convexe désagrégée*, notée DLog dans [Vielma et al., 2010]. Elle permet de représenter n'importe quelle fonction PWL d'une ou de plusieurs variables g non nécessairement continues. Nous prenons les mêmes notations que pour la formulation CC. Le mot *désagrégée* indique que les sommets v appartenant à plusieurs pièces P de g sont représentés comme autant de sommets différents notés (P,v) et donc les variables continues utilisées pour les combinaisons convexes sont $\lambda_{P,v}$ pour tout sommet (P,v) du graphe $\mathcal{P}$ de g au lieu de λ_v. Cette désagrégation est ce qui permet de représenter n'importe quel point sur le graphe d'une des pièces d'une fonction PWL non nécessairement continue puisque dans ce cas, en un point x du domaine commun à plusieurs pièces (sur leur frontière), il peut y avoir plusieurs valeurs ou valeurs limites $g(x)$ suivant la pièce considérée. Nous notons $|\mathcal{P}|$ le nombre de sommets de $\mathcal{P}$, $B : \mathcal{P} \mapsto \{0,1\}^{\lceil\log_2(|\mathcal{P}|)\rceil}$ une fonction injective de l'ensemble des pièces P vers le vecteur binaire $\{0,1\}^{\lceil\log_2(|\mathcal{P}|)\rceil}$, où $\lceil x\rceil$ représente l'arrondi à l'entier supérieur de x. Soient $\mathcal{P}^+(B,l) := \{P \in \mathcal{P} | B(P)_l = 1\}$ l'ensemble de pièces P dont la coordonnée l de $B(P)$ vaut 1, et soit $\mathcal{P}^0(B,l) := \{P \in \mathcal{P} | B(P)_l = 0\}$ l'ensemble de pièces P dont la coordonnée l de $B(P)$ vaut 0. Les contraintes à introduire dans le modèle sont :

$$\sum_{P\in\mathcal{P}} \sum_{v\in\mathcal{V}(\mathcal{P})} \lambda_{P,v} v = x, \tag{1.20}$$

$$\sum_{P\in\mathcal{P}} \sum_{v\in\mathcal{V}(\mathcal{P})} \lambda_{P,v}(m_P v + c_P) = z, \tag{1.21}$$

$$\lambda_{P,v} \geq 0 \quad \forall P \in \mathcal{P}, v \in \mathcal{V}(\mathcal{P}), \tag{1.22}$$

$$\sum_{P\in\mathcal{P}} \sum_{v\in\mathcal{V}(\mathcal{P})} \lambda_{P,v} = 1, \tag{1.23}$$

$$\sum_{P\in\mathcal{P}^+(B,l)} \sum_{v\in\mathcal{V}(\mathcal{P})} \lambda_{P,v} \leq y_l \quad \forall l = 1,...,\lceil\log_2(|\mathcal{P}|)\rceil, \tag{1.24}$$

$$\sum_{P\in\mathcal{P}^0(B,l)} \sum_{v\in\mathcal{V}(\mathcal{P})} \lambda_{P,v} \leq 1 - y_l \quad \forall l = 1,...,\lceil\log_2(|\mathcal{P}|)\rceil, \tag{1.25}$$

$$y_l \in \{0,1\} \quad \forall l = 1,...,\lceil\log_2(|\mathcal{P}|)\rceil. \tag{1.26}$$

Les contraintes (1.20)-(1.23) sont les versions désagrégées des combinaisons convexes des contraintes (1.13)-(1.16) de la formulation CC. Les contraintes (1.24) et (1.25) sont les contraintes linéaires de cardinal logarithmique en le nombre de pièces $|\mathcal{P}|$ qui forcent les variables $\lambda_{P,v}$ à ne pas participer à la combinaison convexe si z n'est pas sur la pièce P.

La *relaxation continue* d'un problème comportant des variables entières est le problème obtenu en retirant les contraintes d'intégrité. Il est difficile de se baser uniquement sur la taille d'une formulation pour en déduire le temps de résolution du problème MILP. En effet, la qualité de la relaxation continue d'une formulation en présence d'autres contraintes linéaires a aussi une grande importance [Vielma et al., 2010, Vielma et Nemhauser, 2011, Vielma, 2015, Huchette et Vielma, 2022]. Pour exemple de propriété sur la qualité de la relaxation continue, les formulations peuvent être *localement idéales* [Padberg, 2000], c'est-à-dire que les sommets du polyèdre de la relaxation continue de la formulation respectent les contraintes d'intégrité. Cette propriété est utile, mais ne permet pas de savoir ce qu'il se passe lors de l'intersection du polyèdre de la relaxation continue avec d'autres contraintes linéaires venant du reste du modèle MILP. Les formulations logarithmiques ont tendance à rendre la résolution du modèle plus rapide dans le cas de fonctions PWL avec un grand nombre de pièces grâce à leurs petites tailles, mais ce temps de résolution souffrent de la mauvaise qualité de leur relaxation continue. Récemment, [Huchette et Vielma, 2022] a proposé les formulations logarithmiques dites *zigzags* qui possédent de meilleures propriétés de relaxation continue, ainsi que la librairie *PiecewiseLinearOpt* dans le langage Julia [Bezanson et al., 2017] qui permet de produire quasiment automatiquement et tester facilement différentes formulations d'une fonction PWL continue.

La dernière caractéristique que nous souhaitons aborder concerne l'applicabilité des formulations. En effet, certaines nécessitent des propriétés particulières de la fonction PWL, comme sa continuité [Vielma et al., 2010], tandis que d'autres peuvent représenter des fonctions non nécessairement continues, comme les formulations désagrégées, et donc notamment DLog. Certaines formulations se décrivent de manière similaire pour les fonctions d'une seule et de plusieurs variables, et c'est notoirement le cas des formulations CC, DLog et de la formulation dite à choix multiples [Vielma et al., 2010]. En revanche pour les autres, soit ce n'est pas possible, soit il faut exhiber une structure particulière. Prenons le cas de fonctions de deux variables : pour la formulation incrémentale il faut trouver un cycle hamiltonien [Wilson, 1998], et pour les formulations zigzags issues de [Huchette et Vielma, 2022] cela demande des calculs lourds via [Huchette et Vielma, 2022, Proposition 2], ou de résoudre un problème particulier sur le graphe de conflit d'une triangulation de forme spécifique. Le mot *triangulation* désigne ici une partition d'un domaine polygonal de $\mathbb{R}^2$ en triangles avec la propriété qu'aucun sommet de triangle ne soit sur le côté d'un autre triangle. [Huchette et Vielma, 2022] compare plusieurs formulations de fonctions PWL de deux variables et donne des exemples de ces triangulations spécifiques. Numériquement, la formulation DLog ne produit pas les formulations les plus rapides à ré-

soudre mais peut modéliser n'importe quelle fonction PWL non nécessairement continue, ce que la formulation logarithmique agrégée correspondante ne peut pas faire. Le récent travail de [Lyu et al., 2023] présente des formulations logarithmiques pour des fonctions PWL de plusieurs variables, mais dont la partition du domaine en domaine des pièces provient d'une subdivision dans chaque dimension, formant des hyperrectangles. Cela limite donc le panel de fonctions PWL pouvant être formulées.

1.2.3 Résolution d'un MILP comportant des fonctions PWL

Après la formulation des fonctions PWL dans le modèle MILP, il faut le résoudre. Les méthodes de résolution reposent généralement sur un algorithme de Branch-and-Bound ou une de ses variantes. Elles utilisent donc le plus souvent une stratégie de branchement sur les variables binaires, et donc en particulier sur les variables binaires introduites par la formulation des fonctions PWL. Il existe des stratégies de branchements spécifiques pour certains groupes de variables binaires, notamment les variables SOS1 (pour *special ordered set of type 1*) [Belotti et al., 2013] et SOS2 (pour *special ordered set of type 2*) [Rebennack, 2016] et les schémas de branchement indépendant dont la théorie est développée dans [Vielma et Nemhauser, 2011, Huchette et Vielma, 2022]. Un groupe de variables continues est dit SOS1 si une seule de ces variables peut être non nulle à la fois. Concernant SOS2, dans un groupe de variables ordonné $\lambda_1, ..., \lambda_n$ au plus deux variables peuvent être non nulles à la fois et elle doivent être consécutives.

Certains solveurs commerciaux proposent de déclarer la fonction PWL en listant par exemple ses points de cassure, pour que le solveur choisisse la formulation mathématique lui-même. Cela a l'avantage de demander moins de travail de modélisation à l'utilisateur et le choix de formulation fait par le solveur devrait être pertinent dans la majorité des cas. Deux articles proposent de gérer les formulations des fonctions PWL de manière spécifique. Dans [Keha et al., 2006], le rôle des variables binaires des formulations est joué par des contraintes générées pendant l'exécution d'un Branch-and-Cut. Aussi, le récent article [Hübner et al., 2023] propose une méthode astucieuse dans le cas des fonctions PWL d'une seule variable. Il décrit un Branch-and-Bound spatial où les fonctions PWL d'une seule variable sont gérées comme des fonctions non linéaires spécifiques, au lieu d'être formulées dans le modèle MILP. Ainsi, pendant le calcul de borne inférieure c'est l'enveloppe convexe des fonctions PWL qui est utilisée. L'intérêt principal est que le nombre de variables et contraintes introduites par la formulation d'une fonction PWL ne dépend plus de son nombre de pièces mais du nombre de pièces de son enveloppe convexe, généralement plus faible. Pour cela, l'article décrit un algorithme calculant efficacement l'enveloppe convexe des fonctions PWL d'une seule variable. Les expériences numériques montrent que plus les fonctions PWL ont de pièces, meilleur est ce Branch-and-Bound spatial comparé à plusieurs formulations logarithmiques.

En dernier point, mentionnons à nouveau le travail de [Zhang et Wang, 2008]. Au lieu d'introduire des variables binaires dans la formulation de fonctions PWL, il propose de résoudre plusieurs problèmes LP (si aucune variable binaire n'est nécessaire pour le reste du modèle), en supposant qu'une seule fonction linéaire peut être choisie à chaque fois. Pour ne pas résoudre autant de problèmes LP que de fonctions linéaires, il introduit une façon de sélectionner seulement certaines fonctions linéaires.

1.2.4 Propriétés d'approximation

En général, résoudre le problème MILP approximant le problème MINLP ne permet pas d'obtenir de garanties par rapport à la solution du MINLP, comme la réalisabilité ou des bornes sur la valeur optimale. Nous décrivons ici des méthodes pour obtenir de telles garanties. Nous supposons dans la suite de cette section que le problème (*MINLP*) qui se formule comme suit :

$$(MINLP) \quad \begin{cases} \min\limits_{x} & f(x) & (1.27) \\ s.c. & g_i(x) \leq 0 \quad \forall i \in I & (1.28) \\ & h_j(x) = 0 \quad \forall j \in J & (1.29) \\ & Ax \leq b & (1.30) \\ & \underline{x} \leq x \leq \bar{x} & (1.31) \\ & x_i \in \mathbb{Z} \text{ pour } i = 1, ..., d_e & (1.32) \end{cases}$$

a été approximé par le problème (*MILP*)

$$(MILP) \quad \begin{cases} \min\limits_{x} & f^{PWL}(x) & (1.33) \\ s.c. & g_i^{PWL}(x) \leq 0 \quad \forall i \in I & (1.34) \\ & h_j^{PWL}(x) = 0 \quad \forall j \in J & (1.35) \\ & Ax \leq b & (1.36) \\ & \underline{x} \leq x \leq \bar{x} & (1.37) \\ & x_i \in \mathbb{Z} \text{ pour } i = 1, ..., d_e & (1.38) \end{cases}$$

où I et J sont des ensembles finis et f^{PWL}, g_i^{PWL} pour tout $i \in I$ et h_j^{PWL} pour tout $j \in J$ sont des fonctions linéaires par morceaux dont une formulation a été incluse dans (*MILP*) sans détailler les ajouts de variables et de contraintes dédiées.

Le résultat exprimé par la Proposition 6 permet d'obtenir des bornes supérieures, sans garantie sur la différence avec la valeur objectif du problème original.

Proposition 6. *Si $g_i^{PWL}(x) \leq g_i(x)$ pour tout x et pour tout $i \in I$, et qu'il n'y a pas de contraintes d'égalité ($J = \varnothing$), alors une solution réalisable de (MILP) est une solution réalisable de (MINLP). Si de plus la fonction $f^{PWL}(x)$ satisfait $f(x) \leq f^{PWL}(x)$, alors la valeur optimale*

de (MILP) est une borne supérieure de (MINLP).

En effet, les contraintes d'inégalités $g_i(x) \leq 0$ sont vérifiées en particulier si x satisfait $g_i^{PWL}(x) \leq 0$ et que $g_i^{PWL} \leq g_i$. À l'inverse, employer des sous-estimateurs PWL pour l'approximation des fonctions non linéaires permet d'obtenir des bornes inférieures [Contardo et Ngueveu, 2021]. Cette proposition décrit un problème MILP qui est une *restriction* du problème MINLP.

Une deuxième manière d'obtenir des garanties sur la solution de (*MINLP*) utilise la définition de l'approximation d'une fonction avec une erreur absolue.

Définition 7 (approximation à erreur absolue près)**.** Soient f et $\tilde{f}$ deux fonctions définies de $D \subset \mathbb{R}^m$ dans $\mathbb{R}$ et la borne sur l'erreur $\delta > 0$. On dit que $\tilde{f}$ est une *approximation absolue de la fonction f à δ près* si : $f(x) - \delta \leq \tilde{f}(x) \leq f(x) + \delta$ pour tout $x \in D$.

De manière équivalente il est possible de dire que $\tilde{f}$ est une δ-approximation absolue de f, ou que $\tilde{f}$ approxime à erreur absolue δ la fonction f. Nous donnons maintenant un résultat d'approximation de la solution d'un problème MINLP dont les contraintes sont linéaires ou exprimées avec des fonctions PWL :

Proposition 8. *Soit $\delta > 0$. Supposons que les fonctions g_i pour tout $i \in I$ et h_j pour tout $j \in J$ sont linéaires par morceaux et que la fonction $f^{PWL}(x)$ est une approximation absolue de f à δ près. Fixons $g_i^{PWL} = g_i$ pour tout i et $h_j^{PWL} = h_j$ pour tout j. Alors les valeurs optimales z^{MILP} de (MILP) et z^{MINLP} de (MINLP) satisfont l'inégalité $z^{MILP} - \delta \leq z^{MINLP} \leq z^{MILP} + \delta$.*

Démonstration. Les solutions réalisables de (*MILP*) et (*MINLP*) sont les mêmes puisque les fonctions décrivant les contraintes sont les mêmes. Soit x^{MILP} une solution optimale de (*MILP*) et x^{MINLP} une solution optimale de (*MINLP*). On démontre les deux inégalités séparément. Par l'absurde, supposons que $z^{MILP} - \delta > z^{MINLP}$. On a alors $\tilde{f}(x^{MINLP}) - \delta \geq \tilde{f}(x^{MILP}) - \delta$ car x^{MILP} est une solution optimale de (*MILP*), et $\tilde{f}(x^{MILP}) - \delta > f(x^{MINLP})$ par hypothèse ce qui conduit à une contradiction puisque $\tilde{f}$ est une approximation absolue à δ près de f. De la même manière, supposons par l'absurde que $z^{MINLP} > z^{MILP} + \delta$. On a alors $f(x^{MILP}) \geq f(x^{MINLP}) > f(x^{MILP}) + \delta$. Ce qui conduit aussi à une contradiction pour la même raison. □

Ce résultat permet d'obtenir une solution optimale à δ près pour le problème (*MINLP*) en résolvant (*MILP*) dans le cas où les contraintes de (*MINLP*) sont linéaires.

Finalement, en modifiant légèrement la formulation des fonctions PWL dans (*MILP*), il est possible d'obtenir une relaxation de (*MINLP*) comme expliqué dans [Geißler et al., 2012, Section 5], même en présence de contraintes d'égalités. Il suffit pour cela d'ajouter une variable continue e dans la formulation de chaque fonction g_i^{PWL} et h_j^{PWL} pour représenter une variation à la fonction non linéaire bornée selon l'erreur d'approximation à respecter. Par exemple, pour

la formulation CC de la fonction h_j^{PWL} utilisée dans une contrainte d'égalité, la contrainte (1.14) est modifiée en

$$e + \sum_{v \in \mathcal{V}(\mathcal{P})} \lambda_v (m_P v + c_P) = z, \tag{1.39}$$

et les contraintes

$$-\sum_{P \in \mathcal{P}} \left(\max_{u \in \mathcal{D}(P)} h_j^{PWL}(u) - h_j(u) \right) y_P \leq e \leq \sum_{P \in \mathcal{P}} \left(\max_{u \in \mathcal{D}(P)} h_j(u) - h_j^{PWL}(u) \right) y_P \tag{1.40}$$

sont ajoutées, où $\mathcal{D}(P)$ est le domaine de la pièce P. Pour une contrainte d'inégalité $g_i^{PWL} \leq 0$, la contrainte de gauche de (1.40) ne doit pas être ajoutée. La Figure 1.5 montre une telle relaxation pour la fonction $x^3 - \frac{9}{2}x^2 + 6x$ dans une contrainte d'égalité, où l'ensemble réalisable est l'union des polygones en pointillé.

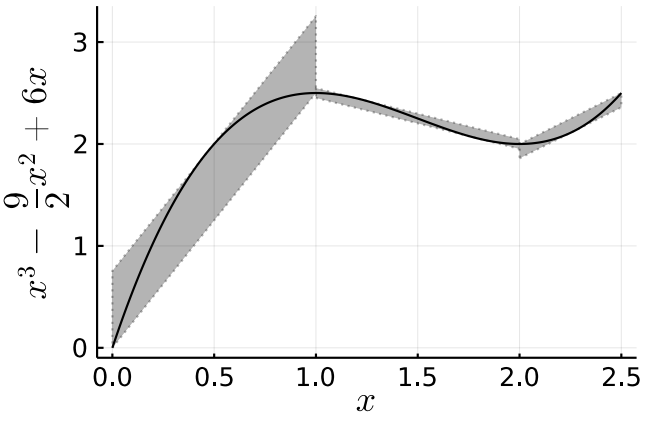


FIGURE 1.5: Enveloppes polygonales de la relaxation de la fonction $x^3 - \frac{9}{2}x^2 + 6x$ sur $[0, 2.5]$ avec les points de cassure 0, 1, 2 et 2.5.

Si les fonctions g_i^{PWL} et h_j^{PWL} sont construites pour qu'une solution de (*MILP*) viole les contraintes correspondantes $g_i(x) \leq 0$ ou $h_j(x) = 0$ de (*MINLP*) d'au plus la tolérance sur cette contrainte, alors une solution x^{MILP} de (*MILP*) respecte les contraintes de (*MINLP*) à la tolérance demandée. x^{MILP} est parfois considéré comme une "solution acceptable" pour (*MINLP*). Aussi, d'après la Proposition 8 le respect du niveau de tolérance sur la valeur optimale peut s'obtenir grâce à une approximation à la tolérance demandée sur la fonction objectif pour laquelle des méthodes sont décrites dans la Section 1.3. Mises bout à bout, l'utilisation de cette relaxation et de l'approximation de la fonction objectif à la tolérance demandée permet de calculer une solution de (*MINLP*) respectant un niveau de tolérance prédéfini en résolvant un seul problème (*MILP*). Cependant, pour des tolérances très faibles sur le respect des contraintes par

exemple (violation de l'ordre de 10^{-6} par exemple), cette approche peut produire des modèles MILP bien trop gros pour être résolu en un temps acceptable.

1.2.5 Méthodes de raffinement des approximations PWL

Comme énoncé précédemment lors de l'approximation d'un problème MINLP avec des fonctions PWL et des niveaux de tolérance très bas, il arrive que le problème MILP résultant demande un temps de résolution rédhibitoire, à cause du grand nombre de pièces des fonctions PWL. Dans ce cas, il est courant d'utiliser une procédure itérative pour obtenir des fonctions PWL avec relativement peu de pièces, mais qui sont de bonnes approximations aux endroits où c'est le plus utile.

Une stratégie d'approximation de (*MINLP*), que nous appelons *méthode de raffinement*, consiste alors à résoudre itérativement des problèmes de la forme (*MILP*) en commençant avec des fonctions PWL qui ont peu de pièces, tout en améliorant localement la qualité de l'approximation de (*MINLP*) à chaque itération. Pour cela, les fonctions PWL constituant (*MILP*) sont mises à jour partiellement pour être de plus en plus proches des fonctions non linéaires correspondantes dans certaines zones. En général, cette mise à jour correspond à remplacer une ou plusieurs pièces d'une fonction PWL par un plus grand nombre de pièces plus proches de la fonction non linéaire : c'est l'étape de *raffinement* des fonctions PWL. Ce sont souvent les pièces utilisées par la solution du problème (*MILP*) qui sont raffinées, parce qu'a priori, c'est le meilleur endroit pour obtenir une bonne approximation de la solution optimale de (*MINLP*). L'itération du raffinement et du calcul de la solution x^{MILP} de (*MILP*) continue jusqu'à ce qu'un certain critère d'arrêt soit satisfait : par exemple, la plus large violation de contraintes en x^{MILP} est plus petite qu'un certain seuil, ou encore la différence entre $f^{PWL}(x^{MILP})$ et $f(x^{MILP})$ est plus petite qu'une certaine valeur. D'autres critères d'arrêt sont possibles.

Nous donnons ci-après quelques exemples de raffinement. L'article [Yue et You, 2014] décrit une telle méthode pour la résolution d'un problème de théorie des jeux avec des fonctions univariées et des termes bilinéaires $f(x, y) = xy$, tandis que [Contardo et Ngueveu, 2021] traite de problèmes dont les fonctions non linéaires sont linéairement séparables. Pour des fonctions de deux variables, la procédure de raffinement appelée *red refinement* est étudiée dans [Burlacu, 2019, Burlacu et al., 2020, Burlacu, 2021] et consiste à diviser un simplexe en quatre simplexes grâce aux points milieux des arêtes du simplexe. La Figure 1.6 montre un exemple de raffinement par cette méthode. Le triangle ABC est raffiné en quatre triangles grâce aux points M_{AB}, M_{BC} et M_{AC} qui sont les points milieux des arêtes du triangle ABC. L'interpolation de la fonction en les sommets des quatre nouveaux triangles est alors plus fidèle que l'interpolation uniquement sur les sommets du triangle ABC. Cette méthode de raffinement peut s'appliquer à des fonctions de m variables avec $m \geq 1$, puisque le raffinement s'effectue sur un simplexe et se généralise à

n'importe quelle dimension.

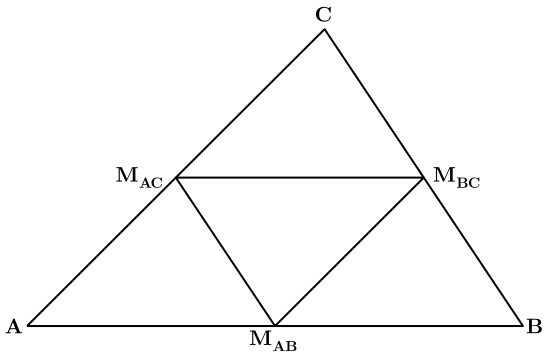


FIGURE 1.6: Exemple de raffinement d'une pièce de domaine triangulaire en quatre pièces par la méthode red refinement

1.3 Linéarisation de fonctions avec borne sur l'erreur d'approximation et minimisation du nombre de pièces

De nombreuses méthodes utilisent un nombre fixé de pièces et minimisent l'erreur d'approximation. Ces approches sont pertinentes lorsque la taille de la fonction PWL à construire est fixée. Plusieurs métriques d'erreurs sont utilisées dans ce cas [Rebennack et Krasko, 2020, Chen et Wang, 2013], parmi lesquelles la somme des différences au carré, la somme des valeurs absolues des différences en un nombre fini de points du domaine, ou encore le maximum des valeurs absolues des différences. En revanche, nous nous concentrons dans ce manuscrit sur la construction d'une fonction PWL dont l'objectif est la minimisation du nombre de pièces tout en respectant une borne sur l'erreur d'approximation. En effet, cela permet de réduire la taille, et donc le nombre de variables et de contraintes de la formulation dans un problème MILP. Quand cette erreur d'approximation est l'erreur absolue, $(\mathcal{P}_{abs})$ décrit ce problème :

$$(\mathcal{P}_{abs}) \quad \begin{cases} \min\limits_{g} & n & (1.41) \\ & g \text{ est une fonction PWL à } n \text{ pièces} & (1.42) \\ s.c. & f(x) - \delta \leq g(x) \leq f(x) + \delta \quad \forall x \in D \subset \mathbb{R}^m, & (1.43) \end{cases}$$

où f est une fonction continue et δ est la borne sur l'erreur d'approximation absolue à respecter. Les contraintes (1.43) assurent que la fonction PWL solution respecte une borne de δ sur l'erreur d'approximation absolue. Pour simplifier notre discours, nous mentionnerons le respect des contraintes (1.43) par l'expression "satisfaire (ou respecter) l'erreur d'approximation".

1.3.1 Fonctions représentées par un ensemble fini de points

Nous traitons d'abord le cas des méthodes résolvant une variante du problème $(\mathcal{P}_{abs})$ où $m = 1$ et c'est un ensemble fini de points de la forme $(x, f(x))$ qui représente la fonction f d'une seule variable [O'Rourke, 1981, Hakimi et Schmeichel, 1991]. L'équivalent pour une courbe plane à la place de la fonction f a aussi été étudié, par exemple dans [Dunham, 1986], mais la métrique d'erreur utilisée n'est pas la même donc ce n'est pas pertinent dans notre cas. Nous présentons ici la méthode très astucieuse de [O'Rourke, 1981] puisqu'elle est réutilisée dans plusieurs des méthodes décrites dans ce manuscrit. Le problème que traite cet article est de construire une fonction PWL g d'une seule variable avec le moins de pièces possible tel que la fonction vérifie des contraintes de la forme $\alpha_k \leq g(x_k) \leq \omega_k$ pour un nombre fini d'indices k, où α_k et ω_k sont des paramètres du problème. Ces intervalles $[\alpha_k, \omega_k]$ permettent notamment de représenter une erreur absolue δ pour une fonction f satisfaisant à la fois $f(x_k) = \frac{\alpha_k+\omega_k}{2}$ pour tout k et $\omega_k - \alpha_k = 2\delta$ pour tout k. L'algorithme développé pour résoudre ce problème utilise des propriétés de l'ensemble réalisable d'un problème LP qui modélise le fait de trouver une fonction linéaire satisfaisant ces contraintes. L'algorithme a une complexité linéaire en le nombre d'intervalles $[\alpha_k, \omega_k]$ à respecter, et il est montré qu'on ne peut pas trouver une meilleure complexité pour ce problème.

1.3.2 Fonctions d'une variable

Pour résoudre $(\mathcal{P}_{abs})$ avec l'hypothèse additionnelle que g est une fonction PWL continue, deux articles ont proposé des méthodes similaires [Rebennack et Krasko, 2020, Kong et Maravelias, 2020]. Ces méthodes itèrent entre deux phases. La première phase consiste à résoudre un MILP obtenu en remplaçant les contraintes (1.43) par des contraintes qui n'imposent de respecter l'erreur d'approximation qu'en un nombre fini de points du domaine. La deuxième phase consiste à vérifier s'il existe des zones du domaine continu où la fonction PWL, solution optimale du MILP, ne respecte pas l'erreur d'approximation. Si c'est le cas alors des points supplémentaires sont pris en compte dans le MILP avant une nouvelle résolution. Sinon l'algorithme s'arrête et la fonction PWL optimale a été trouvée. La phase de vérification requiert généralement la résolution d'un NLP.

À notre connaissance, la première méthode résolvant exactement le problème $(\mathcal{P}_{abs})$ sans hypothèse additionnelle, c'est-à-dire en produisant une fonction PWL non nécessairement continue, est celle de [Ngueveu, 2019]. Cette méthode exacte exploite une approche itérative à base de résolution de NLP, tandis qu'une autre méthode proposée, une heuristique à garantie de performance, évite de résoudre des NLP en décomposant le domaine de la fonction non linéaire en zones convexes et concaves qui seront traitées séparément. Le principe de ces deux méthodes est de produire à chaque itération la fonction linéaire satisfaisant l'erreur d'approximation deman-

dée sur le plus grand intervalle $[a, b_{int}]$ parmi l'intervalle non encore couvert par des fonctions linéaires $[a, b]$. Les travaux de [Codsi et al., 2021] améliorent ces méthodes, en particulier la méthode exacte, en introduisant trois principes. Le premier principe est d'introduire une notion appelée *corridor* qui permet de généraliser les types d'erreurs d'approximation considérés. Le deuxième principe est de caractériser les fonctions linéaires recherchées en termes de tangente et intersection avec les frontières des corridors lorsque l'intervalle du domaine est convexe ou concave. le troisième principe est d'utiliser l'algorithme de [O'Rourke, 1981] lorsque l'intervalle considéré possède au moins une partie convexe et une partie concave. il en résulte des algorithmes drastiquement plus rapides que les plus proches concurrents [Rebennack et Krasko, 2020, Kong et Maravelias, 2020], même si une telle comparaison n'est pas équitable à cause de l'hypothèse de continuité de la fonction PWL qui est respectée par ses concurrents. Elle permet toutefois d'illustrer l'intérêt de s'affranchir de cette hypothèse. La comparaison des solutions produites montre quant à elle que ne pas imposer la continuité de la fonction PWL permet de produire des fonctions PWL avec moins de pièces.

1.3.3 Fonctions de deux variables

Concernant les fonctions de deux variables, commençons par présenter le travail de [Bärmann et al., 2023] qui résout $(\mathcal{P}_{abs})$ spécifiquement pour la fonction non linéaire $f(x, y) = xy$, très courante en pratique. Il décrit une procédure appelée *crossing swords* pour construire une fonction PWL continue dont les fonctions linéaires sont définies sur des triangles interpolant la fonction f et respectant une erreur d'approximation absolue donnée. Il est montré que cette procédure a un ratio d'approximation de $\frac{\sqrt{5}}{2}$ dans le cas particulier d'une interpolation dont la division du domaine en domaine des pièces est une triangulation.

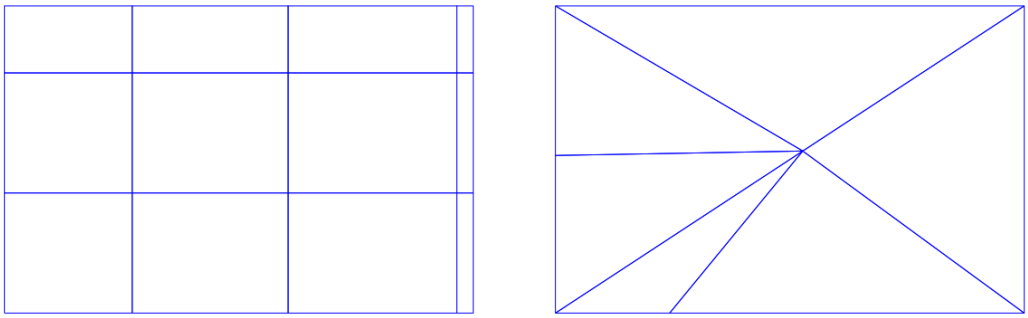
À propos des fonctions de deux variables non spécifiques, il n'existe à ce jour que quelques articles décrivant des méthodes qui sont toutes des heuristiques pour le problème $(\mathcal{P}_{abs})$. Toutes, exceptées celle provenant de notre travail [Duguet et Ngueveu, 2022], prennent l'hypothèse supplémentaire que la fonction PWL solution doit être continue [Rebennack et Kallrath, 2015a, Kazda et Li, 2021, 2023].

[Rebennack et Kallrath, 2015a] présente deux méthodes garantissant une erreur absolue donnée, adaptable à d'autres types d'erreurs. La première, basée sur une décomposition de $(\mathcal{P}_{abs})$ en deux sous-problèmes d'approximation de fonctions d'une variable, est notée *RK1D* et n'est applicable qu'à certaines fonctions non linéaires de deux variables. La seconde, basée sur une manipulation directe de fonctions de deux variables est notée *RK2D* et ne souffre d'aucune restriction (si ce n'est la connaissance d'une expression analytique de la fonction).

Présentons plus en détail la méthode RK1D. Il s'agit d'une méthode en trois phases. La première phase consiste à trouver une transformation h de la fonction f pour laquelle la contri-

bution des deux variables x_1 et x_2 est séparée linéairement : $h(f(x)) = \hat{f}_1(x_1) + \hat{f}_2(x_2)$. Si une telle transformation n'existe pas, la fonction ne peut pas être approximée par RK1D. Puis dans la deuxième phase, il faut calculer des valeurs δ_1 et δ_2 pour lesquelles la troisième phase produit une fonction PWL de deux variables g approximant f avec une erreur absolue δ. Finalement, la troisième phase consiste à calculer les fonctions PWL d'une seule variable g_1 et g_2 qui sont les approximations PWL à δ_1 et δ_2 près des fonctions $\hat{f}_1$ et $\hat{f}_2$ respectivement, grâce à l'Algorithme 4.1 de [Rebennack et Kallrath, 2015b] ; puis construire la fonction g à partir de g_1 et g_2 en effectuant les opérations inverses menant à la transformation $h(f(x)) = \hat{f}_1(x_1) + \hat{f}_2(x_2)$. Donnons deux exemples. Dans le premier exemple, nous traitons le cas le plus simple : celui d'une fonction linéairement séparable $f(x) = f_1(x_1) + f_2(x_2)$. Dans ce cas $h(f(x)) = \hat{f}_1(x_1) + \hat{f}_2(x_2)$ où $\hat{f}_1(x_1) = f_1(x_1)$ et $\hat{f}_2(x_2) = f_2(x_2)$. La transformation est l'identité. Les valeurs δ_1 et δ_2 "se partagent" l'erreur δ tant qu'ils respectent $\delta = \delta_1 + \delta_2$, par exemple avec $\delta_1 = \delta_2 = \frac{\delta}{2}$. Puis les fonctions g_1 et g_2 approximant à erreur absolue δ_1 et δ_2 près les fonctions f_1 et f_2 sont construites, et g est calculée grâce à $g(x) = g_1(x_1) + g_2(x_2)$ qui est une approximation de f avec une erreur absolue $\delta = \delta_1 + \delta_2$. Dans le second exemple, nous traitons le cas du produit de deux fonctions d'une variable strictement positives $f(x) = f_1(x_1)f_2(x_2)$. Dans ce cas $h(f(x)) = ln(f(x)) = ln(f_1(x_1)) + ln(f_2(x_2)) = \hat{f}_1(x_1) + \hat{f}_2(x_2)$. La fonction g sera construite grâce à $g(x) = exp(ln(g_1(x_1)) + ln(g_2(x_2)))$, donc un calcul donne que les erreurs δ_1 et δ_2 induiront une erreur absolue $f(x)(exp(\delta_1 + \delta_2) - 1)$ sur la fonction f. Il suffit ensuite de choisir des valeurs de δ_1 et δ_2 qui respectent $f(x)(exp(\delta_1 + \delta_2) - 1) \leq \delta$. [Rebennack et Kallrath, 2015a] décrit deux autres types de fonctions pour lesquels on peut dériver une expression séparant linéairement les contributions en x_1 et x_2 : les fonctions de la forme $f(x) = \hat{f}(x)^{\bar{f}(x)}$, et les fonctions de la forme $f(x) = \hat{f}(\bar{f}(x))$. Dans chaque cas, la partition du domaine de la fonction f est un ensemble de rectangles dont la base est parallèle à l'axe x_1. La Figure 1.7a illustre cette situation. Cette restriction dans la forme des polygones peut conduire à un nombre important de pièces lors de l'approximation de certaines fonctions.

Présentons à présent la méthode RK2D. Elle produit des fonctions PWL pour lesquelles le domaine des pièces est une triangulation. La méthode s'initialise en divisant le domaine rectangulaire par sa diagonale en deux triangles. Puis, l'algorithme cherche s'il existe une fonction linéaire respectant l'erreur d'approximation absolue δ pour chacun de ces triangles. Cette fonction est recherchée parmi un nombre fini de fonctions linéaires correspondant à la combinaison convexe des trois sommets du triangle de la forme $(x_1, x_2, f(x_1, x_2) + s)$ où s est une valeur dans $[-\delta, \delta]$ représentant un décalage par rapport à la valeur de la fonction. La recherche de cette fonction est faite en résolvant un problème NLP. Si une solution au NLP est trouvée, la fonction correspondante est ajoutée à la PWL en construction. Sinon, le domaine triangulaire est subdivisé. RK2D considère deux procédures de raffinement possibles, la subdivision en trois triangles autour du point où l'erreur d'approximation est maximale, ou la subdivision en quatre



(a) pour la méthode RK1D (b) pour la méthode RK2D

FIGURE 1.7: Exemple de subdivision du domaine de la fonction PWL pour différentes méthodes

triangles comme dans la littérature sous le nom de *red refinement*. La Figure 1.7b illustre un résultat de subdivision de domaine. Une fois les nouveaux domaines obtenus, RK2D les traite séparément en recherchant pour chacun s'il existe une fonction linéaire couvrant le domaine tout en respectant l'erreur d'approximation. Il y a quelques détails techniques permettant d'obtenir une fonction PWL continue à partir de ce processus, mais en ne les considérant pas, la méthode permet d'obtenir des fonctions PWL non nécessairement continues. Le principal défaut de cette heuristique est que les sous-domaines ne sont pas construits pour utiliser "le plus possible" l'erreur d'approximation autorisée, mais seulement pour être acceptés lorsque une fonction linéaire respecte l'erreur d'approximation sur le sous-domaine. Cela contribue à simplifier significativement l'algorithme, mais augmente aussi de manière importante le nombre de pièces de la fonction PWL construite.

[Kazda et Li, 2021] propose une approche différente, que l'on note *KL21*. Plutôt que de produire les pièces de la fonction PWL une par une, elles sont toutes produites en même temps grâce à la résolution d'un ou plusieurs MILP vérifiant les erreurs d'approximation sur un ensemble fini de points du domaine D_q. Puis, la résolution de problèmes NLP calculant le maximum d'erreur d'approximation entre la fonction à approximer et la fonction PWL sur chaque domaine de pièce permet de vérifier si l'erreur d'approximation est respectée en tout point du domaine continu ou pas. En cas de non-respect, de nouveaux points sont ajoutés à D_q, la valeur q est incrémentée pour indiquer le changement d'itération, et le MILP résultant est résolu pour produire une nouvelle fonction PWL. Une autre particularité de la méthode est que la fonction PWL recherchée g est modélisée au sein du MILP sous la forme d'une différence de fonctions PWL continues convexes $g(x) = g^+(x) - g^-(x)$ que l'on appelle *DC CPWL* pour *difference of convex continuous piecewise linear function*. Les deux composants de la fonction DC CPWL sont respectivement appelés le composant convexe (g^+) et le composant concave (g^-), suivant s'ils sont ajoutés ou soustraits pour obtenir la fonction DC CPWL. C'est le fait qu'ils soient convexe et concave qui

permet de les exprimer dans le problème MILP comme le maximum de fonctions linéaires. Le modèle MILP résolu une ou plusieurs fois à chaque itération est décrit dans l'annexe A.

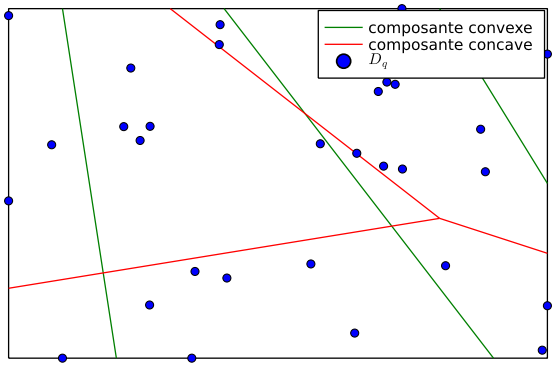


FIGURE 1.8: Représentation du domaine d'une fonction DC CPWL avec les points en lesquels l'erreur d'approximation est respectée

La Figure 1.8 montre une étape de la construction de la fonction DC CPWL. Le domaine de la fonction est le rectangle noir. Les points bleus sont les points dans D_q. La résolution d'un problème MILP a produit une fonction DC CPWL pour laquelle les frontières des pièces du composant convexe sont en vert, tandis que les frontières des pièces du composant concave sont en rouge. L'algorithme se termine en un nombre d'itérations fini car les contraintes du MILP assurant de respecter l'erreur d'approximation sont plus strictes qu'a priori nécessaire : ce sont les contraintes $f(x_i) - \eta\delta \leq g(x_i) \leq f(x_i) + \eta\delta$ pour tout $x_i \in D_q$ et $\eta \in]0, 1[$ qui sont modélisées. Cela permet de forcer la fonction PWL solution du MILP à satisfaire l'erreur sur un sous-domaine ouvert contenant x_i de la fonction f. De plus, il y a deux variables binaires dans le MILP qui permettent d'ajouter une fonction linéaire au composant convexe ou concave en cas d'échec de création d'une fonction PWL satisfaisant l'erreur sur D_q, permettant d'augmenter le nombre de morceaux de la fonction PWL solution. Il y a une remarque importante à faire par rapport à cet ajout de fonctions linéaires aux composants : ajouter une fonction linéaire à un composant n'augmente pas forcément d'une unité le nombre de morceaux de la fonction DC CPWL. L'augmentation peut être de plus d'une unité, ou pas, à cause de l'interaction complexe des fonctions linéaires faisant partie d'une fonction DC CPWL. Notons également que la forme d'un morceau peut être n'importe quel polygone convexe, ce qui permet de mieux s'adapter à l'erreur d'approximation à respecter, et donc potentiellement d'utiliser moins de pièces que si des pièces de domaine triangulaire étaient imposées. Malgré cette flexibilité, les problèmes MILP de taille grandissante à résoudre induisent une limitation importante à l'utilisation de l'algorithme pour des instances de taille moyenne ou grande. De même, l'écriture de la fonction PWL sous la forme DC CPWL ne permet pas de contrôler exactement le nombre de pièces de

Caractéristiques	RK1D	RK2D	KL21	KL23-LP et KL23-LPrelaxed
Principe	Approximations de deux fonctions d'une variable	raffinement du domaine	relaxations MILP + fonction DC CPWL	relaxations LP + fonction DC CPWL
Forme des pièces	rectangles	triangles	polygones convexes	polygones convexes
Régularité	continue	continue	continue	continue
Sous-problèmes	MILP	NLP	MILP et NLP	LP et NLP
Temps d'exécution	court	moyen	long	moyen

TABLE 1.1: Quelques caractéristiques des algorithmes résolvant ($\mathcal{P}_{abs}$)

la fonction PWL en fonction du nombre de pièces des composants convexes et concaves.

Dans [Kazda et Li, 2023], la méthode KL21 est améliorée dans le but de diminuer le temps de calcul, au détriment du nombre de pièces des fonctions PWL solutions. Pour cela, l'itération de résolution de problèmes MILP est remplacée par l'itération de deux étapes : la première tente de trouver une bonne affectation des points de D_q à un ensemble fini de fonctions linéaires en résolvant des problèmes LP, et la deuxième ajoute des fonctions linéaires au composant convexe ou concave. La partie "détermination des nouveaux points où l'erreur d'approximation doit être respectée" ne change pas. Nous appelons cette méthode *KL23-LP*. Une variante de cette méthode est décrite dans le même article, nous l'appelons *KL23-LPrelaxed*. Elle gère certaines des contraintes du modèle LP comme des contraintes lazy, c'est-à-dire en les ajoutant au modèle LP uniquement quand la solution précédente ne les respectent pas, puisque ces problèmes LP demandent un grand nombre de contraintes. Les résultats numériques sur le jeu d'instances de [Rebennack et Kallrath, 2015a] montrent que les méthodes KL23-LP et KL23-LPrelaxed permettent de résoudre des instances bien plus grosses que la méthode KL21. En effet, elles résolvent respectivement 42 et 44 des 45 instances avec les mêmes contraintes de temps que KL21, qui lui en résout seulement 24 sur 45. Le nombre de pièces des fonctions PWL construite est nettement plus petit que pour la méthode RK2D, mais lorsque KL21 trouve une solution, elle a en général quelques pièces de moins que les solutions des méthodes KL23-LP et KL23-LPrelaxed.

La Table 1.1 résume des caractéristiques des méthodes existantes. La ligne *Principe* donne l'idée principale du fonctionnement de la méthode. La *Forme des pièces* est la forme géométrique du domaine des pièces d'une fonction PWL solution de la méthode, et *Régularité* donne la propriété de régularité (continuité) la plus forte respectée par toutes les fonctions PWL construites par la méthode. De plus, la ligne *Sous-problèmes* décrit les classes des problèmes d'optimisation principaux résolus pendant l'exécution de la méthode. Ils permettent de donner une idée du temps d'exécution global, dont une indication est donné dans la ligne *Temps d'exécution*.

1.3.4 Fonctions de trois variables et plus

Toutes les méthodes décrites précédemment pour l'approximation de fonctions de deux variables s'adaptent à trois variables et plus. Les modifications portent principalement pour RK2D, KL21, KL23-LP et KL23-LPrelaxed sur l'adaptation du problème NLP qui calcule si la borne sur l'erreur d'approximation est respectée ou non, et pour RK1D sur la décomposition de la fonction non linéaire en fonctions d'une seule variable pour évaluer les contributions des bornes sur les erreurs univariées à l'erreur globale. Il y a d'ailleurs des expériences numériques sur la résolution d'un problème NLP utilisant la fonction multilinéaire $\prod_{i=1,\dots,m} x_i$ sur $[0,1]^m$ qui est approché avec la méthode KL23-LPrelaxed dans [Kazda et Li, 2023] pour $m = 3, 4, 5$. Cette méthode trouve de meilleures solutions en moins de temps qu'une méthode exacte et qu'une méthode calculant un optimum local pour certaines instances.

1.4 Conclusion

La méthode de résolution de problèmes MINLP par approximation des fonctions non linéaires en fonctions PWL a reçu une attention importante dans la littérature pour chacune de ses étapes : construction de la fonction PWL, formulation de la fonction PWL dans un problème MILP et résolution du problème MILP. Pour la problématique qui nous intéresse, c'est-à-dire l'approximation par des fonctions PWL de deux variables respectant une erreur d'approximation, la plupart des méthodes produisent des fonctions PWL continues, alors que cette contrainte peut être affaiblie à chercher des fonctions non nécessairement continues. De plus, pour des fonctions de deux variables, il n'existe aucune méthode qui garantisse la minimisation du nombre de pièces utilisées par les fonctions PWL, alors que c'est un atout lors de la résolution de l'approximation du problème MINLP. Ainsi, ce manuscrit se concentre sur l'élaboration de méthodes d'approximation par des fonctions PWL non nécessairement continues satisfaisant une borne sur l'erreur d'approximation et utilisant le moins de pièces possible.

Chapitre 2

Le $\mathbb{R}^2$-CFP : Corridor Fitting Problem pour les fonctions de deux variables

Sommaire

Dans le chapitre précédent, nous avons décrit l'état de l'art sur l'approximation linéaire par morceaux de fonctions non linéaires respectant une erreur d'approximation. Dans ce chapitre, nous introduisons le problème que nous étudions dans l'ensemble du manuscrit. Pour cela, nous définissons la notion de corridor, permettant une représentation générique d'une classe d'erreurs, dans la Section 2.1 et nous présentons le problème de $\mathbb{R}^2$-Corridor Fitting Problem dans la Section 2.2.

2.1 De la linéarisation par morceaux respectant une erreur d'approximation...

2.1.1 À la recherche de fonctions PWL non nécessairement continues

Comme mentionné dans la conclusion du Chapitre 1, nous recherchons des fonctions PWL non nécessairement continues. En effet, l'hypothèse de continuité n'est pas nécessaire dans le cas de la résolution d'un MINLP par la linéarisation par morceaux des fonctions non linéaires. Cela limite toutefois les formulations MILP des fonctions PWL utilisables, puisque certaines formulations ne peuvent pas exprimer de fonctions qui ne sont pas continues, comme la formulation incrémentale ou la version agrégée de la formulation DLog. En revanche, d'autres formulations comme la formulation DLog restent applicables et performantes.

Nos travaux visent à utiliser un ensemble de polytopes composant la fonction PWL g pour approximer une fonction non linéaire tout en respectant une erreur d'approximation. Lors de la modélisation du MILP qui approxime le MINLP, il suffit donc de formuler les polytopes eux-mêmes plutôt que la fonction g. La différence est qu'une fonction a une évaluation unique en tout point x de son domaine de définition, alors qu'un ensemble de polytopes dans lequel x appartient à plusieurs éléments peut conduire à plusieurs évaluations différentes, ce qui peut arriver aux points de discontinuité de g.

2.1.2 Représentation générique d'une classe d'erreurs d'approximation

Le type d'erreur d'approximation à considérer dépend de l'application visée. Par exemple, si l'on utilise un appareil de mesure dont l'erreur est bornée en amplitude par la même valeur δ peu importe la valeur mesurée, alors l'erreur absolue est pertinente. En revanche, si l'on considère une incertitude sur un coût économique alors, une erreur absolue ne serait probablement pas pertinente. En effet, par exemple, une erreur absolue d'une unité est inacceptablement grande lorsque le coût est de quelques centimes et inutilement faible si le coût est de quelques milliards. Dans ce cas, une erreur relative comme définie ci-dessous semble plus adaptée. Dans d'autres situations, d'autres types d'erreurs pourraient s'avérer plus pertinentes.

Définition 9 (approximation à erreur relative près)**.** Soient deux fonctions f et $\tilde{f}$ strictement positives définies de $D \subset \mathbb{R}^m$ et la borne sur l'erreur $\epsilon \in [0, 1]$. On dit que $\tilde{f}$ est une *approximation relative de la fonction f à ϵ près* si : $(1-\epsilon)f(x) \leq \tilde{f}(x) \leq (1+\epsilon)f(x)$ pour tout $x \in D$. On définit de manière équivalente l'approximation par erreur relative pour des fonctions $f < 0$.

La Figure 2.1 montre les bornes d'erreurs absolues et relatives pour l'approximation de la fonction $f(x) = x^2$ sur $[1, 5]$. Nous pouvons y constater que pour de petites valeurs de $f(x)$, l'erreur absolue est plus permissive que l'erreur relative, mais pour de grandes valeurs c'est l'inverse.

Nous nous intéressons à une classe d'erreurs pour laquelle "respecter une borne α sur l'erreur d'approximation" revient à construire une fonction g satisfaisant des contraintes

$$l(x) \leq g(x) \leq u(x) \quad \forall x \in D \tag{2.1}$$

pour lesquelles D est le domaine de la fonction f que l'on approxime. Les fonctions u et l y décrivent, pour chaque x dans le domaine, les limites satisfaisant la borne α sur l'erreur d'approximation voulue. L'erreur absolue et l'erreur relative font partie de cette classe. Un exemple d'erreur qui n'en fait pas partie est la somme des valeurs absolues des différences $\sum_{i=1,\ldots,N} |f(x_i) - g(x_i)|$. Nous appelons les contraintes (2.1) les *contraintes de corridor*, puisqu'elles forcent le graphe de la fonction g à être à l'intérieur d'un *corridor* dont les limites sont données par les fonctions u et l :

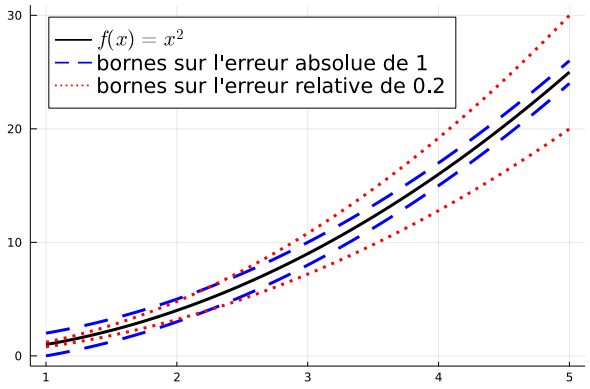
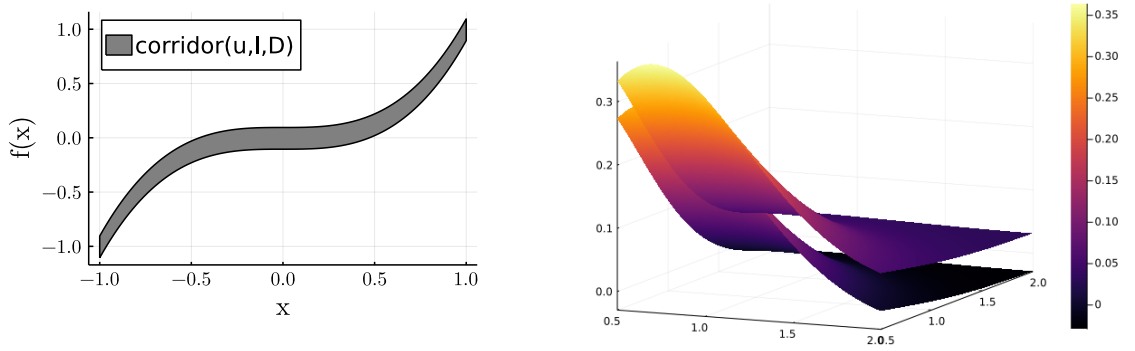


FIGURE 2.1: Seuils d'approximation avec les erreurs absolue et relative

Définition 10 (Corridor)**.** Soient le polytope $D \subset \mathbb{R}^m$ et les fonctions continues u et l de D dans $\mathbb{R}$ vérifiant $l(x) \leq u(x)$ pour tout $x \in D$. On appelle l'ensemble $\mathcal{C} = \{(x, z) \in \mathbb{R}^{m+1} | x \in D, l(x) \leq z \leq u(x)\}$ le *corridor entre* u *et* l *sur* D. On appelle D le *domaine du corridor* $\mathcal{C}$, que l'on note aussi $D(\mathcal{C})$, et la notation $\mathcal{C} = Corridor(u, l, D)$ récapitule les caractéristiques du corridor $\mathcal{C}$.



(a) $\mathbb{R}$-Corridor (b) $\mathbb{R}^2$-Corridor

FIGURE 2.2: Deux exemples de corridors

Nous appelons $\mathbb{R}^m$-*corridor* un corridor dont le domaine est inclus dans $\mathbb{R}^m$. Dans le cas où le domaine D d'un $\mathbb{R}$-*corridor* est un intervalle $[a, b]$, nous appelons $b - a$ la *longueur* du domaine de $\mathcal{C}$. La Figure 2.2 montre deux exemples de corridors. La Figure 2.2b montre les surfaces des fonctions u et l ; le $\mathbb{R}^2$-corridor est l'ensemble des points compris entre ces deux surfaces. Avec ces notations, l'erreur absolue s'exprime ainsi : $u(x) = f(x) + \delta$, $l(x) = f(x) - \delta$, ce qui définit le corridor $Corridor(f + \delta, f - \delta, D)$. L'erreur relative s'exprime grâce à $u(x) = f(x) + \epsilon f(x) = (1 + \epsilon) f(x)$ et $l(x) = f(x) - \epsilon f(x) = (1 - \epsilon) f(x)$ ce qui forme le corridor $Corridor((1 + \epsilon) f, (1 - \epsilon) f, D)$.

La notion de corridor a initialement été introduite par [Codsi et al., 2021] pour les fonctions non linéaires d'une variable. En conséquence, les méthodes de linéarisation par morceaux qui y sont développées s'appliquent aux $\mathbb{R}$-corridor. Les méthodes RK1D, RK2D, KL21, KL23-LP, KL23-LPrelaxed et celles issues de [Kong et Maravelias, 2020, Rebennack et Krasko, 2020] sont décrites spécifiquement pour la linéarisation par morceaux de fonctions avec l'erreur absolue. Elles pourraient également être adaptées aux contraintes de corridor à condition de pouvoir calculer efficacement le point d'erreur maximum sur un polytope.

2.2 ... au $\mathbb{R}^2$-Corridor Fitting Problem

Nous pouvons maintenant introduire le problème d'approximation de fonctions que nous étudions dans ce manuscrit. Soient le domaine polytopique $D \subset \mathbb{R}^m$ et les fonctions continues f, u et l de D dans $\mathbb{R}$ satisfaisant $l(x) \leq f(x) \leq u(x)$ pour tout $x \in D$. Les fonctions u et l définissent les limites acceptables pour une approximation de la fonction f en chaque point du domaine D. Le problème d'approximation que nous traitons est l'approximation de la fonction f par une fonction PWL g entre les bornes u et l. Il s'écrit :

$$(\mathcal{P}) \quad \begin{cases} \min\limits_{g} & n & (2.2) \\ & g \text{ est une fonction PWL à } n \text{ pièces} & (2.3) \\ s.c. & l(x) \leq g(x) \leq u(x) \quad \forall x \in D \subset \mathbb{R}^m. & (2.4) \end{cases}$$

Notons que la fonction f n'apparaît pas explicitement dans la définition du problème $(\mathcal{P})$, seules les fonctions u et l représentant le corridor y sont. Il est donc possible de la retirer de la définition du problème : nous cherchons simplement une fonction PWL bornée par les fonctions u et l. Nous pouvons donc reformuler ce problème avec la notion de corridor, qui définit naturellement l'ensemble des valeurs autorisées pour la fonction PWL g :

Définition 11 (Corridor fitting problem)**.** Soient le domaine polytopique $D \subset \mathbb{R}^m$ et les fonctions continues u et l de D dans $\mathbb{R}$ satisfaisant $l(x) \leq u(x)$ pour tout $x \in D$. Le problème consistant à trouver une fonction g solution de :

$$(CFP) \quad \begin{cases} \min\limits_{g} & n & (2.5) \\ s.c. & g \text{ est une fonction PWL à } n \text{ pièces} & (2.6) \\ & (x, g(x)) \in \mathcal{C}, \quad \forall x \in D, & (2.7) \end{cases}$$

est appelé $\mathbb{R}^m$-CFP (pour Corridor Fitting Problem dans $\mathbb{R}^m$) dans le corridor $Corridor(u, l, D)$.

Le $\mathbb{R}^m$-CFP consiste donc à construire une fonction PWL g avec le nombre minimum de pièces pour laquelle son graphe est inclus dans le corridor $\mathcal{C} = Corridor(u, l, D)$.

Nous définissons ici un vocabulaire spécifique autour de la notion de corridor pour un domaine polytopique D dans $\mathbb{R}^2$ qui sera utile dans le reste du manuscrit. Ce vocabulaire est majoritairement une extension des définitions de (Codsi et al. [2021]) elles-mêmes écrites pour un intervalle borné $D \subset \mathbb{R}$. La plupart de ce vocabulaire est adaptable à un corridor de $\mathbb{R}^m$ pour $m \geq 2$.

Définition 12 (Pièce dans un corridor)**.** Soient un corridor $\mathcal{C}$, une fonction linéaire $L : D_L \subset D(\mathcal{C}) \mapsto \mathbb{R}$ et une pièce $P = \{(x, y, L(x, y))|(x, y) \in D_L\}$. On dit que la *pièce P est dans un corridor $\mathcal{C}$* si et seulement si $P \subset \mathcal{C}$.

En d'autres termes, une pièce P issue d'une fonction linéaire L définie sur le domaine polytopique D_L est dans un corridor si l'ensemble des points de P sont dans le corridor $\mathcal{C}$. Nous introduisons la notion de corridor intérieur PWL, utile dans le Chapitre 3 :

Définition 13 (Corridor PWL)**.** Un corridor $\mathcal{C} = Corridor(u, l, D)$ est dit PWL si et seulement si u et l sont des fonctions PWL.

Nous appelons *corridor intérieur* un corridor qui est inclus dans un autre corridor de la manière suivante :

Définition 14 (Corridor intérieur)**.** Soient les corridors $\mathcal{C}_0 = Corridor(u_0, l_0, D_0)$ et $\mathcal{C} = Corridor(u, l, D)$. On appelle $\mathcal{C}$ un *corridor intérieur de $\mathcal{C}_0$* si et seulement si $D = D_0$ et $l_0(x, y) \leq l(x, y) \leq u(x, y) \leq u_0(x, y)$.

Un corridor intérieur PWL $\mathcal{C} = Corridor(u, l, D)$ d'un corridor $\mathcal{C}_0$ est alors un corridor intérieur de $\mathcal{C}_0$ avec des fonctions u et l qui sont linéaires par morceaux.

Définition 15 (Traversée)**.** On dit qu'une fonction PWL g *traverse* un corridor $\mathcal{C}$ si et seulement si g est définie sur le domaine $D(\mathcal{C})$ et toutes les pièces de g sont dans le corridor $\mathcal{C}$.

En particulier, le concept de traversée s'applique aussi à une fonction linéaire, puisque c'est une fonction PWL à une pièce. On peut donc dire qu'une fonction linéaire traverse un corridor si elle est définie sur le même domaine et que son graphe est dans le corridor. Par abus de langage, on peut aussi dire qu'une pièce traverse un corridor si la fonction linéaire associée traverse le corridor. Avec ces nouvelles définitions, le $\mathbb{R}^2$-CFP consiste à trouver une fonction PWL g de deux variables qui traverse un corridor $\mathcal{C}$ avec le nombre de pièces minimum. Nous précisons maintenant le sens de *pièce maximale*, aussi utile dans le Chapitre 3, pour des corridors de $\mathbb{R}$:

Définition 16 ($\mathbb{R}$-Corridor tronqué)**.** Soient les deux $\mathbb{R}$-corridors : $\mathcal{C}_1 = Corridor(u, l, [a_1, b_1])$ et $\mathcal{C}_2 = Corridor(u, l, [a_2, b_2])$. On appelle $\mathcal{C}_2$ un *corridor tronqué de $\mathcal{C}_1$* si et seulement si $a_1 = a_2$ et $b_2 \leq b_1$.

Cette notion de $\mathbb{R}$-corridor tronqué permet d'isoler la partie du domaine d'un corridor traversée par une pièce.

Définition 17 (Pièce maximale pour un corridor de $\mathbb{R}$)**.** Une *pièce maximale p d'un corridor* $\mathcal{C} = Corridor(u, l, [a, b])$ est une pièce traversant le corridor tronqué $Corridor(u, l, [a, b_p])$ telle qu'il n'existe pas de pièce p' traversant le corridor tronqué $Corridor(u, l, [a, b_{p'}])$ avec $b_{p'} > b_p$.

Cela définit l'intuition qu'une pièce qui traverse le plus grand intervalle possible de $D(\mathcal{C})$ partant de a est une pièce maximale d'un $\mathbb{R}$-corridor. L'algorithme exact de [Codsi et al., 2021] caractérise les pièces maximales des corridors de $\mathbb{R}$ dans les cas où la fonction est convexe ou concave. Il propose aussi un algorithme pour calculer une pièce maximale si la fonction n'est pas convexe ou concave et montre qu'itérer la construction de pièces maximales partant du début du domaine du corridor restant à couvrir permet de résoudre exactement le $\mathbb{R}$-CFP.

2.3 Conclusion

Dans ce chapitre, nous avons défini le $\mathbb{R}^2$-CFP. Il consiste à construire une fonction PWL qui respecte les contraintes de corridor avec le nombre minimum de pièces. L'intérêt de ce problème est que les fonctions PWL recherchées peuvent être discontinues, et l'utilisation des contraintes de corridor permet de représenter dans un même formalisme toute une classe d'erreurs d'approximation. Nous avons ensuite introduit un vocabulaire autour de la notion de corridor, qui servira dans le reste de ce manuscrit.

Chapitre 3

Heuristiques pour la résolution du $\mathbb{R}^2$-CFP

Sommaire

Dans le chapitre précédent, nous avons défini le $\mathbb{R}^2$-CFP et le vocabulaire autour de la notion de corridor. Dans ce chapitre, nous décrivons une *trame* de génération d'heuristiques pour résoudre ce difficile $\mathbb{R}^2$-CFP pour des corridors $Corridor(u, l, D)$ où u et l sont des fonctions continues. Par "trame", nous entendons "algorithme dont le fonctionnement est divisé en plusieurs procédures indépendantes les unes des autres et dont chaque procédure peut être réalisée de différentes manières", que l'on appelle *composants*. Cela permet de facilement remplacer et donc comparer les différents composants.

Le plan du chapitre est le suivant. Tout d'abord, dans la Section 3.1 nous présentons la trame et déroulons un exemple. Dans la Section 3.2 nous décrivons chacune des procédures de la trame, avec un composant simple. La Section 3.3 quant à elle propose des composants plus sophistiqués pour certaines procédures de la trame. Finalement, dans la Section 3.4 nous analysons numériquement les différents composants de la trame, sélectionnons nos meilleurs algorithmes et les comparons à l'état de l'art.

3.1 Une trame pour la construction d'heuristiques pour le $\mathbb{R}^2$-CFP

Trois principes régissent la trame que nous proposons. Les deux premiers permettent de pallier à certains défauts des heuristiques de la littérature RK2D, RK1D et LinA1D, tandis que le troisième a pour but de remplacer un sous-problème par un autre plus facile à résoudre.

- **Principe 1** : autoriser les pièces de la fonction PWL à être des polygones convexes, pas seulement des triangles.
- **Principe 2** : construire des pièces "les plus grandes" dans le sens où ce sont des pièces maximales dans une direction d (définition dans la Section 3.2.2).
- **Principe 3** : remplacer le corridor du $\mathbb{R}^2$-CFP par un corridor intérieur PWL (Définitions 13 et 14) pour calculer une solution réalisable du problème de pièce maximale dans la direction d avec une série de problèmes LP.

Chaque heuristique produite par la trame respecte ces trois principes. La Section 3.1.1 décrit la trame, liste les différents composants implémentés et où ils sont décrits dans le chapitre. Finalement, un exemple du déroulement de la trame est présenté dans la Section 3.1.2.

3.1.1 Trame proposée

L'Algorithme 1 résume la trame proposée et décrit la gestion globale des pièces et du domaine du corridor. Cet algorithme est glouton puisque à chaque itération de la boucle tant que (Ligne 3) une nouvelle pièce est construite et n'est plus modifiée. Plus précisément, à chaque itération de cette boucle tant que, une pièce candidate est calculée pour chaque sommet du domaine du corridor restant $D(\mathcal{C}_r)$ grâce aux Lignes 7-9, que la procédure Score évalue pour ne garder que celle de plus haut score Ligne 11. Les métriques de score proposées sont décrites dans les Sections 3.2.5 et 3.3.2. La procédure ConstructionCorridorInterieurPWL (Ligne 8) calcule un corridor intérieur PWL au corridor $\mathcal{C}_r$ avec le sommet de départ v et la direction de progression d. Cela permet de construire et résoudre dans la procédure CalculPieceMaximale une série de problèmes LP expliquée Section 3.2.3 dont la résolution permet d'obtenir une pièce incluse dans le corridor $\mathcal{C}_r$. La Ligne 12 ajoute la pièce p à l'ensemble de pièces déjà construites $\mathcal{P}$. Finalement, la procédure MajDomaine$(\mathcal{Q}, \mathcal{C}_r, p)$ retire au corridor $\mathcal{C}_r$ la partie du domaine sur

laquelle la pièce p est définie, et renvoie $\mathcal{Q}$. Il peut y avoir plusieurs corridors dans la liste $\mathcal{Q}$ à cause d'une opération faite dans MajDomaine décrite Section 3.3.4 pour éviter un mauvais comportement de l'algorithme.

Algorithme 1 Trame de génération d'heuristiques pour le $\mathbb{R}^2$-CFP

Entrée : le corridor $\mathcal{C}$ à approximer, dont le domaine est un polygone convexe
Sortie : les pièces $\mathcal{P}$ de la fonction PWL qui approxime le corridor $\mathcal{C}$

1: $\mathcal{P} \leftarrow \emptyset$ ▷ *liste des pièces choisies*
2: $\mathcal{Q} \leftarrow \{\mathcal{C}\}$ ▷ *liste de corridors avec des domaines convexes pas encore couverts par des pièces*
3: **tant que** $\mathcal{Q} \neq \emptyset$ **faire**
4: $\mathcal{C}_r = \text{RetirerDernierElement}(\mathcal{Q})$ ▷ $\mathcal{C}_r$ *est appelé le corridor restant*
5: $candidats \leftarrow \emptyset$ ▷ *pièces candidates pour être ajoutées à* $\mathcal{P}$
6: **pour tout** v sommet de $D(\mathcal{C}_r)$ **faire**
7: $d \leftarrow \text{CalculDirectionProgression}(\mathcal{C}_r, v)$ ▷ *voir Sections 3.2.1 et 3.3.3*
8: $C_{int}^{PWL} \leftarrow \text{ConstructionCorridorInterieurPWL}(\mathcal{C}_r, v, d)$ ▷ *voir Sections 3.2.4 et 3.3.1*
9: $p \leftarrow \text{CalculPieceMaximale}(C_{int}^{PWL}, d)$ ▷ *voir Section 3.2.3*
10: $candidats \leftarrow candidats \cup \{p\}$
11: $p \leftarrow \text{argmax}_{p \in candidats} \text{Score}(p)$ ▷ *voir Sections 3.2.5 et 3.3.2*
12: $\mathcal{P} \leftarrow \mathcal{P} \cup \{p\}$
13: $\mathcal{Q} = \text{MajDomaine}(\mathcal{Q}, \mathcal{C}_r, p)$ ▷ *voir Section 3.3.4*

Nous résumons ici les composants utilisés dans la trame ainsi que l'endroit où ils sont décrits :

— la procédure CalculDirectionProgression calcule la direction de progression d avec les composants *DB* ou *ProMA*, décrits dans les Sections 3.2.1 et 3.3.3,
— le calcul du corridor intérieur PWL $\mathcal{C}_{int}^{PWL}$ est effectué dans la procédure ConstructionCorridorInterieurPWL avec le composant *raffinement du niveau d'approximation intérieure* qui dépend d'un réel $\eta \in [0, 1[$, décrit dans la Section 3.3.1,
— la procédure Score évalue la qualité des pièces produites avec les composants *Aire* ou *VaT* dans les Sections 3.2.5 et 3.3.2.

Nous notons $\mathcal{C} = Corridor(u, l, D)$ un corridor avec le domaine polygonal convexe $D \subset \mathbb{R}^2$ pour le reste du chapitre. Le fonctionnement de la trame permet de calculer une solution réalisable au $\mathbb{R}^2$-CFP pour un corridor dont les fonctions u et l sont continues. Cela dit, pour améliorer deux parties de la trame, certains composants avancés qui nécessitent les dérivées partielles premières et secondes de u et l sont utilisées. Dans le premier cas, le composant raffinement du niveau d'approximation intérieure utilise le gradient de u et l, mais il est facile d'utiliser uniquement les valeurs des fonctions à la place au prix d'une augmentation du temps de calcul. Dans le deuxième cas, il est nécessaire que le composant *VaT* utilise la deuxième différentielle de u et l.

3.1.2 Déroulement d'un algorithme généré par la trame

Pour illustrer le déroulement de l'Algorithme 1, considérons la fonction $f_0(x, y) = xe^{-x^2-y^2}$ à approximer avec une erreur absolue de 0.03 sur le domaine $[0.5, 2.0] \times [0.5, 2.0]$. Cela correspond à un corridor $C_0 = Corridor(f_0 + 0.03, f_0 - 0.03, [0.5, 2.0] \times [0.5, 2.0])$ illustré par la Figure 3.1. Les deux surfaces visibles sont les graphes des fonctions u et l.

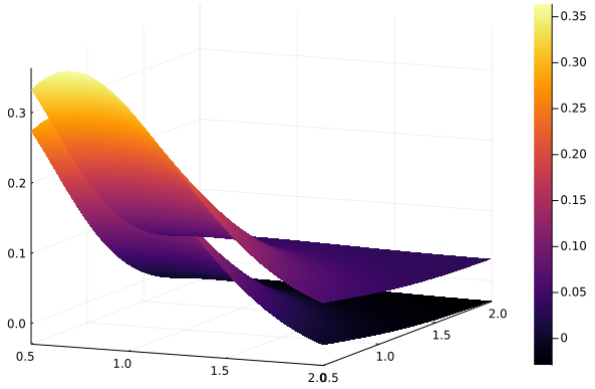
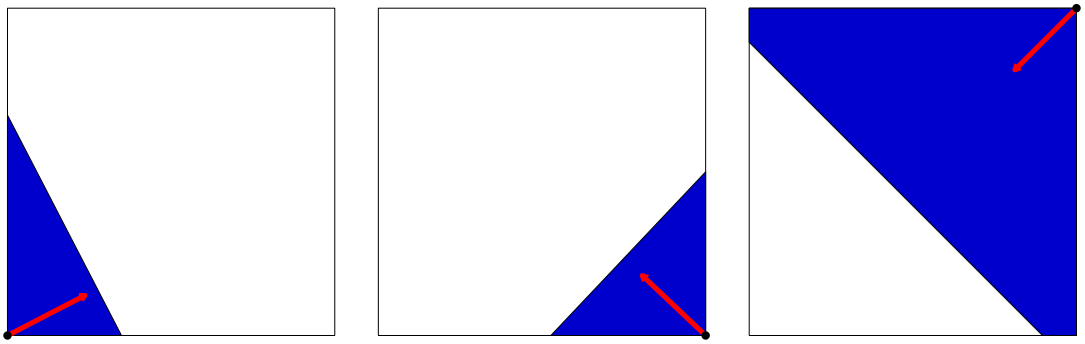


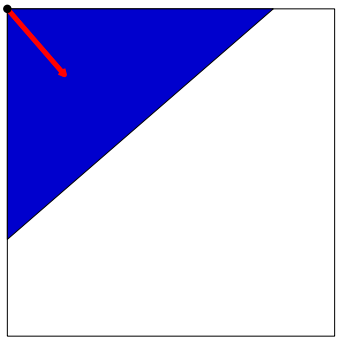
FIGURE 3.1: Corridor pour la fonction $f(x, y) = xe^{-x^2-y^2}$ sur le domaine $[0.5, 2.0] \times [0.5, 2.0]$ et l'erreur absolue de 0.03

Les Figures 3.2 à 3.5 montrent le déroulement de l'Algorithme 1 avec l'instance de corridor C_0. Chaque sous-figure montre l'état du domaine de la fonction PWL en cours de construction après une itération de la boucle pour tout (Ligne 6) : les pièces déjà ajoutées à $\mathcal{P}$ (Ligne 12) sont en vert, la pièce p qui vient d'être calculée par la procédure CALCULPIECEMAXIMALE est en bleu, le sommet v et la direction de progression d utilisés pour calculer p sont, respectivement, le point noir et la flèche rouge.

Pour calculer la première pièce, $D(\mathcal{C}_r)$ a quatre sommets, donc il y a quatre pièces candidates visibles dans la Figure 3.2 et c'est le candidat 3 qui obtient le plus haut score. La Figure 3.3 montre les trois pièces candidates pour devenir la deuxième pièce de la fonction PWL en construction. C'est le candidat 1 qui obtient le plus haut score. La pièce candidate 3 en cyan indique que la pièce avait déjà été calculée pendant un appel précédent à CALCULPIECEMAXIMALE et n'est donc pas recalculée. En l'occurence, c'est la pièce candidate 1 pour la pièce 1. Pour la troisième pièce, il y a quatre pièces candidates comme illustré dans la Figure 3.4, et c'est le candidat 1 qui obtient le plus haut score. Finalement, la Figure 3.5 montre le calcul de la quatrième pièce. Le premier problème de pièce maximale dans une direction d couvre l'ensemble du corridor restant $\mathcal{C}_r$, donc les problèmes de pièces maximales partant des autres sommets du corridor restant n'ont pas besoin d'être calculés. La Figure 3.6 montre la fonction PWL solution, où la couleur indique la valeur de la fonction PWL en chaque point du domaine.

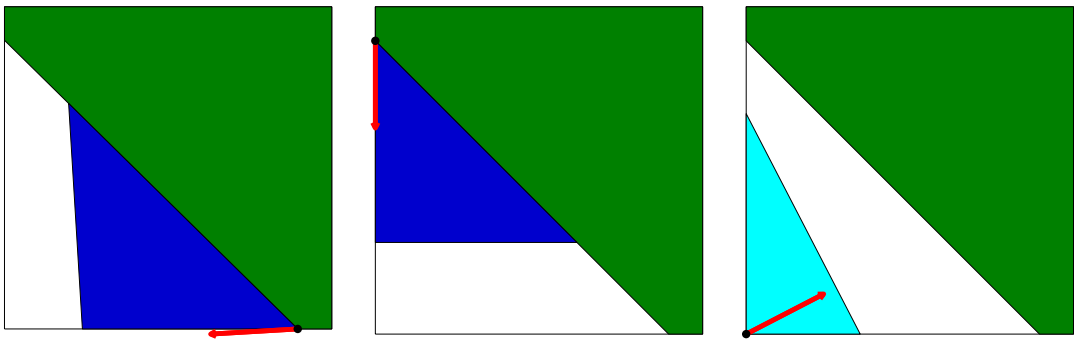


(a) Candidat 1 (b) Candidat 2 (c) Candidat 3, plus haut score



(d) Candidat 4

FIGURE 3.2: Itération 1 : pièces candidates pour la première pièce de la trame

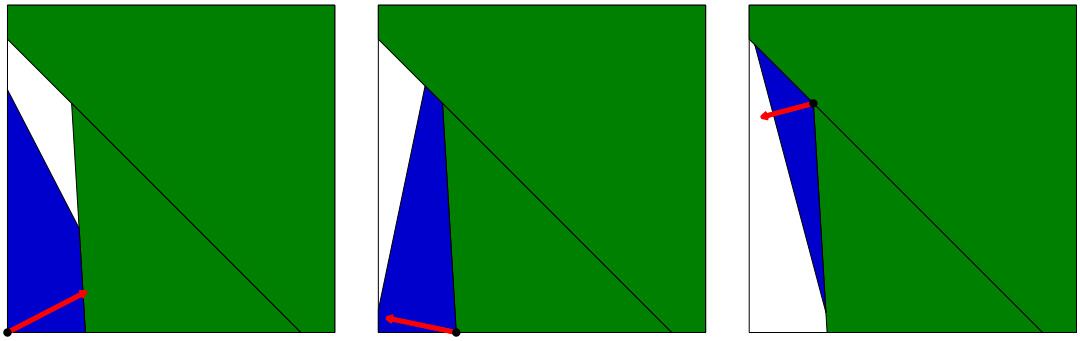


(a) Candidat 1, plus haut score (b) Candidat 2 (c) Candidat 3

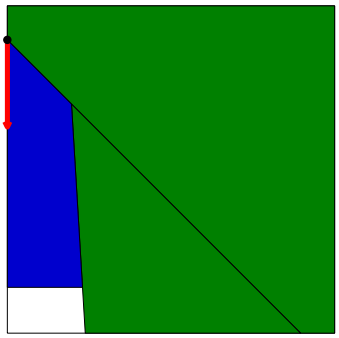
FIGURE 3.3: Itération 2 : pièces candidates pour la deuxième pièce de la trame

3.2 Description de la trame

Cette section décrit les procédures utilisées dans l'Algorithme 1. La Section 3.2.1 explique la procédure CALCULDIRECTIONPROGRESSION avec le composant *DB* pour *direction de la bissectrice*. La Section 3.2.2 présente le problème de pièce maximale dans une direction, tandis que

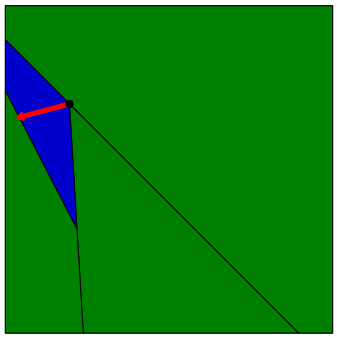


(a) Candidat 1, plus haut score (b) Candidat 2 (c) Candidat 3



(d) Candidat 4

FIGURE 3.4: Itération 3 : pièces candidates pour la troisième pièce de la trame



(a) Candidat 1, pièce qui traverse le corridor restant

FIGURE 3.5: Itération 4 : pièce candidate pour la quatrième pièce de la trame

la Section 3.2.3 détaille le calcul d'une fonction linéaire traversant un corridor intérieur PWL effectué par la procédure CALCULPIECEMAXIMALE. Ensuite, la Section 3.2.4 détaille la procédure CONSTRUCTIONCORRIDORINTERIEURPWL, qui construit un corridor intérieur PWL. La Section 3.2.5 décrit la procédure SCORE avec un composant basique. Finalement, la Section

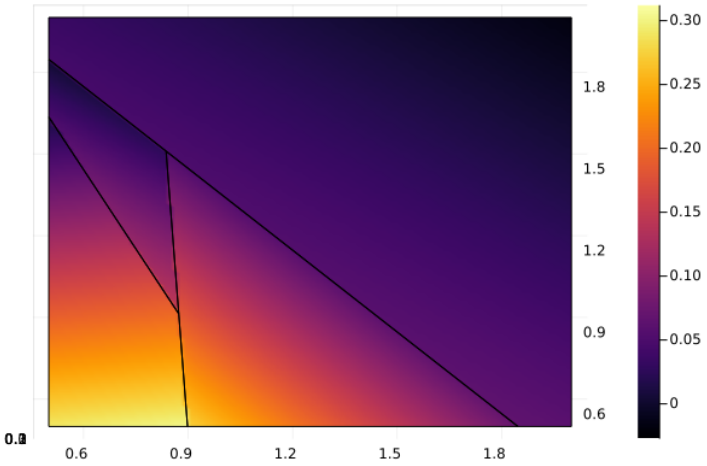


FIGURE 3.6: Valeurs de la fonction PWL construite

3.2.6 prouve que le principe 1 est bien respecté par la trame.

3.2.1 Calcul de la direction de progression

La Ligne 7 de l'Algorithme 1 identifie la direction de progression d en connaissant le sommet de départ v. Deux composants ont été testés pour ce choix. Le composant DB choisit pour direction de progression la bissectrice des deux arêtes de $D(\mathcal{C})$ utilisant v comme point de départ. Cette manière simple de construire le vecteur d nous servira de point de référence. Le deuxième composant, plus sophistiqué, est décrit dans la Section 3.3.3.

3.2.2 Définition du problème de pièce maximale dans la direction d

Nous introduisons la définition de pièce maximale dans la direction d adaptée à un corridor de $\mathbb{R}^2$, qui est immédiatement adaptable à $\mathbb{R}^m$ pour $m \geq 3$.

Définition 18 ($\mathbb{R}^2$-corridor tronqué dans la direction d)**.** Soient deux $\mathbb{R}^2$-corridors $\mathcal{C}_1 = Corridor(u, l, D_1)$ et $\mathcal{C}_2 = Corridor(u, l, D_2)$ et un vecteur $d \in \mathbb{R}^2 \setminus \{0\}$. On dit que $\mathcal{C}_2$ est un *corridor tronqué de* $\mathcal{C}_1$ *dans la direction* d si et seulement s'il existe un réel σ pour lequel $D(\mathcal{C}_2) = D(\mathcal{C}_1) \cap \{(x, y) \in \mathbb{R}^2 | (x, y) \cdot d \leq \sigma\}$.

Cela revient à dire que $D(\mathcal{C}_2)$ est l'intersection de $D(\mathcal{C}_1)$ avec un demi-plan orienté selon le vecteur d. Nous rappelons qu'on dit qu'une fonction ou une pièce traverse un corridor $\mathcal{C}$ si elle est définie sur $D(\mathcal{C})$ et si son graphe est inclus dans $\mathcal{C}$.

Définition 19 (Pièce maximale dans la direction d)**.** Soient $\mathcal{C}$ un $\mathbb{R}^2$-corridor, $d \in \mathbb{R}^2 \setminus \{0\}$ un vecteur, σ un réel et p une pièce définie sur un domaine $D_p = D(\mathcal{C}) \cap \{(x, y) \in \mathbb{R}^2 | (x, y) \cdot d \leq \sigma\}$ qui traverse le corridor tronqué de $\mathcal{C}$ dans la direction d de domaine D_p. La pièce p est une *pièce*

maximale dans la direction d du corridor $\mathcal{C}$ s'il n'existe pas de pièce p' qui traverse un corridor tronqué de $\mathcal{C}$ dans la direction d de domaine $D(\mathcal{C}) \cap \{(x, y) \in \mathbb{R}^2 | (x, y) \cdot d \leq \sigma'\}$ avec $\sigma' > \sigma$.

Une pièce maximale d'un $\mathbb{R}^2$-corridor est donc maximale au sens où elle traverse un corridor tronqué dans la direction d qui est le plus grand possible. Pour simplifier le discours, si le corridor $\mathcal{C}$ sur lequel se calcule la pièce maximale dans la direction d de corridor $\mathcal{C}$ est clair, nous ne le mentionnerons pas et dirons à la place de *pièce maximale dans la direction d*. En revanche, nous mentionnerons toujours *dans la direction d* pour différencier une pièce maximale sur un corridor de domaine dans $\mathbb{R}$ d'une pièce maximale dans la direction d du corridor $\mathcal{C}$ de domaine $\mathbb{R}^2$. La Figure 3.7 montre un exemple de domaine couvert par une pièce obtenue comme solution d'un problème de pièce maximale dans la direction de progression d. Le domaine est partiellement couvert dans sa partie inférieure gauche par une pièce maximale dans la direction d en bleu. Le demi-plan nécessaire au calcul de la pièce maximale dans la direction d est hachuré en noir.

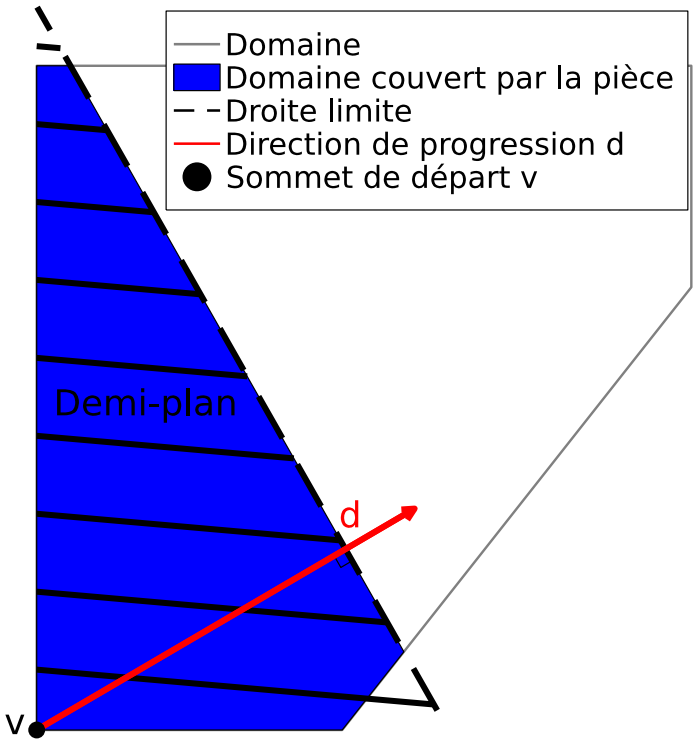


FIGURE 3.7: Représentation d'une solution de problème de pièce maximale dans la direction de progression

Calculer une pièce par la procédure CALCULPIECEMAXIMALE de l'Algorithme 1 consiste à résoudre le problème de pièce maximale dans la direction d, noté (MP_d). Cela revient à calculer le plus grand demi-plan dirigé par le vecteur d tel qu'il existe une fonction linéaire qui traverse

le corridor sur le domaine obtenu comme intersection du demi-plan et du domaine du corridor restant :

$$(MP_d)\begin{cases}\max & \sigma \\ s.c. & \alpha x+\beta y+\gamma \in \mathcal{C}_\sigma^d \quad \forall (x,y)\in D(\mathcal{C}_\sigma^d) \\ & \alpha,\beta,\gamma,\sigma\in\mathbb{R},\end{cases}$$

où $\mathcal{C}_\sigma^d = Corridor(u,l,D(\mathcal{C}_r)\cap\{(x,y)\in\mathbb{R}^2|(x,y)^T\cdot d\leq\sigma\})$, un corridor tronqué de $\mathcal{C}$ dans la direction d. (MP_d) est un problème d'optimisation semi-infinie généralisé (GSIP), c'est-à-dire qu'il a un nombre infini de contraintes, un nombre fini de variables et l'ensemble réalisable des variables dépend lui-même de la valeur des variables (σ dans notre cas). En effet, il y a bien un nombre de contraintes infini, quatre variables : σ pour l'intersection du demi-plan ainsi que trois variables réelles (α,β,γ) pour décrire la fonction linéaire $L(x,y)=\alpha x+\beta y+\gamma$, et les contraintes dépendent de la variable σ à travers le corridor $\mathcal{C}_\sigma^d$. Pour plus d'informations, [Guerra Vázquez et al., 2008] présente une introduction aux GSIP. Choisir une pièce comme solution de (MP_d) permet donc bien de choisir la "pièce la plus grande" dans le sens où son domaine s'étend le plus loin possible dans la direction de progression.

Le problème (MP_d) étant un GSIP, sa résolution exacte requiert a priori un temps de calcul conséquent pour notre algorithme, puisque ce problème doit être résolu plusieurs fois à chaque itération.

Nous proposons de générer des solutions réalisables de (MP_d) en nous focalisant sur un sous-espace de l'ensemble réalisable du GSIP qui peut être décrit par un nombre fini de contraintes. L'idée clef est d'approximer le corridor par l'intérieur avec un corridor PWL, puis d'utiliser ce corridor PWL pour calculer une solution réalisable de (MP_d) en résolvant une série de problèmes LP. L'efficacité de l'approche repose sur le fait qu'un corridor PWL peut être défini par un nombre fini de contraintes linéaires et l'approximation par l'intérieur permet de garantir que la solution obtenue est bien réalisable pour le GSIP initial. Les deux prochaines sous-sections présentent le calcul efficace de solutions réalisables du (MP_d) et la construction d'un corridor intérieur PWL.

3.2.3 Calcul d'une solution réalisable au problème de pièce maximale pour un corridor PWL

Nous allons définir plusieurs problèmes à partir du problème (MP_d). Ils ont pour caractéristique commune que leur résolution fournit une solution réalisable pour (MP_d). Le dernier problème introduit permet de calculer une telle solution réalisable en résolvant une série de problèmes LP. Ce résultat est montré dans le Lemme 20 dont les étapes de la preuve sont mises en avant par des faits.

Soit $\mathcal{C}_{PWL}{}_\sigma^d = Corridor(\tilde{u},\tilde{l},D(\mathcal{C}_\sigma^d))$ un corridor intérieur PWL de $\mathcal{C}_\sigma^d$. Notons $(D_{\tilde{u}}^i)_{i\in I}$ et

$(D_{\tilde{l}}^j)_{i\in J}$ les sous-domaines du corridor $\mathcal{C}_{PWL_\sigma}^d$ sur lesquels $\tilde{u}$ et $\tilde{l}$ sont linéaires respectivement, avec I et J les ensembles d'indices respectifs. Le problème (MP'_d) :

$$(MP'_d)\begin{cases} \max\ \sigma \\ s.c. \\ \alpha x+\beta y+\gamma-\tilde{u}_i(x,y)\leq 0 \quad \forall i\in I, \forall (x,y)\in D(\mathcal{C}_\sigma^d)\cap D_{\tilde{u}}^i, \\ \alpha x+\beta y+\gamma-\tilde{l}_j(x,y)\geq 0 \quad \forall j\in J, \forall (x,y)\in D(\mathcal{C}_\sigma^d)\cap D_{\tilde{l}}^j, \\ \alpha,\beta,\gamma,\sigma\in\mathbb{R}, \end{cases}$$

a des contraintes plus fortes que le problème (MP_d) parce que $\mathcal{C}_\sigma^d$ y est remplacé par un corridor intérieur.

Fait 1 : une solution réalisable de (MP'_d) est une solution réalisable de (MP_d).

Or, sur chaque domaine polygonal $D(\mathcal{C}_\sigma^d)\cap D_{\tilde{u}}^i$, la différence de fonctions $\alpha x+\beta y+\gamma-\tilde{u}(x,y)$ est une fonction linéaire, donc son maximum est atteint en un des sommets du domaine polygonal. Il suffit donc de vérifier $\alpha x+\beta y+\gamma-\tilde{u}(x,y)\leq 0$ pour chaque sommet du domaine $D(\mathcal{C}_\sigma^d)\cap D_{\tilde{u}}^i$ pour vérifier la contrainte $\alpha x+\beta y+\gamma-\tilde{u}(x,y)\leq 0$ sur $D(\mathcal{C}_\sigma^d)\cap D_{\tilde{u}}^i$. Un raisonnement similaire amène au résultat équivalent pour les contraintes utilisant $\tilde{l}$. Alors, le problème (MP''_d) est équivalent au problème (MP'_d), c'est-à-dire qu'il a les mêmes solutions optimales, solutions réalisables et la même valeur optimale. Il a toutefois l'avantage de nécessiter un nombre fini de contraintes linéaires :

$$(MP''_d)\begin{cases} \max\ \sigma \\ s.c. \\ \alpha x+\beta y+\gamma-\tilde{u}_i(x,y)\leq 0 \quad \text{pour chaque sommet } (x,y) \text{ de } D(\mathcal{C}_\sigma^d)\cap D_{\tilde{u}}^i, \forall i\in I \\ \alpha x+\beta y+\gamma-\tilde{l}_j(x,y)\geq 0 \quad \text{pour chaque sommet } (x,y) \text{ de } D(\mathcal{C}_\sigma^d)\cap D_{\tilde{l}}^j, \forall j\in J \\ \alpha,\beta,\gamma,\sigma\in\mathbb{R}. \end{cases}$$

Fait 2 : Une solution réalisable de (MP''_d) est réalisable pour (MP'_d).

De plus, le problème (MP''_d) a des contraintes qui utilisent des intersections de polygones $D(\mathcal{C}_\sigma^d)\cap D_{\tilde{u}}^i$. Le nombre et les positions des sommets de ces intersections dépendent de la variable σ, ce qui suggère d'utiliser des variables binaires dans la modélisation. Une manière de ne pas avoir ces variables binaires est de paramétrer (MP''_d) en fonction de la valeur de σ, pour obtenir le problème LP de réalisabilité $(MP''_{d,\sigma})$:

$$(MP''_{d,\sigma})\begin{cases} \text{Trouver } \alpha,\beta,\gamma \text{ satisfaisant} \\ \alpha x+\beta y+\gamma-\tilde{u}_i(x,y)\leq 0 \quad \text{pour chaque sommet } (x,y) \text{ de } D(\mathcal{C}_\sigma^d)\cap D_{\tilde{u}}^i, \forall i\in I \\ \alpha x+\beta y+\gamma-\tilde{l}_j(x,y)\geq 0 \quad \text{pour chaque sommet } (x,y) \text{ de } D(\mathcal{C}_\sigma^d)\cap D_{\tilde{l}}^j, \forall j\in J \\ \alpha,\beta,\gamma\in\mathbb{R}. \end{cases}$$

Nous pouvons donc calculer une solution réalisable au problème (MP''_d) en calculant la plus grande valeur de σ telle que $(MP''_{d,\sigma})$ a une solution réalisable.

Fait 3 : Une solution réalisable pour $(MP''_{d,\sigma})$ est réalisable pour (MP''_d).

Pour savoir les domaines de pièces $D_{\tilde{u}}^i$ ou $D_{\tilde{l}}^j$ qui s'intersectent avec $D(\mathcal{C}_\sigma^d)$, nous calculons

pour chaque pièce p_k de $\tilde{u}$ ou $\tilde{l}$ un nombre σ_k, avec k dans $I \cup J$, correspondant au plus petit produit scalaire entre la direction de progression d et tous les sommets du domaine de la pièce. Si $\sigma_i < \sigma$, alors au moins une partie d'intérieur non vide de $D^i_{\tilde{u}}$ est dans $D(\mathcal{C}^d_\sigma)$. Sinon, l'intersection est d'intérieur vide. Dans le cas où l'intersection est non vide mais d'intérieur vide, il n'est pas utile d'ajouter les contraintes car il existe une pièce adjacente dont le domaine contient cette intersection. Pour ne pas calculer l'intersection d'un grand nombre de polygones $D^i_{\tilde{u}}$ ou $D^j_{\tilde{l}}$ avec $D(\mathcal{C}^d_\sigma)$, nous définissons le problème $(MP'''_{d,\sigma})$:

$$(MP'''_{d,\sigma}) \begin{cases} \text{Trouver } \alpha, \beta, \gamma \text{ satisfaisant} & \\ \alpha x + \beta y + \gamma - \tilde{u}_i(x,y) \leq 0 & \text{pour chaque sommet } (x,y) \text{ de } D^i_{\tilde{u}}, \forall i \in I \text{ si } \sigma_i < \sigma \\ \alpha x + \beta y + \gamma - \tilde{l}_j(x,y) \geq 0 & \text{pour chaque sommet } (x,y) \text{ de } D^j_{\tilde{l}}, \forall j \in J \text{ si } \sigma_j < \sigma \\ \alpha, \beta, \gamma \in \mathbb{R}. & \end{cases}$$

La différence avec $(MP''_{d,\sigma})$ est que le domaine d'une pièce p_k de $\tilde{u}$ ou $\tilde{l}$ est complètement inclus dans les contraintes du problème dès lors que $\sigma_k < \sigma$ au lieu d'inclure seulement l'intersection du domaine de la pièce avec $D(\mathcal{C}^d_\sigma)$. Ainsi, si (α, β, γ) est une solution de $(MP'''_{d,\sigma})$, alors (α, β, γ) est aussi une solution de $(MP''_{d,\sigma})$.

Fait 4 : Une solution réalisable de $(MP'''_{d,\sigma})$ est une solution réalisable de $(MP''_{d,\sigma})$.

La série de faits prouve le résultat suivant :

Lemme 20. *Soient un corridor $\mathcal{C}$, un vecteur $d \in \mathbb{R}^2$ et un réel σ. Une solution réalisable de $(MP'''_{d,\sigma})$ est une solution réalisable de (MP_d).*

Pour déterminer pour quelles valeurs de σ nous résolvons $(MP'''_{d,\sigma})$, nous procédons de la manière suivante. Nous trions par ordre croissant toutes les pièces p_k de $\tilde{u}$ ou $\tilde{l}$ avec les valeurs σ_k. Soient σ^1 et σ^2 tels que $\sigma^1 < \sigma^2$. Les problèmes (MP'''_{d,σ^1}) et (MP'''_{d,σ^2}) sont les mêmes s'il n'existe pas de k tel que $\sigma_k \in]\sigma^1, \sigma^2]$ puisque les mêmes domaines de pièces sont inclus. Il suffit donc de tester la réalisabilité de $(MP'''_{d,\sigma})$ pour les valeurs de $(\sigma_k)_{k \in I \cup J}$. Nous effectuons une recherche dichotomique sur les valeurs ordonnées de $(\sigma_k)_{k \in I \cup J}$ pour trouver la plus grande valeur σ^* telle que $(MP'''_{d,\sigma})$ a une solution. Cela permet de résoudre $(MP'''_{d,\sigma})$ seulement un nombre de fois grandissant en le logarithme du cardinal de l'ensemble $\{\sigma_k | k \in I \cup J\}$. Le processus suivi prouve que la fonction $L(x,y) = \alpha x + \beta y + c$ traverse le corridor $\mathcal{C}^d_{\sigma^*}$.

3.2.4 Approximation intérieure d'un corridor

Nous décrivons ici le fonctionnement de la procédure ConstructionCorridorInterieurPWL. Dans cette procédure, le calcul du corridor intérieur PWL $\mathcal{C}_{PWL}$ de $\mathcal{C}$ revient à calculer les fonctions PWL $\tilde{u}$ et $\tilde{l}$ vérifiant $l(x,y) \leq \tilde{l}(x,y) \leq \tilde{u}(x,y) \leq u(x,y)$ pour chaque (x,y) dans $D(\mathcal{C})$. Elle utilise pour cela le sommet de départ v, les fonctions u et l du corridor restant, le domaine restant D et la direction de progression d. La construction du corridor intérieur PWL est divisée en quatre étapes, expliquées dans l'ordre ci-dessous.

Étape 1 : calcul du domaine atteignable par la pièce maximale dans la direction d partant du sommet v. Nous définissons le domaine atteignable d'une pièce maximale dans la direction d.

Définition 21 (corridor atteignable)**.** Soient $\mathcal{C} = Corridor(u, l, D)$ et d une direction de progression. On appelle *corridor atteignable* un corridor tronqué de $\mathcal{C}$ dans la direction d qui contient une pièce maximale dans la direction d du corridor $\mathcal{C}$. On appelle domaine atteignable le domaine d'un corridor atteignable.

L'intérêt d'utiliser un corridor atteignable au lieu du corridor original est d'accélérer les calculs en négligeant des zones du domaine que la pièce maximale ne peut pas atteindre, d'où le mot "atteignable". Par exemple, le corridor intérieur PWL du corridor atteignable nécessite moins de pièces qu'un corridor intérieur du corridor original. Cela demande donc moins de calculs parfois coûteux, et les étapes 3 et 4 en demanderont moins aussi. Notre méthode d'obtention du domaine atteignable ne permet pas forcément de trouver le plus petit domaine atteignable, ce qui reviendrait à résoudre directement le problème de pièce maximale dans la direction d. Son but est plutôt de trouver relativement rapidement un domaine atteignable raisonnablement petit par rapport au domaine du corridor restant. Pour cela, nous nous appuyons sur le lemme suivant.

Lemme 22. *Soient le corridor $\mathcal{C}$, un sommet v du domaine $D(\mathcal{C})$ et un vecteur $d \in \mathbb{R}^2$. Soit un segment $[v, v']$ obtenu comme intersection du domaine $D(\mathcal{C})$ et d'une droite passant par v. Soit $[v, e]$ une pièce maximale sur le segment $[v, v']$. Soit le demi-plan $H = \{x | x.d \leq e.d\}$. Si le produit scalaire $(e - v).d$ est positif et $e \neq v'$, alors la pièce maximale dans la direction d du corridor $\mathcal{C}$ est incluse dans le demi-plan H.*

Démonstration. H contient v car $(e - v).d \geq 0$. Supposons par l'absurde que p_{max} la pièce maximale dans la direction d du corridor $\mathcal{C}$ ne soit pas incluse dans le demi-plan H. Alors en particulier le domaine de p_{max} contient strictement le segment $[v, e]$ par définition de H. Cela veut dire qu'il existe une fonction linéaire définie sur un segment contenant strictement $[v, e]$ dont le graphe est inclus dans le corridor $\mathcal{C}$, ce qui contredit la maximalité de $[v, e]$ la pièce maximale sur le segment $[v, v']$. □

Par construction, les composants *DB* et *ProMA* produisent une direction de progression d qui satisfait la condition du lemme : "le produit scalaire $(e - v).d$ est strictement positif" à condition que le domaine du corridor soit convexe, ce qui est toujours le cas grâce au corridor initial et à la fonction MajDomaine.

Notre méthode de construction du corridor atteignable utilise le Lemme 22 ainsi que trois pièces maximales sur un segment partant de v. Deux de ces pièces maximales sur des segments

sont le long des deux demi-droites qui partent de v et longent les arêtes ayant v comme extrémité. La troisième a pour domaine de $\mathbb{R}$-corridor le segment qui relie v au sommet du corridor restant de plus grand produit scalaire avec d. Ainsi, si ce sommet est inclus dans le domaine de la pièce maximale dans la direction d, ce domaine est le domaine du corridor restant. La Figure 3.8 illustre la création de ce domaine atteignable. Les trois segments sur lesquels sont calculés les

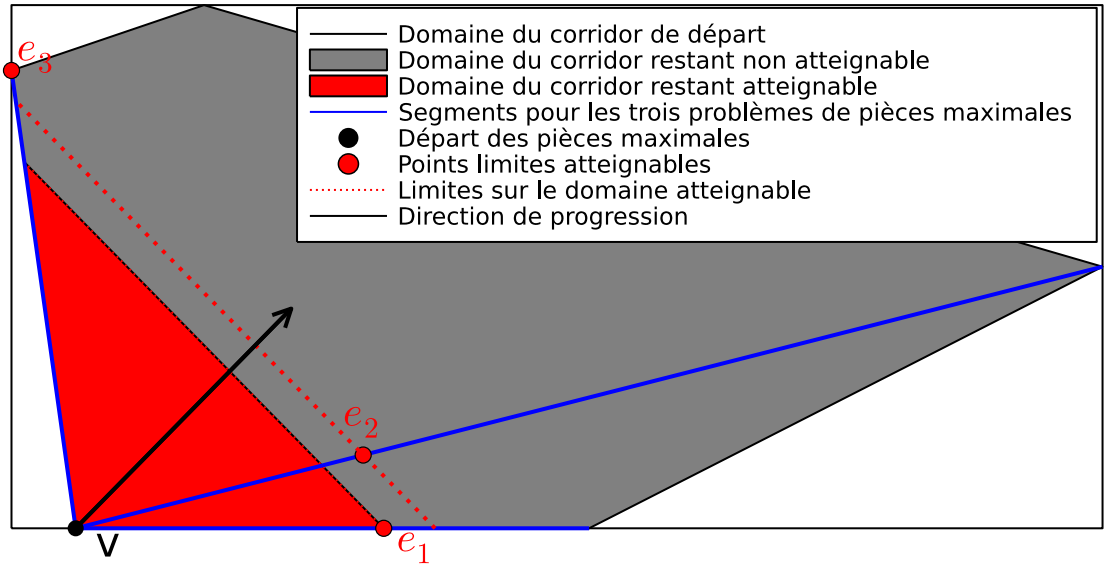


FIGURE 3.8: Construction du domaine atteignable du corridor restant

pièces maximales sur un segment partant de v sont en bleu sur la figure. Nous notons e_1, e_2 et e_3 les extrémités différentes de v du domaine des pièces maximales et h_1, h_2, h_3 les droites ayant d pour vecteur directeur et passant par les points e_1, e_2 et e_3. Ce sont les frontières des demi-plans mentionnés dans le lemme. Dans la figure, la droite h_3 n'est pas définie car la pièce maximale d'extrémité de domaine e_3 atteint le bout de son $\mathbb{R}$-corridor, ce qui a pour conséquence qu'elle ne nous apprend pas l'existence d'une limite au domaine atteignable d'après le lemme. Finalement, le domaine atteignable est l'intersection du domaine restant avec les deux demi-plans définissant des limites au domaine atteignable, qui sont les deux demi-plans séparés par les droites h_1, h_2 et contenant le point v. Il est possible de calculer plus de pièces maximales sur des segments pour mieux estimer le domaine atteignable, mais ces trois pièces nous ont semblé un compromis entre le temps de calcul du domaine atteignable et son utilité à accélérer le calcul de la pièce maximale dans la direction d.

Étape 2 : construction du plus petit rectangle contenant le domaine atteignable avec une base orientée parallèle à la direction de progression d. Nous construisons le rectangle le plus petit contenant le domaine atteignable qui a une base parallèle à la direction de

progression d car le corridor PWL construit à l'étape 3 sera composé de pièces rectangulaires.

Étape 3 : calcul de $\tilde{u}$ et $\tilde{l}$. Nous utilisons un argument d'arithmétique d'intervalle comme expliqué dans le Lemme 23 pour calculer la fonction $\tilde{u}$ composant le corridor intérieur PWL sur le rectangle construit à l'étape 2. Un résultat similaire pour $\tilde{l}$ existe.

Pour ce faire, considérons une fonction $u \in C^1$ défini sur le domaine $\mathcal{D} = [a, b] \times [c, d] \subset \mathbb{R}^2$, les réels $r_x = b - a$ et $r_y = d - c$, le vecteur $(M_x, M_y) := (\frac{a+b}{2}, \frac{c+d}{2})$, le gradient ∇u de u et l'ensemble $[D_x^{low}, D_x^{high}] \times [D_y^{low}, D_y^{high}]$ tel que $\nabla u(x, y) \in [D_x^{low}, D_x^{high}] \times [D_y^{low}, D_y^{high}]$ pour tout (x, y) dans $\mathcal{D}$. De plus, soient les réels :

$$u^-_{(a,c)} := u(M_x, M_y) - D_x^{high} r_x - D_y^{high} r_y, \tag{3.1}$$
$$u^-_{(b,c)} := u(M_x, M_y) + D_x^{low} r_x - D_y^{high} r_y, \tag{3.2}$$
$$u^-_{(b,d)} := u(M_x, M_y) + D_x^{low} r_x + D_y^{low} r_y, \tag{3.3}$$
$$u^-_{(a,d)} := u(M_x, M_y) - D_x^{high} r_x + D_y^{low} r_y. \tag{3.4}$$

Lemme 23. *Si une fonction linéaire L satisfait les quatre contraintes* (3.5)-(3.8) *alors $L(x, y) \leq u(x, y)$ pour tout (x, y) dans $\mathcal{D}$.*

$$L(a, c) \leq u^-_{(a,c)} \tag{3.5}$$
$$L(b, c) \leq u^-_{(b,c)} \tag{3.6}$$
$$L(b, d) \leq u^-_{(b,d)} \tag{3.7}$$
$$L(a, d) \leq u^-_{(a,d)} \tag{3.8}$$

Démonstration. Soit (x_0, y_0) dans $\mathcal{D}$. On a :

$$u(M_x, M_y) - D_x^{high}(M_x - x_0) - D_y^{high}(M_y - y_0) \leq u(x_0, y_0) \quad \text{si } x_0 \leq M_x, y_0 \leq M_y, \tag{3.9}$$
$$u(M_x, M_y) + D_x^{low}(x_0 - M_x) - D_y^{high}(M_y - y_0) \leq u(x_0, y_0) \quad \text{si } x_0 \geq M_x, y_0 \leq M_y, \tag{3.10}$$
$$u(M_x, M_y) + D_x^{low}(x_0 - M_x) + D_y^{low}(y_0 - M_y) \leq u(x_0, y_0) \quad \text{si } x_0 \geq M_x, y_0 \geq M_y, \tag{3.11}$$
$$u(M_x, M_y) - D_x^{high}(M_x - x_0) + D_y^{low}(y_0 - M_y) \leq u(x_0, y_0) \quad \text{si } x_0 \leq M_x, y_0 \geq M_y, \tag{3.12}$$

parce que $[D_x^{low}, D_x^{high}] \times [D_y^{low}, D_y^{high}]$ sont des bornes de ∇u sur le domaine $\mathcal{D}$. Soit L_{PWL} une fonction PWL à quatre pièces, de domaines correspondant aux conditions sur (x_0, y_0) dans les inégalités (3.9)-(3.12). Les fonctions linéaires les décrivant sont les termes de gauche de ces inégalités. En particulier, $L_{PWL} \leq u$ sur le domaine $\mathcal{D}$. De plus, un calcul direct montre que $L(x, y) \leq L_{PWL}(x, y)$ pour tout (x, y) sommet des quatre pièces. Finalement, comme L et L_{PWL} sont linéaires sur chacune des quatre pièces du domaine de L_{PWL}, les inégalités $L \leq L_{PWL} \leq u$ sont vraies sur le domaine $\mathcal{D}$. □

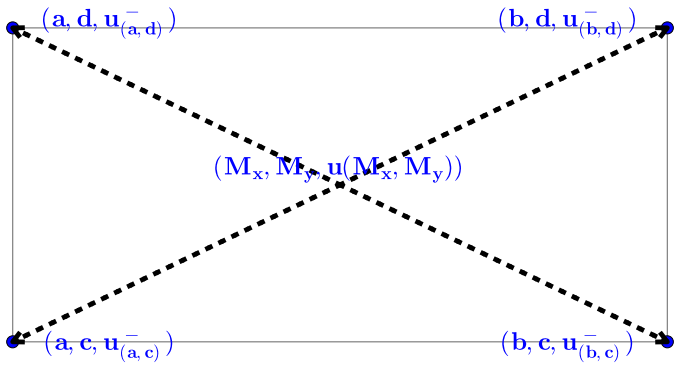


FIGURE 3.9: Schéma illustrant la construction d'une pièce du corridor intérieur PWL d'après le Lemme 23

La Figure 3.9 illustre la fonction PWL L_{PWL} décrite par le Lemme 23. Les droites pleines et en pointillé sont les limites des quatre pièces. Le point $(M_x, M_y, u(M_x, M_y))$ est sur la surface de la fonction u. Nous utilisons les valeurs extrêmes du gradient ∇u et le point $(M_x, M_y, u(M_x, M_y))$ pour construire la troisième coordonnée des points extrémaux du domaine $\mathcal{D}$ pour la fonction L_{PWL}. C'est pour cela que sur la figure des flèches relient $(M_x, M_y, u(M_x, M_y))$ aux sommets des quatre sommets du rectangle. Ainsi, si une fonction linéaire L satisfait les quatre contraintes (3.5)-(3.8), alors elle satisfait aussi $L(x, y) \leq u(x, y)$ pour tout (x, y) dans D.

Le résultat du Lemme 23 est un outil très intéressant pour construire les fonctions PWL $\tilde{u}$ et $\tilde{l}$ qui forment une approximation intérieure du corridor $\mathcal{C}$. Il reste cependant une difficulté : du fait qu'ils sont construits indépendamment l'un de l'autre, il peut arriver que $\tilde{l} \not\leq \tilde{u}$. Une manière d'éviter ce cas de figure revient à vérifier que $\tilde{u} \geq \tilde{l}$ en chaque pièce, et à décomposer les pièces pour lesquelles ce n'est pas le cas. La Section 3.3.1 présente une autre manière de procéder, qui est celle utilisée dans les expériences numériques de la Section 3.4.

Étape 4 : suppression des pièces de $\tilde{u}$ et $\tilde{l}$ qui ne s'intersectent pas avec le domaine atteignable. Nous supprimons les pièces de $\tilde{u}$ et $\tilde{l}$ qui ont une intersection d'intérieur vide avec le domaine atteignable puisque ces pièces n'apportent pas d'information utile au problème de pièce maximale dans une direction d. Concernant les pièces partiellement incluses dans le domaine atteignable, une possibilité d'amélioration de notre code serait de supprimer leurs parties hors domaine atteignable car ces parties sont superflues.

3.2.5 Calcul du score des pièces

Dans l'Algorithme 1, une procédure SCORE évalue la qualité de différentes pièces au travers de leur capacité à couvrir "efficacement" le domaine puisque le but est de produire une fonction PWL avec le moins de pièces possible. La pièce de plus haut score est ajoutée à la fonction PWL

en construction. Nous avons implémenté deux composants pour calculer le score d'une pièce.

Le composant *Aire* mesure simplement l'aire du domaine de la pièce p. C'est une manière simple d'évaluer la qualité d'une pièce à couvrir efficacement le domaine. Elle nous servira de référence à comparer à d'autres procédures Score. Une procédure de calcul de score plus sophistiquée est donnée dans la Section 3.3.2.

3.2.6 Découpage du domaine du corridor en polygones convexes

Pour minimiser le nombre de pièces, le domaine du corridor devrait pouvoir être pavé de formes aussi générales que possible à condition qu'elles puissent être formulées en un MILP. Ces formes sont des polygones, mais nous ajoutons la contrainte que ces formes sont aussi convexes parce que la formulation d'un polygone non convexe dans un MILP introduirait des variables binaires supplémentaires. Il est attendu que l'utilisation de polygones convexes au lieu de triangles participe à produire des fonctions PWL avec moins de morceaux. La Proposition 24 montre que l'Algorithme 1 respecte bien le principe 1.

Proposition 24. *Toute fonction PWL générée par l'Algorithme 1 est composée de pièces dont les domaines sont des polygones convexes.*

Démonstration. Nous utilisons dans la suite plusieurs fois la propriété que l'intersection d'un polygone convexe avec un demi-plan est un polygone convexe. La convexité vient de l'intersection de deux convexes, et c'est un polygone car résultant de l'intersection d'un polygone et d'un demi-plan.

Montrons par récurrence que si $D(\mathcal{C}_r)$ (Ligne 4) est un polygone convexe, alors le domaine de la pièce produite à cette itération et $D(\mathcal{C}_r)$ après l'exécution de la Ligne 13 sont tous les deux des polygones convexes. Le corridor $\mathcal{C}$ en entrée est convexe par hypothèse, donc pendant la première itération de la boucle tant que (Ligne 3), le corridor $\mathcal{C}_r$ issu de la Ligne 2 a un domaine qui est un polygone convexe. Supposons donc que $D(\mathcal{C}_r)$ est un polygone convexe à chaque itération à la Ligne 4. La pièce produite (Ligne 9) est l'intersection de $D(\mathcal{C}_r)$ et du demi-plan $\{(x, y) \in \mathbb{R}^2 | (x, y)^T \cdot d \leq \sigma\}$ comme solution réalisable du problème (MP_d) décrit dans la Section 3.2.2. Son domaine est donc un polygone convexe. De plus, la procédure MajDomaine, décrite dans les Sections 3.1.1 et 3.3.4, effectue deux opérations susceptibles d'ajouter un élément à $\mathcal{Q}$ qui n'est pas un polygone convexe. La première opération (Section 3.1.1) consiste à retirer à $D(\mathcal{C}_r)$ le domaine couvert par la pièce nouvellement construite en intersectant $D(\mathcal{C}_r)$ avec le demi-plan complémentaire de celui utilisé dans l'intersection produisant le domaine de la pièce produite. C'est donc à nouveau convexe comme l'intersection d'un polygone convexe et d'un demi-plan. La deuxième opération (Section 3.3.4) consiste elle à diviser $D(\mathcal{C}_r)$ en deux polygones autour d'une droite passant par deux sommets de $D(\mathcal{C}_r)$, ce qui revient encore à intersecter un polygone convexe avec un demi-plan ou son complémentaire, qui est aussi un demi-plan. □

3.3 Améliorations des composants de la trame

Les deux sections précédentes ont permis de décrire notre trame de génération d'heuristiques. Cette section présente des composants ou des procédures plus avancées que celles présentées dans la Section 3.2 à utiliser dans la trame de génération d'heuristiques. La Section 3.3.1 concerne la construction d'un corridor intérieur PWL, les Sections 3.3.2 et 3.3.3 proposent des composants plus avancés pour le calcul du score d'une pièce et la direction de progression respectivement, et la Section 3.3.4 décrit une manière d'éviter un mauvais comportement de l'Algorithme 1.

3.3.1 Raffinement du niveau d'approximation intérieure d'un corridor

Cette partie décrit un composant de la trame pour effectuer l'étape 3 de la construction d'un corridor intérieur PWL dépendant d'une valeur $\eta \in [0, 1[$. Pour construire un corridor intérieur PWL en utilisant le Lemme 23, nous utilisons une méthode que nous appelons *raffinement du niveau d'approximation intérieure*. Elle est décrite après l'introduction des notions de hauteur de corridor et de niveau d'approximation intérieure.

Définition 25 (Hauteur de corridor)**.** Soit $\mathcal{C} = Corridor(u, l, D)$ un corridor. On appelle $H_{\mathcal{C}}(x, y)) = u(x, y) - l(x, y)$ la *hauteur du corridor* $\mathcal{C}$ *au point* (x, y).

Définition 26 (Corridor de hauteur constante)**.** On dit qu'un corridor $\mathcal{C}$ a une *hauteur constante* si la hauteur du corridor en x_0 est la même pour tout $x_0 \in D(\mathcal{C})$.

Remarque 27. L'erreur d'approximation la plus couramment utilisée pour une fonction f est l'erreur absolue. Cela induit un corridor $\mathcal{C}$ tel que $l(x, y) = f(x, y) - \delta$ et $u(x, y) = f(x, y) + \delta$ avec $\delta > 0$, qui a donc une hauteur constante de 2δ en chaque point du corridor.

Proposition 28. *Soit un corridor* $\mathcal{C} = Corridor(u, l, D)$ *de hauteur constante* α*. Il existe une fonction* f *et une valeur* $\alpha_f \geq 0$ *tel que le corridor issu de* f *par l'erreur d'approximation absolue* α_f *est le corridor* $\mathcal{C}$*.*

Démonstration. Définissons $\alpha_f = \frac{\alpha}{2}$ et $f(x) = \frac{u(x)+l(x)}{2}$ pour tout $x \in D$. Alors, le corridor $\mathcal{C}_f = Corridor(f + \alpha_f, f - \alpha_f, D)$ est égal au corridor $\mathcal{C}$ puisque pour tout $x \in D$:

$$f(x) + \alpha_f = \frac{u(x) + l(x) + 2\alpha_f}{2} = \frac{2u(x)}{2} = u(x),$$

et l'égalité $f(x) - \alpha_f = l(x)$ se montre de manière équivalente. □

Définition 29 (niveau d'approximation intérieure η)**.** Soient $\mathcal{C}$ un corridor de domaine polygonal $D(\mathcal{C})$ dans $\mathbb{R}^2$ et $\mathcal{C}_{PWL}$ un corridor intérieur PWL de $\mathcal{C}$. On dit que $\mathcal{C}_{PWL}$ atteint un *niveau d'approximation intérieure de* $\eta \in [0, 1[$ *pour* $\mathcal{C}$ si le ratio de hauteur au point (x, y) entre le

corridor intérieur PWL et le corridor $\mathcal{C}$, calculé par $\frac{H_{\mathcal{C}_{PWL}}(x,y))}{H_{\mathcal{C}}(x,y))}$, est supérieur ou égal à η pour chaque (x, y) dans le domaine $D(\mathcal{C})$.

Proposition 30. *Soit $\mathcal{C}$ un corridor de domaine polygonal $D(\mathcal{C})$ dans $\mathbb{R}^2$ de hauteur constante, soit $\mathcal{C}_{PWL} = Corridor(\tilde{u}, \tilde{l}, D(\mathcal{C}))$ un corridor intérieur PWL de $\mathcal{C}$ tel que le domaine de chaque pièce de $\tilde{u}$ est aussi le domaine d'une pièce de $\tilde{l}$, et soit $\eta \in [0, 1[$ un niveau d'approximation intérieure. $\mathcal{C}_{PWL}$ atteint un niveau d'approximation intérieure de η pour $\mathcal{C}$ si et seulement si le ratio de hauteur de corridor $\frac{H_{\mathcal{C}_{PWL}}(x,y)}{H_{\mathcal{C}}(x,y)}$ est supérieur ou égal à η pour chaque sommet (x, y) du domaine des pièces de $\tilde{u}$ et de $\tilde{l}$.*

Démonstration. Soit un corridor $\mathcal{C}$ de hauteur constante. Alors pour chaque pièce du corridor intérieur PWL $\mathcal{C}_{PWL}$ de $\mathcal{C}$, le ratio minimum de hauteur est sur un point extrémal du domaine. En effet, la hauteur du corridor intérieur PWL $\mathcal{C}_{PWL}$ sur le domaine d'une pièce de $\tilde{u}$ (ou $\tilde{l}$) est une fonction linéaire en (x, y), donc il y a un sommet pour lequel l'extrémum est atteint. Or, ce ratio est borné inférieurement sur les sommets des pièces de $\tilde{u}$ et $\tilde{l}$ par hypothèse. □

Ce résultat est valable seulement pour les corridors de hauteur constante, c'est-à-dire issus d'une erreur d'approximation absolue. L'Algorithme 2 produit un corridor intérieur PWL qui atteint le niveau d'approximation intérieure η pour le corridor $\mathcal{C}$ de hauteur constante d'après la Proposition 30.

Algorithme 2 Raffinement du niveau d'approximation intérieure d'un corridor rectangulaire de hauteur constante

Entrée :
- le corridor $\mathcal{C} = Corridor(u, l, D)$ de hauteur constante à approximer
- le niveau d'approximation intérieure $\eta \in [0, 1[$

Sortie : les pièces $\mathcal{P}_{\tilde{u},\tilde{l}}$ des fonctions PWL $\tilde{u}$ et $\tilde{l}$ qui réalisent un corridor intérieur PWL de $\mathcal{C}$ sur D de niveau d'approximation intérieure η

1: $[p_{\tilde{u}}, p_{\tilde{l}}] \leftarrow$ EncadrementParGradient$(\mathcal{C}, D, 1, 1)$ ▷ *applique le lemme 23 pour u et l*
2: $Q \leftarrow [p_{\tilde{u}}, p_{\tilde{l}}]$ ▷ *Q est la liste de pièces à vérifier*
3: $\mathcal{P}_{\tilde{u},\tilde{l}} \leftarrow \emptyset$ ▷ *pièces du corridor PWL satisfaisant le niveau d'approximation intérieure η*
4: **tant que** $Q \neq \emptyset$ **faire**
5: $[p_{\tilde{u}}, p_{\tilde{l}}] \leftarrow RetirerDernierElement(Q)$
6: **si** niveau d'approximation intérieure sur les sommets des pièces p $< \eta$ **alors**
7: $Q \leftarrow Q \cup$ EncadrementParGradient$(\mathcal{C}, D(p), 2, 2)$ ▷ *raffine la pièce en 4 nouvelles pièces*
8: **sinon**
9: $\mathcal{P}_{\tilde{u},\tilde{l}} \leftarrow \mathcal{P}_{\tilde{u},\tilde{l}} \cup [p_{\tilde{u}}, p_{\tilde{l}}]$
10: **retourne** $\mathcal{P}_{\tilde{u},\tilde{l}}$

Le mot *pièce* décrit ici une pièce des fonctions $\tilde{u}$ et $\tilde{l}$. Ce n'est en aucun cas une pièce de la fonction PWL solution du $\mathbb{R}^2$-CFP construite par la trame de génération d'heuristiques.

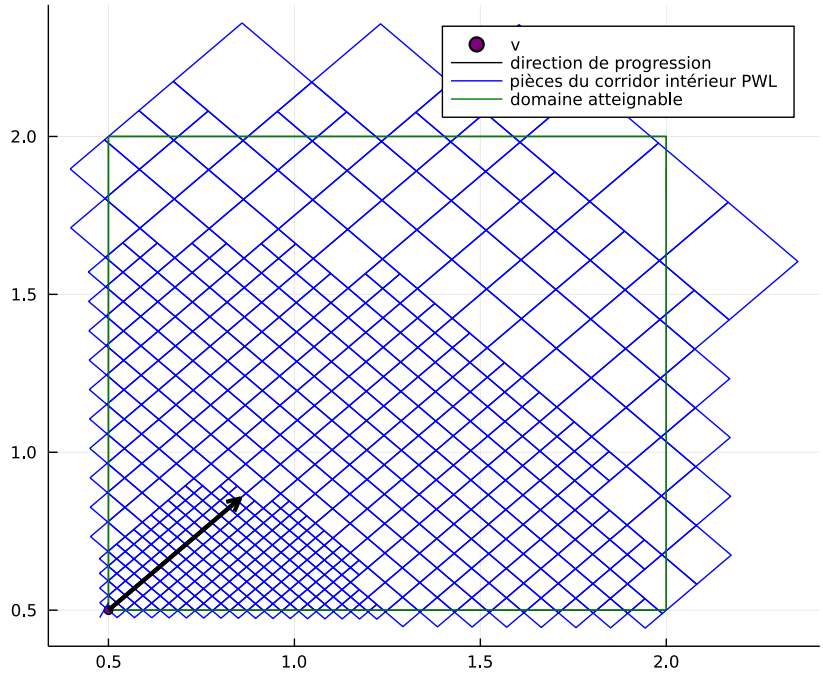


FIGURE 3.10: Un exemple de domaine des pièces du corridor intérieur PWL

Le domaine est un rectangle $\mathcal{D}$ contenant le domaine atteignable. Dans un premier temps, la procédure ENCADREMENTPARGRADIENT crée des fonctions PWL $\tilde{u}$ et $\tilde{l}$ à une pièce qui peuvent être invalides dans le sens où $\tilde{l} \not\leq \tilde{u}$. Ensuite, cette procédure raffine itérativement les pièces qui ne satisfont pas le niveau d'approximation intérieure η aux quatre sommets. Pour cela, elle divise ces pièces en quatre plus petites pièces rectangulaires de tailles égales (selon une grille 2x2), et revérifie le niveau d'approximation intérieure η de ces nouvelles pièces. Cela continue jusqu'à ce que chacune des pièces satisfasse le niveau d'approximation intérieure η du corridor $\mathcal{C}$. La Figure 3.10 montre un exemple de corridor intérieur PWL dans laquelle les domaines des pièces n'ont pas la même aire.

L'Algorithme 2 ne donne la garantie sur le niveau d'approximation intérieure η que pour des corridors à hauteur constante, et donc issus d'une erreur d'approximation absolue. Néanmoins, cet algorithme peut être utilisé sans garantie pour des corridors qui n'ont pas la propriété de hauteur constante. C'est pour cela que la trame utilise cet algorithme pour calculer le corridor intérieur PWL peu importe le corridor.

Dans cet algorithme, le paramètre η doit être ajusté en fonction de la qualité de l'approximation du corridor demandée. Pour produire une très bonne approximation du corridor $\mathcal{C}$, il suffit d'utiliser une valeur η proche de 1, mais dans ce cas le nombre de pièces formant $\mathcal{C}_{PWL}$ sera d'autant plus grand que η est proche de 1, tout comme le temps pour résoudre le problème $(MP'''_{d,\beta})$. Il y a donc un équilibre à trouver entre la qualité de l'approximation intérieure du

corridor et le temps de résolution de $(MP'''_{d,\beta})$, qui est étudié dans les expériences numériques Section 3.4.2.

3.3.2 Composant alternatif pour le calcul du score

Le composant *Aire* permet de calculer de manière simple le score d'une pièce. Nous présentons ici une version plus sophistiquée, basée sur la définition de la *variation totale* d'une fonction.

Il s'agit d'une mesure de la variation d'une fonction sur son domaine $\mathcal{D}$. La variation totale de la fonction g sur $\mathcal{D}$ est égale à l'intégrale $\int_{\mathcal{D}} ||\nabla g(x)||_2 dx$, où ∇ désigne le gradient. Le composant *Variation Totale des dérivées partielles* (VaT) approxime la somme des variations totales de toutes les dérivées partielles de u et l sur le domaine de la pièce p. Les fonctions u et l doivent être C^2.

Le composant VaT de la procédure Score est pensé comme une adaptation aux fonctions de deux variables du Théorème 1 de (Frenzen et al. [2010]) qui concerne les fonctions d'une seule variable. En effet, ce résultat explique que pour un corridor $\mathcal{C}$ de hauteur constante 2δ avec $D(\mathcal{C}) = [a, b]$, le nombre de pièces minimum $s(\delta)$ du $\mathbb{R}$-CFP satisfait une approximation asymptotique $s(\delta) \sim \frac{1}{4\delta} \int_a^b \sqrt{|u''(x, y)|} dx$. Concernant cette formule, nous remarquons d'une part que $u''(x, y) = l''(x, y)$ parce que c'est un corridor à hauteur constante, et d'autre part que l'intégrale calculée est la variation totale de la fonction u', et donc aussi celle de l'. Cette valeur augmente avec l'aire du domaine de la pièce et avec les variations du gradient de u et de l sur ce domaine. Ces deux relations correspondent à ce que l'on attend de la procédure Score puisque plus l'aire augmente ou plus les variations du gradient de u et l augmentent, plus il est difficile de trouver une pièce compatible avec le corridor. Comme pour un domaine de corridor dans $\mathbb{R}$ la variation totale permet de calculer une approximation du plus petit nombre de pièces possible, il est attendu que le composant VaT donne de meilleurs résultats que le composant *Aire* pour de petites valeurs de hauteur de corridor. Pour de grandes valeurs, la variation totale devrait être moins pertinente, donc dans ce cas le composant VaT pourrait être moins efficace que le composant *Aire*.

En raison de son coût algorithmique, nous ne calculons pas la variation totale de manière exacte, mais seulement une approximation de celle-ci. Le détail de la procédure est dans l'Annexe B.1.

3.3.3 Composant alternatif pour le calcul de la direction de progression

Le composant *ProMA* consiste à calculer deux pièces maximales de $\mathbb{R}$-corridor, puis à choisir comme direction de progression d le vecteur orthogonal à la ligne joignant les extrémités des deux pièces maximales. La Figure 3.11 indique les deux $\mathbb{R}$-corridor en bleu, le long des arêtes du

domaine $D(\mathcal{C})$ partant du sommet v, ainsi que les deux extrémités e_1 et e_2 des pièces maximales, la ligne reliant ces deux extrémités en rouge et le vecteur directeur orthogonal à cette ligne en noir.

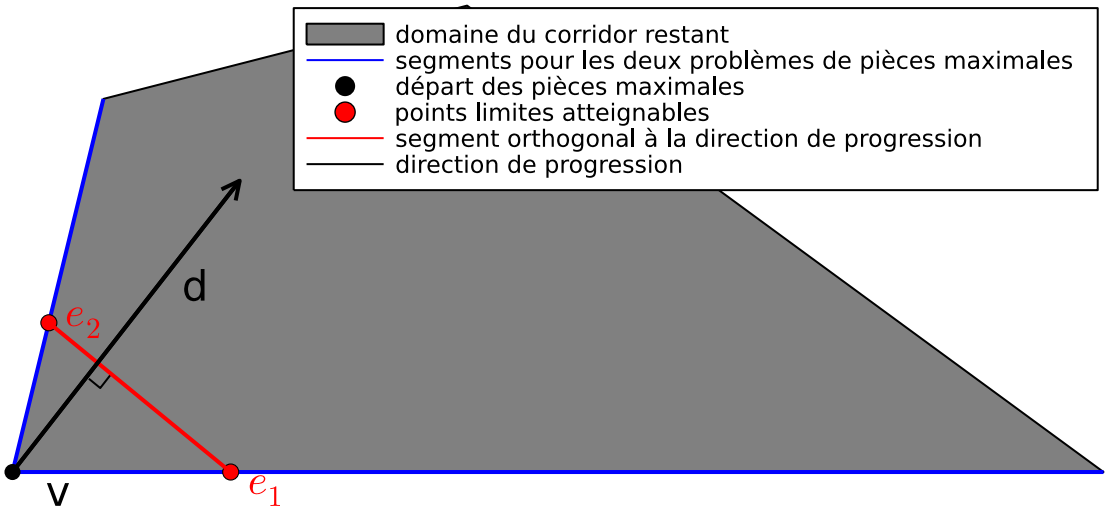
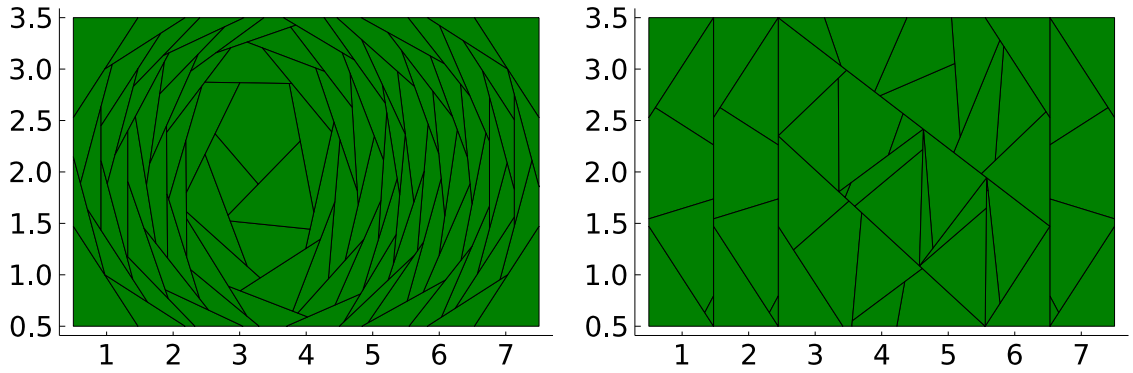


FIGURE 3.11: Construction de la direction de progression pour le composant *ProMA*

Le composant *DB* sert de référence, tandis que *ProMA* est pensé pour prendre en compte la "difficulté" d'une pièce à respecter les contraintes d'approximations le long des deux arêtes partant de v. Nous supposons qu'il sera plus efficace que le composant *DB* parce qu'il utilise des informations du corridor à couvrir qui sont plus proches du calcul de pièce maximale dans la direction d que ne le fait le composant *DB*.

3.3.4 Subdivision du corridor restant

Nous avons introduit au cours de MAJDOMAINE (Ligne 13) une procédure qui vérifie si le nouveau corridor restant $\mathcal{C}_r$ ne possède que des angles plus grands que 90°. Si c'est le cas, alors $\mathcal{C}_r$ est divisé en deux corridors qui seront traités séparément. En effet, appliquer nos procédures de calcul de pièces sur des corridors dont le domaine comprend des angles proches de 180° produit des pièces très aplaties, couvrant moins efficacement le domaine, et donc plus nombreuses. Ceci est illustré dans la Figure 3.12a où la fonction PWL obtenue sans subdivision a 97 pièces. L'application de notre procédure de subdivision préalable permet de pallier à ce défaut, comme illustré par la Figure 3.12b, où la fonction PWL n'a que 48 pièces. Cet exemple est issu d'une résolution du $\mathbb{R}^2$-CFP où le corridor est l'approximation absolue à $\delta = 0.25$ près de la fonction $f(x, y) = x^2 + y^2$ sur le domaine $[2, 8] \times [2, 4]$. Le paramètre η vaut 0.95, et les composants *VaT* et *DB* sont utilisés. Les segments longeant plusieurs pièces sont typiquement introduits lors de la subdivision.



(a) Résultat en l'absence de subdivision (b) Résultat lorsque la subdivision est appliquée

FIGURE 3.12: Effet de notre procédure de subdivision du corridor restant sur la linéarisation par morceaux de $f(x, y) = x^2 + y^2$ sur le domaine $[2, 8] \times [2, 4]$

3.4 Expériences numériques pour une erreur d'approximation absolue

Nous présentons les instances et les paramètres utilisés dans nos expériences dans la Section 3.4.1, puis nous analysons les différents composants de l'Algorithme 1 et formons des heuristiques à partir des meilleures combinaisons de composants dans la Section 3.4.2. Finalement, nous comparons nos heuristiques à l'état de l'art dans la Section 3.4.3. Nos expériences numériques se focalisent sur l'erreur d'approximation absolue car les méthodes de l'état de l'art sont implémentées uniquement pour cette erreur.

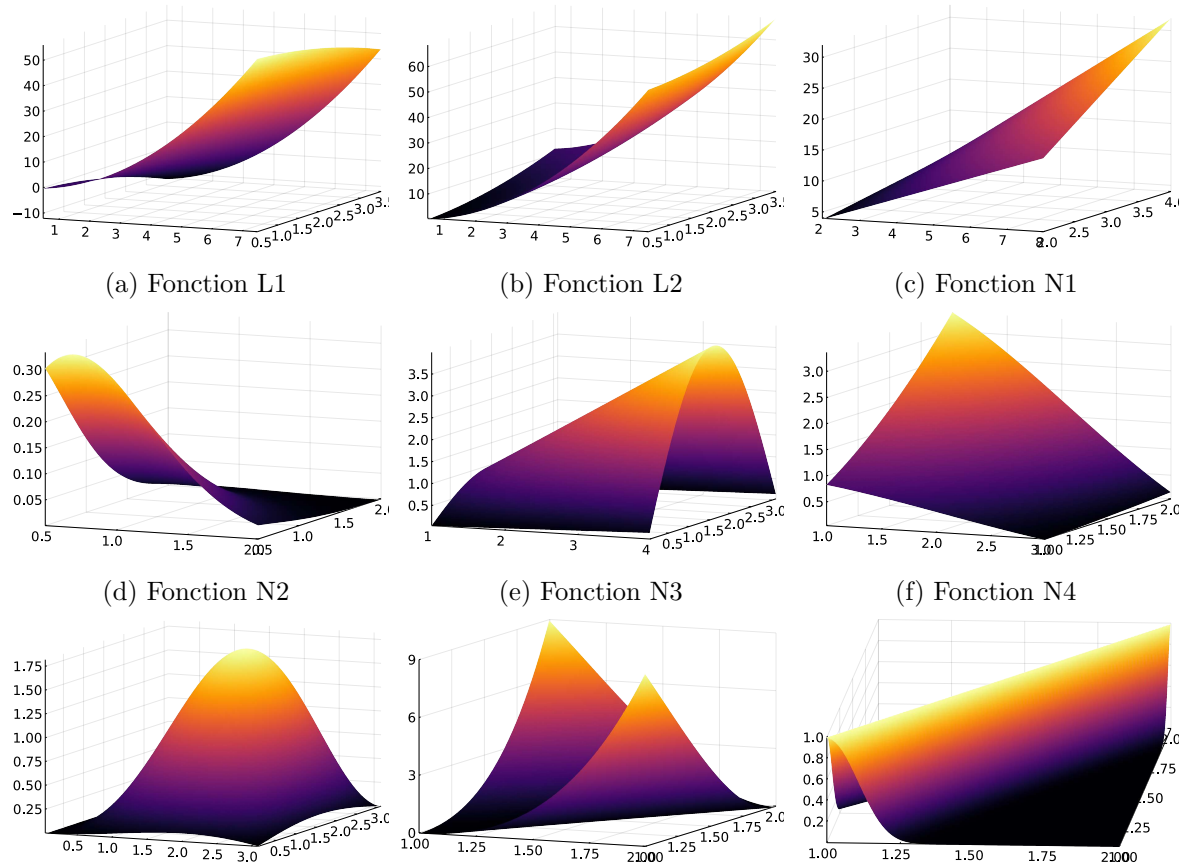
3.4.1 Instances et paramètres

Instances. Nous utilisons le jeu d'instances de [Rebennack et Kallrath, 2015a], qui consiste en 45 instances obtenues à partir de neuf fonctions et de cinq erreurs d'approximation absolues, différentes pour chaque fonction. L'expression et le domaine de définition de ces fonctions sont décrits dans la Table 3.1, et la Figure 3.13 exhibe les neuf fonctions du jeu d'instances.

ref	expression	domaine	ref	expression	domaine
L1	$x^2 - y^2$	$[0.5, 7.5] \times [0.5, 3.5]$	N3	$xsin(y)$	$[1, 4] \times [0.05, 3.1]$
L2	$x^2 + y^2$	$[0.5, 7.5] \times [0.5, 3.5]$	N4	$\frac{sin(x)}{x} y^2$	$[1, 3] \times [1, 2]$
			N5	$xsin(x)sin(y)$	$[0.05, 3.1] \times [0.05, 3.1]$
N1	xy	$[2, 8] \times [2, 4]$	N6	$(x^2 - y^2)^2$	$[1, 2] \times [1, 2]$
N2	$xe^{-x^2-y^2}$	$[0.5, 2] \times [0.5, 2]$	N7	$e^{-10(x^2-y^2)^2}$	$[1, 2] \times [1, 2]$

TABLE 3.1: Expression et domaine des fonctions du jeu d'instances

Les deux premières fonctions du jeu d'instances sont des fonctions linéairement séparables



(a) Fonction L1 (b) Fonction L2 (c) Fonction N1

(d) Fonction N2 (e) Fonction N3 (f) Fonction N4

(g) Fonction N5 (h) Fonction N6 (i) Fonction N7

FIGURE 3.13: Toutes les fonctions du jeu d'instances de [Rebennack et Kallrath, 2015a]

que l'on appelle L1 et L2 tandis que les sept autres sont non linéairement séparables et nous les appelons N1, ..., N7.

Paramètres. Le paramètre N_{VaT} utilisé dans les expériences est fixé à 200.

Nom des heuristiques. Le nom d'une heuristique basée sur la trame de l'Algorithme 1 est donné par la concaténation des composants utilisés dans l'ordre *ProMA*/*DB* puis *VaT*/*Aire* puis 100η. Ainsi, l'heuristique utilisant les composants *VaT*, *DB* et $\eta = 0.95$ est nommée *DB_VaT_*95.

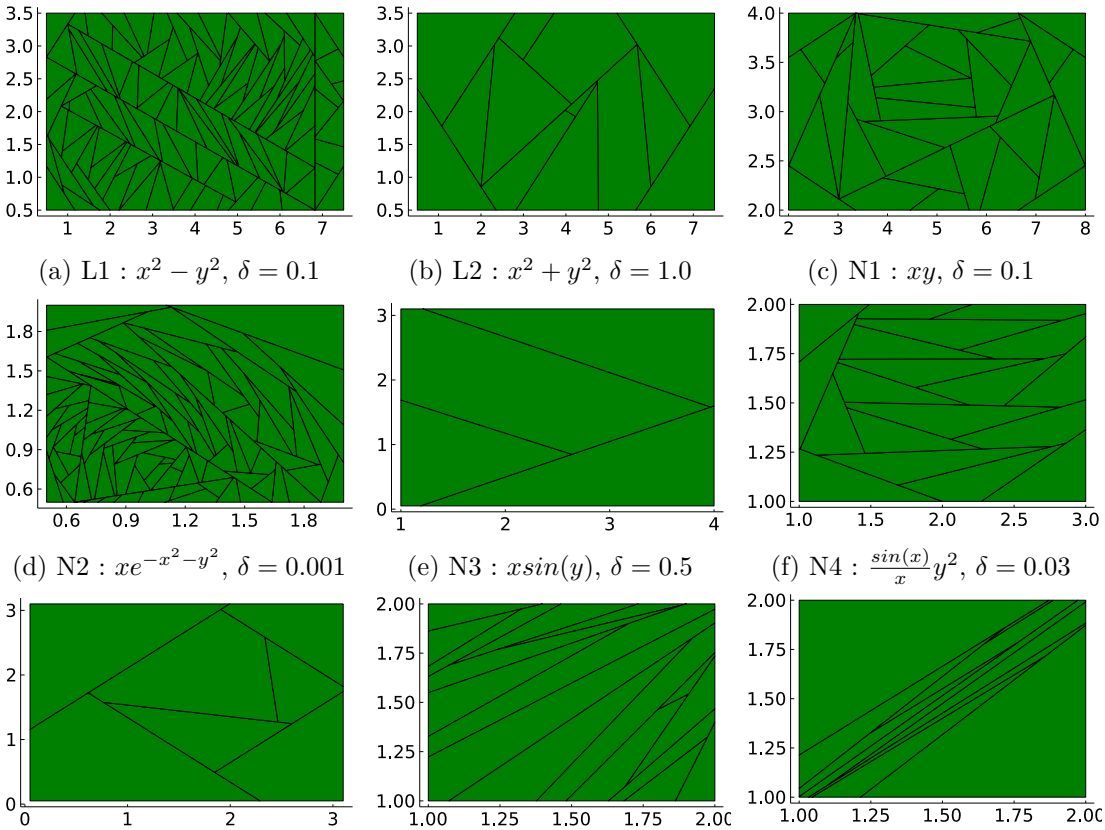
Récapitulatif des composants. La Table 3.2 résume les composants construits pour l'Algorithme 1 et les sections où ils sont décrits.

Procédure	Composants	Description
CalculDirectionProgression	*DB*, *ProMA*	Sections 3.2.1 et 3.3.3
Score	*Aire*, *VaT*	Sections 3.2.5 et 3.3.2
ConstructionCorridorInterieurPWL	$\eta \in [0, 1[$	Section 3.3.1

Table 3.2: Récapitulatif des composants de l'Algorithme 1

3.4.2 Analyse numérique des composants des heuristiques

Dans cette section, nous présentons des exemples de solutions obtenues par une heuristique issue de la trame, puis nous analysons numériquement l'effet de chaque composant de l'Algorithme 1 en comparant les uns aux autres les composants d'une même procédure.



(a) L1 : $x^2 - y^2$, $\delta = 0.1$ (b) L2 : $x^2 + y^2$, $\delta = 1.0$ (c) N1 : xy, $\delta = 0.1$

(d) N2 : $xe^{-x^2-y^2}$, $\delta = 0.001$ (e) N3 : $xsin(y)$, $\delta = 0.5$ (f) N4 : $\frac{sin(x)}{x}y^2$, $\delta = 0.03$

(g) N5 : $xsin(x)sin(y)$, $\delta = 0.5$ (h) N6 : $(x^2 - y^2)^2$, $\delta = 0.1$ (i) N7 : $e^{-10(x^2-y^2)^2}$, $\delta = 0.05$

Figure 3.14: Exemples de solutions obtenues avec la méthode *ProMA_VaT_*95

Exemple de solutions. La Figure 3.14 montre la division en domaines des pièces de la fonction PWL pour 9 solutions obtenues par l'heuristique *ProMA_VaT_*95.

	APP60	*TSM*
moyenne géométrique du nombre de pièces	14.5	**10.9**
moyenne géométrique du temps de calcul (s)	**7.2**	21.4
solution avec le plus petit nombre de pièces	28.9%	**97.8%**

TABLE 3.3: Agrégation des résultats de quatre heuristiques utilisant les composants *APP60* et *TSM*

Évaluation de l'intérêt de calculer une pièce par sommet du corridor restant. Une pièce candidate par sommet du domaine du corridor restant est calculée à chaque itération de la boucle tant que (Ligne 3) de l'Algorithme 1. La procédure SCORE sélectionne ensuite la pièce qui semble la plus intéressante parmi ces pièces. Nous avons voulu évaluer l'intérêt de cette approche en la comparant avec une approche qui se contenterait de calculer une seule pièce pour l'ensemble du domaine. Pour cela, nous proposons l'ajout d'un composant à l'Algorithme 1 pour ne calculer qu'une seule pièce, toujours avec les Lignes 7-9, mais seulement avec le sommet v correspondant à l'angle du domaine du corridor restant le plus proche de 60°. Cette valeur est choisie pour former a priori des pièces dont le domaine est plus "équilibré", c'est-à-dire plus proche du triangle équilatéral s'il s'agissait de triangles. Pour distinguer les heuristiques utilisant ce composant, nous préfixons à leur nom le terme *APP60* pour *angle le plus près de 60°*. Si ce préfixe n'est pas présent, c'est l'Algorithme 1 non modifié qui est utilisé, et nous appelons ce composant *TSM* pour *tout sommet*.

Pour évaluer l'intérêt du composant *TSM*, nous comparons 2 heuristiques utilisant le composant *APP60*, et 2 heuristiques utilisant *TSM* avec les mêmes autres composants : il s'agit de *APP60_DB_Aire_*95, *APP60_DB_Aire_*99, *DB_Aire_*95 et *DB_Aire_*99. Notons qu'avec *APP60*, le composant utilisé pour sélectionner la meilleure pièce n'a pas d'importance puisqu'il n'y a qu'un choix. Les composants *DB* et *Aire* communs à ces heuristiques, permettent une comparaison sur une construction des pièces simple, pour interpréter plus facilement les résultats.

La Table 3.3 présente des statistiques sur les résultats agrégés des heuristiques utilisant *APP60*, et celles utilisant *TSM*. Elle présente les moyennes géométriques des temps de calcul et du nombre de pièces, ainsi que la proportion de solutions qui comporte au plus autant de pièces que la solution de la même instance par l'heuristique avec le composant complémentaire dans *APP60*/*TSM*. On peut voir que les heuristiques utilisant *TSM* produisent des fonctions PWL avec 25% moins de pièces, mais nécessitent un temps de calcul trois fois plus long. De plus, 97.8% des solutions des heuristiques utilisant *TSM* ont le plus petit nombre de pièces, alors que ce n'est le cas que pour 28.9% des solutions utilisant *APP60*. Nous en concluons que le temps de calcul disponible pour approximer une fonction non linéaire est une caractéristique importante pour déterminer si le composant *APP60* doit être utilisé à la place de *TSM*.

η	0.5	0.7	0.9	0.95	0.99
moyenne géométrique du nombre de pièces	21.1	15.9	12.0	10.8	**9.7**
moyenne géométrique du temps de calcul (s)	22.0	17.4	**14.3**	17.0	36.4
solution avec le plus petit nombre de pièces	2.2%	6.1%	21.1%	39.4%	**94.4%**

TABLE 3.4: Agrégation des résultats de 20 heuristiques pour l'analyse de l'influence du ratio d'approximation intérieure η

	DB	*ProMA*
moyenne géométrique du nombre de pièces	10.7	**9.8**
moyenne géométrique du temps de calcul (s)	**20.8**	29.6
solution avec le plus petit nombre de pièces	61.7%	**77.8%**

TABLE 3.5: Agrégation des résultats de huit heuristiques pour la comparaison des composants *DB* et *ProMA* qui construisent la direction de progression

Évaluation de l'impact du ratio d'approximation intérieure η. Analysons l'influence du ratio d'approximation intérieur η sur le temps de calcul et le nombre de pièces d'une fonction PWL. Il est attendu qu'augmenter η diminue le nombre de pièces puisque le corridor intérieur PWL sera approximé avec une plus grande précision. Pour cette analyse, nous utilisons les cinq valeurs $\eta = 0.5, 0.7, 0.9, 0.95, 0.99$ et les deux couples de composants *ProMA*/*DB* et *VaT*/*Aire* pour former un total de 20 heuristiques.

Nous comparons les heuristiques avec des valeurs différentes de η et les mêmes deux autres composants. Les résultats sont présentés dans la Table 3.4. Ils montrent que lorsque η augmente, le nombre de pièces diminue de manière importante : 54% de pièces en moins pour $\eta = 0.99$ par rapport à $\eta = 0.5$. De plus, les heuristiques utilisant $\eta = 0.99$ ont presque toujours la fonction PWL avec le moins de pièces. Cette proportion décroît très rapidement lorsque η décroît. Concernant le temps de calcul moyen, nous constatons qu'il est le plus bas pour $\eta = 0.9$. Ce résultat s'expliquerait par un équilibre entre l'influence de η sur le temps de calcul d'une pièce et le nombre de pièces total à calculer. En conclusion, la valeur de η a une grande influence sur le temps de calcul et le nombre de pièces de la fonction PWL solution. Pour le jeu d'instance utilisé, une valeur de η proche de 0.9 est la plus adaptée si le temps de calcul est le principal critère, et 0.99 sinon.

Évaluation de l'impact de la direction de progression. Analysons la méthode de calcul de la direction de progression par les composants *DB* ou *ProMA*. Pour cela, comparons les résultats de huit heuristiques utilisant les composants choisis parmi les trois couples *ProMA*/*DB*, *VaT*/*Aire* et $\eta = 0.95, 0.99$.

La Table 3.5 montre les statistiques de ces expériences. Les heuristiques utilisant *ProMA* produisent en moyenne des solutions avec un peu moins de pièces mais qui demandent sensi-

	Aire	*VaT*
moyenne géométrique du nombre de pièces	10.4	**10.1**
moyenne géométrique du temps de calcul (s)	25.9	**23.9**
solution avec le plus petit nombre de pièces	80.6%	**90.0%**

TABLE 3.6: Agrégation des résultats de huit heuristiques pour la comparaison des composants *Aire* et *VaT* qui calculent le score d'une pièce

	ProMA_VaT	*ProMA_Aire*	*DB_VaT*	*DB_Aire*
moyenne géométrique du nombre de pièces	**10.2**	10.4	11.2	11.5
moyenne géométrique du temps de calcul (s)	18.1	24.9	**13.4**	13.8
solution avec le plus petit nombre de pièces	**66.7%**	60.0%	60.0%	48.9%

TABLE 3.7: Agrégation des résultats de quatre heuristiques pour la détermination de l'heuristique produisant des solutions avec le moins de pièces pour $\eta = 0.95$

blement plus de temps de calcul. L'utilisation du composant *ProMA* semble donc produire des solutions de légèrement meilleure qualité au prix d'un temps de calcul bien plus élevé.

Évaluation de l'impact du score des pièces. Analysons la méthode pour calculer le score des pièces avec les composants *Aire* ou *VaT*. Pour cela, nous utilisons huit heuristiques formées à partir des couples de composants *ProMA*/*DB*, *VaT*/*Aire* et $\eta = 0.95, 0.99$.

La Table 3.6 présente les statistiques de cette analyse. Le nombre de pièces et la proportion de solutions avec le plus petit nombre de pièces issues des heuristiques utilisant les deux composants sont sensiblement les mêmes, avec un léger avantage pour le composant *VaT*. Cet avantage est aussi présent sur le temps de calcul.

Identification des heuristiques produisant le moins de pièces. Nous déterminons les heuristiques issues de l'Algorithme 1 produisant des fonctions PWL avec le moins de pièces pour $\eta = 0.95$ et $\eta = 0.99$. Pour ce faire, nous comparons les résultats de quatre heuristiques utilisant les composants *ProMA*/*DB* et *VaT*/*Aire*. Les meilleures heuristiques seront ensuite comparées aux méthodes de l'état de l'art dans la Section 3.4.3. Le critère de sélection de ces heuristiques est le plus petit nombre de pièces des solutions et ne prend pas en compte le temps de calcul. En effet, la comparaison des méthodes de l'état de l'art est basée sur le nombre de pièces et non sur le temps de calcul, puisque les méthodes de l'état de l'art ont des temps d'exécution très différents, allant de moins d'une seconde à plusieurs heures.

Les résultats pour $\eta = 0.95$ (resp. 0.99) sont donnés dans la Table 3.7 (resp. 3.8). Ils montrent que l'heuristique *ProMA_VaT_*95 (resp. *ProMA_VaT_*99) est celle produisant les solutions avec en moyenne le moins de pièces et avec la plus grande proportion de meilleures solutions. Ceci dit, l'écart avec *ProMA_Aire_*95 (resp. *ProMA_Aire_*99) n'est pas important, en conséquence

	ProMA_VaT	*ProMA_Aire*	*DB_VaT*	*DB_Aire*
moyenne géométrique du nombre de pièces	**9.2**	9.3	10.1	10.3
moyenne géométrique du temps de calcul (s)	43.3	39.5	**30.9**	33.1
solution avec le plus petit nombre de pièces	**68.9%**	64.4%	66.7%	55.6%

TABLE 3.8: Agrégation des résultats de quatre heuristiques pour la détermination de l'heuristique produisant des solutions avec le moins de pièces pour $\eta = 0.99$

Paramètres	trame, LinA1D	RK2D	KL21	KL23-LP, KL23-LPrelaxed
langage de programmation	Julia 1.6.2	-	Python 3.6.10	Python 3.6.10
langage de modélisation	JuMP 0.21.10	GAMS 23.6	Pyomo 5.6.4	Pyomo 5.6.4
solveur	Gurobi 9.1.1	LindoGlobal 23.6.5	CPLEX 12.8 COUENNE 0.5.8/Baron	CPLEX 22.1.0
CPU	i5-10310U de 1.7GHz	i7 de 2.93GHz	3.6GHz	3.6GHz
RAM	32Go	12Go	32Go	32Go
remarques	limité à un seul cœur	-	Baron utilisé seulement pour la fonction $N7$	10 cœurs pour résoudre les problèmes MILP/LP

TABLE 3.9: Matériel utilisé pour les expériences numériques

il est possible que ce résultat soit dépendant des instances utilisées.

3.4.3 Comparaison à l'état de l'art pour une erreur d'approximation absolue

Nous décrivons maintenant les expériences numériques ayant permis de comparer notre trame de génération d'heuristiques aux autres méthodes de linéarisation par morceaux de l'état de l'art. Pour cela, nous utilisons les heuristiques *ProMA_VaT_*95 et *ProMA_VaT_*99 issues de l'Algorithme 1 qui ont été identifiées comme donnant des solutions avec le moins de pièces dans la Section 3.4.2. Pour simplifier les notations, nous les appelons dans cette section *DN95* et *DN99* respectivement.

Les heuristiques de l'état de l'art sont RK1D, RK2D, KL21, KL23-LP et KL23-LPrelaxed, décrites dans la Section 1.3.3. La méthode RK1D est cependant remplacée par une nouvelle méthode baptisée *LinA1D*. Elle est basée sur la même décomposition que RK1D, mais utilise la librairie LinA de ([Codsi et al., 2021]) pour calculer les approximations univariées. LinA1D a de meilleures performances que RK1D puisque la librairie LinA calcule des approximations PWL optimales parmi les fonctions PWL non nécessairement continues, tandis que RK1D construit des approximations PWL optimales parmi les fonctions PWL continues. Ainsi, RK1D n'est pas présente dans nos comparatifs.

La Table 3.9 montre le matériel utilisé par chaque méthode. La méthode "trame" fait référence à nos deux heuristiques construites à partir de l'Algorithme 1. Les méthodes comparées ici ont été exécutées sur des ordinateurs différents, et les informations sur le matériel utilisé ne sont pas toujours complètes. Nous considérons cependant que ce n'est pas un problème, parce que les conclusions de notre analyse ne dépendent que marginalement du temps de calcul. Concernant la limite de temps des méthodes, les heuristiques LinA1D et RK2D n'en ont pas mais ne dépassent pas 3600 secondes de calcul, tandis que KL21, KL23-LP et KL23-LPrelaxed

ont une limite de temps de 3600 secondes par résolution de problèmes MILP ou LP. Ainsi, nous choisissons d'allouer 3600 secondes aux heuristiques DN95 et DN99 pour résoudre chaque problème LP de la procédure CalculPieceMaximale.

La Table 3.10 présente le nombre de pièces des solutions pour chaque instance et pour chaque méthode. Les deux premières colonnes indiquent l'instance par la référence de la fonction et le paramètre δ définissant l'erreur absolue. Les autres colonnes montrent le nombre de pièces obtenu par une méthode particulière. Les entiers en gras signalent une solution avec le plus petit nombre de pièces obtenu parmi les différentes méthodes pour cette instance. La mention "TL" indique qu'aucune solution n'a été trouvée lorsque la limite de temps a été atteinte.

La Table 3.11 résume les résultats de la Table 3.10 en affichant le nombre de fois que chaque méthode obtient le plus petit nombre de pièces sur le sous-groupe d'instances avec des fonctions linéairement séparables et celui avec des fonctions non linéairement séparables. Cette répartition des instances en deux sous-groupes permet une analyse plus fine des résultats. Concernant les instances avec des fonctions linéairement séparables, c'est l'heuristique LinA1D qui produit le plus souvent la solution avec le moins de pièces. Pour les instances avec des fonctions non linéairement séparables, c'est notre heuristique DN99 qui obtient le plus souvent les meilleurs résultats, suivie par KL21, KL23-LP et KL23-LPrelaxed. Les instances avec des fonctions linéairement séparables sont les cas d'applications les plus naturels pour la méthode LinA1D puisqu'il n'y a pas besoin de transformation pour obtenir une expression linéairement séparable. Cela peut expliquer pourquoi LinA1D performe mieux dans ce sous-groupe d'instances que dans l'autre, comparativement aux autres méthodes. En ce qui concerne la méthode KL21, elle produit sur l'ensemble des instances le plus petit nombre de pièces 19 fois mais atteint la limite de temps 21 fois, ce qui montre que cette heuristique est plutôt adaptée aux petites instances. En effet, les problèmes MILP de taille croissante que doit résoudre KL21 causent une augmentation rapide du temps de calcul en fonction du nombre de pièces de la solution. Même si les méthodes KL23-LP et KL23-LPrelaxed, les méthodes dérivées de KL21, résolvent la plupart des instances, elles ne produisent pas de solutions de suffisamment bonne qualité pour battre DN99 dans le sous-groupe d'instances avec des fonctions non linéairement séparables.

La Table B.1 en annexe donne les temps de calcul pour les résultats présentés dans la Table 3.10. Le temps de calcul, le matériel et les codes sont différents suivant les algorithmes, donc comparer les temps de calculs nécessaires pour chaque heuristique est biaisé. Par contre, nous pouvons considérer l'ordre de grandeur des temps de calcul par heuristique, avec LinA1D prenant systématiquement moins d'une seconde, RK2D, DN95 et DN99 quelques secondes à quelques minutes, tandis que KL21, KL23-LP et KL23-LPrelaxed prennent quelques secondes à quelques heures.

ref	δ	DN95	DN99	RK2D	LinA1D	KL21	KL23-LP	KL23-LPrelaxed
	1.5	7	**6**	16	**6**	**6**	**6**	**6**
	1.0	12	9	20	8	**6**	10	8
L1	0.5	20	18	48	**15**	TL	18	16
	0.25	40	37	80	**21**	TL	36	32
	0.1	106	93	224	**60**	TL	98	83
	1.5	7	**6**	24	**6**	TL	7	**6**
	1.0	13	9	28	8	**6**	11	7
L2	0.5	24	24	84	15	TL	20	**14**
	0.25	41	42	121	**21**	TL	TL	25
	0.1	126	119	351	**60**	TL	TL	TL
	1.0	**3**	**3**	4	6	4	6	4
	0.5	7	**6**	12	8	**6**	10	10
N1	0.25	16	14	20	18	**12**	16	14
	0.1	32	**28**	59	45	TL	36	30
	0.05	67	59	94	91	TL	64	**57**
	0.1	**1**	**1**	2	4	**1**	**1**	**1**
	0.05	**2**	**2**	6	9	**2**	3	4
N2	0.03	**4**	**4**	10	12	**4**	6	**4**
	0.01	11	**10**	31	30	TL	19	18
	0.001	100	**88**	350	238	TL	TL	181
	1.0	3	3	5	12	TL	**2**	3
	0.5	4	4	8	16	TL	6	**3**
N3	0.25	7	**6**	16	22	8	10	8
	0.1	17	**15**	44	68	TL	**15**	19
	0.05	34	30	85	120	TL	36	**27**
	0.5	**2**	**2**	**2**	3	**2**	**2**	**2**
	0.25	**3**	**3**	4	8	4	4	4
N4	0.1	7	**6**	9	21	**6**	**6**	**6**
	0.05	13	**11**	23	27	12	14	14
	0.03	19	**17**	40	44	TL	25	26
	1.0	2	**1**	6	20	**1**	**1**	**1**
	0.5	7	5	6	42	**4**	**4**	**4**
N5	0.25	11	14	21	72	**5**	13	6
	0.1	26	24	96	168	TL	24	**21**
	0.05	56	49	274	340	TL	**43**	44
	1.0	**3**	**3**	6	9	**3**	**3**	**3**
	0.5	5	5	9	9	**4**	**4**	**4**
N6	0.25	8	7	12	16	**6**	14	10
	0.1	17	**15**	40	49	TL	22	16
	0.05	37	**28**	87	81	TL	50	38
	1.0	**1**	**1**	2	4	**1**	**1**	**1**
	0.5	**2**	**2**	4	4	**2**	**2**	**2**
N7	0.25	4	**3**	6	9	4	19	12
	0.1	8	7	84	16	**5**	19	8
	0.05	**10**	11	86	36	TL	19	14

TABLE 3.10: Comparaison du nombre de pièces des solutions des heuristiques de l'état de l'art utilisant une erreur d'approximation absolue δ

	DN95	DN99	RK2D	LinA1D	KL21	KL23-LP	KL23-LPrelaxed
# Meilleur sur L1-L2	0	2	0	**7**	3	1	3
# Meilleur sur N1-N7	10	**22**	1	0	16	12	14

TABLE 3.11: Nombre d'instances avec le plus petit nombre de pièces pour chaque heuristique sur les deux sous-groupes d'instances

3.4.4 Comparaison avec l'algorithme crossing swords dédié à la fonction $f(x, y) = xy$

Nous complétons ici les résultats de la comparaison numérique de la trame de génération d'heuristiques avec l'état de l'art du $\mathbb{R}^2$-CFP dans le cas d'un corridor issu d'une erreur d'approximation absolue. En effet, l'algorithme crossing swords [Bärmann et al., 2023] est construit spécifiquement pour l'approximation de la fonction $N1(x, y) = xy$ avec une erreur d'approximation absolue et produit une fonction PWL par interpolation, donc la fonction PWL obtenue est continue et son évaluation sur le sommet (x, y) d'une pièce est égale à l'évaluation de la fonction xy. C'est un algorithme d'approximation de la restriction du $\mathbb{R}^2$-CFP à des fonctions PWL obtenues par interpolation de facteur $\frac{\sqrt{5}}{2} \approx 1.118$ pour des valeurs $\delta = \frac{1}{16i}$, $i \in \mathbb{N}$ pour d'autres valeurs de δ, le facteur est de $\frac{\sqrt{5}}{2} + 4\sqrt{5}\delta$. Les résultats de [Bärmann et al., 2023] sont obtenus sur le domaine $[0, 6] \times [0, 2]$ alors que nous les obtenons pour $[2, 8] \times [2, 4]$ tel que défini dans [Rebennack et Kallrath, 2015a]. Nous montrons que le résultat pour l'instance de $\mathbb{R}^2$-CFP avec le domaine $[0, 6] \times [0, 2]$ permet de construire une fonction PWL avec autant de pièces solution de l'instance avec le domaine $[2, 8] \times [2, 4]$.

Proposition 31. *Soient les corridors* $\mathcal{C}_1 = Corridor(xy + \delta, xy - \delta, [2, 8] \times [2, 4])$ *et* $\mathcal{C}_2 = Corridor(xy + \delta, xy - \delta, [0, 6] \times [0, 2])$. *Soient l'application affine* $T_{aff} : (x, y, z) \to (x - 2, y - 2, z - 2x - 2y + 4)$ *et une fonction PWL* g *de domaine* $[2, 8] \times [2, 4]$. *Si* g *est dans le corridor* $\mathcal{C}_1$, *alors* $T_{aff}(g)$ *est une fonction PWL de domaine* $[0, 6] \times [0, 2]$ *incluse dans le corridor* $\mathcal{C} = Corridor(xy + \delta, xy - \delta, [0, 6] \times [0, 2])$ *avec autant de pièces que* g.

Démonstration. Voici le plan de la preuve. Nous montrons d'abord que l'image du corridor $\mathcal{C}_1 = Corridor(xy + \delta, xy - \delta, [2, 8] \times [2, 4])$ par T_{aff} est le corridor $\mathcal{C}_2 = Corridor(xy + \delta, xy - \delta, [0, 6] \times [0, 2])$. Puis nous montrons que l'image par T_{aff} d'un ensemble dont la projection sur les deux premières coordonnées est égal à $[2, 8] \times [2, 4]$ donne un ensemble dont la projection sur les deux premières coordonnées est égal à $[0, 6] \times [0, 2]$. Finalement, l'image d'un polytope par une application affine est un polytope, donc l'image d'une pièce de g par T_{aff} est une pièce, ce qui conclut la preuve.

T_{aff} est constituée du changement de base $T : (x, y, z) \to (x, y, z-2x-2y)$ et de la translation de vecteur $(-2, -2, 4)$. Montrons que $T_{aff}(\mathcal{C}_1) = \mathcal{C}_2$. Soit $(x, y, z) \in \mathcal{C}_1$. Cela équivaut à (x, y, z) est de la forme $(x, y, xy + \alpha)$ avec $\alpha \in [-\delta, \delta]$. On a $T_{aff}(x, y, xy + \alpha) = (x - 2, y - 2, xy -$

δ	1.0	0.5	0.25	0.1	0.05
Crossing swords	**3**	7	**12**	31	60
DN99	**3**	**6**	14	**28**	**59**

TABLE 3.12: Comparaison du nombre de pièces obtenue par DN99 et crossing swords sur la fonction N1

$2x - 2y + 4 + \alpha) = (x - 2, y - 2, (x - 2)(y - 2) + \alpha) = (u, v, uv + \alpha) \in \mathcal{C}_2$ pour $(u, v) = (x-2, y-2)$, $(u, v) \in [0, 6] \times [0, 2]$. Comme T est une base, il existe une base T^{-1} et une fonction affine T_{aff}^{-1} réalisant l'opération inverse de T_{aff}. L'inclusion $T_{aff}^{-1}(\mathcal{C}_2) \subset \mathcal{C}_1$ se montre de manière équivalente. Ensuite, soit un ensemble $E \in \mathcal{C}_1$ tel que $\{(x, y)|(x, y, z) \in E\} = [2, 8] \times [2, 4]$. $T_{aff}(E) = \{(x-2, y-2, z-2x-2y+4)|(x, y, z) \in E\}$, donc la projection sur les deux premières coordonnées de $T_{aff}(E)$ est $\{(x - 2, y - 2)|(x, y, z) \in E\}$ qui contient $[0, 6] \times [0, 2]$. □

Donc nous pouvons comparer le nombre de pièces des solutions de crossing swords avec DN99. La Table 3.12 présente ces résultats. Nous y constatons que l'heuristique DN99 produit une solution avec strictement moins de pièces 3 fois sur 5 tandis que crossing swords ne bat DN99 qu'une seule fois. En théorie, l'algorithme crossing swords a pour avantage que c'est un algorithme dédié à la fonction $f(x, y) = xy$, quand DN99 est générique. À l'inverse, DN99 a pour avantage d'autoriser des domaines de pièces qui sont des polygones convexes ainsi que des fonctions PWL non nécessairement continues quand crossing swords utilise des triangles et une interpolation. Les résultats numériques en faveur de DN99 confortent l'idée que chercher à produire des fonctions PWL aussi générales que possible est très important pour obtenir un petit nombre de pièces.

3.5 Conclusion

Nous avons conçu et présenté une trame de génération d'algorithmes de linéarisation par morceaux pour la résolution du $\mathbb{R}^2$-CFP. Parmi les éléments clefs de notre trame et les contributions scientifiques associées, citons notamment la construction de solutions réalisables d'un problème GSIP à l'aide de la résolution d'une série de problèmes LP, ou encore la proposition d'une procédure SCORE pour évaluer à quel point une pièce couvre un domaine "difficile à couvrir". Finalement, les expériences numériques sur un jeu d'instances de la littérature pour une erreur d'approximation absolue montrent que dans le cas des instances utilisant des fonctions linéairement séparables, c'est la méthode LinA1D qui est la meilleure. Dans le cas des instances utilisant des fonctions non linéairement séparables, notre trame de génération d'heuristiques permet de produire des méthodes qui surpassent l'état de l'art. Nous souhaitons aussi souligner que cette trame est valide pour construire des fonctions PWL de plus de deux variables. Parmi les pistes pour continuer ce travail, nous proposons la diminution du temps de calcul des heu-

ristiques de la trame, la création de nouveaux composants ou l'application de ces heuristiques à la résolution de problèmes MINLP pour montrer l'intérêt du $\mathbb{R}^2$-CFP en pratique.

Chapitre 4

Bornes inférieures pour le $\mathbb{R}^2$-CFP

Sommaire

Dans le chapitre précédent, nous avons décrit des heuristiques qui battent l'état de l'art sur les fonctions non linéairement séparables. Il existe notamment des instances pour lesquelles notre meilleure méthode trouve une solution avec deux fois moins de pièces que la meilleure méthode qui n'est pas issue de notre trame de génération d'heuristiques. Ainsi, il paraît intéressant d'évaluer l'écart à l'optimum des solutions retournées par des heuristiques. Pour cette raison, ce chapitre se concentre sur le calcul de bornes inférieures au $\mathbb{R}^2$-CFP.

Le plan du chapitre est le suivant. Dans la Section 4.1, nous introduisons une famille d'inégalités valides du $\mathbb{R}^2$-CFP issue d'un problème de coloration d'hypergraphe ainsi qu'une relaxation les utilisant, basée sur la discrétisation du domaine. Puis, nous décrivons dans la Section 4.2 quatre relaxations successives du $\mathbb{R}^2$-CFP et un algorithme pour résoudre chacune de ces relaxations dans la Section 4.3. Finalement, dans la Section 4.4 nous comparons nos algorithmes

sur les instances de la littérature et prouvons pour la première fois l'optimalité de certaines solutions non triviales pour plusieurs de ces instances.

4.1 Une famille d'inégalités valides du $\mathbb{R}^2$-CFP à partir d'une discrétisation du domaine

Nous introduisons une famille d'inégalités valides du $\mathbb{R}^2$-CFP obtenues comme contraintes d'un problème de coloration d'hypergraphe.

4.1.1 Quelques définitions de théorie des graphes

Nous rappelons quelques notions de théorie des graphes qui seront utilisées pour établir des inégalités valides du $\mathbb{R}^2$-CFP.

Définition 32 (graphe)**.** On appelle $G = (S, A)$ un *graphe* s'il possède un ensemble de sommets S et un ensemble d'arêtes A contenant des paires de sommets $\{s, s'\}$ avec $s, s' \in S$. On dit que l'arête $\{s, s'\}$ relie s et s', ou que s et s' sont adjacents. Le graphe G est dit *simple* s'il n'y a pas d'arêtes reliant un sommet avec lui-même, et s'il n'y a pas plusieurs arêtes reliant la même paire de sommets.

Dans le reste du chapitre, tous les graphes considérés sont des graphes simples. Nous étudions en particulier le problème de coloration de graphe :

Définition 33 (Coloration de graphe)**.** Soit $G = (S, A)$ un graphe et n un entier positif. On appelle *problème de coloration du graphe* G le problème consistant à trouver une coloration des sommets $c : S \mapsto \{1, ..., n\}$ telle que deux sommets reliés par une arête ne sont pas de la même couleur. On appelle *nombre chromatique* du graphe G le plus petit nombre n tel qu'il existe une coloration de G à n couleurs.

C'est un problème NP-complet pour $n \geq 3$ [Garey et Johnson, 1979] qui a des applications dans de nombreux domaines. On dit qu'une coloration c définit une *partition* des sommets du graphe. Soit $G = (S, A)$ un graphe, une modélisation MILP du problème : *existe-t-il une coloration du graphe* G *à* n *couleurs ?* est donnée dans :

$$b_{i,j} \in \{0, 1\} \quad \forall i = 1, ..., |S|, \forall j = 1, ..., n \tag{4.1}$$

$$\sum_{j=1,...,n} b_{i,j} = 1 \quad \forall i = 1, ..., |S| \tag{4.2}$$

$$b_{i,j} + b_{i',j} \leq 1 \quad \forall (i, i') \in A, \forall j = 1, ..., n \tag{4.3}$$

où $|S|$ est le nombre de sommets dans S et où $b_{i,j} = 1$ indique que le sommet i est coloré de la couleur j.

Dans la suite, nous nous intéressons aux hypergraphes, une extension des graphes à des arêtes reliant plus de deux sommets :

Définition 34 (Hypergraphe)**.** Un *hypergraphe* H est un couple (S, A) où S est un ensemble de sommets S et A un ensemble d'arêtes, où chaque arête $a_i \in A$ est un sous-ensemble de S à au moins deux sommets.

Un hypergraphe peut posséder des arêtes qui relient plus de deux sommets. Il ne doit pas être confondu avec un multigraphe, où il peut y avoir plusieurs arêtes qui connectent la même paire de sommets. De la même manière que pour un graphe, Nous définissons le *problème de coloration d'hypergraphe* :

Définition 35 (Coloration d'hypergraphe)**.** Soit H un hypergraphe et n un entier positif. On appelle *problème de coloration de l'hypergraphe* H le problème consistant à trouver une coloration des sommets $c : S \mapsto \{1, ..., n\}$ telle que tous les sommets d'une arête $a \in A$ ne sont pas de la même couleur. On appelle nombre chromatique de l'hypergraphe H le plus petit nombre n tel qu'il existe une coloration de H à n couleurs.

Soit $H = (S, A)$ un hypergraphe. De la même manière que pour la coloration de graphe, il existe une modélisation MILP du problème : *existe-t-il une coloration de l'hypergraphe* H *à* n *couleurs ?*, où les contraintes (4.3) sont remplacées par les contraintes :

$$\sum_{k \in A} b_{k,j} \leq |a| \quad \forall a \in A, \forall j = 1, ..., n \tag{4.4}$$

pour modéliser les contraintes associées aux arêtes des hypergraphes. La Figure 4.1 donne un exemple de solution réalisable d'un problème de coloration d'hypergraphe. Les segments bleus sont des arêtes qui relient deux sommets, et le triangle jaune est une arête qui relie les trois sommets à l'intérieur. Les trois sommets reliés par l'arête jaune ne sont pas tous les trois de la même couleur.

Les sommets des graphes et hypergraphes que nous considérons dans ce chapitre ont un correspondent à des points de $\mathbb{R}^2$. Ainsi, il nous arrivera de parler de l'enveloppe convexe de plusieurs sommets d'un graphe, désignant alors l'enveloppe convexe des points de $\mathbb{R}^2$ auxquels sont associés les sommets du graphe.

4.1.2 Une famille d'inégalités valides issues du problème de coloration d'hypergraphe

Le problème de $\mathbb{R}^2$-CFP sur le corridor $\mathcal{C} = Corridor(u, l, D)$ possède un nombre infini indénombrable de contraintes de la forme $l(x) \leq g(x) \leq u(x)$ pour $x \in D$. Ces contraintes assurent que la fonction PWL g solution du problème a bien son graphe dans le corridor.

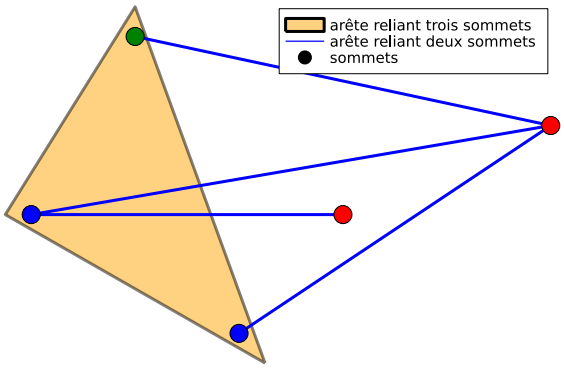


FIGURE 4.1: Exemple de coloration d'hypergraphe

Un sous-ensemble de ces contraintes construit grâce à un polygone convexe D_0 inclus dans le domaine du corridor D donne d'autres informations, et notamment permet de répondre à la question : *est-ce qu'il existe une pièce qui traverse la totalité du corridor* $Corridor(u, l, D_0)$ *induit par le sous-domaine* D_0 *?* Si la réponse est non, alors toutes les solutions réalisables du $\mathbb{R}^2$-CFP sur le corridor $\mathcal{C}$ vérifient qu'*il y a au moins deux pièces dont le domaine intersecte* D_0. L'ensemble des polygones convexes D_0 inclus dans D pour lesquels la réponse à la question est *non* est donc une famille de contraintes implicitement décrites dans le $\mathbb{R}^2$-CFP. Nous rappelons que l'on dit qu'une fonction PWL g *traverse* un corridor $\mathcal{C}$ si et seulement si g est définie sur le domaine $D(\mathcal{C})$ et toutes les pièces de g sont dans le corridor $\mathcal{C}$. La Définition 36 propose un nom au problème à résoudre pour répondre à la question.

Définition 36 (traversée linéaire de $\mathbb{R}^m$-corridor)**.** Soient $C_0 = Corridor(u, l, D_0)$, avec $D_0 \subset \mathbb{R}^m$ et g une fonction définie sur D_0. On dit que *la fonction* g *traverse linéairement le corridor* C_0 si g est une fonction linéaire et traverse le corridor C_0.

Nous dirons également qu'une pièce traverse un corridor si la fonction linéaire associée à la pièce traverse le corridor. Prenons un exemple. Soient x_1 et x_2 deux points du domaine D. Nous définissons le corridor $\mathcal{C}_{12} = Corridor(u, l, Conv(x_1, x_2))$, où $Conv(S)$ représente l'enveloppe convexe des points dans S. S'il n'existe pas de fonctions qui traverse linéairement le corridor $\mathcal{C}_{12}$, alors dans une solution du $\mathbb{R}^2$-CFP sur le corridor $\mathcal{C}$, il y a au moins deux pièces différentes dont le domaine intersecte le segment $Conv(x_1, x_2)$. Cela veut dire en particulier que les pièces dont les domaines couvrent x_1 et x_2 sont deux pièces différentes. La contrainte exprimant que x_1 et x_2 ne peuvent pas appartenir à la même pièce est vérifiée par toutes les solutions du $\mathbb{R}^2$-CFP de corridor $\mathcal{C}$, mais n'est pas écrite dans le problème : nous l'appelons provisoirement une *contrainte implicite* du $\mathbb{R}^2$-CFP. Elle a le même sens qu'une contrainte de coloration de graphe : les points (ou sommets) x_1 et x_2 ne peuvent pas appartenir à (ou être 'colorés' par) la même pièce (ou couleur) dans une solution réalisable du $\mathbb{R}^2$-CFP (coloration du graphe). De la même

manière, n'importe quel sous-ensemble fini a de points du domaine D pour lequel il n'existe aucune pièce traversant le corridor $Corridor(u, l, Conv(a))$ induit une contrainte implicite pour le problème $\mathbb{R}^2$-CFP sur le corridor $\mathcal{C}$. Elle indique que tous les points de a ne peuvent pas appartenir à (être colorés par) la même pièce. Comme ces contraintes implicites ont le même sens qu'une contrainte de coloration d'hypergraphe, il est possible de les exprimer de la même manière grâce aux contraintes (4.1), (4.2) et (4.4). Dans un modèle MILP adéquat, ce sont donc des contraintes linéaires. Pour cette raison, nous appelons maintenant ces contraintes implicites du $\mathbb{R}^2$-CFP des *inégalités valides* du $\mathbb{R}^2$-CFP.

Hypergraphe pour représenter les inégalités valides de coloration de graphe. Il est possible de créer des inégalités valides à une instance du $\mathbb{R}^2$-CFP en choisissant d'une part n'importe quel sous-ensemble fini I de points du domaine qui représente les sommets d'un hypergraphe H, et d'autre part des arêtes a correspondant à des sous-ensembles de points de I pour lesquels il n'existe pas de pièces traversant $Corridor(u, l, Conv(a))$. Comme une telle arête décrit une "incompatibilité" entre ses sommets, nous définissons les arêtes d'incompatibilité pour l'hypergraphe H par rapport au $\mathbb{R}^2$-CFP :

Définition 37 (arête d'incompatibilité)**.** Soient un corridor $\mathcal{C} = Corridor(u, l, D)$, un sous-ensemble fini I de D et un sous-ensemble a de I. On dit que a est une *arête d'incompatibilité* pour $\mathcal{C}$ s'il n'existe pas de pièce traversant le corridor $Corridor(u, l, Conv(a))$

Ainsi, un hypergraphe H de sommets I ne possédant que des arêtes d'incompatibilité représente uniquement des inégalités valides pour le $\mathbb{R}^2$-CFP sur le corridor $\mathcal{C}$.

4.1.3 Une relaxation du $\mathbb{R}^2$-CFP par discrétisation du domaine

Nous décrivons ici une relaxation du $\mathbb{R}^2$-CFP. Nous rappelons la formulation $(\mathcal{P})$ du $\mathbb{R}^2$-CFP pour simplifier la discussion qui suit.

$$(\mathcal{P}) \quad \begin{cases} min & n & (4.5) \\ s.c. & g \text{ est une fonction PWL à } n \text{ pièces} & (4.6) \\ & l(x) \leq g(x) \leq u(x) \quad \forall x \in D \subset \mathbb{R}^2. & (4.7) \end{cases}$$

Soit $\mathcal{C} = Corridor(u, l, D)$ un corridor et soit $H = (I, A)$ un hypergraphe tel que $I \subset D$, et A ne contient que des arêtes d'incompatibilité pour le $\mathbb{R}^2$-CFP sur le corridor $\mathcal{C}$. Le problème suivant est le $DCFP_H^{\mathcal{C}}$.

$$(DCFP_H^{\mathcal{C}}) \quad \begin{cases} min & n & (4.8) \\ s.c. & g : D' \mapsto \mathbb{R},\ D' \subset D, \text{ est une fonction PWL à } n \text{ pièces} & (4.9) \\ & \text{chaque pièce de } g \text{ correspond à une couleur dans la coloration} \\ & \text{de l'hypergraphe } H = (I, A) \text{ en } n \text{ couleurs,} & (4.10) \\ & l(x_i) \leq g(x_i) \leq u(x_i) \quad \forall x_i \in I \subset D. & (4.11) \end{cases}$$

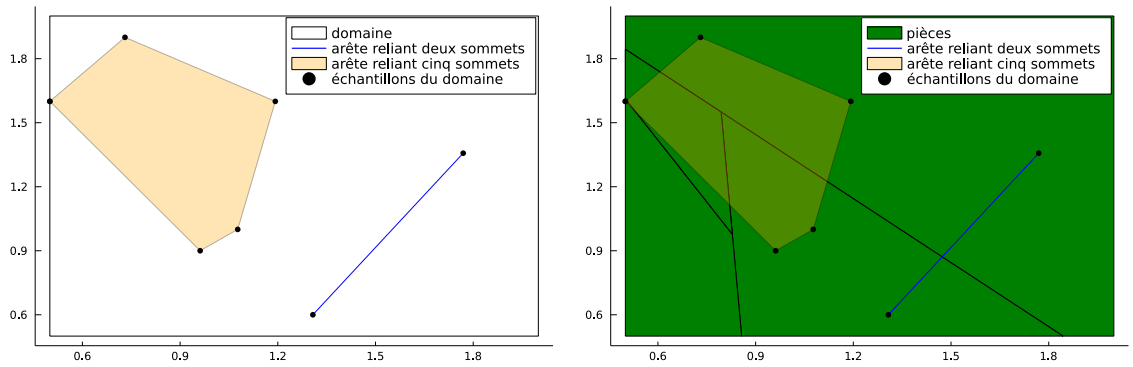
La contrainte (4.10) indique que chaque pièce de g est associée à une couleur de la coloration de l'hypergraphe H. Le domaine D' de la fonction g dans la contrainte (4.9) est l'union du domaine de chaque pièce de g, qui est l'enveloppe convexe des sommets $x \in I$ colorés de cette couleur. Ainsi, par définition de fonction linéaire par morceaux le domaine D' est un ensemble de polygones convexes inclus dans le domaine D dont l'intersection deux à deux est d'intérieur vide. De plus, les sommets I de l'hypergraphe H sont dans le domaine D'. Les contraintes (4.7) de $(\mathcal{P})$ sont remplacées par les contraintes (4.10) et (4.11). Remarquons que les contraintes (4.11) qui impose des contraintes de corridors sur un ensemble fini de points sont utilisées dans plusieurs travaux dont [Kazda et Li, 2021, Rebennack et Krasko, 2020, Kong et Maravelias, 2020], mais c'est la première fois que les contraintes (4.10) sont employées.

Proposition 38. *Soient le corridor $\mathcal{C} = Corridor(u, l, D)$, l'ensemble fini $I \subset D$ et l'hypergraphe $H = (I, A)$ tel que l'ensemble d'arêtes A ne contient que des arêtes d'incompatibilité pour $\mathcal{C}$. Le problème $DCFP_H^{\mathcal{C}}$ est une relaxation du $\mathbb{R}^2$-CFP sur le corridor $\mathcal{C}$.*

Démonstration. Le domaine D' est inclus dans le domaine D, donc la contrainte (4.9) est une relaxation de la contrainte (4.6). De plus, les contraintes (4.11) sont incluses dans les contraintes (4.7) puisque $I \subset D$. Finalement, nous avons montré dans la Section 4.1.2 que la contrainte (4.10) est une famille d'inégalités valides du problème de $\mathbb{R}^2$-CFP pour le corridor $\mathcal{C}$ car A ne contient que des arêtes d'incompatibilité. □

Résoudre le $DCFP_H^{\mathcal{C}}$ avec les contraintes de coloration d'hypergraphe de H est une relaxation du $\mathbb{R}^2$-CFP parce que H ne contient que des arêtes d'incompatibilité. En revanche, il n'est pas nécessaire de mettre toutes les arêtes d'incompatibilité existantes dans H. Nous exploitons ce fait par la suite.

La Figure 4.2a représente le domaine d'un corridor, avec un hypergraphe H dont les sommets sont les points noirs et les arêtes sont en bleu et en jaune. Ces deux arêtes sont des arêtes d'incompatibilité pour l'instance du $\mathbb{R}^2$-CFP pour laquelle elles ont été calculées. Plus précisément, pour n'importe quelle solution réalisable de cette instance du $\mathbb{R}^2$-CFP chacune de ces arêtes devra être couverte par au moins deux pièces. La Figure 4.2b représente une solution réalisable de cette instance en vert à laquelle nous avons superposé les arêtes de l'hypergraphe



(a) Représentation d'un hypergraphe H avec deux arêtes d'incompatibilité

(b) Superposition d'arêtes d'incompatibilité et d'une solution réalisable du $\mathbb{R}^2$-CFP

FIGURE 4.2: Représentation de l'hypergraphe H dans le $DCFP_H^{\mathcal{C}}$

H. Les deux arêtes sont bien couvertes par au moins deux pièces différentes dans cette solution.

Dans la suite de ce chapitre, nous considérons le corridor $\mathcal{C} = Corridor(u, l, D)$, le problème $DCFP_H^{\mathcal{C}}$ avec I un sous-ensemble fini de points de D et un hypergraphe $H = (I, A)$ dont les sommets sont I et toutes les arêtes $a \in A$ sont des arêtes d'incompatibilité du $\mathbb{R}^2$-CFP sur le corridor $\mathcal{C}$.

La Proposition 39 décrit une condition nécessaire et suffisante pour qu'une arête soit une arête d'incompatibilité pour un corridor :

Proposition 39. *Soient un corridor $\mathcal{C} = Corridor(u, l, D)$, un sous-ensemble fini I de D et un hypergraphe H' contenant l'arête a. a est une arête d'incompatibilité pour le corridor $\mathcal{C}$ si et seulement s'il n'existe pas de fonction linéaire $L : Conv(a) \mapsto \mathbb{R}$ satisfaisant $l(x) \leq L(x) \leq u(x)$ pour tout $x \in Conv(a)$.*

Démonstration. Soit a une arête qui n'est pas une arête d'incompatibilité. Par définition, cela est équivalent à *il existe une pièce traversant le corridor* $\mathcal{C}$, qui est équivalent à *il existe une fonction linéaire L traversant le corridor* $\mathcal{C}$. Or, cela se traduit par $l(x) \leq L(x) \leq u(x)$ pour tout $x \in Conv(a)$. Donc, l'arête a est une arête d'incompatibilité si et seulement s'il n'existe pas de fonction linéaire $L : Conv(a) \mapsto \mathbb{R}$ satisfaisant $l(x) \leq L(x) \leq u(x)$ pour tout $x \in Conv(a)$. □

La Section 4.1.4 détaille comment nous calculons les arêtes reliant deux sommets, tandis que la Section 4.1.5 explique comment nous identifions des arêtes reliant au moins trois sommets.

4.1.4 Calcul des arêtes reliant deux sommets

Soit $\mathcal{C}$ le corridor $Corridor(u, l, D)$. L'enveloppe convexe de deux points $x_1 = (x_1^1, x_1^2)$ et $x_2 = (x_2^1, x_2^1)$ dans $\mathbb{R}^2$ est un segment que l'on note s. Une fonction de deux variables f se

restreint à une fonction à une variable sur un segment de la manière suivante : $f_{|s}(x,y) = f(x(t),y(t)) = h(t)$ avec une combinaison convexe des deux extrémités du segment s :

$$(x(t),y(t)) = (tx_1^1 + (1-t)x_2^1, tx_1^2 + (1-t)x_2^2).$$

Ainsi, nous pouvons définir le corridor $C_s = Corridor(u,l,s)$ de domaine le segment s.

S'il existe une pièce qui traverse le corridor C_s, alors d'après la Proposition 39, l'arête correspondant aux points x_1 et x_2 n'est pas une arête d'incompatibilité pour le corridor $\mathcal{C}$, et ne représente donc pas une inégalité valide du $\mathbb{R}^2$-CFP. À l'inverse, s'il n'existe pas de pièces traversant le corridor $\mathcal{C}_s$ alors l'arête correspondant aux points x_1 et x_2 est une arête d'incompatibilité.

Pour calculer toutes les arêtes d'incompatibilité reliant deux sommets d'un hypergraphe à n sommets, il suffit donc de résoudre $\frac{n(n-1)}{2}$ problèmes de traversée linéaire de $\mathbb{R}$-corridor. Pour cela, il suffit de vérifier si le $\mathbb{R}$-CFP correspondant possède une solution à une pièce. Dans la pratique, résoudre tous ces problèmes peut être long. Il est possible de gagner du temps au prix de l'éventuel non détection de certaines arêtes d'incompatibilité reliant deux sommets en résolvant une relaxation du problème de traversée linéaire de $\mathbb{R}$-corridor utilisant lui aussi une discrétisation du corridor :

$$(\mathbb{R}-LDRFP) \quad \begin{cases} \textit{Existe-t-il des réels } a \textit{ et } b \textit{ tels que} & (4.12) \\ l(x(t_i),y(t_i)) \leq at_i + b \leq u(x(t_i),y(t_i)) \quad \forall t_i \in T \subset [0,1] & (4.13) \end{cases}$$

où T est un échantillonnage régulier du segment s à $N_{echantillons}$ points. Le nom du problème signifie problème de traversée d'intervalles par une fonction linéaire de $\mathbb{R}$ (Linear Data Range Fitting Problem). C'est une relaxation du problème de traversée linéaire de $\mathbb{R}$-corridor puisque la fonction linéaire solution $L(x) = ax + b$ doit vérifier $l \leq L \leq u$ seulement sur un ensemble fini de points dans le $\mathbb{R} - LDRFP$. S'il n'existe pas de réels a et b solution du $\mathbb{R} - LDRFP$, cela prouve que l'arête $a = \{x_1, x_2\}$ est une arête d'incompatibilité ; s'il existe de tels réels, nous ne savons pas si c'est une arête d'incompatibilité ou pas. Pour résoudre le $\mathbb{R} - LDRFP$, nous utilisons l'algorithme de [O'Rourke, 1981] modifié pour s'arrêter dès qu'il est montré qu'aucun segment ne peut couvrir tous les intervalles. Il est de complexité linéaire en le nombre d'éléments dans T et est donc très performant pour résoudre ce problème.

4.1.5 Calcul des arêtes reliant au moins trois sommets

Soit le corridor $\mathcal{C} = Corridor(u,l,D)$. Une méthode pour savoir si une arête a reliant plus de deux sommets est une arête d'incompatibilité pour le $\mathbb{R}^2$-CFP de corridor $\mathcal{C}$ est de résoudre

le problème de traversée linéaire de $\mathbb{R}^2$-corridor suivant :

$$\begin{cases} & \textit{Existe-t-il des réels } a, b, c \textit{ tels que} \\ & l(x) \leq ax^1 + bx^2 + c \leq u(x) \quad \forall x = (x^1, x^2) \in Conv(a). \end{cases} \tag{4.14}$$

Une des difficultés que pose ce problème est qu'il y a un nombre infini de contraintes dans cette formulation. Au prix de l'éventuelle non détection de certaines arêtes d'incompatibilité, nous considérons une relaxation de ce problème ne vérifiant l'appartenance au corridor que sur un sous-ensemble fini E de points de $Conv(a)$:

$$(\mathbb{R}^2 - LDRFP) \quad \begin{cases} & \textit{Existe-t-il des réels } a, b, c \textit{ tels que} \\ & l(x_i) \leq ax_i^1 + bx_i^2 + c \leq u(x_i) \quad \forall x_i = (x_i^1, x_i^2) \in E. \end{cases} \tag{4.15}$$

Le nom $\mathbb{R}^2 - LDRFP$ signifie problème de traversée d'intervalles par une fonction linéaire de $\mathbb{R}^2$. Ce problème peut être formulé directement en un problème de programmation linéaire à 3 variables et $2N$ contraintes, avec N le nombre de points dans l'ensemble I.

L'algorithme 3 décrit comment un ensemble d'arêtes reliant au moins trois sommets est calculé. Il revient à essayer de construire une arête d'incompatibilité contenant chaque sommet s_i de S un nombre $k_{repetition}$ de fois, implémenté grâce à la boucle pour présente à la Ligne 2.

Algorithme 3 Algorithme de calcul d'arêtes reliant au moins trois sommets

Entrée :
- S l'ensemble de sommets,
- A^2 l'ensemble des arêtes reliant deux sommets,
- N le nombre de points de I
- $k_{repetition}$ le nombre de tentatives par sommet

Sortie : nouvelles_aretes un ensemble d'arêtes d'incompatibilité reliant au moins trois sommets

```
1:  nouvelles_aretes = {}
2:  pour i allant de 1 à n faire
3:    pour k_rep allant de 1 à k_repetition faire
4:      ordre = i ∪ MELANGE({1, ..., i − 1, i + 1, ..., N})          ▷ i = ordre_1
5:      a = {i, ordre_2}        ▷ a contient les indices des sommets de l'arête en construction
6:      pour j allant de 3 à N faire
7:        si (ordre_j, s) ∉ A^2  ∀s ∈ a alors
8:          si S[ordre_j] ∉ Conv(S[a]) alors
9:            a = GARDERSEULEMENTPOINTSEXTREMAUX(S, a ∪ ordre_j)        ▷ enlève les
              points non extrémaux
10:           test = CALCULERFONCTIONLINEAIRE(S[a]) ;
11:           si test == Faux alors
12:             nouvelles_aretes = nouvelles_aretes ∪ a
13:             break
```

L'Algorithme 3 consiste à ajouter à l'ensemble a défini Ligne 5 des sommets s_j un par un

jusqu'à vérifier grâce à la résolution d'une instance de $\mathbb{R}^2 - LDRFP$ si l'arête a est une arête d'incompatibilité. Pour créer des arêtes différentes, les sommets font l'objet d'un tri aléatoire grâce à la Ligne 4. S'il existe déjà une arête d'incompatibilité a' telle que $a' \subset a$, alors ajouter l'arête a dans l'hypergraphe est inutile. En effet, si un ensemble de points a' ne peut pas être couvert par la même pièce, alors l'ensemble a le contenant ne peut pas non plus être couvert par la même pièce. Le test Ligne 7 s'assure de ne pas ajouter d'arêtes incluses dans une arête d'incompatibilité reliant deux sommets. Nous notons $Conv(S[a])$ l'enveloppe convexe des sommets de S dont les indices sont dans a. Ajouter un sommet qui est dans l'enveloppe convexe des sommets de a ne change pas l'enveloppe convexe et donc ne rend pas la contrainte plus forte. C'est pour cela que le test Ligne 8 n'est satisfait que si le sommet à ajouter $S[ordre_j]$ n'est pas dans l'enveloppe convexe. La fonction GarderSeulementPointsExtremaux retourne un ensemble contenant seulement les sommets qui sont des points extrémaux de l'arête a en construction pour qu'elle contienne le moins de sommets possibles, puisque cela donne une contrainte plus forte. De plus, la fonction CalculerFonctionLineaire construit et résout le $\mathbb{R}^2 - LDRFP$ sur le domaine $Conv(S[a])$ et retourne $Vrai$ s'il existe une fonction linéaire solution, et $Faux$ sinon.

Construction de l'ensemble E. Soit a un sous-ensemble de sommets de l'hypergraphe H à au moins 3 sommets. L'ensemble E des Contraintes (4.15) est construit à l'aide de $Conv(a)$: nous calculons n^E_{arete} points répartis uniformément sur chaque côté de $Conv(a)$, et nous ajoutons n^E_{int} points à l'intérieur de $Conv(a)$ de la même manière que pour l'échantillonnage utilisé dans le calcul de VaT décrit dans l'Annexe B.1.

4.2 Quatre relaxations successives du $\mathbb{R}^2$-CFP

Nous décrivons ici quatre relaxations du $\mathbb{R}^2$-CFP basées sur le $DCFP^{\mathcal{C}}_H$. Ces relaxations sont introduites de la plus forte à la plus faible.

4.2.1 Relaxation par discrétisation du domaine

Le problème $\mathcal{R}_1 - DCFP^{\mathcal{C}}_H$ est en fait le problème $DCFP^{\mathcal{C}}_H$. Nous lui donnons ce nouveau nom pour indiquer explicitement son rang dans les relaxations successives : c'est la relaxation la plus forte parmi les quatre relaxations du $\mathbb{R}^2$-CFP, celle dont la valeur optimale est la plus élevée. Ce problème s'écrit de la manière suivante :

$$(\mathcal{R}_1 - DCFP_H) \begin{cases} min & n & (4.16) \\ s.c. & g : D' \mapsto \mathbb{R},\ D' \subset D, \text{ est une fonction PWL à } n \text{ pièces} & (4.17) \\ & \text{chaque pièce de } g \text{ correspond à une couleur dans la} & \\ & \text{coloration de l'hypergraphe } H = (I, A) \text{ en } n \text{ couleurs} & (4.18) \\ & l(x_i) \leq g(x_i) \leq u(x_i) \quad \forall x_i \in I \subset D, & (4.19) \end{cases}$$

où A est un ensemble d'arêtes d'incompatibilité pour le $\mathbb{R}^2$-CFP sur le corridor $\mathcal{C} = Corridor(u, l, D)$.

4.2.2 Relaxation en coloration d'hypergraphe

Pour obtenir la relaxation $\mathcal{R}_2 -$ Coloration, nous partons du problème (4.16)-(4.19) et relâchons les contraintes (4.17) et (4.19). Le problème $\mathcal{R}_2 -$ Coloration se résume à encoder les inégalités valides décrites dans H, ce qui revient à utiliser uniquement les contraintes de coloration de l'hypergraphe H :

Quel est le nombre chromatique de l'hypergraphe H ? ($\mathcal{R}_2 -$ Coloration)

4.2.3 Relaxation en problèmes de cliques

Nous décrivons ici deux relaxations du problème de coloration d'hypergraphe $\mathcal{R}_2-$Coloration basées sur la recherche de cliques :

Définition 40 (Clique, clique maximum)**.** Soit $G = (S, A)$ un graphe. On appelle *clique* du graphe G un sous-ensemble S' des sommets de G pour lequel tous les sommets de $s \in S'$ sont adjacents dans G. On appelle un sous-ensemble S' de G une *clique maximum* du graphe G une clique de cardinal maximum.

Ainsi, trouver une clique à n sommets est une preuve qu'il faut au moins n couleurs pour colorier un graphe. En effet, deux sommets de la clique ne peuvent pas être colorés de la même couleur puisqu'ils sont adjacents. La clique maximum est donc la clique qui prouve la plus grande borne inférieure du nombre chromatique d'un graphe. Cependant, calculer une clique maximum dans un graphe est un problème NP-difficile [Karp, 1972], donc une borne inférieure au problème de coloration de graphe peut être coûteuse à obtenir en terme de temps de calcul. Il existe un autre type de clique qui est beaucoup plus facile à calculer et qui peut donc être exploité pour obtenir une borne inférieure :

Définition 41 (Clique maximale)**.** Soit $G = (S, A)$ un graphe. On appelle *clique maximale* du graphe G un sous-ensemble S' de sommets de G qui est une clique, et pour lequel il n'existe pas de sommet $s \in S \setminus S'$ qui soit adjacent à tous les sommets de S'.

En d'autres termes, une clique maximale est une clique qui n'est incluse dans aucune autre clique de G. Il peut y en avoir un grand nombre dans un graphe, et de tailles très différentes de celle d'une clique maximum.

Les cliques sont définies sur des graphes et pas sur des hypergraphes donc les relaxations à base de cliques du $\mathbb{R}^2$-CFP utilisent seulement des graphes. Nous définissons alors le graphe H' comme l'hypergraphe H auquel on a retiré les arêtes reliant au moins trois sommets. Ces relaxations sont $\mathcal{R}_3 -$ CliqueMaximum et $\mathcal{R}_4 -$ CliqueMaximale :

Quel est le cardinal d'une clique maximum du graphe H' ?
($\mathcal{R}_3 -$ CliqueMaximum)

Quel est le cardinal d'une clique maximale du graphe H' ?
($\mathcal{R}_4 -$ CliqueMaximale)

Comme la taille d'une clique maximale est inférieure ou égale à la taille d'une clique maximum, $\mathcal{R}_4 -$ CliqueMaximale est bien une relaxation de $\mathcal{R}_3 -$ CliqueMaximum.

4.3 Algorithmes pour résoudre les relaxations

Nous présentons dans cette section quatre algorithmes pour résoudre les quatre relaxations définies dans la section précédente. Ces algorithmes permettent donc de calculer des bornes inférieures du problème de $\mathbb{R}^2$-CFP pour le corridor $Corridor(u, l, D)$. Dans le but de les comparer numériquement sur des temps de calcul équivalents, ces algorithmes ont en commun le fait de répéter la résolution de la relaxation correspondante avec des instances de plus en plus grandes jusqu'à atteindre la limite de temps.

4.3.1 Algorithme pour la relaxation $\mathcal{R}_1 - DCFP_H^{\mathcal{C}}$

MILP de coloration d'hypergraphe avec contraintes de convexité du domaine des pièces. Nous décrivons ici un modèle MILP qui fournit une solution réalisable du $DCFP_H^{\mathcal{C}}$ à n pièces tout en respectant une n-coloration de l'hypergraphe H. Cela représente une fonction PWL définie sur un sous-ensemble de polygones convexes $D' \subset D$ respectant les contraintes de corridor sur un nombre fini de points du domaine D du corridor. Nous utilisons ici indifféremment les expressions *coloration d'un sommet par la couleur j* ou *couverture d'un sommet par une pièce j* puisque le problème de coloration revient à affecter des sommets aux pièces. La Table 4.1 décrit les paramètres utilisés dans ce modèle MILP. Les paramètres $a_{min}, a_{max}, b_{min}, b_{max}, c_{min}, c_{max}$ sont calculées en raisonnant sur les valeurs possibles des coefficients des fonctions linéaires des pièces en fonction des points, des paires de points et des triplets de points de la discrétisation du domaine.

paramètres	rôles
$Corridor(u, l, D)$	corridor du $\mathbb{R}^2$-CFP
N	nombre de sommets de l'hypergraphe $H = (S, A)$
n	nombre de couleurs à utiliser dans la coloration d'hypergraphe
x, y	vecteurs contenant les positions dans $(x_i, y_i) \in D$ des sommets du graphe $H = (S, A)$
A	ensemble contenant toutes les arêtes d'incompatibilité calculées
$cliqueMaximale$	clique maximale calculée avec les arêtes reliant deux sommets de H
a_{min}, a_{max} b_{min}, b_{max} c_{min}, c_{max}	bornes sur les coefficients des fonctions linéaires

TABLE 4.1: Paramètres du problème MILP de coloration d'hypergraphe

variable	indices	type	rôle
$b_{i,j}$	$i = 1, ..., N,\ j = 1, ..., n$	binaire	vaut 1 si le point i est couvert par la pièce j
$a_j^{cor}, b_j^{cor}, c_j^{cor}$	$j = 1, ..., n$	continue	coefficients de la fonction linéaire de la pièce j d'expression : $a_j^{cor}x + b_j^{cor}y + c_j^{cor}$
$a_{j,j'}^{sep,+}, b_{j,j'}^{sep,+}$	$j, j' = 1, ..., n, j < j'$	continue	coefficients positifs du vecteur directeur de la séparation linéaire des pièces j et j'
$a_{j,j'}^{sep,-}, b_{j,j'}^{sep,-}$	$j, j' = 1, ..., n, j < j'$	continue	coefficients négatifs du vecteur directeur de la séparation linéaire des pièces j et j'
$sign_{j,j'}^{a}, sign_{j,j'}^{b}$	$j, j' = 1, ..., n, j < j'$	binaire	indique le signe du coefficient $a_{j,j'}^{sep}$ ou $b_{j,j'}^{sep}$ du vecteur directeur de la séparation linéaire
$w_{j,j'}$	$j, j' = 1, ..., n, j < j'$	continue	valeur seuil de la séparation linéaire des pièces j et j'

TABLE 4.2: Variables du problème MILP de coloration d'hypergraphe

La Table 4.2 liste quant à elle les variables définies dans le modèle.

Elle précise l'ensemble des indices, le type, et le rôle de la variable dans le modèle. Pour faciliter la lecture, nous utilisons une expression dans la description des contraintes du modèle pour les variables qui ont en exposant les mots '*sep,+*' ou '*sep,-*' :

$$a_{j,j'}^{sep} := a_{j,j'}^{sep,+} - a_{j,j'}^{sep,-} \tag{4.20}$$

$$b_{j,j'}^{sep} := b_{j,j'}^{sep,+} - b_{j,j'}^{sep,-} \tag{4.21}$$

$$c_{j,j'}^{sep} := c_{j,j'}^{sep,+} - c_{j,j'}^{sep,-} \tag{4.22}$$

Parmi les rôles des variables, il y en a qui assurent la *séparation linéaire* entre les domaines de deux pièces, que nous définissons maintenant.

Définition 42 (séparation linéaire)**.** Soient A,$B \subset \mathbb{R}^m$. Les ensembles A et B sont *séparés linéairement* si :

$$\exists v \in \mathbb{R}^m,\ \exists b \in \mathbb{R}\ :\ \forall x \in A,\ \forall y \in B : x.v \leq b \leq y.v.$$

La séparation linéaire des ensembles A et B consiste donc à séparer $\mathbb{R}^m$ en deux demi-espaces,

de façon à ce que le premier demi-espace contienne A et le deuxième B. Il ne faut pas confondre une fonction linéairement séparable et deux ensembles séparés linéairement puisqu'ils ne traitent pas du même type d'objets. Dans le premier cas c'est une fonction, et dans le deuxième ce sont deux ensembles. Le modèle MILP de coloration d'hypergraphe est le modèle (4.23)-(4.40).

$$a_{min} \leq a_j^{cor} \leq a_{max} \quad \forall j = 1, ..., n \tag{4.23}$$
$$b_{min} \leq b_j^{cor} \leq b_{max} \quad \forall j = 1, ..., n \tag{4.24}$$
$$c_{min} \leq c_j^{cor} \leq c_{max} \quad \forall j = 1, ..., n \tag{4.25}$$
$$(b_{i,j} = 1) \Rightarrow l(x_i, y_i) \leq a_j^{cor} x_i + b_j^{cor} y_i + c_j^{cor} \quad \forall i = 1, ..., N, \forall j = 1, ..., n \tag{4.26}$$
$$(b_{i,j} = 1) \Rightarrow u(x_i, y_i) \geq a_j^{cor} x_i + b_j^{cor} y_i + c_j^{cor} \quad \forall i = 1, ..., N, \forall j = 1, ..., n \tag{4.27}$$
$$(b_{i,j} = 1) \Rightarrow a_{j,j'}^{sep} x_i + b_{j,j'}^{sep} y_i \leq w_{j,j'} \quad \forall i = 1, ..., N, \forall j, j' = 1, ..., n, j < j' \tag{4.28}$$
$$(b_{i,j'} = 1) \Rightarrow a_{j,j'}^{sep} x_i + b_{j,j'}^{sep} y_i \geq w_{j,j'} \quad \forall i = 1, ..., N, \forall j, j' = 1, ..., n, j < j' \tag{4.29}$$
$$a_{j,j'}^{sep,+} \leq sign_{j,j'}^{a} \quad \forall j, j' = 1, ..., n, j < j' \tag{4.30}$$
$$a_{j,j'}^{sep,-} \leq 1 - sign_{j,j'}^{a} \quad \forall j, j' = 1, ..., n, j < j' \tag{4.31}$$
$$b_{j,j'}^{sep,+} \leq sign_{j,j'}^{b} \quad \forall j, j' = 1, ..., n, j < j' \tag{4.32}$$
$$b_{j,j'}^{sep,-} \leq 1 - sign_{j,j'}^{b} \quad \forall j, j' = 1, ..., n, j < j' \tag{4.33}$$
$$a_{j,j'}^{sep,+} + a_{j,j'}^{sep,-} + b_{j,j'}^{sep,+} + b_{j,j'}^{sep,-} \geq 1 \quad \forall j, j' = 1, ..., n, j < j' \tag{4.34}$$
$$0 \leq a_{j,j'}^{sep,+}, a_{j,j'}^{sep,+}, a_{j,j'}^{sep,+}, a_{j,j'}^{sep,+} \leq 1 \quad \forall j, j' = 1, ..., n, j < j' \tag{4.35}$$
$$\sum_{j=1,...,n} b_{i,j} = 1 \quad \forall i = 1, ..., N \tag{4.36}$$
$$\sum_{i=1,...,N} b_{i,j} \geq 1 \quad \forall j = 1, ..., n \tag{4.37}$$
$$\sum_{i \in a} b_{i,j} \leq |a| - 1 \quad \forall a \in A \tag{4.38}$$
$$b_{cliqueMaximale_j, j} = 1 \quad \forall j \in [1, |cliqueMaximale|] \tag{4.39}$$
$$b_{i,j} = 0 \quad \forall i \in cliqueMaximale, i \neq cliqueMaximale_j, \forall j \in [1, |cliqueMaximale|] \tag{4.40}$$

Les contraintes (4.23)-(4.25) limitent le domaine des variables décrivant l'expression de la fonction linéaire de la pièce j. Les contraintes (4.26)-(4.29) décrivent des contraintes indicatrices. L'écriture d'une contrainte indicatrice $(b = 1) \Rightarrow L \leq R$ indique que la contrainte $L \leq R$ doit être respectée si la variable binaire b vaut 1, et pas forcément si b vaut 0. Cela remplace une contrainte utilisant une formulation big-M et certains solveurs MILP savent gérer ce type de contraintes. Un des avantages des contraintes indicatrices sur les formulations big-M est que l'on n'a pas à fournir les valeurs des big-M. Les contraintes (4.26)-(4.27) forcent la pièce j à respecter le corridor au point i si la variable binaire $b_{i,j}$ vaut 1. De plus, les contraintes (4.28)-

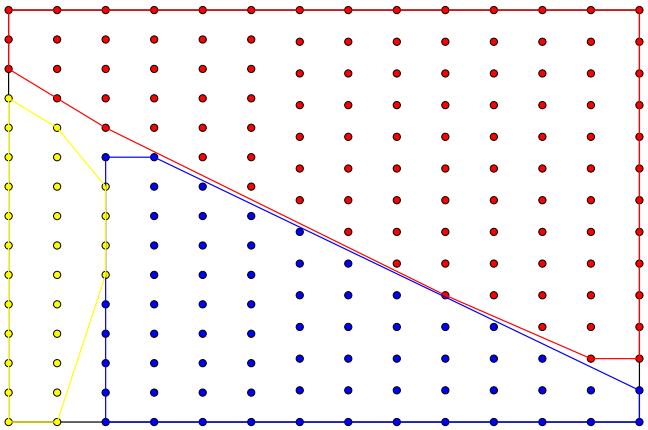


FIGURE 4.3: Exemple de solution du modèle MILP (4.23)-(4.40)

(4.29) assurent que si la variable binaire $b_{i,j}$ vaut 1, alors tous les points de la pièce j doivent être dans le demi-plan $\{(x,y) \mid a^{sep}_{j,j'}x + b^{sep}_{j,j'}y \leq w_{j,j'}\}$ et tous les points de la pièce j' dans le demi-plan complémentaire $\{(x,y) \mid a^{sep}_{j,j'}x + b^{sep}_{j,j'}y \geq w_{j,j'}\}$. La droite $a^{sep}_{j,j'}x + b^{sep}_{j,j'}y = w_{j,j'}$ sépare donc les pièces j et j'. Cela permet de reconstruire le polygone convexe représentant le domaine de chaque pièce à partir de la solution du modèle MILP. Pour j et j' fixées, les contraintes (4.30)-(4.35) assurent que le vecteur $(a^{sep}_{j,j'}, b^{sep}_{j,j'})$ prend des valeurs dans le domaine $D_{sep} = \{(a,b) \in [-1,1] \mid |a|+|b| \geq 1\}$. Le but de cela est que le vecteur $(a^{sep}_{j,j'}, b^{sep}_{j,j'})$ ne prenne pas la valeur $(0,0)$ puisqu'il représente un vecteur directeur de droite dans $\mathbb{R}^2$. Les contraintes (4.36) vérifient que chaque point i est couvert par une pièce j, tandis que les contraintes (4.37) forcent chaque pièce j à couvrir au moins un point i pour utiliser toutes les pièces. Les contraintes (4.38) décrivent les contraintes de coloration des arêtes de l'hypergraphe $H = (S, A)$. Pour chacune des arêtes a de l'hypergraphe, tous ses sommets ne peuvent pas appartenir à la même pièce j. Pour réduire le nombre de variables binaires $b_{i,j}$, nous utilisons la clique maximale $cliqueMaximale$ pour définir les contraintes (4.39) et (4.40). Cela permet de fixer à une pièce j différente chacun des sommets i de la clique $cliqueMaximale$ avec $b_{i,j} = 1$.

La Figure 4.3 montre un exemple de solution du modèle MILP (4.23)-(4.40) pour $n = 3$ couleurs. Les sommets de l'hypergraphe H sont colorés en jaune, bleu ou rouge suivant la pièce qui les contient. L'enveloppe convexe des sommets de chaque couleur indique le domaine de la pièce. Grâce aux contraintes assurant la séparation linéaire, ces enveloppes convexes ne s'intersectent pas ou uniquement sur leurs frontières. Cela correspond bien à une fonction PWL définie sur un sous-ensemble de polygones convexes $D' \subset D$ respectant les contraintes de corridor sur un nombre fini de points du domaine D du corridor.

Algorithme basé sur la résolution d'un MILP de coloration d'hypergraphe. L'Algorithme Alg_1-CHCC, pour coloration d'hypergraphe avec contraintes de convexité, utilise le modèle MILP de coloration d'hypergraphe avec contraintes de convexité (4.23)-(4.40) pour calculer une borne inférieure BI au $\mathbb{R}^2$-CFP sur le $Corridor(u, l, D)$. Il est décrit dans l'Algorithme 4.

Algorithme 4 Algorithme Alg_1-CHCC basé sur un MILP de coloration d'hypergraphe avec contraintes de convexité

Entrées :
- l'instance $\mathcal{C} = Corridor(u, l, D)$ de $\mathbb{R}^2$-CFP
- n_{max} une borne supérieure
- $k_{repetition}$ paramètre de l'Algorithme 3 du même nom
- (N_0, N_{add}) coefficients décrivant le nombre de sommets $N = N_0 n + N_{add}$ en fonction du nombre de pièces n à utiliser
- $\alpha > 1$ multiplicateur pour le nombre de sommets N
- limite_temps la limite de temps d'exécution de l'algorithme
- $n_{cliques\_maximales}$ le nombre de calculs de cliques maximales dans la fonction HeuristiqueCliqueMaximale

Sortie :
BI la borne inférieure au $\mathbb{R}^2$-CFP sur le corridor $Corridor(u, l, D)$

1: $BI = 1$
2: $k_{fail} = 0$
3: **tant que** $limite\_temps -$ temps écoulé > 0 **faire**
4: $N =$ ArrondiSuperieur$((N_0 BI + N_{add}) \times \alpha^{k_{fail}})$
5: $S =$ CalculSommets$(\mathcal{C}, N)$
6: $A^2 =$ CalculAretesTaille2$(\mathcal{C}, S)$
7: $cliqueMaximale =$ HeuristiqueCliqueMaximale$(A^2, n_{cliques_maximales})$
8: **si** Taille$(cliqueMaximale) > n$ et temps écoulé $> limite\_temps$ **alors**
9: $BI =$ Taille$(cliqueMaximale)$
10: **sinon**
11: $A =$ CalculAretes$(\mathcal{C}, S, A^2, N, k_{repetition}) \cup A^2$
12: $statut =$ MILP$(\mathcal{C}, S, A, BI, cliqueMaximale, limite_temps -$ temps écoulé$)$
13: **si** temps écoulé $> limite\_temps$ **alors**
14: Retourner BI
15: **si** $statut = irréalisable$ **alors**
16: $BI = BI + 1$
17: **sinon**
18: **si** $statut = optimal$ **alors**
19: $k_{fail} = k_{fail} + 1$

Le nombre de sommets N de l'hypergraphe H est calculé Ligne 4. Sa valeur est une fonction linéaire de la borne inférieure BI courante multipliée par un coefficient α à la puissance du paramètre k_{fail}. Ce paramètre dépend du nombre de résolutions du MILP qui se sont terminées par l'obtention d'une solution optimale. L'appel à la fonction CalculSommets$(\mathcal{C}, N)$ Ligne 5

retourne N points du domaine D du $Corridor(u, l, D)$. La fonction CalculAretesTaille2 Ligne 6 calcule les arêtes reliant deux sommets de l'hypergraphe dont les sommets sont S comme expliqué dans la Section 4.1.4. Le sous-ensemble de sommets $cliqueMaximale$ calculé par la fonction HeuristiqueCliqueMaximale Ligne 7 sert à fixer des variables binaires du modèle MILP décrit par les contraintes (4.23)-(4.40). Il est calculé à partir d'une heuristique gloutonne de construction de clique maximale : à chaque itération, l'heuristique ajoute à la clique le sommet avec le plus d'arêtes qui est relié à tous les sommets déjà ajoutés à la clique. Les égalités sont départagées par un tirage aléatoire. Cette heuristique s'exécute très rapidement par rapport à d'autres éléments de l'algorithme donc elle est utilisée $n_{cliques\_maximales}$ fois et la clique maximale avec le plus de sommets est retournée. Trouver une clique à m sommets dans le graphe (S, A^2) prouve qu'il faut au moins m pièces pour couvrir l'hypergraphe $H = (S, A)$ tel que $A^2 \subset A$. Donc si la clique $cliqueMaximale$ trouvée contient strictement plus de BI sommets, la borne inférieure BI est ajustée au nombre de sommets de la clique dans les Lignes 8 et 9. Dans ce cas, le modèle MILP n'a pas besoin d'être résolu pour l'ancienne valeur de BI. Sinon, la fonction CalculAretes Ligne 11 calcule un certain nombre d'arêtes reliant au moins trois sommets grâce à l'Algorithme 3 en fonction du paramètre $k_{repetition}$.

Finalement, le modèle MILP (4.23)-(4.40) est lancé avec une limite de temps $limite\_temps-$ temps écoulé, correspondant au temps limite $limite\_temps$ auquel on soustrait le temps déjà écoulé depuis le début de l'algorithme. Nous récupérons le statut de la résolution du modèle MILP (4.23)-(4.40) avec la sortie de la fonction MILP Ligne 12. Si le modèle est irréalisable alors il faut strictement plus de BI pièces pour résoudre cette instance du $\mathbb{R}^2$-CFP, et BI est incrémenté, sinon c'est k_{fail} qui est incrémenté. Le paramètre k_{fail} agit sur le nombre de sommets N aux prochaines itérations de la boucle tant que Ligne 3. Si le MILP a été résolu à l'optimum, il y a deux possibilités. Soit BI est la valeur optimale de l'instance de $\mathbb{R}^2$-CFP, soit les contraintes retirées pendant la discrétisation de l'instance de $\mathbb{R}^2$-CFP font que la borne inférieure trouvée est strictement inférieure à la solution optimale du $\mathbb{R}^2$-CFP.

Pour tenter d'améliorer la qualité de la borne inférieure, il est possible d'augmenter le nombre de sommets via les paramètres (N_0, N_{add}), ou d'augmenter le nombre d'arêtes reliant au moins trois sommets avec $k_{repetition}$. En contrepartie, le temps de calcul des fonctions de l'algorithme impactées augmentera. Le MILP peut aussi se terminer en atteignant le temps limite autorisé $limite\_temps -$ temps écoulé ; cela ne permet pas de tirer d'informations sur la borne inférieure BI.

4.3.2 Algorithme pour la relaxation $\mathcal{R}_2$ – Coloration

L'Algorithme Alg_2-Coloration calcule une borne inférieure à la relaxation ($\mathcal{R}_2-$Coloration) pour des hypergraphes avec un nombre croissant de sommets jusqu'à épuisement du temps de

calcul donné par le paramètre *limite_temps*. L'Algorithme 5 le décrit. Le nombre N de som-

Algorithme 5 Algorithme Alg_2-ColorationHypergraphe basé sur la coloration d'hypergraphe

Entrée :
- l'instance $\mathcal{C} = Corridor(u, l, D)$ du $\mathbb{R}^2$-CFP
- N_0 le nombre de sommets du graphe initial discrétisant le domaine D
- $k_{repetition}$ le nombre de tentatives de création d'arêtes reliant plus de deux sommets pour chaque sommet de l'hypergraphe (S, A) construit dans l'algorithme
- $\alpha > 1$ le multiplicateur pour le nombre de sommets de l'hypergraphe
- N_{add} le nombre additionnel de sommets par pièce dans la borne inférieure prouvée
- limite_temps la limite de temps d'exécution de l'algorithme

Sortie : BI une borne inférieure au $\mathbb{R}^2$-CFP sur $Corridor(u, l, D)$

1: $BI = 1$
2: $k_{fail} = 0$
3: **tant que** *limite_temps* − temps écoulé > 0 **faire**
4: $N =$ ArrondiSuperieur$((N_0 + N_{add}BI) \times \alpha^{k_{fail}})$
5: $S =$ CalculSommets$(\mathcal{C}, N)$
6: $A^2 =$ CalculAretesTaille2$(\mathcal{C}, S)$
7: $A =$ CalculAretes$(\mathcal{C}, S, A^2, N, k_{repetition}) \cup A^2$
8: $n =$ ColorationHypergraphe$(S, A,$ *limite_temps* − temps écoulé$)$
9: **si** $n > BI$ et *limite_temps* − temps écoulé > 0 **alors**
10: $BI = n$
11: **sinon**
12: $k_{fail} = k_{fail} + 1$

mets de l'hypergraphe calculé Ligne 4 est croissant avec la borne inférieure déja prouvée BI et le nombre de fois que la résolution de la relaxation n'a pas amélioré la borne inférieure à l'aide de k_{fail}. La fonction ColorationHypergraphe Ligne 8 calcule le nombre chromatique de l'hypergraphe $H = (S, A)$ avec une limite de temps de *limite_temps* − temps écoulé, le temps restant avant épuisement du temps alloué à l'algorithme par le biais de *temps_limite*. Plus précisément, cette fonction est basée sur l'algorithme de coloration de graphe $gc - cdcl$ décrit dans l'article [Hebrard et Katsirelos, 2020]. Il combine des techniques de programmation par contraintes [Van Hentenryck, 2003] et de résolution de problème de satisfaction de contraintes [Ghedira, 2013] pour trouver le nombre chromatique d'un graphe. La règle de branchement utilisée dans le Branch-and-Bound de $gc - cdcl$ utilise la récurrence de Zykov [Hebrard et Katsirelos, 2020]. Elle consiste à considérer dans un des nœuds fils que deux sommets ont la même couleur, et dans l'autre que les deux mêmes sommets n'ont pas la même couleur. Cela permet de modifier le graphe à colorier suivant le choix de branchement : si les deux sommets doivent avoir la même couleur, ils sont fusionnés en un seul nœud ; et si deux sommets ne doivent pas avoir la même couleur, une arête les reliant est ajoutée. La contrainte de coloration associée à une arête d'au moins trois sommets est exprimable dans un problème de satisfaction de contraintes.

Ainsi, l'algorithme de coloration de graphe s'adapte en algorithme de coloration d'hypergraphe simplement en ajoutant ces contraintes et en retirant le calcul de borne supérieure spécifique à la coloration d'un graphe. De plus, au cours de l'algorithme $gc - cdcl$, lorsque une nouvelle borne inférieure est prouvée, elle est sauvegardée pour pouvoir être récupérée si l'algorithme fini sans trouver la solution optimale.

4.3.3 Algorithmes pour les relaxations à base de cliques

Nous implémentons ici deux algorithmes pour résoudre les problèmes $\mathcal{R}_3-$CliqueMaximum et $\mathcal{R}_4-$CliqueMaximale. L'Algorithme Alg_3-CliqueMaximum calcule de manière itérative des cliques maximums dans des graphes avec de plus en plus de sommets à partir du corridor $Corridor(u, l, D)$ de l'instance de $\mathbb{R}^2$-CFP jusqu'à atteindre la limite de temps d'exécution. Il est décrit dans l'Algorithme 6.

Algorithme 6 Algorithme Alg_3-CliqueMaximum basé sur le calcul de cliques maximums

Entrée :
- l'instance $\mathcal{C} = Corridor(u, l, D)$ du $\mathbb{R}^2$-CFP
- N le nombre de sommets du graphe initial discrétisant le domaine D
- $\alpha > 1$ un multiplicateur pour le nombre de sommets initial N
- limite_temps la limite de temps d'exécution de l'algorithme

Sortie : BI une borne inférieure au $\mathbb{R}^2$-CFP sur le corridor $Corridor(u, l, D)$

1: $BI = 1$
2: **tant que** $limite\_temps -$ temps écoulé > 0 **faire**
3: $S =$ CalculSommets$(\mathcal{C}, N)$
4: $A^2 =$ CalculAretesTaille2$(\mathcal{C}, S)$
5: $n =$ CalculCliqueMaximum(S, A)
6: **si** $n > BI$ et $limite\_temps -$ temps écoulé > 0 **alors**
7: $BI = n$
8: $N =$ ArrondiSuperieur$(N\alpha)$

L'Algorithme Alg_4-CliqueMaximale est décrit dans l'Algorithme 7. Il fonctionne sur le même principe, avec le calcul de $n_{cliques\_maximales}$ cliques maximales à la place du calcul d'une clique maximum. Pour cela, l'algorithme glouton HeuristiqueCliqueMaximale décrit dans l'Algorithme 4 produit une clique maximale à la place de la fonction CalculCliqueMaximum Ligne 5 de l'Algorithme Alg_3-CliqueMaximum. La construction d'une clique maximale de cette manière a une complexité en $O(|S||A|)$ ce qui permet de manipuler des graphes avec plus de sommets que pour les trois autres algorithmes calculant une borne inférieure du $\mathbb{R}^2$-CFP.

Algorithme 7 Algorithme Alg_4-CliqueMaximale basé sur le calcul de clique maximale

Entrée :
- l'instance $\mathcal{C} = Corridor(u, l, D)$ du $\mathbb{R}^2$-CFP
- N le nombre de sommets du graphe initial discrétisant le domaine D
- $\alpha > 1$ un multiplicateur pour le nombre de sommets initial N
- limite_temps la limite de temps d'exécution de l'algorithme
- $n_{cliques\_maximales}$ le nombre de répétitions de la construction d'une clique maximale à chaque itération

Sortie : BI une borne inférieure au $\mathbb{R}^2$-CFP sur le corridor $Corridor(u, l, D)$

1: $BI = 1$
2: **tant que** $limite\_temps -$ temps écoulé > 0 **faire**
3: $S =$ CalculSommets$(\mathcal{C}, N)$
4: $A^2 =$ CalculAretesTaille2$(\mathcal{C}, S)$
5: $n = 1$
6: $clique =$ HeuristiqueCliqueMaximale$(A^2, n_{cliques_maximales})$
7: **si** $taille(clique) > BI$ et $limite\_temps -$ temps écoulé > 0 **alors**
8: $BI = taille(clique)$
9: $N =$ ArrondiSuperieur$(N\alpha)$

4.4 Expériences numériques

Le jeu d'instances utilisé est le même que celui du Chapitre 3.4, provenant de [Rebennack et Kallrath, 2015a]. Pour la reproductibilité des résultats, toutes les étapes aléatoires des algorithmes utilisés dans cette section sont initialisées avec la même graine. Nous utilisons la librairie JuMP (version 0.21.10) du langage julia comme langage de modélisation, et le solveur Gurobi (version 9.1.1) pour la résolution de problèmes MILP/LP. L'ordinateur sur lequel les expériences ont été faites a un CPU i5-10310U de 1.7 GHz, et 32 Go de RAM.

4.4.1 Choix de paramètres

Le paramètre $n_{cliques\_maximales}$ utilisé dans les Algorithmes Alg_1-CHCC et Alg_4-CliqueMaximale est fixé à 100. Pour le calcul du nombre de sommets dans S présents dans les quatre algorithmes, les paramètres α, N_0 et N_{add} sont fixés à $2^{\frac{1}{3}}$, 100 et 20. Le temps total alloué à chaque algorithme est de 900 secondes. Dans la fonction CalculerFonctionLineaire de l'Algorithme 3, l'ensemble discrétisant le domaine $Conv(S[a])$ utilisé dans les Contraintes (4.15) comporte deux paramètres. Le paramètre n^E_{arete} prend la valeur 16 et le paramètre n^E_{int} la valeur 200.

4.4.2 Détermination numérique de paramètres

Le nombre d'échantillons $N_{echantillons}$ pour les $\mathbb{R}$-corridors. Nous avons effectué des expériences numériques pour déterminer une valeur $N_{echantillons}$ d'échantillons à utiliser dans

	4	8	16	32	64	128	256	512	1024
temps moyens relatifs (s)	0.0032	0.0072	0.0156	0.0322	0.0614	0.1280	0.2532	0.5055	1*
nombres moyens d'arêtes relatifs	0.8881	0.9850	0.9990	0.9997	1*	1*	1*	1*	1*

TABLE 4.3: Proportion d'arêtes trouvées et temps de calcul en fonction du paramètre $N_{echantillons}$

la résolution du $\mathbb{R} - LDRFP$. Le but est de déterminer l'influence du paramètre $N_{echantillons}$ sur le nombre d'arêtes d'incompatibilité trouvées et le temps de calcul. Pour cela, nous avons compté le nombre d'arêtes produites par la fonction CALCULARETESTAILLE2 pour toutes les instances pour $|S| = 50$ points dans le domaine. La Table 4.3 présente les résultats. Pour faciliter la comparaison, les temps et nombres d'arêtes obtenus ont été divisés par le temps et le nombre d'arêtes trouvés avec la valeur du paramètre $N_{echantillons}$ fixée à 1024 ; ce sont donc des temps et nombres d'arêtes *relatifs*. Ces valeurs sont arrondies à 10^{-4}.

L'astérisque $*$ indique que c'est la valeur exacte. Le temps relatif semble augmenter linéairement avec le paramètre, ce qui est raisonnable puisque l'algorithme utilisé pour résoudre le $\mathbb{R} - LDRFP$ a une complexité linéaire en le paramètre $N_{echantillons}$ [O'Rourke, 1981]. La Figure 4.4 montre les valeurs de la Table 4.3 dans un graphique dont l'axe des x est en échelle logarithmique pour mieux séparer les valeurs. Le nombre d'arêtes relatif en fonction de $N_{echantillons}$ augmente nettement jusqu'à 16 puis n'augmente presque plus pour de plus grandes valeurs.

Nous choisissons la valeur $N_{echantillons} = 64$ puisque c'est la plus petite valeur pour laquelle CALCULARETESTAILLE2 trouve autant d'arêtes que pour $N_{echantillons} = 1024$. Dans une utilisation où le temps de calcul de CALCULARETESTAILLE2 serait crucial, une valeur plus petite est tout à fait possible. Par exemple, $N_{echantillons} = 16$ s'exécute quatre fois plus vite et identifie pratiquement autant d'arêtes d'incompatibilité reliant deux sommets.

Intérêt des inégalités valides de coloration de graphe dans l'algorithme Alg_1-CHCC. Pour montrer l'intérêt d'utiliser la famille d'inégalités valides issues d'un problème de coloration d'hypergraphe, nous proposons la comparaison de deux algorithmes basés sur Alg_1-CHCC. Ils correspondent à deux variantes de Alg_1-CHCC avec et sans l'utilisation des inégalités valides et sans certaines caractéristiques avancées pour que l'analyse soit moins biaisée. Le premier algorithme que nous appelons SIV pour "sans inégalités valides" correspond à l'Algorithme Alg_1-CHCC sans le calcul des arêtes ni la clique maximale, et pour lequel la fonction MILP est changée. Ce changement correspond à la suppression des contraintes (4.23)-(4.25) bornant les variables a_j^{cor}, b_j^{cor} et c_j^{cor}, les contraintes (4.38) de coloration d'hypergraphe et les contraintes (4.39) et (4.40) fixant certaines variables binaires grâce à la clique maximale calculée. Le deuxième algorithme que nous appelons AIV pour "avec inégalités valides" est comme SIV sauf que les arêtes reliant deux sommets sont calculées et les contraintes (4.38) sont déclarées dans le problème MILP.

ref	δ	SIV	AIV
L1	1.5	3	**4**
	1.0	3	**5**
	0.5	3	**7**
	0.25	4	**10**
	0.1	3	**15**
L2	1.5	3	**5**
	1.0	3	**6**
	0.5	4	**8**
	0.25	3	**14**
	0.1	3	**17**
N1	1.0	**3**	**3**
	0.5	3	**4**
	0.25	3	**5**
	0.1	3	**7**
	0.05	3	**10**
N2	0.1	**1**	**1**
	0.05	**2**	**2**
	0.03	**3**	**3**
	0.01	3	**5**
	0.001	3	**17**
N3	1.0	**2**	**2**
	0.5	**3**	**3**
	0.25	3	**4**
	0.1	3	**6**
	0.05	3	**9**
N4	0.5	**2**	**2**
	0.25	**2**	**2**
	0.1	3	**4**
	0.05	3	**5**
	0.03	3	**7**
N5	1.0	**1**	**1**
	0.5	3	**4**
	0.25	3	**4**
	0.1	3	**7**
	0.05	3	**12**
N6	1.0	**3**	**3**
	0.5	3	**4**
	0.25	3	**5**
	0.1	3	**7**
	0.05	3	**12**
N7	1.0	**1**	**1**
	0.5	**1**	**1**
	0.25	**3**	**3**
	0.1	3	**5**
	0.05	3	**7**

TABLE 4.4: Borne inférieures prouvées en fonction de la présence (AIV) ou non (SIV) d'inégalités valides dans deux algorithmes modifiés de Alg_1-CHCC

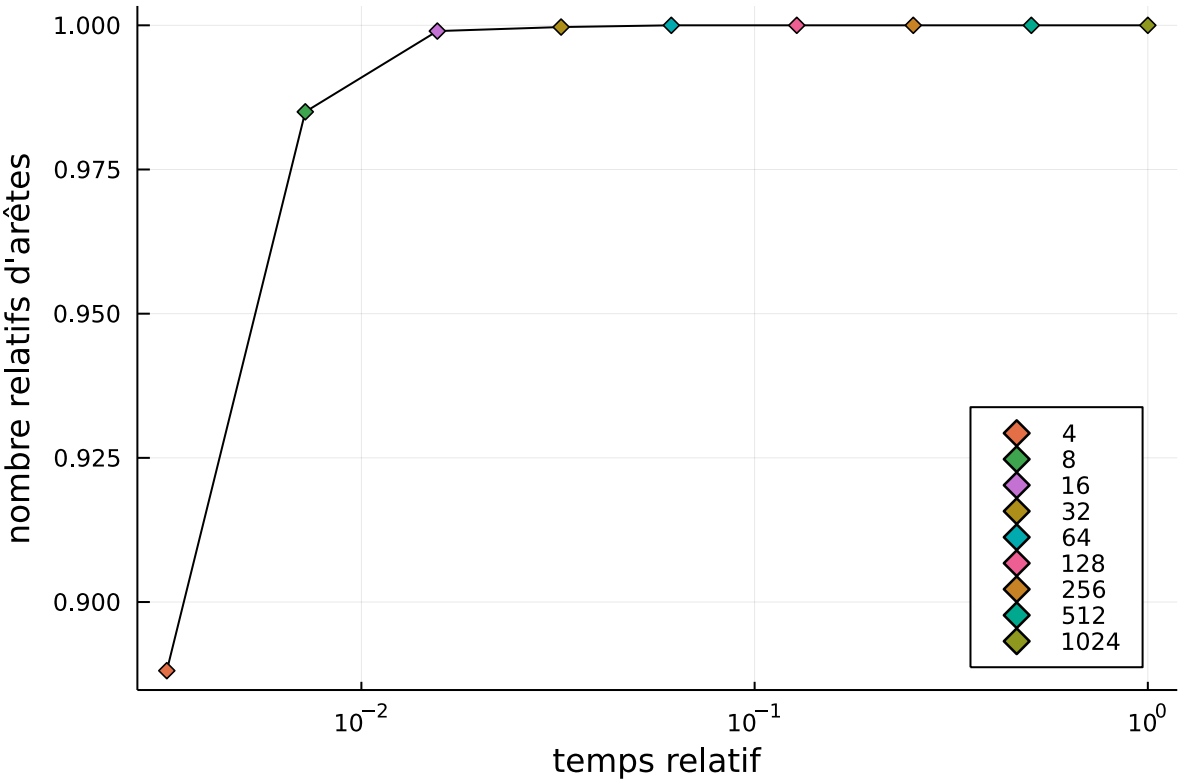


FIGURE 4.4: Proportion d'arêtes trouvées et temps de calcul en fonction du paramètre $N_{echantillons}$

La Table 4.4 montre les résultats sur le jeu d'instances complet. La plus haute borne inférieure pour une instance donnée est marquée en gras. Comme on pouvait s'y attendre, l'Algorithme AIV fournit une borne inférieure au moins aussi bonne que celle de l'Algorithme SIV pour toutes les instances. De plus, l'Algorithme SIV ne fournit que des bornes inférieures $BI \leq 4$. Cela confirme de manière très claire que l'ajout d'inégalités valides du $\mathbb{R}^2$-CFP est bénéfique pour calculer des bornes inférieures de bonne qualité.

Le paramètre $k_{repetition}$ pour l'algorithme Alg_1-CHCC. Nous cherchons expérimentalement quelle valeur du paramètre $k_{repetition}$ est la plus adaptée à l'Algorithme Alg_1-CHCC.

La Table 4.5 montre les résultats de l'Algorithme Alg_1-CHCC pour huit paramétrages différents que l'on identifie par le doublet de paramètres (présence de bornes, $k_{repetition}$). Le paramètre *présence de bornes* vaut "Oui" ou "Non" suivant si l'on déclare ou non les contraintes (4.23)-(4.25) donnant des bornes aux variables qui représentent les paramètres des fonctions linéaires de la fonction PWL. En effet, le calcul de ces bornes prend un temps important à cause du grand nombre de triplets de points de la discrétisation du domaine utilisée. Le paramètre $k_{repetition}$ vaut 0, 1, 4 ou 16. Cette suite de valeurs augmente exponentiellement parce qu'il est

supposé que la différence de performance entre par exemple des valeurs 4 et 5 est négligeable par rapport à la différence de performance entre 0 et 1.

Les deux premières colonnes identifient l'instance avec la fonction et l'erreur d'approximation absolue utilisée. Pour chaque paramétrage, la borne inférieure prouvée est en noir et le temps de calcul en secondes est en gris. Les nombres en gras indiquent les meilleures bornes inférieures par instance lorsque ce ne sont pas les mêmes pour les huit paramétrages différents. Sur les 45 instances, il n'y en a que 5 pour lesquelles les résultats sont différents, et dans ces cas, les bornes inférieures ont une différence inférieure ou égale à 1. Les résultats sont donc relativement proches. Les deux paramétrages avec le plus de meilleures bornes inférieures sont $(Non, 0)$ et $(Oui, 4)$ avec 43 meilleures bornes inférieures sur 45. Celui qui demande le moins de temps pour les instances résolues à l'optimum est $(Non, 0)$, donc c'est ce paramétrage qui est utilisé dans les nouvelles expériences avec l'Algorithme Alg_1-CHCC. Il semble qu'utiliser les inégalités valides provenant d'arêtes reliant au moins trois sommets à travers $k_{repetition} > 0$ ne donne pas un avantage comparé à $k_{repetition} = 0$. Cela peut être dû au fait que le temps de calcul des arêtes reliant au moins trois sommets est important. Ainsi, le temps de calcul qui n'est pas utilisé à calculer des arêtes reliant au moins trois sommets permet à la place de faire plus d'itérations dans la boucle tant que de l'Algorithme Alg_1-CHCC, avec un nombre de sommets N plus grand.

Le paramètre $k_{repetition}$ pour l'algorithme Alg_2-Coloration. Nous cherchons expérimentalement quelle valeur du paramètre $k_{repetition}$ est la plus adaptée à l'Algorithme Alg_2-Coloration. Il est attendu qu'une plus grande valeur de $k_{repetition}$ produise une borne inférieure plus élevée, au prix d'un plus grand temps de calcul. Ainsi, dans le cadre d'un temps de calcul limité, l'effet de ce paramètre est inconnue à priori. La Table 4.6 montre la borne inférieure trouvée par l'Algorithme Alg_2-Coloration pour chaque instance. La première colonne donne le nom de la fonction, la deuxième indique la valeur de δ utilisée pour l'erreur d'approximation absolue, tandis que les quatre dernières colonnes donnent les bornes inférieures obtenues pour quatre valeurs différentes du paramètre $k_{repetition}$: 0, 1, 4 et 16. Les nombres sont en gras pour indiquer les meilleures bornes inférieures par instance lorsque ce ne sont pas les quatre mêmes. Il y a 8 instances sur les 45 pour lesquelles les 4 bornes inférieures ne sont pas identiques. C'est la méthode avec $k_{repetition} = 1$ qui donne le plus grand nombre de meilleures bornes inférieures et donc c'est cette valeur de $k_{repetition}$ que nous utilisons pour la comparaison des algorithmes Section 4.4.3.

4.4.3 Résultats sur le jeu d'instances de la littérature

Nous comparons ici les quatre Algorithmes Alg_1-CHCC, Alg_2-Coloration, Alg_3-CliqueMaximum et Alg_4-CliqueMaximale. En suivant les résultats obtenus dans la Section 4.4.2, le paramètre $k_{repetition}$ vaut 0 pour Alg_1-CHCC, 1 pour Alg_2-Coloration et les contraintes (4.23)-(4.25)

coefs bornés		Non								Oui							
$k_{repetition}$		0		1		4		16		0		1		4		16	
f	δ	BI	time	BI	time	BI	time	BI	time	BI	time	BI	time	BI	time	BI	time
L1	1.5	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL
	1.0	6	35.5	6	62.9	6	66.0	6	175.4	6	46.1	6	51.1	6	100.2	6	210.8
	0.5	8	TL	8	TL	8	TL	8	TL	8	TL	8	TL	8	TL	8	TL
	0.25	12	TL	12	TL	12	TL	12	TL	12	TL	12	TL	12	TL	12	TL
	0.1	22	TL	22	TL	22	TL	22	TL	22	TL	22	TL	22	TL	22	TL
L2	1.5	5	TL	5	TL	5	TL	5	TL	5	TL	5	TL	5	TL	5	TL
	1.0	6	1.4	6	2.0	6	1.6	6	1.7	6	2.4	6	1.6	6	1.7	6	1.6
	0.5	10	TL	10	TL	10	TL	10	TL	10	TL	10	TL	10	TL	10	TL
	0.25	17	TL	17	TL	17	TL	17	TL	17	TL	17	TL	17	TL	17	TL
	0.1	33	TL	33	TL	33	TL	33	TL	33	TL	33	TL	33	TL	33	TL
N1	1.0	3	7.6	3	11.1	3	23.4	3	77.6	3	16.2	3	18.2	3	26.3	3	92.9
	0.5	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL
	0.25	6	TL	6	TL	6	TL	6	TL	6	TL	6	TL	6	TL	6	TL
	0.1	**9**	TL	8	TL	**9**	TL	**9**	TL	**9**	TL	**9**	TL	**9**	TL	**9**	TL
	0.05	12	TL	12	TL	12	TL	12	TL	12	TL	12	TL	12	TL	12	TL
N2	0.1	1	0.0	1	0.0	1	0.0	1	0.0	1	0.0	1	0.0	1	0.0	1	0.0
	0.05	2	2.2	2	2.0	2	2.7	2	2.6	2	3.1	2	2.9	2	2.4	2	2.5
	0.03	3	TL	3	TL	3	TL	3	TL	3	TL	3	TL	3	TL	3	TL
	0.01	6	TL	6	TL	6	TL	6	TL	6	TL	6	TL	6	TL	6	TL
	0.001	**25**	TL	24	TL	24	TL	24	TL	24	TL	24	TL	24	TL	24	TL
N3	1.0	2	TL	2	TL	2	TL	2	TL	2	TL	2	TL	2	TL	2	TL
	0.5	3	TL	3	TL	3	TL	3	TL	3	TL	3	TL	3	TL	3	TL
	0.25	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL
	0.1	8	TL	8	TL	8	TL	8	TL	8	TL	8	TL	8	TL	8	TL
	0.05	11	TL	11	TL	11	TL	11	TL	11	TL	11	TL	11	TL	11	TL
N4	0.5	2	2.7	2	22.7	2	60.5	2	217.4	2	7.6	2	20.7	2	61.4	2	269.4
	0.25	2	TL	2	TL	2	TL	2	TL	2	TL	2	TL	2	TL	2	TL
	0.1	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL
	0.05	6	TL	6	TL	6	TL	6	TL	6	TL	6	TL	6	TL	6	TL
	0.03	8	TL	8	TL	8	TL	8	TL	8	TL	8	TL	8	TL	8	TL
N5	1.0	1	0.0	1	0.0	1	0.0	1	0.0	1	0.0	1	0.0	1	0.0	1	0.0
	0.5	4	12.6	4	28.8	4	61.7	4	163.0	4	14.2	4	26.1	4	48.9	4	209.4
	0.25	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL	4	TL
	0.1	9	TL	9	TL	9	TL	9	TL	9	TL	9	TL	**10**	TL	9	TL
	0.05	14	TL	14	TL	14	TL	14	TL	14	TL	14	TL	14	TL	14	TL
N6	1.0	3	2.5	3	11.0	3	2.6	3	2.3	3	2.4	3	3.3	3	2.3	3	3.3
	0.5	4	1.6	4	1.7	4	1.8	4	2.0	4	1.8	4	3.1	4	1.7	4	1.7
	0.25	5	TL	5	TL	5	TL	5	TL	5	TL	5	TL	5	TL	5	TL
	0.1	9	TL	9	TL	9	TL	9	TL	9	TL	9	TL	9	TL	9	TL
	0.05	13	TL	**14**	TL	13	TL	13	TL	13	TL	13	TL	13	TL	13	TL
N7	1.0	1	0.0	1	0.0	1	0.0	1	0.0	1	0.0	1	0.0	1	0.0	1	0.0
	0.5	1	TL	1	TL	1	TL	1	TL	1	TL	1	TL	1	TL	1	TL
	0.25	3	TL	3	TL	3	TL	3	TL	3	TL	3	TL	3	TL	3	TL
	0.1	**5**	16.1	**5**	56.2	**5**	215.8	**5**	836.2	**5**	27.2	**5**	73.0	**5**	230.8	4	TL
	0.05	7	TL	7	TL	7	TL	7	TL	7	TL	7	TL	7	TL	7	TL

TABLE 4.5: Bornes inférieures et temps en fonction du paramètre $k_{repetition}$ et de la présence ou non de bornes aux coefficients déterminant les fonctions linéaires dans le MILP pour l'Algorithme Alg_1-CHCC

f	δ	$k_{repetition}$ 0	1	4	16
L1	1.5	3	3	3	3
	1.0	5	5	5	5
	0.5	8	8	8	8
	0.25	**14**	13	13	13
	0.1	20	20	20	20
L2	1.5	**5**	**5**	**5**	4
	1.0	6	6	6	6
	0.5	**10**	**10**	9	9
	0.25	**18**	**18**	**18**	17
	0.1	32	32	32	32
N1	1.0	2	2	2	2
	0.5	3	3	3	3
	0.25	5	5	5	5
	0.1	**9**	**9**	**9**	8
	0.05	13	13	13	13
N2	0.1	1	1	1	1
	0.05	2	2	2	2
	0.03	3	3	3	3
	0.01	5	5	5	5
	0.001	23	23	23	23
N3	1.0	2	2	2	2
	0.5	3	3	3	3
	0.25	3	**4**	**4**	**4**
	0.1	7	7	7	7
	0.05	11	11	11	11
N4	0.5	1	**2**	**2**	**2**
	0.25	2	2	2	2
	0.1	4	4	4	4
	0.05	5	5	5	5
	0.03	8	8	8	8
N5	1.0	1	1	1	1
	0.5	3	3	3	3
	0.25	4	4	4	4
	0.1	9	9	9	9
	0.05	13	13	13	**14**
N6	1.0	3	3	3	3
	0.5	4	4	4	4
	0.25	5	5	5	5
	0.1	8	8	8	8
	0.05	15	15	15	15
N7	1.0	1	1	1	1
	0.5	1	1	1	1
	0.25	3	3	3	3
	0.1	4	4	4	4
	0.05	7	7	7	7

TABLE 4.6: Bornes inférieures prouvées en fonction du paramètre $k_{repetition}$ dans Alg_2-COLORATION

du problème MILP de l'Algorithme Alg_1-CHCC ne sont pas prises en compte. La Table 4.7 présente les bornes inférieures obtenues sur chaque instance et pour chaque algorithme pour une limite de temps de 900 secondes. Les noms des algorithmes sont abrégés en Alg_1, Alg_2, Alg_3 et Alg_4 pour la lisibilité de la table. La dernière ligne indique le nombre d'instances où chacun des algorithmes obtient la plus grande borne inférieure parmi les quatre algorithmes.

L'algorithme Alg_1-CHCC obtient une grande majorité du temps la meilleure solution. Les autres algorithmes sont peut-être moins performants parce que les problèmes qu'ils résolvent ont perdu encore plus d'informations au fil des relaxations successives du $\mathbb{R}^2$-CFP. Par exemple, les contraintes de corridor aux points $x \in I$ ne sont plus présentes.

Les 5 instances pour lesquelles Alg_1-CHCC ne produit pas la meilleure borne inférieure sont des instances où δ est parmi les deux plus petits pour cette fonction. Cela indique que c'est plutôt pour de "grandes" instances que l'algorithme est moins performant par rapport aux autres. Nous n'avons pas d'explications à ce fait.

Solutions optimales du $\mathbb{R}^2$-CFP. Nous rappelons que ce chapitre décrit les premières méthodes pour établir des bornes inférieures au $\mathbb{R}^2$-CFP. Les bornes inférieures établies par les algorithmes Alg_1-CHCC, Alg_2-Coloration, Alg_3-CliqueMaximum et Alg_4-CliqueMaximale ainsi que les bornes supérieures connues sur le jeu d'instances de [Rebennack et Kallrath, 2015a] nous permettent de prouver que certaines solutions sont des solutions optimales du $\mathbb{R}^2$-CFP. La Table 4.8 montre les meilleures bornes inférieures et supérieures trouvées dans les Tables 3.10 et 4.7.

Les résultats en bleu indiquent que la borne supérieure est égale à la borne inférieure et donc qu'une solution optimale à l'instance a été trouvée. Cela concerne 15 instances, même si pour 3 d'entres elles, qui ont une borne supérieure de 1, une solution optimale était déjà prouvée. Ce sont majoritairement des "petites" instances au sens où la valeur de δ utilisée dans l'instance fait partie des 2 plus grandes valeurs de δ pour cette fonction, et elles n'ont pas plus de 6 pièces. Bien entendu, une analyse des instances permettrait probablement de prouver l'optimalité d'instances supplémentaires. Par exemple, pour l'instance utilisant la fonction $N7$ et $\delta = 0.5$: la fonction PWL $g(x) = 0.5$ est une solution à une seule pièce puisque les valeurs de $N7$ sont incluses dans $[0, 1]$. Toutefois, ce n'est pas l'objectif du travail de ce chapitre que de trouver des solutions optimales à partir d'une analyse spécifique à une fonction.

Le plus grand ratio BS/BI est $57/13 \approx 4.4$ et a lieu pour l'instance avec la fonction $N1(x, y) = xy$ et $\delta = 0.05$. Il serait intéressant de trouver une fonction non linéaire pour laquelle on connaît le nombre de pièces minimum pour des valeurs très différentes de δ afin d'observer les différences avec les bornes inférieures et supérieures que les algorithmes présentés dans les deux derniers chapitres produisent. Cela pourrait donner des pistes pour leur amélioration.

f	δ	Alg_1	Alg_2	Alg_3	Alg_4
L1	1.5	**4**	3	3	3
	1.0	**6**	5	5	4
	0.5	**8**	**8**	**8**	7
	0.25	12	**13**	12	12
	0.1	22	20	21	**23**
L2	1.5	**5**	**5**	**5**	**5**
	1.0	**6**	**6**	**6**	**6**
	0.5	**10**	**10**	9	9
	0.25	17	**18**	15	17
	0.1	**33**	32	29	**33**
N1	1.0	**3**	2	2	2
	0.5	**4**	3	3	3
	0.25	**6**	5	4	4
	0.1	**9**	**9**	8	7
	0.05	12	**13**	11	10
N2	0.1	**1**	**1**	**1**	**1**
	0.05	**2**	**2**	**2**	**2**
	0.03	**3**	**3**	**3**	**3**
	0.01	**6**	5	5	4
	0.001	**25**	23	22	24
N3	1.0	**2**	**2**	**2**	**2**
	0.5	**3**	**3**	**3**	**3**
	0.25	**4**	**4**	3	3
	0.1	**8**	7	7	6
	0.05	**11**	**11**	10	10
N4	0.5	**2**	**2**	1	1
	0.25	**2**	**2**	**2**	**2**
	0.1	**4**	**4**	3	3
	0.05	**6**	5	4	4
	0.03	**8**	**8**	7	6
N5	1.0	**1**	**1**	**1**	**1**
	0.5	**4**	3	3	3
	0.25	**4**	**4**	**4**	**4**
	0.1	**9**	**9**	8	8
	0.05	**14**	13	13	13
N6	1.0	**3**	**3**	**3**	**3**
	0.5	**4**	**4**	**4**	**4**
	0.25	**5**	**5**	**5**	**5**
	0.1	**9**	8	8	8
	0.05	13	**15**	14	12
N7	1.0	**1**	**1**	**1**	**1**
	0.5	**1**	**1**	**1**	**1**
	0.25	**3**	**3**	**3**	**3**
	0.1	**5**	4	4	4
	0.05	**7**	**7**	**7**	**7**
# meilleurs		**40**	30	18	19

TABLE 4.7: Bornes inférieures produites par les algorithmes issues des quatres relaxations

f	δ	BI	BS
L1	1.5	4	6
	1.0	6	6
	0.5	8	15
	0.25	13	21
	0.1	23	60
L2	1.5	5	6
	1.0	6	6
	0.5	10	14
	0.25	18	21
	0.1	33	60
N1	1.0	3	3
	0.5	4	6
	0.25	6	12
	0.1	9	28
	0.05	13	57
N2	0.1	1	1
	0.05	2	2
	0.03	3	4
	0.01	6	10
	0.001	25	88
N3	1.0	2	2
	0.5	3	3
	0.25	4	6
	0.1	8	15
	0.05	11	27
N4	0.5	2	2
	0.25	2	3
	0.1	4	6
	0.05	6	11
	0.03	8	17
N5	1.0	1	1
	0.5	4	4
	0.25	4	5
	0.1	9	21
	0.05	14	43
N6	1.0	3	3
	0.5	4	4
	0.25	5	6
	0.1	9	15
	0.05	15	28
N7	1.0	1	1
	0.5	1	2
	0.25	3	3
	0.1	5	5
	0.05	7	10

TABLE 4.8: Meilleures bornes inférieures et supérieures du $\mathbb{R}^2$-CFP trouvées dans les Chapitres 3 et 4

4.5 Conclusion

Dans ce chapitre, nous avons étudié le $DCFP_H^{\mathcal{C}}$, une relaxation du problème $\mathbb{R}^2$-CFP basée sur la discrétisation du domaine du corridor. Pour cela, nous avons d'abord identifié une famille d'inégalités valides du $\mathbb{R}^2$-CFP et montré comment les calculer en pratique. Puis nous avons proposé trois autres problèmes qui sont des relaxations successives du $DCFP_H^{\mathcal{C}}$ et qui utilisent ces inégalités valides, faisant intervenir notamment les problèmes de coloration d'hypergraphe, de clique maximum et de clique maximale. Enfin, nous avons construit un algorithme pour chacune de ces relaxations.

Dans une comparaison expérimentale, c'est l'algorithme issu de la relaxation la plus forte, utilisant un problème MILP avec des contraintes de coloration d'hypergraphe et de convexité sur la forme des pièces, qui fournit les meilleurs résultats. Les bornes inférieures trouvées à ce chapitre et les bornes supérieures du Chapitre 3 permettent de prouver que l'on a identifié des solutions optimales de 15 des 45 instances du jeu d'instances de [Rebennack et Kallrath, 2015a], dont 12 pour la première fois.

Nous proposons deux pistes pour améliorer les calculs de bornes inférieures du $\mathbb{R}^2$-CFP. Premièrement, l'ajout de points dans la discrétisation du domaine à la manière de la méthode KL21 a l'air prometteur. Deuxièmement, toujours dans l'idée d'un processus itératif, il est possible de se servir des pièces des fonctions PWL solutions de bornes inférieures ne respectant pas les contraintes du $\mathbb{R}^2$-CFP pour obtenir de nouvelles inégalités valides qui interdiraient la dernière solution.

Chapitre 5

Cas particulier de la linéarisation par morceaux de la norme euclidienne de $\mathbb{R}^2$

Sommaire

Dans les deux chapitres précédents, nous avons décrit des méthodes calculant des bornes supérieures et inférieures au $\mathbb{R}^2$-CFP dans le cas général. Dans ce chapitre, nous étudions une méthode de linéarisation de la norme euclidienne de $\mathbb{R}^2$ dans un problème MINLP et en déduisons des solutions optimales du $\mathbb{R}^2$-CFP pour ce cas particulier. Nous nous appuyons pour cela sur le travail fait par [Camino et al., 2019], qui permet d'approximer une classe de problèmes MINLP non convexes par un MILP.

Notre travail sur ce sujet concerne l'approximation de fonctions dont le domaine est le plan $\mathbb{R}^2$ tout entier, qui n'est pas un polytope parce qu'il n'est pas borné. Donc nous utilisons une définition de fonction PWL légèrement différente de celle donnée dans la Section 1.2 : les fonctions PWL de ce chapitre sont définies sur le plan $\mathbb{R}^2$ et sont formées de pièces qui sont des polyèdres au lieu d'être des polytopes. Cela correspond à changer la famille D_i de polytopes par une famille de polyèdres. Signalons qu'une partie des formulations MILP classiques des fonctions PWL de domaine un polytope n'est pas applicable sur un domaine qui est un polyèdre. C'est notamment le cas des formulations qui utilisent explicitement les points extrémaux des pièces. En effet, les polyèdres qui ne sont pas des polytopes ne peuvent pas s'exprimer comme une combinaison convexe d'un nombre fini de points.

Les problèmes MINLP faisant intervenir la norme euclidienne de $\mathbb{R}^2$ peuvent par exemple contenir des objectifs ou des contraintes de distances entre objets, ou bien des contraintes d'appartenance ou de non-appartenance à un disque. Comme applications récentes, nous pouvons citer le positionnement d'infrastructures de service de réseau 5G [Santoyo-González et Cervelló-Pastor, 2018], le déploiement de nœuds de relais dans des réseaux sans fil [Zhou et al., 2018] et l'optimisation de placement de faisceaux en télécommunication satellite [Camino et al., 2014, 2019].

Le chapitre est organisé comme suit. La Section 5.1 reprend la linéarisation de la norme euclidienne de [Camino et al., 2019] et prouve qu'elle fournit un encadrement de l'ensemble réalisable d'une contrainte spécifique grâce à un ensemble de contraintes linéaires avec le nombre minimum de contraintes pour satisfaire une erreur d'approximation particulière. De plus, la Section 5.2 montre que cette linéarisation peut être utilisée dans la fonction objectif d'un problème MINLP, et qu'elle peut être adaptée à des normes dont les lignes de niveaux sont des ellipses. Dans le Chapitre 6 nous montrerons l'intérêt pratique de tous ces résultats en linéarisant des contraintes décrivant des zones elliptiques sur un problème de télécommunication satellite de placement de faisceaux.

5.1 Linéarisation de la norme euclidienne

Dans cette section, nous décrivons la méthode de linéarisation de la norme euclidienne de $\mathbb{R}^2$ introduite par [Camino et al., 2019], puis nous présentons nos contributions : nous définissons une erreur d'approximation spécifique à cette méthode, et nous prouvons que cette linéarisation peut être utilisée pour construire un encadrement linéaire par morceaux avec le nombre de pièces minimum satisfaisant cette erreur d'approximation. Pour le reste du manuscrit, le terme *linéarisation de la norme euclidienne* désigne la méthode de linéarisation de la norme euclidienne de $\mathbb{R}^2$ proposée par [Camino et al., 2019], malgré le fait qu'il existe d'autres linéarisations spécifiques à la norme euclidienne, dont certaines d'ailleurs sont décrites dans [Camino et al., 2019].

5.1.1 Description de la méthode

La linéarisation de la norme euclidienne est basée sur l'évaluation de plusieurs produits scalaires entre le point auquel est appliqué la norme euclidienne et une famille de vecteurs unitaires particulière. Nous rappelons que $\|\cdot\|_2$ désigne la norme euclidienne de $\mathbb{R}^2$. Pour $x \in \mathbb{R}^2$ et Δ dans $\mathbb{R}^+$, les contraintes possibles sont :

$$\|x\|_2 \leq \Delta, \tag{5.1}$$

$$\|x\|_2 \geq \Delta. \tag{5.2}$$

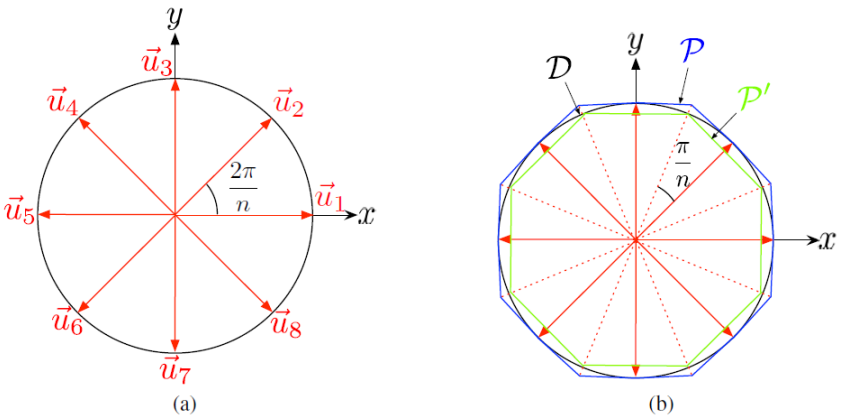


FIGURE 5.1: Linéarisation de la norme euclidienne pour 8 directions Camino et al. [2019]

Linéarisation de $\|x\|_2 \leq \Delta$. La contrainte (5.1) est satisfaite si x est dans un disque de rayon Δ centré en l'origine. Soit p le paramètre de désignant le nombre de produits scalaires qui sont évalués pour la linéarisation de la norme euclidienne. Nous définissons les vecteurs unitaires $u_i := (\cos \frac{2i\pi}{p}, \sin \frac{2i\pi}{p})$ pour $i = 1, ..., p$. Ils sont représentés sur la Figure 5.1(a). La linéarisation de la norme euclidienne consiste à remplacer la contrainte (5.1) soit par les contraintes

$$x.u_i \leq \Delta \quad \forall i = 1, ..., p, \tag{5.3}$$

soit par les contraintes

$$x.u_i \leq \Delta \cos \frac{\pi}{p} \quad \forall i = 1, ..., p, \tag{5.4}$$

où . désigne le produit scalaire usuel. Si les contraintes (5.3) sont satisfaites, alors le point x est dans le polygone bleu de la Figure 5.1(b), et selon ([Camino et al., 2019, Proposition 1]) la borne supérieure suivante sur $\|x\|_2$ est satisfaite :

$$\|x\|_2 \leq \frac{\Delta}{\cos \frac{\pi}{p}}. \tag{5.5}$$

De plus, si l'une des contraintes linéaires de (5.3) n'est pas satisfaite, alors le point x n'est pas dans le polygone bleu et ([Camino et al., 2019, Proposition 2]) donne

$$\|x\|_2 \geq \Delta. \tag{5.6}$$

Les contraintes (5.3) sont une relaxation de la contrainte (5.1), car il existe au moins un point x satisfaisant les contraintes (5.3) mais ne satisfaisant pas la contrainte (5.1). À l'inverse, il est

aussi possible d'obtenir des contrainte plus fortes que la contrainte (5.1) en la linéarisant avec les contraintes (5.4), qui ont le polygone vert de la Figure 5.1(b) comme ensemble réalisable.

Remarquons que les polygones créés par la linéarisation de la contrainte (5.1) en les contraintes (5.3) ou les contraintes (5.4) forcent le point x à être dans un polygone régulier à p côtés si $p \geq 3$, parce que leurs ensembles réalisables sont convexes, fermés et bornés, et de frontière un polygone équiangulaire et équilatéral.

Linéarisation de $\|x\|_2 \geq \Delta$. La linéarisation de la norme euclidienne permet aussi de linéariser la contrainte (5.2), mais elle diffère de celle de la contrainte (5.1) parce que ce n'est pas une contrainte convexe. Les contraintes linéarisant (5.2) sont

$$x.u_i \geq \Delta - M_i(1 - b_i), \quad \forall i = 1, ..., p, \tag{5.7}$$

$$\sum_{i=1}^{p} b_i = 1, \tag{5.8}$$

$$b_i \in \{0, 1\}, \quad \forall i = 1, ..., p, \tag{5.9}$$

qui utilisent la méthode du big-M mais dont le principe reste le même que pour les contraintes (5.3) ou (5.4). Ici, le réel positif M_i doit être une borne supérieure valide de $\Delta - x.u_i$, qui peut être facilement obtenu si x est borné.

Le polygone bleu de la Figure 5.1 a l'air d'être le "plus petit" polygone régulier à 8 côtés contenant le disque noir parce que chaque point au milieu d'un côté touche le bord du disque et que les vecteurs unitaires u_i sont répartis uniformément parmi les vecteurs unitaires. A l'inverse, le polygone vert a l'air d'être le "plus grand" polygone régulier à 8 côtés d'aire maximale à l'intérieur du disque noir parce que chacun de ces sommets touche la frontière du disque. Ces intuitions guident la preuve que la linéarisation de la norme euclidienne utilise le plus petit nombre de pièces possible satisfaisant l'erreur d'approximation décrite dans la Section 5.1.2. Dans la suite, nous nous concentrons sur l'approximation de la frontière des ensembles réalisables des contraintes $\|x\|_2 \leq \Delta$ et $\|x\|_2 \geq \Delta$. Par abus de langage, nous dirons que des contraintes *sont une forme géométrique* en faisant référence à la forme de l'ensemble de points satisfaisant ces contraintes. Dans la Figure 5.1, le polygone bleu est appelé une *approximation extérieure* du disque noir, tandis que le polygone vert est appelé une *approximation intérieure* du même disque. Ces deux termes sont définis plus proprement dans la Section 5.1.2.

5.1.2 Erreur d'approximation spécifique

Nous définissons une erreur d'approximation adaptée à la linéarisation de la norme euclidienne dans les contraintes avec la Définition 43, basée sur les ensembles satisfaisant la contrainte original ou son approximation. Intuitivement, cette erreur mesure le plus grand rapport de norme

euclidienne entre deux points de l'ensemble égal à la différence ensembliste du disque approximé et du polygone qui l'approxime. Si un polygone approximant le disque le contient, il est appelé une approximation extérieure du disque ; s'il est inclut dans le disque, il est appelé une approximation intérieure du disque. La définition suivante est illustrée après son énoncé dans la Figure 5.2.

Définition 43 (approximation intérieure, approximation extérieure, encadrement)**.** Soient le vecteur $x_0 \in \mathbb{R}^2$, le disque $B(x_0, \Delta) = \{x \in \mathbb{R}^2 | \|x - x_0\|_2 \leq \Delta\}$ de rayon Δ et de centre x_0 avec $\Delta \in \mathbb{R}^+$. Soient le polygone P^- et le réel positif $a^- \in \mathbb{R}^+$ tels que $P^- \subset B(x_0, \Delta)$ et $a^- := \inf\{\|x - x_0\|_2 | x \in B(x_0, \Delta) \backslash P^-\}$. Soient le polygone P^+ et le réel positif $a^+ \in \mathbb{R}^+$ tels que $B(x_0, \Delta) \subset P^+$ et $a^+ := \sup\{\|x - x_0\|_2 | x \in P^+ \backslash B(x_0, \Delta)\}$. Nous appelons :

— P^- une *approximation intérieure* du disque $B(x_0, \Delta)$ d'erreur $\epsilon := \frac{\Delta}{a^-} - 1$,

— P^+ une *approximation extérieure* du disque $B(x_0, \Delta)$ d'erreur $\epsilon := \frac{a^+}{\Delta} - 1$,

— P^+ et P^- un *encadrement* du disque $B(x_0, \Delta)$ d'erreur $\epsilon := \max\{\frac{a^+}{\Delta}, \frac{\Delta}{a^-}\} - 1$.

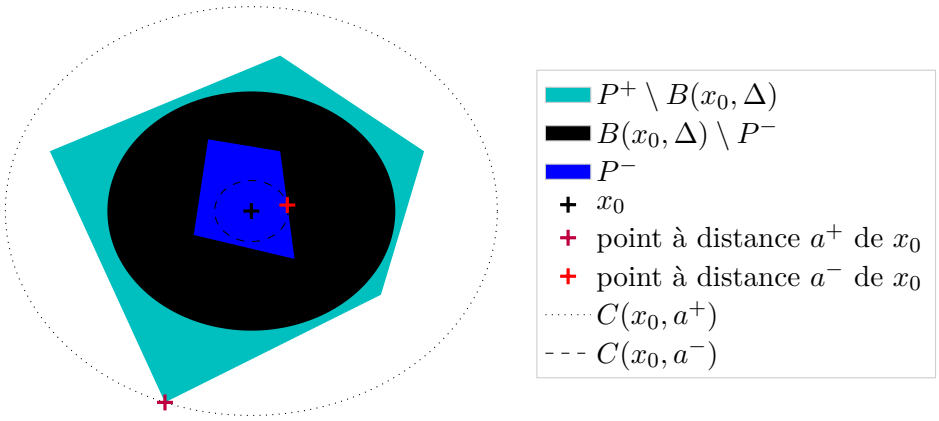


FIGURE 5.2: Illustration de la Définition 43

Le terme -1 de la formule de calcul de l'erreur ϵ est ajouté pour qu'une approximation d'erreur $\epsilon = 0$ soit en fait sans erreur. Cette erreur d'approximation est une erreur dans le pire cas puisqu'elle se base sur le point x pour lequel la norme euclidienne de $x - x_0$ est le plus éloigné de Δ, le rayon du disque à approximer. Une définition similaire peut être faite pour des contraintes $\|x - x_0\|_2 \geq \Delta$.

Dans la Figure 5.2, le polygone bleu est une approximation intérieure du disque centré en x_0 contenant les zones noire et bleue. L'erreur d'approximation intérieure est représentée par la plus grande proportion de norme euclidienne entre deux points de la différence des deux ensembles, c'est-à-dire la zone noire : la croix rouge est le point de plus petite norme euclidienne a^- tandis qu'un point de plus grande norme euclidienne est sur le cercle de rayon Δ donc il est de norme Δ. D'après la Définition 43 cette erreur d'approximation intérieure vaut alors $\frac{\Delta}{a^-} - 1$.

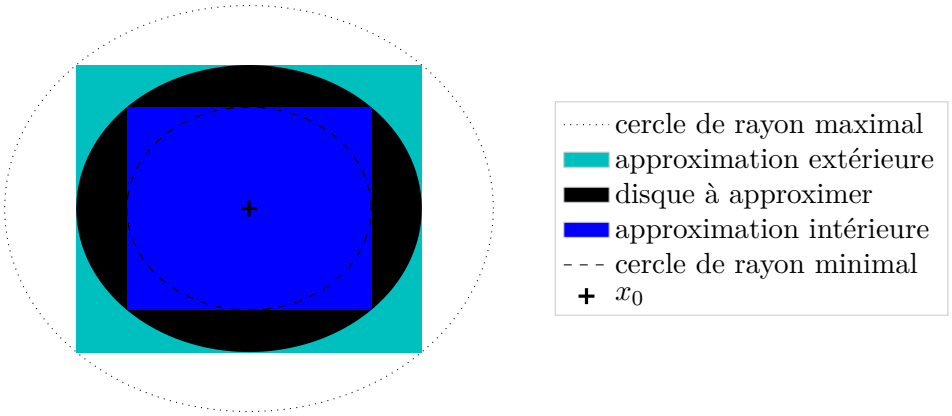


FIGURE 5.3: Encadrement du disque par les contraintes (5.3) et (5.4) pour $p = 4$ produits scalaires

De la même manière, pour l'erreur d'approximation extérieure, le polygone regroupant les zones bleue, noire et cyan est une approximation extérieure du disque noir. Le point de plus grande norme euclidienne dans la différence ensembliste du polygone et du disque est la croix violette de norme a^+. Inversément, un point de plus petite norme est sur le cercle de rayon Δ. L'erreur d'approximation extérieure est alors $\frac{a^+}{\Delta} - 1$. En ce qui concerne l'encadrement du disque noir par les deux polygones, il fournit une erreur d'approximation qui est le maximum des erreurs d'approximation intérieure et extérieure.

La Figure 5.3 illustre les approximations intérieures et extérieures issues de la linéarisation de la norme euclidienne. Les carrés cyan et bleu représentant les ensembles réalisables induits par la linéarisation des contraintes (5.3) et (5.4) pour $p = 4$. Ils forment un encadrement du disque regroupant les zones bleue et noire. La norme maximale du carré d'approximation extérieure (resp. la norme minimale du carré d'approximation intérieure) est donnée par le rayon du disque en pointillés (resp. le cercle en tirets). Ainsi, selon la Définition 43, l'erreur d'approximation de cet encadrement est le maximum entre les deux rapports *rayon du cercle en pointillés* sur *rayon du disque noir* et *rayon du disque noir* sur *rayon du cercle en tiret*, le tout moins un.

Un calcul donne que les erreurs d'approximation associées aux contraintes (5.3) et (5.4) sont toutes les deux $\frac{1}{\cos\frac{\pi}{p}} - 1$ parce que les normes euclidiennes des points dans les différences ensemblistes entre le disque à approximer et les approximations extérieures et intérieures sont $[\Delta, \frac{\Delta}{\cos\frac{\pi}{p}}]$ et $[\Delta\cos\frac{\pi}{p}, \Delta]$ respectivement. Nous faisons la remarque que l'erreur d'approximation tend vers 0 lorsque le nombre de produits scalaires p tend vers l'infini, et donc que la linéarisation de la norme euclidienne peut être aussi précise que nécessaire en choisissant p assez grand.

5.1.3 Approximation intérieure

Dans cette section, nous montrerons que, pour un nombre fixé de côtés p, les polygones réguliers dont tous les sommets sont sur la frontière du disque sont les seuls polygones à p côtés produisant l'erreur minimale d'approximation intérieure d'un disque. Nous appelons *polygone inscrit dans un cercle* un polygone dont les sommets sont tous sur un cercle donné. Pour simplifier le discours, nous disons qu'un polygone est *inscrit dans un disque* s'il est inscrit dans le cercle constituant la frontière du disque. Aussi, nous notons *disque unité* un disque de rayon 1, et *disque centré* un disque centré en l'origine du plan $(0, 0)$. Enfin, nous introduisons une notation utile pour l'erreur d'approximation d'un disque par un polygone : Soit $\mathcal{P}$ un polygone à l'intérieur d'un disque $\mathcal{D}$ centré. Nous notons $\epsilon(\mathcal{P})$ l'erreur d'approximation du disque $\mathcal{D}$ par le polygone $\mathcal{P}$ suivant la Définition 43. Le contexte permet alors de savoir si nous parlons de l'erreur d'approximation intérieure, extérieure ou de l'erreur d'encadrement. Dans cette section, c'est l'approximation intérieure.

Lemme 44. *Soit $\mathcal{D}$ le disque unité centré. Soit $\mathcal{P} \subset \mathcal{D}$ un polygone non convexe simple et soit $\mathcal{P}' = Conv(\mathcal{P})$ l'enveloppe convexe de $\mathcal{P}$. Alors, $\mathcal{P}'$ est inclus dans $\mathcal{D}$, $\mathcal{P}'$ a strictement moins de côtés que $\mathcal{P}$ et les erreurs d'approximation intérieure des polygones vérifient $\epsilon(\mathcal{P}) \geq \epsilon(\mathcal{P}')$.*

Démonstration. Premièrement, $\mathcal{P} \subset \mathcal{D}$ implique que $\mathcal{P}' \subset \mathcal{D}$ parce que $\mathcal{D}$ est convexe et que $\mathcal{P}' = Conv(\mathcal{P})$ est le plus petit convexe contenant $\mathcal{P}$. Deuxièmement, un polygone est convexe si chaque angle (orienté vers l'intérieur) de ce polygone est plus petit que π. Comme $\mathcal{P}$ n'est pas convexe, son enveloppe convexe $\mathcal{P}'$ a strictement moins de côtés parce qu'au moins un sommet de $\mathcal{P}$ a un angle strictement plus grand que π et donc ne sera pas un sommet de $\mathcal{P}'$. Troisièmement, $\mathcal{P} \subset \mathcal{P}' = Conv(\mathcal{P})$ implique que l'ensemble $\mathcal{D}\backslash\mathcal{P}'$ sur lequel est calculé le minimum a^- de la Définition 43 est inclus dans l'ensemble $\mathcal{D}\backslash\mathcal{P}$, et donc que $\epsilon(\mathcal{P}) \geq \epsilon(Conv(\mathcal{P}))$. □

Selon le Lemme 44, un polygone non convexe à p côtés ne pourra pas approximer mieux un disque que son enveloppe convexe. Donc, il y existe un polygone convexe à p côtés qui produit l'erreur d'approximation minimale du disque. Nous notons $\mathcal{A}_p^-$ l'ensemble des polygones convexes à p côtés inclus dans le disque unité centré $\mathcal{D}$. Le symbole "$-$" désigne l'approximation intérieure et nous considérons $p \geq 3$ pour que les polygones soient bien définis. Précisons que le disque peut être supposé de rayon 1 et centré sans perte de généralité et pour simplifier les développements, puisque de toute façon le nombre de côtés d'un polygone est invariant par homothétie et translation.

Le but dans la suite de la section est de montrer le Théorème 45.

Theorem 45. *Les polygones réguliers à p côtés inscrits dans le disque unité centré sont les*

seules solutions optimales du problème d'optimisation $(\mathcal{P}_p^-)$ *:*

$$(\mathcal{P}_p^-) \quad \begin{cases} \min & \epsilon(\mathcal{P}) \\ s.c. & \mathcal{P} \in \mathcal{A}_p^-. \end{cases}$$

Nous introduisons une transformation spécifique de polygone comme premier argument pour démontrer la validité de ce théorème. Elle utilise le terme *intérieur d'un ensemble* qui désigne le plus grand ouvert inclus dans cet ensemble et que l'on note $Int(\cdot)$.

Définition 46 (gonflement)**.** Soient un polygone $\mathcal{P} \in \mathcal{A}_p^-$ dont on note les sommets v_i pour $i = 1, ..., p$ et un point x dans $Int(\mathcal{P})$. On appelle *gonflement de* $\mathcal{P}$ *par* x le polygone $\mathcal{P}_{gonf(x)}$ dont les sommets w_i pour $i = 1, ..., p$ sont l'intersection du cercle frontière du disque unité centré $\mathcal{D}$ et de la demi-droite partant du point x dans la direction du sommet v_i.

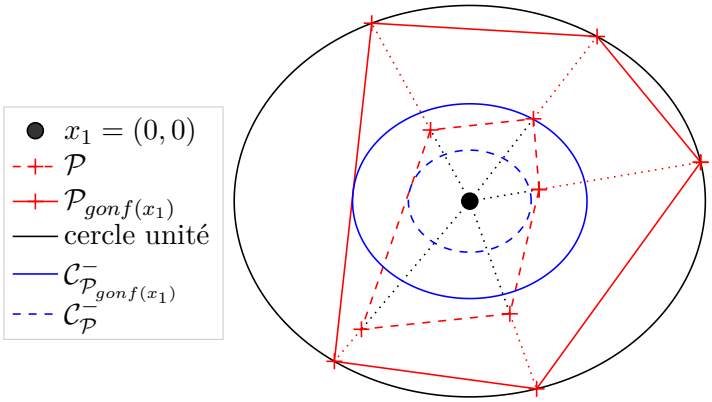


FIGURE 5.4: gonflement de $\mathcal{P}$ par $x_1 = (0, 0)$

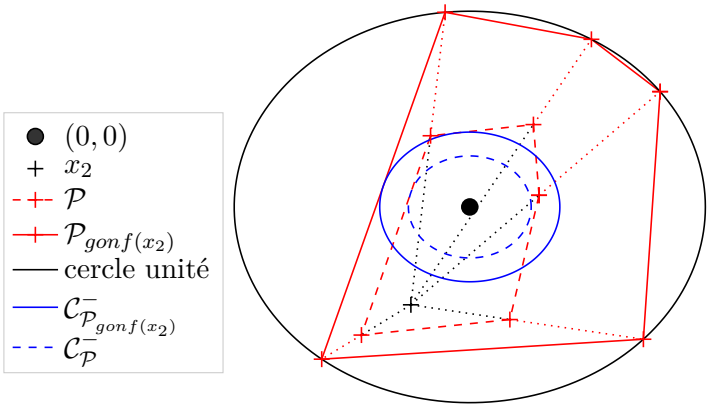


FIGURE 5.5: gonflement de $\mathcal{P}$ par $x_2 = (-0.25, -0.5)$

Les Figures (5.4) et (5.5) illustrent la Définition 46. Elles montrent deux gonflements du

polygone rouge en tirets $\mathcal{P}$ par les points $x_1 = (0, 0)$ et $x_2 = (-0.25, -0.5)$ respectivement. Les polygones obtenus par gonflement $\mathcal{P}_{gonf(x_1)}$ et $\mathcal{P}_{gonf(x_2)}$ sont en rouge, avec des traits en pointillés pour montrer la construction de leurs sommets par rapport aux sommets du polygone $\mathcal{P}$. Finalement, les cercles bleus en tirets et continus montrent la norme minimale de points des polygones $\mathcal{P}$ et de leur gonflement par x_1 ou x_2 respectivement.

Lemme 47. *Soit l'entier $p \geq 3$ et le polygone $\mathcal{P}_{gonf(x)}$ obtenu par gonflement du polygone $\mathcal{P} \in \mathcal{A}_p^-$ par le point $x \in Int(\mathcal{P})$. On a :*

1. *$\mathcal{P}_{gonf(x)}$ est un polygone simple,*
2. *$Int(\mathcal{P}) \subset Int(\mathcal{P}_{gonf(x)})$,*
3. *les sommets $\mathcal{P}_{gonf(x)}$ sont sur le cercle,*
4. *$\mathcal{P}_{gonf(x)}$ est convexe,*
5. *$\mathcal{P}_{gonf(x)}$ a autant de côtés que $\mathcal{P}$,*
6. *$\epsilon(\mathcal{P}_{gonf(x)}) \leq \epsilon(\mathcal{P})$,*

Démonstration. 1. Par définition, un polygone est un ensemble de points du plan délimité par une chaîne fermée de segments. Ces segments sont aussi appelés côtés. Un polygone est dit simple si deux de ses côtés ne s'intersectent qu'en leurs extrémités s'ils sont voisins dans la chaîne. Comme les sommets de $\mathcal{P}_{gonf(x)}$ sont construits à partir des sommets de $\mathcal{P}$, les côtés de $\mathcal{P}_x$ sont obtenus comme ceux induits par les côtés de $\mathcal{P}$ et nous obtenons bien une chaîne fermée de côtés, donc $\mathcal{P}_{gonf(x)}$ est un polygone. Pour montrer qu'il est aussi simple, il suffit de montrer que les intersections aux sommets sont entre deux côtés voisins, et qu'il n'y a pas d'intersection de côtés en-dehors des sommets. Remarquons que les p demi-droites utilisées pour construire le gonflement s'intersectent en x. De plus, comme le polygone $\mathcal{P}$ est simple, il est possible de diviser le plan $\mathbb{R}^2$ en p secteurs angulaires, où un secteur angulaire est associé à un côté de la manière suivante : il est construit avec les deux demi-droites issues des deux sommets qui sont les extrémités d'un côté de $\mathcal{P}$. Pour construire les sommets de $\mathcal{P}_{gonf(x)}$, les sommets sont déplacés dans ce même secteur angulaire, donc les seules intersections de côtés sont entre deux côtés voisins en leur sommet commun.
2. Les sommets de $\mathcal{P}$ utilisés pour construire ceux de $\mathcal{P}_{gonf(x)}$ sont forcément "déplacés" dans une direction opposée au point x, donc $\mathcal{P} \subset \mathcal{P}_{gonf(x)}$, d'où l'inclusion des intérieurs.
3. Par définition du gonflement.
4. Il suffit de montrer que l'angle en chaque sommet de $\mathcal{P}_{gonf(x)}$ a un angle donnant sur l'intérieur du polygone plus petit que π radian. Considérons la tangente au cercle en un sommet s de $\mathcal{P}_{gonf(x)}$. Elle sépare le plan en deux demi-plans dont celui contenant x contient aussi $\mathcal{P}_{gonf(x)}$ car il contient entre autre le cercle. Cela implique que l'angle en s n'est pas plus grand que π radian, sinon au moins un des sommets voisin de s ne serait pas dans le même demi-plan.

5. Par définition, $\mathcal{P}_{gonf(x)}$ n'a pas plus de p sommets. Par contradiction, supposons que $\mathcal{P}_{gonf(x)}$ a au moins un sommet de moins que $\mathcal{P}$. Alors, il existe un sommet v' de $\mathcal{P}_{gonf(x)}$ qui a été obtenu à partir de deux sommets différents v_1 et v_2 de $\mathcal{P}$. Cela veut dire que ces deux sommets sont sur la même demi-droite partant de x. Donc v_1 (ou v_2) est une combinaison convexe de x et v_2 (ou v_1), ce qui est impossible car x est dans l'intérieur de $\mathcal{P}$.
6. Par le Point 2, nous savons que $\mathcal{D}\backslash Int(\mathcal{P}') \subset \mathcal{D}\backslash Int(\mathcal{P})$, donc $\min\{\|x\|_2 | x \in \mathcal{D}\backslash Int(\mathcal{P}')\} \geq \min\{\|x\|_2 | x \in \mathcal{D}\backslash Int(\mathcal{P})\}$, ce qui amène au point 6.

□

Les points 3, 4, 5 et 6 du Lemme 47 montrent qu'il y a un polygone inscrit dans le cercle qui est solution du problème $(\mathcal{P}_p^-)$. Ainsi, nous pouvons nous concentrer sur de tels polygones pour trouver les solutions du problème. L'erreur d'approximation est calculée avec les points de normes minimales de $\mathcal{D}\backslash Int(\mathcal{P})$. Le Lemme 48 explicite où se situent ces points dans l'ensemble $\mathcal{D}\backslash Int(\mathcal{P})$.

Lemme 48. *Soit $\mathcal{P}$ un polygone de l'ensemble $\mathcal{A}_p^-$ inscrit dans le disque unité centré $\mathcal{D}$. Si l'origine $(0,0)$ est dans l'intérieur de $\mathcal{P}$, alors un point de norme minimale $x \in \mathcal{D}\backslash Int(\mathcal{P})$ est p_m, le point au milieu du côté le plus long de $\mathcal{P}$.*

Démonstration. Il est clair qu'un point de norme minimale est sur un côté de $\mathcal{P}$ puisque l'origine $(0,0)$ est dans l'intérieur de $\mathcal{P}$ et que l'on cherche dans l'ensemble $\mathcal{D}\backslash Int(\mathcal{P})$. Chaque côté du polygone $\mathcal{P}$ a des extrémités de norme euclidienne 1, et un calcul donne que le point de norme minimale d'un côté avec cette propriété est le point au milieu p_m. Nous nous concentrons sur un côté. Nous créons un triangle avec pour sommets l'origine et les deux sommets associés au côté considéré. Notons α l'angle du triangle au sommet qui est l'origine. Grâce à la trigonométrie, l'expression de la norme du point p_m au milieu du côté de $\mathcal{P}$ est la fonction $g(\alpha) := \cos\frac{\alpha}{2}$. Comme $\alpha \in]0, \pi[$, nous avons $g'(\alpha) < 0$ et donc la norme de p_m décroît lorsque α augmente. Or, ceci est équivalent à : la norme de p_m décroît lorsque la longueur du côté augmente. □

L'hypothèse faite pour le Lemme 48 n'est pas limitante car si l'origine n'est pas dans l'intérieur du polygone $\mathcal{P}$, alors l'erreur d'approximation de $\mathcal{P}$ est infinie, ce qui n'est clairement pas une solution de $(\mathcal{P}_p^-)$.

Lemme 49. *Les solutions optimales de $(\mathcal{P}_p^-)$ sont les polygones réguliers à p côtés et ont pour valeur optimale $\frac{1}{\cos\frac{\pi}{p}} - 1$.*

Démonstration. Les points 3, 4, 5 and 6 du Lemme 47 donnent qu'il y a un polygone inscrit dans le disque qui est solution de $(\mathcal{P}_p^-)$. Considérons une partition du polygone $\mathcal{P}$ en p triangles chacun associé à un côté de la même manière que dans la preuve du Lemme 48. Soit $(\alpha_1, ..., \alpha_p)$ les angles de ces triangles en l'origine $(0,0)$. Avoir le plus petit plus long côté revient à avoir le

plus petit plus grand angle α_i, donc une solution de $(\mathcal{P}_p^-)$ qui est inscrite dans le disque doit aussi être solution du problème d'optimisation

$$\min_{\mathcal{P}\in\mathcal{A}_p^-} \max_{i=1,...,p} \alpha_i, \tag{5.10}$$

qui minimise l'angle maximum α_i du polygone $\mathcal{P}$. La seule solution est $\alpha_i = \frac{2\pi}{p},\ i = 1, ..., p$ qui veut dire que les p triangles ont le même angle en l'origine $(0, 0)$ et donc le polygone a p segments de même longueur. Or, un polygone équilatéral et inscrit dans le disque ne peut être qu'un polygone régulier à p côtés. Donc chaque polygone régulier à p côtés est une solution réalisable de $(\mathcal{P}_p^-)$ de valeur $\frac{1}{\cos\frac{\pi}{p}} - 1$ grâce à un calcul de la norme des points au milieu des côtés. De plus, un polygone inscrit dans le cercle qui n'est pas régulier n'est pas solution du problème 5.10. Ainsi, la valeur optimale est $\frac{1}{\cos\frac{\pi}{p}} - 1$ et les polygones réguliers à p côtés inscrits dans le disque sont les seules solutions optimales. □

Ce Lemme conclut la preuve du Théorème 45.

5.1.4 Minimalité du nombre de contraintes

Le Théorème 50 énonce que la linéarisation de la norme euclidienne permet de produire l'encadrement d'un disque à l'erreur d'approximation demandée, par deux polygones qui utilisent le nombre minimum de côtés.

Theorem 50. *Soit $\epsilon_0 > 0$ une erreur d'approximation à respecter par l'approximation d'un disque $\mathcal{D}$ par un polygone. Soit l'entier positif p tel que*

$$p = \min\{k \in \mathbb{N}^* | \frac{1}{\cos\frac{\pi}{k}} - 1 \leq \epsilon_0, k \geq 3\}.$$

Un polygone contenu dans (resp. contenant) le disque $\mathcal{D}$ satisfaisant une erreur d'approximation ϵ_0 a au moins p côtés, et l'approximation intérieure (resp. extérieure) du disque avec p produits scalaires donnés par la linéarisation de la norme euclidienne produit un polygone avec p côtés satisfaisant l'erreur d'approximation ϵ_0.

La preuve du Théorème 50 traite séparément l'approximation intérieure et l'approximation extérieure. Le cas de l'approximation intérieure est montrée, puis une justification que l'approximation extérieure se traite d'une manière similaire est proposée.

Démonstration. Le Lemme 49 donne que l'erreur d'approximation minimale respectée par un polygone à n côtés est croissante en n, donc p est le nombre minimum de côtés d'un polygone satisfaisant une erreur d'approximation ϵ_0. Comme l'approximation intérieure du disque à p produits scalaires par la linéarisation de la norme euclidienne correspond à un polygone régulier à p côtés, elle satisfait bien l'erreur d'approximation ϵ_0.

Si $\mathcal{D}$ n'est pas le disque unité, il existe une transformation linéaire qui donne le disque unité centré lorsque elle est appliquée au disque $\mathcal{D}$. La même transformation appliquée au polygone donné par la linéarisation de la norme euclidienne ne changera pas son nombre de côtés, et l'erreur d'approximation du disque unité centré par ce nouveau polygone sera la même, donc le résultat précédent est valide pour n'importe quel disque.

Finalement, le cas de l'approximation extérieure se traite d'une manière similaire. La différence majeure est qu'il faut remplacer la Définition 46 sur le gonflement d'un polygone par une définition de *dégonflement* d'un polygone : les côtés du polygone d'approximation extérieure sont rapprochés du point à l'intérieur du disque jusqu'à toucher la frontière du disque, tout en restant parallèle à leur position initiale pour donner le polygone dégonflé. L'utilisation des angles en l'origine des triangles partitionnant le polygone est remplacée par les angles directement en les sommets du polygone. Cela permet de montrer qu'un polygone régulier dont le milieu des côtés touchent le bord du disque est solution du problème similaire à $(\mathcal{P}_p^-)$ pour le cas de l'approximation extérieure. □

5.2 Extensions de la linéarisation de la norme euclidienne

Dans cette section, nous proposons deux extensions de la linéarisation de la norme euclidienne pour lesquelles le Théorème 50 est transposable. La première extension, expliquée dans la Section 5.2.1, est la linéarisation de la norme euclidienne de $\mathbb{R}^2$ directement dans la fonction objectif, et non plus à l'intérieur d'une contrainte. Nous construisons une fonction PWL utilisant le nombre minimum de pièces pour une erreur d'approximation demandée. Ce résultat est basé sur la propriété d'absolue homogénéité de la norme euclidienne et sur le Théorème 50. La deuxième extension décrite dans la Section 5.2.2 permet de remplacer une contrainte utilisant une norme avec des lignes de niveaux en forme d'ellipse d'une manière équivalente à la linéarisation de la norme euclidienne. Ce résultat est basé sur le fait qu'une ellipse peut s'obtenir comme transformation linéaire d'un cercle. Une application de ce résultat est traitée dans la Section 6.1. Finalement, la Section 5.2.3 présente les liens entre les deux extensions proposées et la résolution du $\mathbb{R}^2$-CFP pour des corridors particuliers.

5.2.1 Linéarisation dans la fonction objectif

Après avoir introduit l'équation paramétrique d'un *cône de révolution* que nous abrégeons en *cône* pour la section, nous construisons un encadrement linéaire par morceaux de la norme euclidienne de $\mathbb{R}^2$ utilisant le moins de pièces possible pour une erreur d'approximation donnée.

La surface $\mathcal{C}$ générée par la norme euclidienne de $\mathbb{R}^2$ est définie par

$$\begin{aligned}\mathcal{C} &= \{(x_1, x_2, x_3) | x_3 = \|(x_1, x_2)\|_2, x_1 \in \mathbb{R}, x_2 \in \mathbb{R}\} \\ &= \{(x_1, x_2, x_3) \in \mathbb{R}^3 | x_1^2 + x_2^2 - x_3^2 = 0, x_3 \geq 0\},\end{aligned} \tag{5.11}$$

qui est l'intersection d'un cône avec le demi-espace $x_3 \geq 0$. Nous appelons $\mathcal{C}$ le *cône positif*. Avant d'introduire l'encadrement du cône positif, nous rappelons la définition des vecteurs unitaires régulièrement espacés u_i utilisés dans la linéarisation de la norme euclidienne, avec un entier $p \geq 3$:

$$u_i = \left(\cos\frac{2i\pi}{p}, \sin\frac{2i\pi}{p}\right) \text{ for } i = 1, ..., p. \tag{5.12}$$

Soient les ensembles $\mathcal{C}_p^+$ et $\mathcal{C}_p^-$ définis comme

$$\mathcal{C}_p^+ = \{(x_1, x_2, f_p^+(x_1, x_2)) | x = (x_1, x_2) \in \mathbb{R}^2\} \text{ et} \tag{5.13}$$

$$\mathcal{C}_p^- = \{(x_1, x_2, f_p^-(x_1, x_2)) | x = (x_1, x_2) \in \mathbb{R}^2\} \tag{5.14}$$

avec les fonctions $f_p^+(x_1, x_2)$ et $f_p^-(x_1, x_2)$ définies comme

$$f_p^+(x_1, x_2) = \max_{i=1,...,p} \frac{(x_1, x_2)^T . u_i}{\cos\frac{\pi}{p}}, \tag{5.15}$$

$$f_p^-(x_1, x_2) = \max_{i=1,...,p} (x_1, x_2)^T . u_i. \tag{5.16}$$

La surface induite par l'approximation intérieure de p produits scalaires est appelée $\mathcal{C}_p^+$ parce que c'est une *surestimation* de $\mathcal{C}$ dans le sens où pour un point $(x_1, x_2) \in \mathbb{R}^2$ donné, avec x_3 tel que $(x_1, x_2, x_3) \in \mathcal{C}$ et x_3^+ tel que $(x_1, x_2, x_3^+) \in \mathcal{C}_p^+$, x_3^+ est supérieur à x_3.

De manière similaire, la surface induite par l'approximation extérieure de p produits scalaires est une *sous-estimation* de $\mathcal{C}$ et est notée $\mathcal{C}_p^-$. Les termes de surestimation, sous-estimation et d'encadrement sont définis plus précisément dans la Définition 51. Nous introduisons juste avant la notion de *ligne de niveau*. Nous appelons ligne de niveau de la fonction f de valeur $v \in \mathbb{R}$ l'ensemble des points en lesquels la fonction f prend la valeur v. La Définition 51 introduit du vocabulaire lié à l'encadrement d'une fonction à une erreur d'approximation près.

Définition 51 (sous-estimation, surestimation, encadrement)**.** Soient la fonction f de $\mathbb{R}^2$ dans $\mathbb{R}^+$, l'ensemble $F_0 := \{x \in \mathbb{R}^2 | f(x) = 0\}$ la ligne de niveau de la fonction f de valeur 0 et les fonctions f^-, f^+ de $\mathbb{R}^2$ dans $\mathbb{R}^+$ qui ont pour lignes de niveaux de valeur 0 l'ensemble F_0. Si la fonction f^- satisfait

$$\exists \epsilon > 0,\ \forall x \in \mathbb{R}^2 \setminus F_0,\ \frac{f(x) - f^-(x)}{f^-(x)} \in [0, \epsilon], \tag{5.17}$$

alors c'est une *sous-estimation* de la fonction f d'erreur d'approximation ϵ. Si la fonction f^+ satisfait

$$\exists \epsilon > 0,\ \forall x \in \mathbb{R}^2 \setminus F_0,\ \frac{f^+(x) - f(x)}{f(x)} \in [0, \epsilon], \tag{5.18}$$

alors c'est une *surestimation* de la fonction f d'erreur d'approximation ϵ. De plus, si les fonctions f^- et f^+ satisfont respectivement (5.17) et (5.18), alors elles forment un *encadrement* de la fonction f d'erreur d'approximation ϵ. Finalement, si f^- et f^+ sont en plus des fonctions linéaires par morceaux, elles forment un *encadrement linéaire* de f.

Les fonctions présentes dans les dénominateurs des équations (5.17) et (5.18) permettent d'obtenir une valeur de l'erreur positive.

Remarque 52. L'erreur d'approximation utilisée dans la définition de surestimation est l'erreur d'approximation relative. Par contre, celle utilisée dans la définition de sous-estimation n'est pas l'erreur d'approximation relative car sinon la propriété à vérifier serait :

$$\exists \epsilon > 0,\ \forall x \in \mathbb{R}^2 \setminus F_0,\ \frac{f(x) - f^-(x)}{f(x)} \in [0, \epsilon]. \tag{5.19}$$

Un complément à cette discussion est donné dans la Section 5.2.3.

Proposition 53. *Soit $p \in \mathbb{N}$, $p \geq 3$. La norme euclidienne $\|x\|_2$ est encadrée linéairement par f_p^- et f_p^+ avec l'erreur d'approximation $\frac{1}{\cos\frac{\pi}{p}} - 1$.*

Démonstration. Nous décrivons ici la preuve que la fonction f est surestimée par f_p^+ avec l'erreur d'approximation $\frac{1}{\cos\frac{\pi}{p}} - 1$. Le cas pour la sous-estimation par f_p^- se traite de manière similaire. Prouver la propriété

$$\frac{f_p^+(x)}{\|x\|_2} \in [1, \frac{1}{\cos\frac{\pi}{p}}], \quad \forall x \in \mathbb{R}^2 \tag{5.20}$$

est suffisant puisque c'est équivalent à la propriété (5.18) pour $f^+(x) = f_p^+(x)$, $f(x) = \|x\|_2$ et $\epsilon = \frac{1}{\cos\frac{\pi}{p}} - 1$.

Remarquons que l'ensemble $\mathcal{C}_p^+$ est stable par rotation d'angle $\frac{2\pi}{p}$ autour de l'axe x_3 parce que le polygone régulier à p côtés utilisé pour construire $\mathcal{C}_p^+$ a aussi cette propriété. Donc prouver la propriété (5.20) pour le point x dans le secteur angulaire $X_p = \{(\alpha\cos\beta, \alpha\sin\beta) | \alpha \geq 0, \beta \in [-\frac{\pi}{p}, \frac{\pi}{p}]\}$ est suffisant puisque l'ensemble $Z_p^+ := \{(x_1, x_2, f_p^+(x_1, x_2)) | (x_1, x_2) \in X_p\}$ contient les points dans une face de l'approximation intérieure $\mathcal{C}_p^+$. De plus, l'équation

$$Z_p^+ = \{(\alpha_+\cos\frac{\pi}{p}, \beta_+, \alpha_+) | \alpha_+ \geq 0, \beta_+ \in [-\alpha_+\sin\frac{\pi}{p}, \alpha_+\sin\frac{\pi}{p}]\} \tag{5.21}$$

montre une paramétrisation de la face Z_p^+. Les vecteurs dans l'ensemble Z_p^+ peuvent s'écrire de manière unique comme $\alpha_+ u + \beta_+ v$ avec les réels $\alpha_+ \geq 0$ et β_+ dans l'intervalle $[-\alpha_+ \sin\frac{\pi}{p}, \alpha_+ \sin\frac{\pi}{p}]$. Soit $x \in X_p$.

Nous allons montrer l'inégalité $1 \leq \frac{f_p^+(x)}{\|x\|_2} \leq \frac{1}{\cos\frac{\pi}{p}}$. Soient $\alpha \geq 0$ et $\beta \in [-\frac{\pi}{p}, \frac{\pi}{p}]$ tels que $x = (\alpha\cos\beta, \alpha\sin\beta)$. Nous savons que $f(x) = \alpha$. Pour trouver le point $(\alpha_+ \cos\frac{\pi}{p}, \beta_+, \alpha_+)$ de Z_p^+ qui a pour deux premières coordonnées x, il suffit de résoudre le système linéaire

$$\begin{cases} \alpha\cos\beta = \alpha_+ \cos\frac{\pi}{p} \\ \alpha\sin\beta = \beta_+ \end{cases} \tag{5.22}$$

qui a pour solution

$$\begin{cases} \alpha_+ = \alpha\frac{\cos\beta}{\cos\frac{\pi}{p}} \\ \beta_+ = \alpha\sin\beta, \end{cases} \tag{5.23}$$

et ainsi :

$$\frac{f^+(x)}{\|x\|_2} = \frac{\cos\beta}{\cos\frac{\pi}{p}} \in [1, \frac{1}{\cos\frac{\pi}{p}}] \quad \forall\beta \in [-\frac{\pi}{p}, \frac{\pi}{p}] \tag{5.24}$$

et la borne supérieure de l'intervalle $[1, \frac{1}{\cos\frac{\pi}{p}}]$ est atteinte pour $\beta = 0$. Ainsi, l'erreur d'approximation est $\epsilon = \frac{1}{\cos\frac{\pi}{p}}$. □

Corollaire 54. *Soit l'entier positif $p \geq 3$ tel que l'erreur d'approximation ϵ à respecter est dans l'intervalle $[\frac{1}{\cos\frac{\pi}{p}}, \frac{1}{\cos\frac{\pi}{p-1}}[$. Alors l'encadrement linéaire de la norme euclidienne par f_p^- et f_p^+ utilise le nombre minimum de pièces.*

Démonstration. Le nombre de côtés du polygone induit par la linéarisation de la norme euclidienne pour une contrainte est minimum d'après le Théorème 50. De plus, les surestimations f_p^+ et sous-estimations f_p^- de l'encadrement linéaire de la norme euclidienne utilisent le même nombre de pièces pour la même valeur d'erreur d'approximation. Comme les fonctions f_p^+ et f_p^- sont construites à partir du polygone, le nombre de pièces du polygone est une borne inférieure au nombre de pièces de f_p^+ et f_p^-, borne inférieure qui est donc atteinte. □

Ainsi, la linéarisation de la norme euclidienne peut être utilisée pour calculer un encadrement linéaire de la norme euclidienne de $\mathbb{R}^2$ qui utilise le nombre de pièces minimum. La Figure 5.6 montre un tel exemple pour $p = 4$ avec $\mathcal{C}$ en rouge, $\mathcal{C}_p^-$ en cyan et $\mathcal{C}_p^+$ en bleu.

5.2.2 Extension aux normes de lignes de niveaux elliptiques

Nous montrons ici qu'il est possible d'adapter la linéarisation de la norme euclidienne à des contraintes de même forme que les contraintes (5.1) et (5.2) faisant apparaître des normes

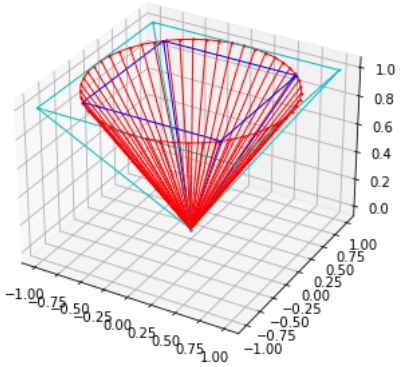


FIGURE 5.6: Encadrement linéaire du cône positif, vu de dessus, pour $p = 4$ produits scalaires

qui ont des lignes de niveaux qui sont des ellipses. Pour simplifier la notation, nous appelons de telles contraintes des *contraintes elliptiques*. Pour cela, nous commençons par introduire des définitions utiles pour manipuler les ellipses, puis nous décrivons comment construire les contraintes approximant une contrainte elliptique à une erreur d'approximation donnée.

Nous appelons *ellipse* $\mathcal{E}$ une courbe plane fermée contenant les points pour lesquels la somme des distances à deux points fixes (les *foyers*) est égale à une certaine valeur. *L'axe focal* est la droite passant par les deux foyers. Nous appelons *sommets principaux* les deux points de l'ellipse intersectant l'axe focal, et *sommets secondaires* les deux points de l'ellipse intersectant la médiatrice du segment reliant les deux foyers. De plus, nous nommons *grand axe* (resp. *petit axe*) le segment reliant les sommets principaux (resp. secondaires) tout comme sa longueur. L'intersection du grand axe et du petit axe est le *centre de l'ellipse*. La Figure 5.7 illustre une partie du vocabulaire sur l'ellipse. Le point rouge appartient à l'ellipse et la longueur additionnée des deux traits rouges en pointillés reliant ce point aux foyers est la même pour chaque point de l'ellipse.

Nous pouvons maintenant définir l'ellipse d'une manière équivalente, qui est plus pratique pour nos développements. Nous appelons ellipse $\mathcal{E}$ de centre (u, v), de demi grand axe a et de demi petit axe b l'ensemble de $\mathbb{R}^2$ tel que

$$\frac{(x-u)^2}{a^2} + \frac{(y-v)^2}{b^2} = 1 \quad \forall (x, y) \in \mathbb{R}^2. \tag{5.25}$$

L'axe focal de l'ellipse et l'axe des abscisses peuvent s'intersecter avec un angle $\theta \neq 0$, auquel cas nous disons que l'ellipse est tournée d'un angle θ, et son équation s'écrit maintenant

$$\frac{((x-u)\cos\theta + (y-v)\sin\theta)^2}{a^2} + \frac{(-(x-u)\sin\theta + (y-v)\cos\theta)^2}{b^2} = 1$$
$$\forall (x, y) \in \mathbb{R}^2. \tag{5.26}$$

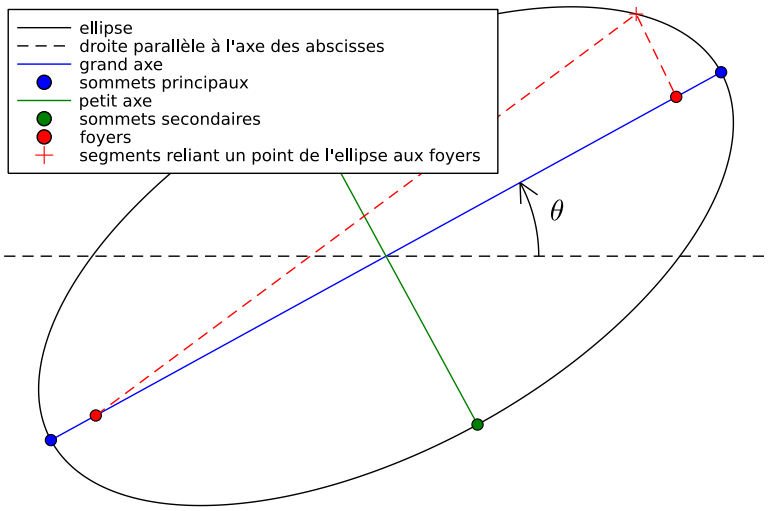


FIGURE 5.7: Représentation du vocabulaire de l'ellipse

Dans cette partie, les résultats sur l'erreur d'approximation de la linéarisation de la norme euclidienne sont étendus aux termes utilisant des normes dont les lignes de niveaux sont des ellipses. Nous notons une telle norme $\|\cdot\|_{a,b,\theta}$, définie par

$$\|x\|^2_{a,b,\theta} = \frac{(x_1 \cos\theta + x_2 \sin\theta)^2}{a^2} + \frac{(-x_1 \sin\theta + x_2 \cos\theta)^2}{b^2}$$
$$\forall x = (x_1, x_2) \in \mathbb{R}^2,\ a, b \in \mathbb{R}^+_*. \tag{5.27}$$

Comme l'ellipse est une courbe, et que l'on souhaite approximer des contraintes du type "x est à l'intérieur d'une ellipse", où l'intérieur de l'ellipse est une surface, le terme ellipse pourra suivant les besoins désigner la courbe ou la surface correspondant à l'intérieur de la courbe. Nous définissons quelques termes supplémentaires pour nos besoins :

— la *forme* d'une ellipse est notée (a, b) où a et b sont les demi grand axe et demi petit axe respectivement (donc $a \geq b$),
— l'angle θ d'une ellipse désigne l'angle entre l'axe des abscisses et le grand axe, avec $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$.

Remarquons que n'importe quelle ellipse non dégénérée (a,$b > 0$) centrée en (u, v), de forme (a, b) et d'angle θ peut être obtenue comme l'image par la transformation linéaire $\phi_{a,b,\theta}$ d'un

cercle de rayon 1 et centré en l'origine, auquel nous ajoutons la translation $(u, v)^T$, avec

$$\phi_{a,b,\theta} : \mathbb{R}^2 \to \mathbb{R}^2 \tag{5.28}$$

$$(x, y) \mapsto (ax\cos\theta - by\sin\theta, ax\sin\theta + by\cos\theta) \tag{5.29}$$

$$= \begin{pmatrix} x & y \end{pmatrix} \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}, \tag{5.30}$$

qui est la composition d'une homothétie $(x, y) \mapsto (ax, by)$ et d'une rotation d'angle θ. La transformation linéaire $\phi_{a,b,\theta}$ admet un inverse linéaire puisque a et b sont strictement positifs, et préserve les ratios de normes entre vecteurs colinéaires puisqu'une application linéaire vérifie que $f(\lambda x) = \lambda f(x)$ pour tout vecteur x et scalaire $\lambda \in \mathbb{R}$. Ainsi, le polygone régulier à p côtés obtenu par la linéarisation de la norme euclidienne est, après transformation par $\phi_{a,b,\theta}$, un polygone qui utilise le minimum de côtés pour satisfaire l'erreur d'approximation d'une ellipse centrée en (u, v), de forme (a, b) et d'angle θ avec p choisie comme dans le Théorème 50. La Figure 5.8 montre le passage de la linéarisation d'un disque à la linéarisation d'une ellipse.

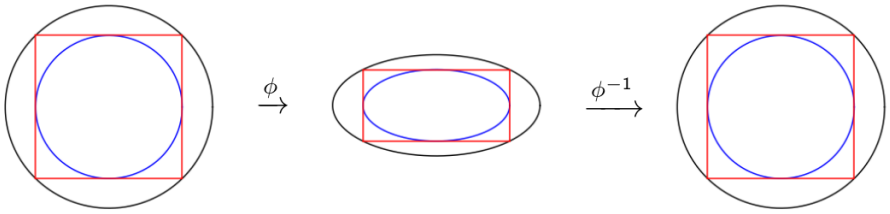


FIGURE 5.8: Transformation linéaire entre un cercle et une ellipse

Cette analyse montre que les contraintes utilisant des normes elliptiques peuvent aussi être linéarisées par la linéarisation de la norme euclidienne en ajoutant un changement de base adapté. La procédure pour linéariser la contrainte $\|x\|_{a,b,\theta} \leq \Delta$ est donc de la remplacer par les contraintes :

$$\phi^{-1}_{a,b,\theta}(x).u_i \leq d\Delta \quad \forall i = 1, ..., p, \tag{5.31}$$

où d est égal à $\frac{1}{\cos\frac{\pi}{p}}$ pour une approximation intérieure, et 1 pour une approximation extérieure. Un désavantage de cette linéarisation est que a, b et θ doivent être des constantes du modèle pour que les contraintes introduites soient linéaires. Cela veut dire qu'appliquer la linéarisation de la norme euclidienne à la contrainte "x est dans l'ellipse non dégénérée $\mathcal{E}$", lorsque la forme et l'angle de l'ellipse $\mathcal{E}$ ne sont pas déjà fixés et se modélisent à l'aide de variables, donnera des contraintes non linéaires. Une manière d'éviter cette non-linéarité est de prédéfinir les ellipses possibles. En revanche, cela génère un grand nombre de contraintes : s'il y a n ellipses différentes et p produits scalaires, il faudra np contraintes avec des big-M, au lieu de p contraintes sans big-

M pour linéariser le disque pour la même erreur d'approximation. Cette linéarisation n'est donc pas bien adaptée pour une discrétisation fine des ellipses. La Section 6.1 montre une application où la linéarisation d'une contrainte de type "x est dans une ellipse" est utile.

5.2.3 Résultats pour le $\mathbb{R}^2$-CFP

Les résultats des Sections 5.2.1 et 5.2.2 permettent de résoudre des cas particuliers du $\mathbb{R}^2$-CFP. Le Corollaire 54 est reformulé en termes de $\mathbb{R}^2$-CFP dans la Remarque 55, et la linéarisation de la fonction $\|x\|_{a,b,\theta}$ pour $x \in \mathbb{R}^2$ est réinterprétée dans la Remarque 56.

Remarque 55. Soient l'erreur relative $\epsilon > 0$ et l'entier positif $p \geq 3$ tel que ϵ est dans l'intervalle $[\frac{1}{\cos\frac{\pi}{p}} - 1, \frac{1}{\cos\frac{\pi}{p-1}} - 1[$. Le $\mathbb{R}^2$-CFP dans le corridor $Corridor(\|\cdot\|_2, (1+\epsilon)\|\cdot\|_2, \mathbb{R}^2)$ a pour solution optimale f_p^+. Ce corridor est issu d'une erreur d'approximation relative de la norme euclidienne de $\mathbb{R}^2$. La sous-estimation f_p^- n'est pas issue d'une erreur d'approximation relative comme expliqué dans la Remarque 52. De plus, les résultats sur f_p^- ne permettent pas de déduire un corridor issu de l'approximation d'une fonction par une autre erreur, car le corridor correspondant serait alors défini en partie par la fonction PWL f_p^-.

Remarque 56. Soient l'erreur relative $\epsilon > 0$, l'entier positif $p \geq 3$ tel que ϵ est dans l'intervalle $[\frac{1}{\cos\frac{\pi}{p}}, \frac{1}{\cos\frac{\pi}{p-1}}[$ et la fonction

$$T_{a,b,\theta} : \mathbb{R}^3 \to \mathbb{R}^3 \tag{5.32}$$

$$(x, y, z) \mapsto (\phi_{a,b,\theta}(x, y), z). \tag{5.33}$$

Le $\mathbb{R}^2$-CFP dans le corridor $Corridor(\|\cdot\|_{a,b,\theta}, (1+\epsilon)\|\cdot\|_{a,b,\theta}, \mathbb{R}^2)$ a pour solution optimale $T_{a,b,\theta}(f_p^+(x, y))$.

5.3 Conclusion

Ce travail sur la linéarisation de contraintes et/ou objectifs de problèmes MINLP non convexes propose des résultats théoriques qui s'appuient sur une linéarisation de la norme euclidienne de [Camino et al., 2019]. Nous avons montré que, étant donné une erreur d'approximation à respecter, cette linéarisation utilise le nombre minimum de produits scalaires pour approximer une contrainte d'appartenance ou de non-appartenance à un disque. De plus, la linéarisation de la norme euclidienne permet de déduire une solution optimale au $\mathbb{R}^2$-CFP lorsque le corridor est issu de l'approximation de la norme euclidienne et d'une erreur relative. Finalement, cette linéarisation permet aussi de linéariser des contraintes faisant apparaître une norme dont les lignes de niveaux sont des ellipses, ce qui permet d'utiliser la linéarisation de la norme euclidienne dans une plus grande classe de problèmes MINLP.

Ce travail peut être continué de plusieurs façons. D'une part, il serait intéressant de l'étendre à d'autres fonctions que les normes traitées dans ce chapitre. D'autre part, les contraintes produites par la linéarisation de la norme euclidienne pourraient être utilisées comme coupes dans une méthode de plans sécants pour mieux contrôler la taille du modèle MILP à résoudre.

Chapitre 6

Applications à la résolution de problèmes d'optimisation mixte non linéaire en nombres entiers

Sommaire

Dans le chapitre précédent, nous avons étudié une méthode de linéarisation de la norme euclidienne dans un problème MINLP. Dans ce chapitre, nous présentons deux applications de travaux de linéarisation par morceaux à erreur d'approximation bornée pour résoudre des problèmes MINLP. La première application traite d'un problème de placement de faisceaux satellites sur la Terre. Des contraintes qui font intervenir la norme euclidienne ou des normes de lignes de niveaux elliptiques sont approximées grâce aux résultats du Chapitre 5. Dans la deuxième application, un problème de théorie des jeux pour calculer des investissements en cybersécurité est modélisé. Une méthode de calcul d'équilibres de Nash approchés est développée et appliquée à ce problème.

6.1 Application au problème de placement de faisceaux

Nous montrons l'intérêt de notre extension de la linéarisation de la norme euclidienne grâce à un problème MINLP utilisant des contraintes d'appartenance ou de non-appartenance à une ellipse, que nous dénommons comme dans le Chapitre 5 des *contraintes elliptiques*. Il s'agit d'un problème de placement de faisceaux dans le domaine des télécommunications satellites. Le problème original a déjà été traité dans [Camino et al., 2014, 2016, 2019] sans contraintes

elliptiques. [Camino et al., 2019] présente une revue de littérature de ce problème. De manière informelle, un satellite multifaisceaux doit satisfaire le maximum de trafic d'un certain service en orientant ses faisceaux sur la surface terrestre. La Figure 6.1 illustre cette situation. Dans le problème original, la portion de surface terrestre qu'un faisceau peut desservir est un disque. Ainsi, une contrainte d'appartenance à un disque (ou contrainte de proximité) permet d'assurer qu'un utilisateur est bien desservi. Il y a aussi des contraintes de séparations entre certains faisceaux, donnant des contraintes de non appartenance à des disques. Par contre, les systèmes multifaisceaux modernes sont capables de couvrir une portion de surface terrestre différente d'un disque, puisque des faisceaux reconfigurables permettent d'obtenir d'autres formes, comme des ellipses [Rao et al., 2006]. Ajouter la possibilité d'utiliser des faisceaux de forme elliptique permet d'améliorer le potentiel de couverture des stations par l'ensemble des faisceaux. En revanche, cela augmente également la complexité du problème à résoudre, à cause de la modélisation des contraintes de proximité et de séparation qui prennent en compte les faisceaux potentiellement elliptiques. Le but de notre étude expérimentale est de déterminer si l'augmentation de complexité du modèle est compensée par le gain de potentiel de couverture des stations pour un temps CPU limité, par rapport au modèle autorisant seulement les faisceaux couvrant un disque.

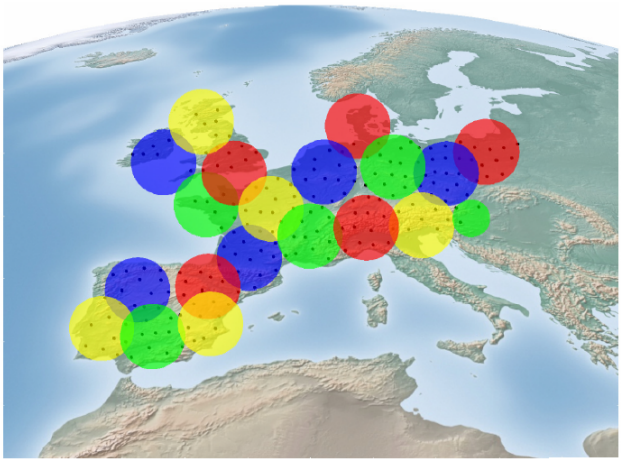


FIGURE 6.1: Exemple de solution du problème de placement de faisceaux avec seulement des faisceaux circulaires Camino et al. [2019]

La Section 6.1.1 explique la modélisation du problème et sa linéarisation, tandis que la Section 6.1.2 décrit les résultats numériques.

6.1.1 Définition et modélisation du problème

Un satellite multifaisceaux est un satellite de télécommunication qui utilise des faisceaux avec une zone de couverture relativement petite pour desservir les utilisateurs sur Terre. Il a plusieurs

réflecteurs contenant chacun plusieurs faisceaux qui influent sur les contraintes de séparation. Dans le problème de placement de faisceaux considéré, un satellite multifaisceaux fourni un service par le biais des faisceaux $f \in F$, chacun étant associé à un réflecteur $r \in R$. Cette appartenance à un réflecteur définit des contraintes de séparation des faisceaux particulières [Camino et al., 2014]. En effet, les portions de surface couvertes par les faisceaux associés à un même réflecteur multipliés par un facteur κ ne doivent pas s'intersecter. Sur Terre, les utilisateurs sont des stations $s \in S$ de coordonnées (X_s, Y_s). Ils sont caractérisés par leur demande en trafic $T_s \in \mathbb{R}^+$. Une station s est considérée couverte par le satellite si l'un des faisceaux f couvre la station et lui transmet un service. Un faisceau ne peut satisfaire une demande totale en trafic plus grande que sa capacité maximale $\gamma \in \mathbb{R}^+$. Chaque faisceau couvre une portion de la surface de la Terre de la forme d'une ellipse de paramètre prédéfinis $(Wa_k, \frac{W}{a_k})$, $a_k \geq 1$ avec $k \in K$, et W le rayon si $a_k = 1$. Ce choix de forme d'ellipses leur permet d'avoir toutes la même surface, pour qu'une plus grande surface n'avantage pas une forme de faisceaux particulière.

Modèle MINLP. Le modèle du problème de placement de faisceaux avec des faisceaux elliptiques est donné par les équations (6.1)-(6.15).

$$\max \sum_{f\in F, s\in S} T_s \alpha_{s,f} \tag{6.1}$$

$$\text{s.t.} \sum_{f\in F} \alpha_{s,f} \leq 1 \qquad \forall s \in S, \tag{6.2}$$

$$\sum_{r\in R} \beta_{f,r} \leq 1 \qquad \forall f \in F, \tag{6.3}$$

$$\sum_{s\in S} \alpha_{s,f} \leq |S| \sum_{r\in R} \beta_{f,r} \qquad \forall f \in F, \tag{6.4}$$

$$\sum_{s\in S} \alpha_{s,f} \geq \sum_{r\in R} \beta_{f,r} \qquad \forall f \in F, \tag{6.5}$$

$$\sum_{s\in S} T_s \alpha_{s,f} \leq \gamma \qquad \forall f \in F, \tag{6.6}$$

$$\sum_{k\in K, \theta\in\Theta} \delta_{f,k,\theta} = 1 \qquad \forall f \in F, \tag{6.7}$$

$$\|(x_f, y_f) - (X_s, Y_s)\|_{a_k, \frac{1}{a_k}, \theta} \leq W + (2 - \alpha_{s,f} - \delta_{f,k,\theta}) M_s$$

$$\forall s \in S, \forall f \in F, \forall k \in K, \forall \theta \in \Theta, \tag{6.8}$$

$$\eta_{f,f'} \geq \textstyle\sum_{r'\in R} \beta_{f,r'} + \sum_{r'\in R} \beta_{f',r'} + \beta_{f,r} + \beta_{f',r} - 3$$

$$\forall f, f' \in F, f' > f, \forall r \in R, \tag{6.9}$$

$$\|(x_f, y_f) - (x_{f'}, y_{f'})\|_{a_k, \frac{1}{a_k}, \theta} \geq 2\kappa W - N(1 - \eta_{f,f'})$$

$$\forall f, f' \in F, f' > f, \tag{6.10}$$

$$x_f, y_f \geq 0 \qquad \forall f \in F, \tag{6.11}$$

$$\alpha_{s,f} \in \{0,1\} \qquad \forall s \in S, \forall f \in F, \tag{6.12}$$

$$\beta_{f,r} \in \{0,1\} \qquad \forall f \in F, \forall r \in R, \tag{6.13}$$

$$\eta_{f,f'} \in \{0,1\} \qquad \forall f, f' \in F, f' > f, \tag{6.14}$$

$$\delta_{f,k,\theta} \in \{0,1\} \qquad \forall f \in F, \forall k \in K, \forall \theta \in \Theta. \tag{6.15}$$

L'objectif (6.1) est la maximisation du trafic couvert par les faisceaux, avec $\alpha_{s,f}$ une variable binaire égale à 1 si la station s est couverte par le faisceau f. Les contraintes (6.2) forcent une station à être couverte par au plus un faisceau. Les contraintes (6.3) associent au plus un réflecteur r à un faisceau f grâce à la variable binaire $\beta_{f,r}$. Les contraintes (6.4) et (6.5) forcent un faisceau non utilisé à n'être associé à aucune station et un faisceau utilisé à être associé à au moins une station. Les contraintes (6.6) assurent que chaque faisceau ne couvre pas plus de trafic que sa capacité de couverture maximale γ. Les contraintes (6.7) associent une forme $(Wa_k, \frac{W}{a_k})$ et un angle θ à un faisceau f grâce à la variable binaire $\delta_{f,k,\theta}$. Les contraintes (6.8) vérifient que chaque station s est dans le faisceau f de centre (x_f, y_f) auquel il est affecté. Le réel positif M_s est la constante de big-M associée. La fonction $\|\cdot\|_{a_k, \frac{1}{a_k}, \theta}$ est la norme définie dans l'équation (5.27) avec une lignes de niveau de valeur 1 qui est une ellipse de forme $(a_k, \frac{1}{a_k})$ et d'angle θ. Les contraintes (6.9) forcent chaque variable binaire $\eta_{f,f'}$ à être égale à 1 si les faisceaux f et f' sont associés au même réflecteur r. Les contraintes (6.10) assurent que deux faisceaux associés ne sont pas trop près. Finalement, les domaines de définitions des variables sont donnés par les équations (6.11)-(6.15). La non linéarité du modèle vient des contraintes (6.8) et (6.10).

Modèle MILP. La linéarisation du modèle (6.1)-(6.15) produit un problème MILP. Comme expliqué par l'équation (5.31), la procédure de linéarisation demande d'appliquer la transformation linéaire adaptée pour revenir au cas de la linéarisation du cercle, puis d'appliquer la linéarisation de la norme euclidienne. De plus, les contraintes (6.9) doivent être adaptées à la linéarisation. Ainsi, les contraintes qui changent à cause de la linéarisation sont (6.8), (6.9) et (6.10).

Soit p le nombre de produits scalaires utilisés pour la linéarisation. Il est choisi en fonction de l'erreur d'approximation ϵ à respecter. Soit $\phi^{-1}_{a_k, \frac{1}{a_k}, \theta}$ la transformation linéaire définie comme dans (5.31) pour les paramètres $(a_k, \frac{1}{a_k}, \theta)$. La linéarisation des contraintes (6.8) avec la

linéarisation de la norme euclidienne produit les contraintes

$$\forall s \in S, \forall f \in F, \forall k \in K, \forall \theta \in \Theta, \forall i = 1, .., p,$$
$$\phi_{a_k, \frac{1}{a_k}, \theta} \left(\begin{pmatrix} x_f - X_s \\ y_f - Y_s \end{pmatrix} \right) \cdot \begin{pmatrix} \cos \frac{2\pi i}{p} \\ \sin \frac{2\pi i}{p} \end{pmatrix} \leq W \cos \frac{\pi}{p} + (2 - \alpha_{s,f} - \delta_{f,k,\theta}) M_s, \tag{6.16}$$

où i est l'indice du vecteur u_i utilisé pour les produits scalaires. La partie de droite des contraintes (6.16) contient le terme $\cos \frac{\pi}{p}$ pour assurer la réalisabilité puisque nous utilisons l'approximation intérieure du disque.

Les contraintes (6.10) qui interdisent à deux ellipses de s'intersecter ne sont pas aussi faciles à linéariser. En effet, dans le cas d'une contrainte qui empêche deux disques de s'intersecter, il suffit de borner inférieurement la distance entre les centres avec la linéarisation de la norme euclidienne. Mais cette utilisation des distances entre centres ne fonctionnent plus dans le cas des ellipses. En revanche, la convexité d'une ellipse permet de modéliser cette non-intersection grâce à la séparation linéaire d'une manière similaire à [Kallrath et Rebennack, 2014].

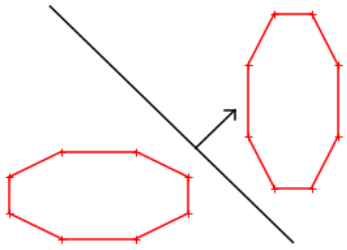


FIGURE 6.2: Séparation linéaire de l'approximation extérieure de deux ellipses par la droite noire

La séparation linéaire pour deux approximations d'ellipses avec $p = 8$ produits scalaires est illustrée dans la Figure 6.2 : les deux octogones sont séparés linéairement parce qu'il est possible de tracer une droite d entre eux. Précisément, la droite $d = \{u \in \mathbb{R}^2 | u.v = b\}$ de vecteur directeur v sépare le plan en deux demi-plans, avec chacun des deux octogones dans un demi-plan différent. Les approximations extérieures d'ellipses sont des polygones convexes donc il suffit de vérifier la séparation linéaire entre les sommets de ces polygones. Le dernier ingrédient pour appliquer la linéarisation est que des termes quadratiques sont formés par des produits scalaires entre des vecteurs qui dépendent tous les deux linéairement de variables. Ainsi, nous choisissons de tester la séparation linéaire seulement pour un nombre fini N_{dir} de droites de vecteurs normaux $v_j = (\cos \frac{2\pi j}{p}, \sin \frac{2\pi j}{p})$ for $j = 1, ..., N_{dir}$. Cela veut dire que l'on vérifie dans la linéarisation une contrainte strictement plus forte que la séparation linéaire, mais avec une valeur N_{dir} raisonnablement haute cette nouvelle approximation a probablement peu d'influence par rapport aux autres approximations du modèle.

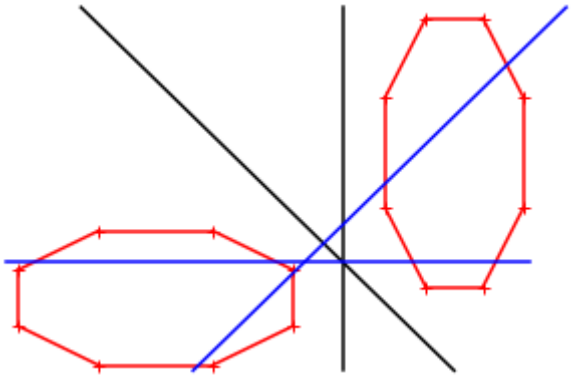


FIGURE 6.3: La séparation linéaire fonctionne avec les deux lignes noires mais ne fonctionne pas avec les deux lignes bleues

Les contraintes (6.9) sont remplacées par les contraintes

$$\sum_{j=1}^{N_{dir}} \eta_{f,f',j} \geq \sum_{r\in R} \beta_{f,r} + \sum_{r\in R} \beta_{f',r} + \beta_{f,r} + \beta_{f',r} - 3 \tag{6.17}$$

$$\forall r \in R, \forall f, f' \in F \text{ tels que } f' > f$$

$$\eta_{f,f',j} \in \{0,1\} \qquad \forall f, f' \in F, f' > f, j, \tag{6.18}$$

où les nouvelles variables binaires $\eta_{f,f',j}$ remplacent les variables $\eta_{f,f'}$ parce que dans cette linéarisation, il faut connaître l'indice j de la droite associée au vecteur normal v_j qui assure la séparation linéaire. Finalement, la linéarisation des contraintes (6.10) devient

$$\begin{pmatrix} S_i^{x,f} \\ S_i^{y,f} \end{pmatrix} \cdot \begin{pmatrix} \cos\frac{2\pi j}{N_{dir}} \\ \sin\frac{2\pi j}{N_{dir}} \end{pmatrix} \leq w_{f,f'} + N(1 - \eta_{f,f',j}) \tag{6.19}$$

$$\begin{pmatrix} S_i^{x,f'} \\ S_i^{y,f'} \end{pmatrix} \cdot \begin{pmatrix} \cos\frac{2\pi j}{N_{dir}} \\ \sin\frac{2\pi j}{N_{dir}} \end{pmatrix} \geq w_{f,f'} + N(1 - \eta_{f,f',j}) \tag{6.20}$$

$$\forall f, f' \in F \text{ tels que } f' \neq f, \forall i = 1, ..., p, \forall j = 1, ..., N_{dir}$$

$$w_{f,f'} \in \mathbb{R} \qquad \forall f, f' \in F \text{ tels que } f' \neq f \tag{6.21}$$

où la valeur N est la constante de big-M des contraintes et $(S_i^{x,f}, S_i^{y,f})^T$ est les coordonnées du sommet i de l'approximation extérieure du faisceau f agrandi d'un rapport κ. Nous utilisons l'approximation extérieure du disque pour assurer la réalisabilité. De plus, les variables continues $w_{f,f'}$ ont le rôle de séparer les valeurs des produits scalaires en deux groupes pour la séparation linéaire des ellipses. Les produits scalaires avec tous les sommets de l'approximation extérieure du faisceau f doivent être plus petits que $w_{f,f'}$, tandis que ceux associés au faisceau f' doivent être plus grands.

6.1.2 Expériences numériques

Le but de cette partie est d'évaluer le potentiel de l'extension de la linéarisation de la norme euclidienne aux ellipses avec le problème de placement de faisceaux.

Les instances [1] ainsi que les logiciels et ressources informatiques utilisés pour les expériences sont décrits. Ensuite, nous comparons deux modèles MILP résultant de la linéarisation et deux modèles MINLP pour déterminer si une résolution directe du MINLP est plus efficace que la résolution de la linéarisation. Finalement, nous montrons l'intérêt de l'extension de la linéarisation de la norme euclidienne aux ellipses par la comparaison de deux modèles : le modèle *LinE*, pour linéarisation des ellipses, où les faisceaux couvrent des ellipses ; et le modèle *LinD*, pour linéarisation des disques, où les faisceaux couvrent des disques comme dans le modèle original de [Camino et al., 2019].

6.1.2.1 Description des instances et des paramètres

Une caractéristique des instances que nous utilisons est la *densité* des stations sur la surface terrestre. Cela correspond à un nombre de stations par faisceau, calibré par une instance initiale dont la densité a été fixée à 100. Ainsi, il est possible de diviser la densité par deux en retirant la moitié des stations, puisque le nombre de stations couvertes par un faisceau est en moyenne divisé par deux. Sur le même principe, la densité est multipliée par 4 en utilisant des faisceaux de rayon deux fois plus grand, puisqu'ils couvrent une surface 4 fois plus grande.

Nous considérons un jeu de 100 instances dont les paramètres qui varient sont le nombre de stations $|S|$ et la densité d des stations sur Terre. Le nombre de stations $|S|$ prend des valeurs dans $\{20, 40, 60, 80, 100\}$ avec un nombre de faisceaux $|B| = \min\{\lceil \frac{|S|}{8} \rceil, 10\}$, où $\lceil x \rceil$ est x arrondi au supérieur. Cela veut dire que $|B|$ vaut respectivement $3, 5, 7, 10$ et 10. La densité d peut prendre les valeurs 30 et 70. Il y a donc dix instances par couple (nombre de stations - densité). De plus, pour les méthodes *LinD* et *LinE*, chaque instance est résolue pour 5 valeurs différentes du nombre de contraintes p par linéarisation : $p \in \{4, 8, 12, 16, 20\}$. Le but est d'analyser l'influence de p sur la résolution.

Pour chaque instance, les positions des stations sont tirées aléatoirement depuis un ensemble de 157 positions venant d'une instance réelle. Nous utilisons trois réflecteurs donc $R = \{1, 2, 3\}$. Les ellipses que les faisceaux peuvent couvrir dans le modèle *LinE* ont pour caractéristiques de forme et d'angle $(Wa, \frac{W}{a}, \theta)$, avec $a \in A = \{\frac{1}{2}, 1, 2\}$ et $\theta \in \Theta = \{\frac{2\pi}{3}, \frac{4\pi}{3}, 0\}$. La Figure 6.4 montre les neuf ellipses autorisées. Les trois ellipses issues de $a = 1$ représentent le même ensemble, mais leurs approximations par la linéarisation de la norme euclidienne pour un nombre de produits scalaires p non multiple de 3 sont différentes puisque ces ellipses ont des angles différents. Enfin, la capacité maximale de couverture γ est fixée à 500 pour chaque instance. Cette valeur de γ

1. Toutes les données et modèles sont disponibles sur demande.

est une limitation importante de la valeur objectif puisque la borne supérieure naïve consistant à additionner les capacités maximales de tous les faisceaux est relativement proche de la valeur objectif de nombreuses d'instances.

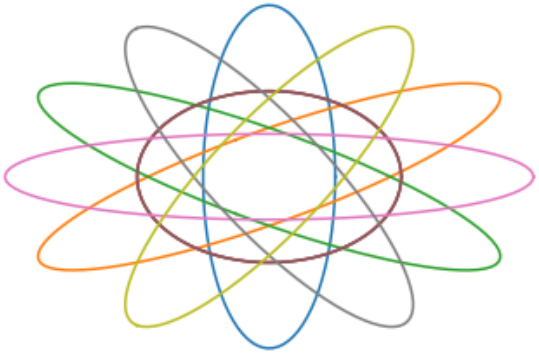


FIGURE 6.4: Ellipses autorisées pour les instances résolues par le modèle *LinE*

Deux exemples de solutions sont donnés dans les Figures 6.5 et 6.6. Les points noirs sont les stations non desservies, les points colorés les stations desservies, et leur couleur indique le réflecteur du faisceau qui dessert la station. L'avantage principal d'utiliser les ellipses est qu'il y a beaucoup plus de possibilités pour couvrir les stations comparé au modèle *LinD* n'utilisant que les disques.

Nous rappelons que le paramètre N_{dir} donne le nombre de droites pour la séparation linéaire de deux polygones approximant des ellipses dans le modèle *LinE*. Nous fixons la valeur de N_{dir} à 10. En effet, des tests préliminaires ont montré que l'influence de ce paramètre était limitée sur ce jeu d'instances à la fois pour le temps de calcul et pour la valeur objectif.

Pour les modèles *LinE* et *LinD*, chaque instance est résolue sur un seul cœur d'un CPU Xeon E5-2695 v3 de 2.30GHz avec une utilisation de la RAM limitée à 3.5 Go. Le solveur utilisé est CPLEX 12.9 avec le paramètre *global thread count* fixé à 1. Concernant la résolution du modèle MINLP, nous utilisons un des meilleurs solveurs commercial de problèmes MINLP : le solveur BARON 21.1.13. Ce solveur est lancé depuis un des ordinateurs du serveur NEOS [Czyzyk et al., 1998, Dolan, 2001, Gropp et Moré, 1997], avec une RAM de 3 Go, un seul thread, les paramètres *absolute termination tolerance* à 10^{-6} et *relative termination tolerance* à 10^{-4} pour correspondre aux équivalents de CPLEX *absolute MIP gap tolerance* et *relative MIP gap tolerance*. Les CPU utilisés par le serveur NEOS sont différents suivant les expériences, et différents de celui utilisé pour la résolution des modèles *LinE* et *LinD*, mais comme les résultats sont très en faveur des modèles *LinE* et *LinD*, ce n'est pas cette différence de CPU qui crée cette différence de résultats. Les deux modèles MINLP sont désignés par *BARON-E* et *BARON-D* respectivement pour le modèle avec des faisceaux elliptiques et celui avec des faisceaux en

forme de disque. Ils correspondent au modèle (6.1)-(6.15) avec les simplifications évidentes pour *BARON-D* puisque seulement les faisceaux en forme de disque y sont autorisés. Cela fait donc un total de 12 modèles, puisque *LinD* et *LinE* sont utilisés pour 5 valeurs de p chacun.

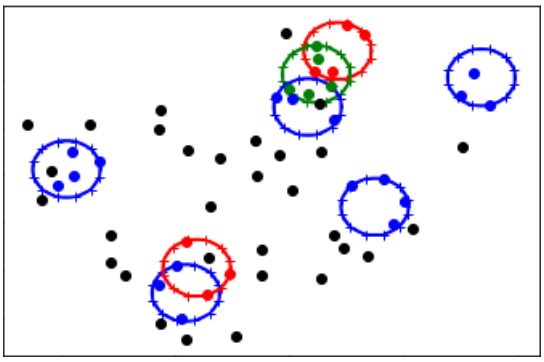


FIGURE 6.5: Exemple de solution réalisable avec le modèle *LinD*

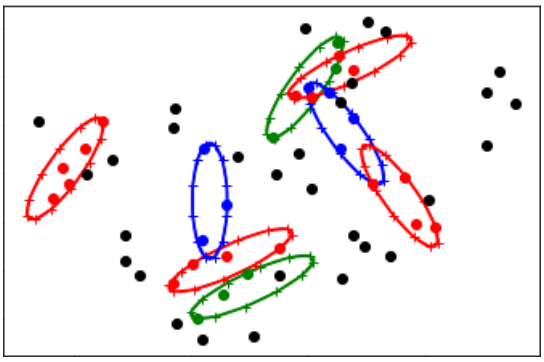


FIGURE 6.6: Exemple de solution réalisable avec le modèle *LinE*

La Table 6.1 montre le nombre de variables binaires, variables continues, contraintes linéaires et contraintes non linéaires de tous les modèles MINLP et de certains des modèles MILP, en fonction du nombre de produits scalaires p utilisés. Il y a nettement plus de contraintes dans les modèles MILP que MINLP, et aussi largement plus de contraintes dans les modèles autorisant les faisceaux elliptiques que les autres. Ces différences viennent des linéarisations des contraintes (6.8) en (6.16), (6.9) en (6.17)-(6.18) et (6.10) en (6.19)-(6.21). Comparativement, le nombre de variables binaires varie moins suivant le modèle.

6.1.2.2 Comparaison avec un solveur MINLP

Toutes les instances dont les résultats sont montrés dans les Tables 6.2, 6.3 et 6.4 sont résolues avec un temps limite de 3600 secondes. Les résultats pour les instances avec une densité de 30 et de 70 sont agrégés tout comme pour les instances avec le même nombre de stations $|S|$, donc les instances sont groupées par paquets de 20.

	BARON-D				BARON-E			
$\|S\|$	#bin	#continues	#Lcont	#NLcont	#bin	#continues	#Lcont	#NLcont
20	72	6	41	63	99	6	44	567
40	225	10	90	210	270	10	95	1890
60	532	16	176	508	604	16	184	4572
80	970	20	255	845	965	20	265	7605
100	1075	20	275	1045	1165	20	285	9405

	LinD avec $p = 4$			LinD avec $p = 20$		
$\|S\|$	#bin	#continues	#cont	#bin	#continues	#cont
20	81	6	293	129	6	1301
40	255	10	930	415	10	4290
60	616	16	2208	1064	16	10336
80	1010	20	3635	1730	20	17155
100	1210	20	4455	1930	20	21175

	LinE avec $p = 4$			LinE avec $p = 20$		
$\|S\|$	#bin	#continues	#cont	#bin	#continues	#cont
20	126	12	2444	126	12	12044
40	360	30	8095	360	30	40095
60	856	72	19704	856	72	97784
80	1370	110	32665	1370	110	162265
100	1570	110	39885	1570	110	198285

TABLE 6.1: Nombre de variables binaires, variables continues, contraintes linéaires et contraintes non linéaires pour les différents modèles

		$\|S\| = 20$		$\|S\| = 40$		$\|S\| = 60$	
model	p	BARON-D	BARON-E	BARON-D	BARON-E	BARON-D	BARON-E
LinD	4	0 / **20**	8 / **12**	7 / **13**	**20** / 0	**19** / 1	**20** / 0
	8	4 / **20**	**12** / 8	**14** / 6	**20** / 0	**20** / 0	**20** / 0
	12	12 / **20**	**15** / 6	**17** / 4	**20** / 0	**20** / 0	**20** / 0
	16	15 / **20**	**15** / 6	**19** / 3	**20** / 0	**20** / 0	**20** / 0
	20	15 / **20**	**15** / 5	**20** / 2	**20** / 0	**20** / 0	**20** / 0
LinE	4	**15** / 5	**20** / 0	**13** / 7	**20** / 0	**20** / 0	**20** / 0
	8	**18** / 2	**20** / 0	**19** / 1	**20** / 0	**20** / 0	**20** / 0
	12	**18** / 2	**20** / 0	**20** / 0	**20** / 0	**20** / 0	**20** / 0
	16	**19** / 1	**20** / 0	**20** / 0	**20** / 0	**20** / 0	**20** / 0
	20	**19** / 1	**20** / 0	**20** / 0	**20** / 0	**20** / 0	**20** / 0

		$\|S\| = 80$		$\|S\| = 100$	
model	p	BARON-D	BARON-E	BARON-D	BARON-E
LinD	4	**20** / 0	**20** / 0	**20** / 0	**20** / 0
	8	**20** / 0	**20** / 0	**20** / 0	**20** / 0
	12	**20** / 0	**20** / 0	**20** / 0	**20** / 0
	16	**20** / 0	**20** / 0	**20** / 0	**20** / 0
	20	**20** / 0	**20** / 0	**20** / 0	**20** / 0
LinE	4	**20** / 0	**20** / 0	**20** / 0	**20** / 0
	8	**20** / 0	**20** / 0	**20** / 0	**20** / 0
	12	**20** / 0	**20** / 0	**20** / 0	**20** / 0
	16	**20** / 0	**20** / 0	**20** / 0	**20** / 0
	20	**20** / 0	**20** / 0	**20** / 0	**20** / 0

TABLE 6.2: Comparaison entre les modèles MINLP et MILP : nombre d'instances avec la plus grande valeur objectif

$\|S\|$	BARON-D	BARON-E	LinD					LinE				
			4	8	12	16	20	4	8	12	16	20
20	20	0	20	20	20	20	20	20	20	17	13	16
40	0	0	20	20	18	15	12	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0	0	0	0	0

TABLE 6.3: Nombre d'instances résolues à l'optimum pour différents modèles

La Table 6.2 [2] montre une comparaison des résultats paquet par paquet pour 12 modèles différents. Elle compare instance par instance les solutions retournées en terme de valeur objectif. Chaque cellule de la table montre deux nombres. Le premier est le nombre de fois (sur 20 instances) où la valeur objectif du modèle MILP est plus grande ou égale à celle du modèle MINLP, tandis que le deuxième est le nombre de fois où c'est le modèle MINLP qui a une valeur objectif plus grande ou égale à celle du modèle MILP. Le plus grand de ces deux nombres est en gras pour faciliter l'interprétation.

Les modèles *LinE* ont un plus grand nombre de meilleures valeurs objectifs pour chaque paquet. Concernant les modèles *LinD*, ils ont le plus grand nombre de meilleures valeurs objectif dans 43 des 50 paquets. Parmi les autres 7 paquets, il y en a 6 pour un nombre de stations de 20 et 1 pour 40.

La Table 6.3 montre le nombre d'instances résolues à l'optimum pour chaque modèle. Ces modèles résolvent des problèmes différents, donc les valeurs optimales sont différentes. Le modèle *BARON-E* n'en résout aucune à l'optimum, les modèles *LinE* et *BARON-D* n'en résolvent exactement que pour des instances avec le plus petit nombre de stations, tandis que les modèles *LinD* en résolvent à l'optimum pour des instances ayant jusqu'à 40 stations. Donc de manière générale, les modèles MILP arrivent à résoudre à l'optimum des instances avec plus de stations.

Finalement, la Table 6.4 montre le nombre d'instances pour lesquelles la valeur objectif à la fin du temps de calcul est 0, ce qui veut dire qu'aucune solution non triviale n'a été trouvée. Pour la plupart des instances avec au moins 60 stations, le modèle *BARON-D* ne trouve pas de solutions non triviales, et *BARON-E* ne trouve des solutions non triviales qu'avec le nombre minimum de stations. Cela montre que les modèles MINLP sont trop difficiles à résoudre pour un temps limite de 3600 secondes.

Pour conclure, la comparaison numérique des modèles MINLP par rapport aux modèles MILP montre que la linéarisation du modèle de placement de faisceaux est une option raison-

2. Les Tables 6.2 et 6.4 présentent les résultats corrigés des tables correspondantes dans [Duguet et al., 2022]. En effet, une erreur a été commise pendant la récupération des résultats numériques des méthodes *LinD* et *LinE*. Toutefois, cela affecte seulement les résultats numériques pour plus de 60 stations, et la conclusion de l'analyse reste inchangée.

$\|S\|$	BARON-D	BARON-E	LinD					LinE				
			4	8	12	16	20	4	8	12	16	20
20	0	3	0	0	0	0	0	0	0	0	0	0
40	0	20	0	0	0	0	0	0	0	0	0	0
60	17	20	0	0	0	0	0	0	0	0	0	0
80	17	20	0	0	0	0	0	0	0	0	0	0
100	14	20	0	0	0	0	0	0	0	0	0	0

TABLE 6.4: Nombre d'instances avec une valeur objectif de 0 pour les différents modèles

nable. Et ce d'autant plus que les modèles MINLP échouent à produire des solutions exploitables pour une large majorité des instances, alors que les modèles MILP retournent des solutions peu importe le nombre de stations des instances.

6.1.2.3 Comparaison entre les modèles *LinE* et *LinD*

Chacune des 100 instances est résolue avec les modèles *LinE* et *LinD*, pour trois limites de temps de calcul différentes, 600, 1800 et 3600 secondes, et pour des linéarisations avec les nombres de produits scalaires p suivants : $4, 8, 12, 16, 20$. La Table 6.3 indique que les modèles *LinE* prouvent l'optimalité dans 17% des instances, et uniquement pour des instances à 20 stations. Les modèles *LinD* quant à eux la prouve dans 37% des cas, pour des instances à 20 ou 40 stations. Comme les linéarisations *LinE* et *LinD* ne produisent pas les mêmes problèmes MILP, les valeurs optimales ne sont pas les mêmes. Dans la suite, les solutions optimales trouvées ne sont plus mentionnées, et les comparaisons se font seulement à partir des valeurs objectifs obtenues dans le temps imparti.

Les Figures 6.7 et 6.8 montrent les résultats de la résolution des instances à faible densité et à forte densité respectivement. Chaque point représente la moyenne de valeur objectif obtenue pour les 10 instances avec les paramètres p et $|S|$ comme indiqué. L'échelle de l'axe des ordonnées est le pourcentage manquant à la borne supérieure naïve donnée par la capacité maximale de couverture de l'ensemble des faisceaux. Cette borne supérieure se calcule en additionnant la capacité maximale de couverture de l'ensemble des faisceaux $|F|\gamma$, où $|F|$ est le nombre de faisceaux. Cette borne supérieure est affichée comme la ligne en pointillés noirs. Les moyennes de valeur objectif sont reliées aux autres moyennes issues de la même méthode, de la même limite de temps de calcul et du même nombre de stations pour montrer l'influence du nombre de produits scalaires dans la linéarisation.

Il y a plusieurs points à remarquer sur les Figures 6.7 et 6.8. Pour un grand nombre de stations, le modèle *LinE* bénéficie vraiment de l'augmentation des temps de calcul alors que les valeurs objectifs ne progressent pas visiblement pour le modèle *LinD* entre 600 et 3600 secondes de temps de calcul peu importe le nombre de stations. Donc, pour des instances avec 40 stations

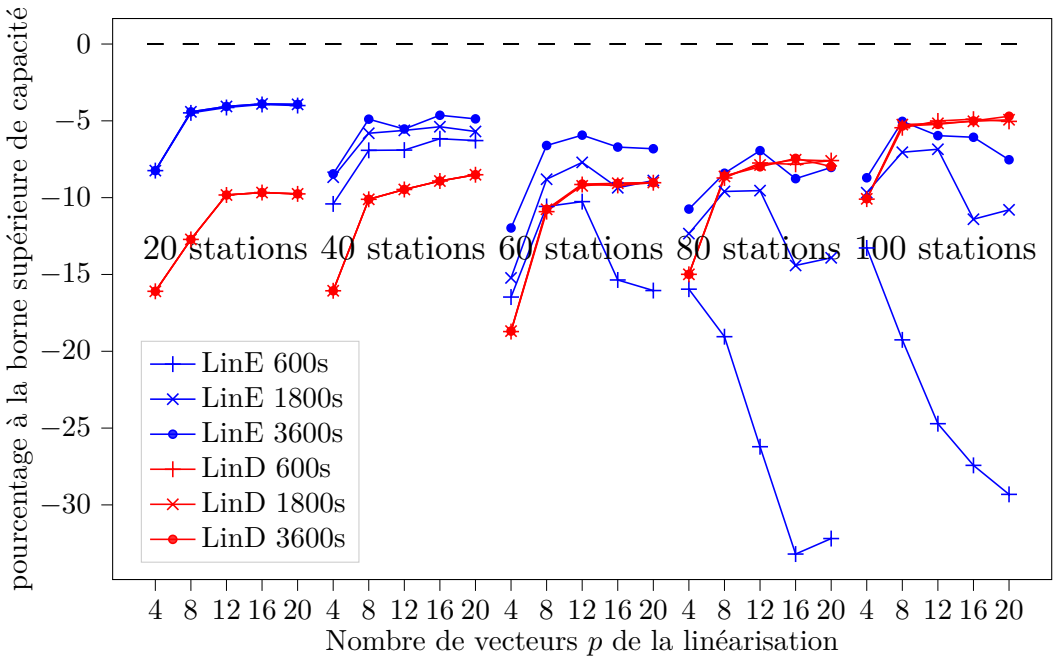


FIGURE 6.7: Résultats pour les instances à faible densité

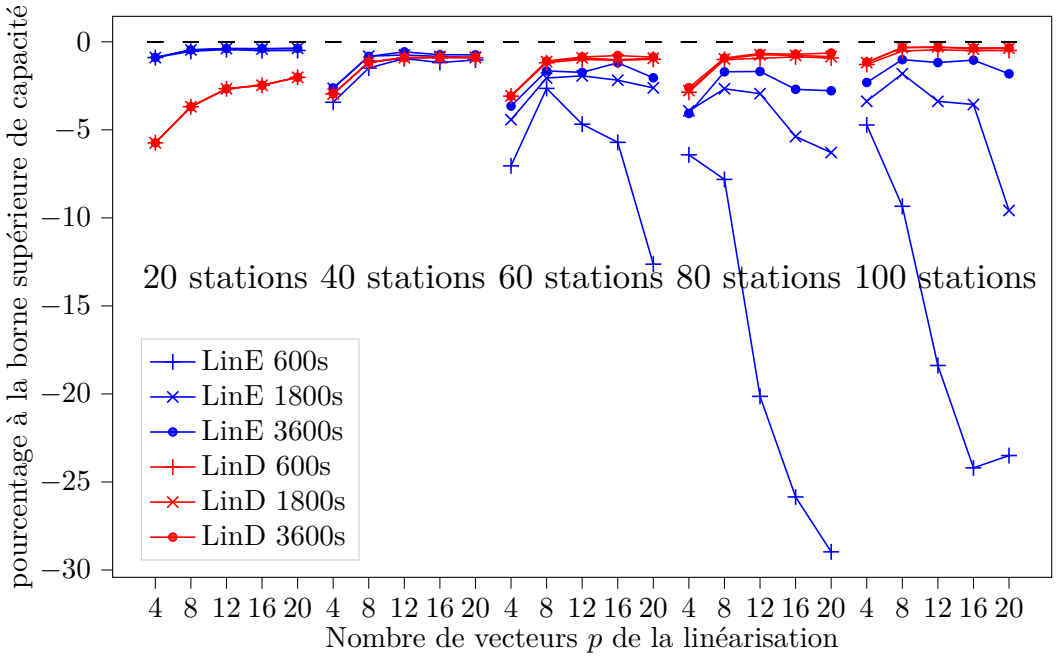


FIGURE 6.8: Résultats pour les instances à forte densité

ou moins et 3600 secondes, le modèle *LinE* est meilleur que le modèle *LinD*.

Une autre tendance est visible en fonction de la densité : les instances à forte densité sont

d \ time (s)	600	1800	3600
30	40	60	80
70	20	40	40

TABLE 6.5: Nombre maximum de stations pour lequel le modèle *LinE* a une meilleur moyenne de valeur objectif que le modèle *LinD*

model \ p	4	8	12	16	20
LinE	4	5	12	7	2
LinD	0	1	4	11*	16*

TABLE 6.6: Nombre de meilleures moyennes en fonction du modèle et du nombre de produits scalaires p de la linéarisation

nettement plus proches de la borne supérieure naïve (la majorité est à entre 0% et 5% de cette borne), alors que les instances à faible densité sont rarement à moins de 5% de la borne. La combinatoire des modèles peut expliquer ce point : une plus forte densité augmente le nombre de stations qu'un faisceau peut couvrir, ce qui a tendance à augmenter le trafic maximum couvert par un faisceau. Remarquons aussi que plus le nombre de stations augmente, plus les résultats du modèle *LinD* sont bons par rapport à ceux du modèle *LinE*.

La Table 6.5 montre le nombre maximum de stations pour lequel le modèle *LinE* donne de meilleurs résultats que le modèle *LinD* en fonction de la densité et de la limite de temps. Plus précisément, pour une densité et une limite de temps donnée, nous considérons la plus haute moyenne de valeur objectif parmi les 5 valeurs de p, et le meilleur modèle est celui avec la plus grande valeur. Les performances du modèle *LinE* par rapport au modèle *LinD* se dégradent avec l'augmentation de la densité et la diminution du temps.

La Table 6.6 donne des indications sur l'impact du nombre de produits scalaires p dans la linéarisation. Il y a 30 paquets de 10 instances par modèle, séparés suivant le nombre de stations, la densité et le temps de calcul. Les valeurs indiquées pour un modèle donné sont le nombre de fois que la valeur de p correspondante donne de meilleurs moyennes de valeur objectif que toutes les autres valeurs de p à densité et limite de temps égale. Le caractère $*$ indique que les résultats pour deux paquets avec $p = 16$ et $p = 20$ étaient les mêmes, ils ont donc tous les deux été comptés comme meilleur résultat.

6.1.3 Conclusion

Cette application au problème de placement de faisceaux compare notre extension de la linéarisation de la norme euclidienne aux contraintes elliptiques à la linéarisation de la norme euclidienne et à une résolution directe du MINLP grâce au solveur BARON. Elle montre que

notre extension permet d'augmenter les possibilités de couverture des stations grâce à une plus grande variété de formes de faisceaux. Cela permet d'obtenir de meilleures solutions lorsque le temps de calcul est suffisant pour obtenir une solution proche de l'optimum. Sinon, la plus petite taille du modèle MILP issu de la linéarisation de la norme euclidienne lui permet de produire de meilleures solutions.

6.2 Application à la résolution de jeux non linéaires

Dans cette section, nous décrivons une application numérique sur un jeu d'investissement en cybersécurité. Le jeu concerné fait partie de la classe des *Integer Programming Games* (IPG) qui font eux-mêmes partie des jeux simultanés et non coopératifs à information complète.

Un jeu est dit *non coopératif à information complète* si chaque joueur y maximise sa fonction d'utilité et a connaissance de toutes les informations décrivant la fonction d'utilité et les contraintes des autres joueurs. Un concept prédominant de solution d'un tel jeu est alors celui de l'équilibre de Nash [Nash, 1951], qui se définit comme une solution dont aucun joueur n'a intérêt à dévier seul. Il s'agit donc de solutions stables pour lesquelles il n'est pas profitable pour un joueur de changer unilatéralement ses décisions.

Ces dernières années, plusieurs auteurs ont proposé des algorithmes pour calculer des équilibres de Nash dans des IPG [Carvalho et al., 2021, 2022, Crönert et Minner, 2022, Dragotto et Scatamacchia, 2023, Sagratella, 2016, Schwarze et Stein, 2022]. La majorité de ces algorithmes suppose que les fonctions d'utilité sont linéaires, quadratiques ou ont des propriétés de convexités. Le calcul des équilibres de Nash dans les IPG avec des fonctions d'utilité non linéaires (en les variables de décision de chaque joueur) est un sujet peu exploré.

Le plan de cette section est le suivant. Nous introduisons dans la Section 6.2.1 les définitions nécessaires à la compréhension de la suite ainsi que notre méthodologie de calcul d'une variante d'équilibre de Nash dite "δ-équilibre de Nash" pour des IPG avec des fonctions d'utilité non linéaires, inspirée de la méthode d'approximation des problèmes MINLP grâce aux fonctions PWL. Puis dans la Section 6.2.2 nous comparons numériquement notre méthodologie avec l'unique méthode comparable de la littérature sur un jeu d'investissement en cybersécurité. Enfin, nous tirons les conclusions de cette application numérique dans la Section 6.2.3.

6.2.1 Définitions et méthode de résolution du problème

6.2.1.1 Vocabulaire et notations

Définition 57 (IPG)**.** Un *IPG* est un jeu simultané et non coopératif à information complète avec un nombre fini de joueurs $M = \{1, 2..., m\}$ tel que chaque joueur $p \in M$ résout un problème

d'optimisation paramétrique

$$\max_{x^p} \quad \Pi^p(x^p; x^{-p}) \tag{6.22a}$$

$$\text{s.c.} \quad x^p \in X^p := \{A^p x^p \leq b^p \mid x^p \in \mathbb{R}^{n_p^c} \times \mathbb{Z}^{n_p^i}\}, \tag{6.22b}$$

où $x^{-p} = (x^1, \ldots, x^{p-1}, x^{p+1}, \ldots, x^m)$ est un vecteur contenant les stratégies de chaque joueur excepté le joueur p, X^p et Π^p sont les *ensembles de stratégies* et la *fonction d'utilité* du joueur p, et A^p et b^p sont une matrice de rationnels et un vecteur des dimensions appropriées respectivement.

Nous appelons X l'ensemble des combinaisons des stratégies des joueurs, c'est-à-dire le produit cartésien $X = \prod_{p=1}^m X^p$ et nous appelons $x \in X$ un *profil de stratégies*. Pour chaque joueur p, nous disons que $x^p \in X^p$ est une *stratégie pure* pour p. Lorsque les joueurs jouent aléatoirement parmi plusieurs stratégies pures, ils jouent ce que l'on appelle une *stratégie mixte*, c'est-à-dire une probabilité de distribution σ^p sur l'ensemble des stratégies pures X^p. Soit Δ^p l'espace des distributions de probabilités sur X^p pour le joueur p. Un *profil de stratégies mixtes* est un vecteur $\sigma = (\sigma^1, \ldots, \sigma^m)$ de stratégies mixtes tel que $\sigma^p \in \Delta^p$. Nous appelons $\Pi^p(\sigma^p; \sigma^{-p})$ l'espérance de l'utilité du joueur p associée avec σ.

Définition 58 (équilibre de Nash d'un IPG)**.** Étant donné une instance d'IPG et un scalaire $\delta \in \mathbb{R}^+$, un profil de stratégies mixtes $\hat{\sigma}$ est un *équilibre de Nash* si aucun joueur n'a d'intérêt à s'en éloigner, c'est-à-dire si, pour chaque joueur $p \in M$, nous avons

$$\Pi^p(\hat{\sigma}^p; \hat{\sigma}^{-p}) \geq \Pi^p(\bar{x}^p; \hat{\sigma}^{-p}), \quad \forall \bar{x}^p \in X^p. \tag{6.23}$$

Cette définition assure que pour chaque joueur p, il n'existe pas de stratégie $\bar{x}^p$ pour laquelle la fonction d'utilité du joueur augmente si les autres joueurs gardent leur stratégie $\hat{\sigma}^{-p}$.

Comme expliqué dans le tutoriel [Carvalho et al., 2023], le calcul de la meilleure réponse, c'est-à-dire la résolution du problème d'optimisation du joueur p (6.22) sachant que les autres joueurs jouent l'ensemble de stratégies x^{-p}, est essentiel pour l'identification correcte et le calcul d'un équilibre de Nash. Or, d'un point de vue computationnel, la forme de la fonction d'utilité Π^p influence profondément la difficulté du calcul. Ainsi une fonction d'utilité Π^p qui n'est pas linéaire en x^p, induit un calcul de la meilleure réponse revenant à résoudre un problème MINLP, qui comme nous l'avons vu est en général difficile à résoudre.

Définition 59 (δ-équilibre de Nash)**.** Étant donné une instance d'IPG et un scalaire $\delta \in \mathbb{R}_+$, un profil de stratégies mixtes $\hat{\sigma}$ est un *δ-équilibre de Nash* si aucun joueur ne peut unilatéralement s'en écarter et voir son utilité augmenter de plus de δ, c'est-à-dire si, pour chaque joueur $p \in M$, nous avons

$$\Pi^p(\hat{\sigma}^p; \hat{\sigma}^{-p}) + \delta \geq \Pi^p(\bar{x}^p; \hat{\sigma}^{-p}), \quad \forall \bar{x}^p \in X^p. \tag{6.24}$$

Notons que le cas où $\delta = 0$ est équivalent à l'équilibre de Nash (que l'on pourrait alors appeler équilibre de Nash *exact*).

Dans notre travail les termes *δ-équilibre* et *équilibre* sont parfois utilisés à la place de δ-équilibre de Nash et équilibre de Nash respectivement, puisque dans notre cas la solution d'un jeu est toujours définie à partir de la notion d'équilibre de Nash.

6.2.1.2 L'algorithme SGM

L'algorithme SGM (pour *sample generation method* [Carvalho et al., 2022]) est à notre connaissance la seule méthode permettant de calculer un δ-équilibre à un IPG quand la fonction d'utilité a des termes non linéaires en les variables du joueur sans propriétés particulières. C'est une méthode itérative qui alterne entre deux phases, avec un processus qui pourrait s'apparenter à la génération de colonnes [Desrosiers et Lübbecke, 2005]. La première phase consiste à calculer l'équilibre σ d'un jeu où l'ensemble de stratégies de chaque joueur p est restreint à un sous-ensemble de X^p. En pratique, nous supposons ces sous-ensembles non vides et le jeu résultant est appelé *jeu approché intérieurement*. La deuxième étape consiste à calculer la meilleure réponse à σ^{-p} pour chaque joueur p, étant donné l'équilibre σ du jeu approché intérieurement. Le but est de vérifier si, pour chaque joueur, il existe une stratégie déviant de σ et augmentant sa fonction d'utilité de plus de δ. Dans le cas où pour chaque joueur p et les stratégies des adversaires σ^{-p}, l'utilité d'une meilleure réponse $\bar{x}^p$ est inférieure ou égale à celle pour σ augmentée de δ, alors σ est un δ-équilibre pour l'IPG. Le critère d'arrêt de SGM est donc que $\Pi^p(\sigma^p; \sigma^{-p}) + \delta \geq \Pi^p(\bar{x}^p; \sigma^{-p})$ soit vrai pour chaque joueur p. En revanche, s'il existe au moins une déviation $\bar{x}^p$ suffisamment profitable, alors SGM ajoute cette stratégie $\bar{x}^p$ au sous-ensemble de stratégies de X^p de la première phase et itère à nouveau.

6.2.1.3 Approximation des équilibres et des fonctions d'utilité

Calculer un équilibre de Nash est une tâche difficile. Par exemple, même pour des jeux sous forme normale à deux joueurs [Chen et Deng, 2006], qui forment une sous-classe de jeux finis inclus dans les IPG, calculer un équilibre de Nash n'est pas facile. De plus, dans le cas des IPG, même calculer une meilleure réponse demande de résoudre un MINLP. Une manière de réduire la difficulté de la tâche de déterminer un équilibre de Nash est de considérer les δ-équilibres à la place des équilibres exacts.

Sans perte de généralité, nous représentons la fonction d'utilité du joueur p comme

$$\Pi^p(x^p; x^{-p}) = f^p(x^p) + g^p(x^p; x^{-p}),$$

où f^p est une fonction regroupant ce que l'on appelle les *termes individuels*, c'est-à-dire tous les termes qui dépendent seulement de la stratégie du joueur p, et g^p regroupe les termes

dépendant de la stratégie du joueur p et des stratégies des autres joueurs. Nous nous concentrons sur l'approximation de f^p à l'aide des fonctions PWL. Nous faisons la remarque que nous n'approximons pas la fonction g^p car elle contient les variables des autres joueurs x^{-p} et donc son approximation PWL ferait apparaître les variables x^{-p} dans les contraintes du joueur p. En conséquence, approximer g^p demanderait d'étendre la définition d'équilibre aux *équilibres de Nash généraux*, c'est-à-dire aux équilibres de jeux où l'ensemble de stratégies possibles pour chaque joueur dépend de la stratégie des adversaires. Or, ces jeux ne sont pas des IPG et cela ne nous permettrait pas d'utiliser les algorithmes disponibles pour résoudre les IPG. Nous considérons les IPG où les fonctions g^p sont linéaires en x^p.

Nous présentons un lien entre les δ-équilibres et les approximations de fonctions d'utilité. Plus précisément, nous montrons que, étant donné une instance G d'IPG, nous pouvons calculer un δ-équilibre par le calcul d'un équilibre du jeu $\hat{G}$ obtenu en approximant la fonction non linéaire f^p par une approximation PWL respectant une erreur absolue.

Proposition 60. *Soit G un IPG où le problème d'optimisation de chaque joueur p est de la forme* (6.22)*. Supposons que $\hat{G}$ est un IPG pour chaque joueur p qui résout le problème d'optimisation*

$$\max_{x^p}\{\hat{\Pi}^p(x^p;x^{-p}) := \hat{f}^p(x^p) + g^p(x^p;x^{-p}) | x^p \in X^p\},$$

où $\hat{f}^p$ est une δ-approximation absolue de f^p dans X^p. Alors, (i.) un équilibre de Nash $\hat{\sigma}$ de $\hat{G}$ est un 2δ-équilibre de G et, (ii.) un δ_M-équilibre de $\hat{G}$ est un $(2\delta+\delta_M)$-équilibre de G.

Démonstration. Nous montrons d'abord que (ii.) est vraie. Considérons arbitrairement un joueur p de G. Nous remarquons que x^{-p} est un paramètre du problème d'optimisation du joueur p, donc la fonction $\hat{\Pi}^p$ dépend seulement de x^p. $\hat{\Pi}^p(x^p;x^{-p}) = \hat{f}^p(x^p) + g^p(x^p;x^{-p})$ est alors une δ-approximation absolue de $\Pi^p(x^p;x^{-p}) = f^p(x^p) + g^p(x^p;x^{-p})$ pour n'importe quel x^{-p} parce que $\hat{f}^p$ est une δ-approximation absolue de f^p. Il suit que :

$$\begin{aligned}
\Pi^p(\hat{\sigma}^p;\hat{\sigma}^{-p}) &\geq \hat{\Pi}^p(\hat{\sigma}^p;\hat{\sigma}^{-p}) - \delta, && && (6.25)\\
&\geq \hat{\Pi}^p(x^p;\hat{\sigma}^{-p}) - \delta - \delta_M && \forall x^p \in X^p, && (6.26)\\
&\geq \Pi^p(x^p;\hat{\sigma}^{-p}) - 2\delta - \delta_M && \forall x^p \in X^p, && (6.27)
\end{aligned}$$

où nous exploitons le fait que $\hat{\Pi}$ est une δ-approximation absolue de Π dans (6.25) et (6.27), et le fait que $\hat{\sigma}$ est un δ_M-équilibre de $\hat{G}$ dans (6.26). Cela montre que $\hat{\sigma}$ est un $(2\delta+\delta_M)$-équilibre de G. Finalement, (i.) est vraie comme cas particulier du cas (ii.) en prenant $\delta_M = 0$. □

Intuitivement, la Proposition 60 assure que l'équilibre du jeu approché $\hat{G}$ est un δ-équilibre du jeu G. Comme nous l'expliquons dans la Section 6.2.1.4, nous employons cette proposition pour calculer un δ-équilibre à des jeux où les fonctions d'utilité sont non linéaires. L'Annexe C

présente un résultat similaire pour une erreur relative.

6.2.1.4 Calcul de δ-équilibres

Dans cette section, nous utilisons les ingrédients algorithmiques introduits plus tôt pour présenter notre algorithme pour le calcul de δ-équilibres. Plus précisément, nous proposons deux procédures algorithmiques différentes pour calculer un δ_f-équilibre à l'IPG G basé sur SGM et des approximations à erreur absolue près des fonctions d'utilité des joueurs.

Approximation directe. Pour résumer, notre *procédure d'approximation directe* calcule une et une seule fois une approximation des fonctions d'utilité des joueurs, avant d'utiliser SGM pour calculer un équilibre du jeu approché. En particulier, SGM utilise les fonctions approximées dans le calcul des meilleures réponses. L'approximation directe fonctionne de la manière suivante. Premièrement, pour chaque joueur $p \in M$ et un paramètre $\mu \in]0,1[$, il approxime les fonctions f^p avec des fonctions PWL $\mu\frac{\delta_f}{2}$-approximations absolues $\hat{f}^p$ pour obtenir un jeu approché $\hat{G}$ de G. Deuxièmement, il calcule un $(1-\mu)\delta_f$-équilibre de $\hat{G}$. Selon la Proposition 60, $\hat{\sigma}$ est une $2\mu\frac{\delta_f}{2} + (1-\mu)\delta_f = \delta_f$-équilibre de G. Le paramètre μ détermine la proportion de la tolérance δ_f qui est utilisée pour chacune des deux opérations demandant une tolérance : l'approximation PWL et SGM. Cela permet de contrôler le compromis entre ces deux tolérances pour que d'un point de vue pratique, l'approximation PWL soit facile à calculer et SGM termine. En effet, une valeur de μ trop proche des bornes nuira fortement à l'une de ces deux opérations.

Approximation à deux niveaux. À l'inverse de la procédure d'approximation directe, nous introduisons la *procédure d'approximation à deux niveaux* où nous raffinons une fois les approximations PWL. L'idée est de produire tout d'abord un δ_0-équilibre avec $\delta_0 > \delta_f$ pour qu'il soit calculé plus rapidement que le δ_f-équilibre que nous cherchons, puis de l'utiliser comme initialisation pour produire le δ_f-équilibre. L'initialisation devrait réduire drastiquement le nombre d'itérations à l'intérieur de SGM et donc diminuer le temps de calcul du deuxième appel à SGM permettant d'aboutir à une procédure globale plus rapide que la procédure d'approximation directe. Nous décrivons cette procédure dans l'Algorithme 8. Étant donné une instance G d'IPG,

Algorithme 8 Algorithme pour la procédure d'approximation à deux niveaux

Entrée : un IPG G, une tolérance initiale δ_0, une tolérance finale δ_f et un paramètre $\mu \in]0,1[$
Sortie : Un δ_f-équilibre $\hat{\sigma}$

1: $\hat{G}_0 = \textsc{ApproximeIPG}(G, \delta_0)$ ▷ *crée un jeu approché* $\hat{G}_0$
2: $\hat{\sigma}_0 = \textsc{AppelSGM}(\hat{G}_0, (1-\mu)\delta_f)$ ▷ *trouve un* $(1-\mu)\delta_f$*-équilibre de* $\hat{G}$
3: $\hat{G} = \textsc{ApproximeIPG}(G, \mu\frac{\delta_f}{2})$
4: $\hat{\sigma} = \textsc{AppelSGM}(\hat{G}, (1-\mu)\delta_f, \text{initialisation} = \hat{\sigma}_0)$ ▷ *utilise* $\hat{\sigma_0}$ *pour l'initialisation*

un paramètre $\mu \in]0,1[$, un paramètre δ_f et un paramètre $\delta_0 \geq \mu\frac{\delta_f}{2}$, l'algorithme retourne un δ_f-équilibre de G. Cet algorithme utilise deux fonctions. La fonction ApproximeIPG(G, δ) calcule une δ-approximation absolue pour chaque fonction d'utilité des joueurs du jeu G et retourne le jeu approché utilisant les approximations. Il en résulte un jeu approché $\hat{G}$ avec le même ensemble de joueurs et de stratégies, et la fonction d'utilité donnée par les δ-approximations absolues. Concernant la fonction AppelSGM(G, δ), elle calcule un δ-équilibre au jeu G grâce à SGM. Cette fonction peut aussi avoir en troisième paramètre un profil de stratégies mixtes fonctionnant comme une initialisation dans le problème MILP de [Sandholm et al., 2005] utilisé dans SGM, pour accélérer la résolution.

Adaptation de SGM utilisé au sein des deux procédures. Pour résoudre un jeu approché intérieurement, SGM peut appliquer n'importe quel algorithme de la littérature adapté. Pour une implémentation plus facile et à cause de l'existence de solveurs de problèmes MILP efficaces, nous utilisons [Sandholm et al., 2005, feasibility Program 1] [3]. Cela a aussi l'avantage de permettre d'utiliser un équilibre calculé dans les itérations précédentes pour initialiser la procédure et gagner du temps. Avec cette formulation, résoudre un jeu approché intérieurement revient à résoudre un problème MILP qui n'est pas réalisable s'il n'existe pas d'équilibres. Puisque c'est un problème de faisabilité, nous ajoutons aussi une fonction objectif qui minimise la taille du support, c'est-à-dire le nombre de stratégies pures utilisées avec probabilité positive dans un profil de stratégies mixtes. Nous utilisons cet objectif parce qu'il y a des arguments expérimentaux suggérant que les équilibres ont en général des supports de petites tailles [Porter et al., 2008].

6.2.2 Application numérique

Nous montrons dans cette partie l'intérêt de la méthodologie développée dans la Section 6.2.1. Pour cela, nous commençons par décrire le jeu d'investissement en cybersécurité que nous résolvons puis nous décrivons la configuration expérimentale et enfin nous analysons les résultats.

6.2.2.1 Modélisation du jeu d'investissement en cybersécurité

Le jeu d'investissement en cybersécurité (JIC) est un jeu non coopératif où un ensemble de joueurs (les fournisseurs) vendent des produits équivalents sur leur propre site internet. Pendant l'achat du produit en ligne par des revendeurs, la transaction est sujette aux cyberattaques, qui abîmerait la réputation du joueur et diminuerait ses profits si elles réussissaient. Le jeu étend le modèle de [Nagurney et al., 2017] en incluant un coût d'ouverture de marché entre joueurs et

3. C'est une formulation adaptée à un jeu à deux joueurs qui se généralise à m joueurs.

revendeurs et en assouplissant les conditions sur les fonctions représentant les fonctions d'investissement en cybersécurité. Nous nous référons à [Nagurney et al., 2017] pour une explication complète des motivations du modèle.

Dans le JIC, l'ensemble de joueurs $p \in M$ représente les fournisseurs qui vendent des produits équivalents à un ensemble de revendeurs $J = \{1, ..., n\}$. Chaque fournisseur $p \in M$ décide la quantité Q_{pj} de produits vendus à chaque revendeur $j \in J$ tel que Q_{pj} ne dépasse pas la borne supérieure $\bar{Q}_{pj}$. Comme la transaction prend place en ligne, un attaquant peut potentiellement lancer une cyberattaque grâce à la transaction. Si l'attaque est un succès, le fournisseur est considéré responsable et subit des dégâts financiers et de réputation. Pour pallier à cela, chaque fournisseur décide son niveau de cybersécurité $s_p \in [0, \bar{s}_p]$ pour protéger ses transactions, avec $\bar{s}_p < 1$. Plus le niveau de cybersécurité s_p est haut, moins il y a de chance qu'une cyberattaque réussisse. En particulier, si $s_p = 1$, aucune cyberattaque ne peut réussir. De plus, lorsque le joueur p fait une transaction avec le revendeur j, il doit dépenser un coût de lancement de marché, activé par une variable binaire b_{pj} qui vaut 1 si et seulement si le joueur p vend des produits au revendeur j. Tout considéré, le problème d'optimisation du joueur p est un problème MINLP avec seulement des contraintes sur les bornes des variables.

$$\max_{X^p} \sum_{j\in J} \hat{\rho}_j(Q,s)Q_{pj} - c_p^{\text{prod}} \sum_{j\in J} Q_{pj} - \sum_{j\in J} c_{pj}^{\text{setup}} b_{pj} - \sum_{j\in J} c_{pj}(Q_{pj}) - h_p(s_p) - p(s)D_p \quad (6.28a)$$

$$\text{s.c.} \quad 0 \le Q_{pj} \le b_{pj}\bar{Q}_{pj}, \quad b_{pj} \in \{0,1\} \qquad \forall j \in J, \quad (6.28b)$$

$$0 \le s_p \le \bar{s}_p. \quad (6.28c)$$

La fonction objectif 6.28a contient les termes suivants :

1. **Profits.** Le terme $\sum_{j\in J} \hat{\rho}_j(Q,s)Q_{pj}$ représente le profit du joueur p. La variable $Q := (Q_{ij})_{i\in M, j\in J}$ est l'agrégation des variables Q_{ij} pour tous les joueurs i et revendeurs j, tandis que la variable s est l'agrégation de tous les niveaux de cybersécurité des joueurs $s = (s_1, ..., s_m)$. La fonction $\hat{\rho}_j(Q,s)$ est le prix de vente unitaire fixé par le revendeur j donné par la fonction de demande $(q_j + r_j M(s)) - m_j \sum_{i\in M} Q_{ij}$, où $M(s)$ est le niveau moyen de cybersécurité des joueurs, et q_j, m_j et r_j sont des paramètres. Le prix décroît avec la quantité totale de produits vendus, et augmente avec le niveau moyen de cybersécurité des joueurs. Seul le niveau moyen de cybersécurité est utilisé parce que l'on considère que les revendeurs ne connaissent pas spécifiquement le niveau de sécurité des fournisseurs, mais seulement le niveau moyen de l'ensemble des fournisseurs.
2. **Coût de production.** Le terme $c_p^{\text{prod}} \sum_{j\in J} Q_{pj}$ représente le coût de production du joueur p, où $c_p^{\text{prod}} \in \mathbb{R}_+$ est un paramètre.
3. **Coût de lancement de marché.** Le terme $\sum_{j\in J} c_{pj}^{\text{setup}} b_{pj}$ représente le coût de lan-

cement de marché que le joueur p paye pour échanger avec chaque revendeur j, avec $c_{pj}^{\text{setup}} \in \mathbb{R}_+$ un paramètre.

4. **Coût de transaction variable.** La fonction $c_{pj}(Q_{pj}) = c_{pj}^{\text{quad}} Q_{pj}^2 + c_{pj}^{\text{lin}} Q_{pj}$ représente le coût de transaction variable du joueur p, qui est une somme de termes linéaires et quadratiques en Q_{pj}. Les paramètres c_{pj}^{quad} et c_{pj}^{lin} sont des valeurs positives.
5. **Coût en cybersécurité.** Le terme $h_p(s_p)$ est une fonction représentant le coût en cybersécurité du joueur p borné par le budget de cybersécurité maximum B_p. Nous détaillerons ce terme plus tard.
6. **Dégâts dues à une cyberattaque réussie.** le terme $p(s)D_p$ représente l'espérance du coût que le joueur p doit payer à cause des cyberattaques. La fonction $p(s)$ est la probabilité d'attaque réussie lorsque les niveaux de cybersécurité sont $s = (s_1, ..., s_m)$ et D_p est le coût estimé d'une cyberattaque réussie sur le joueur p. La fonction $p(s)$ est donnée par $(1 - s_p)(1 - M(s))$.

La difficulté de résolution du problème (6.28a)-(6.28c) vient des variables entières b_{pj}, les termes quadratiques des fonctions d'utilité des joueurs et le coût de cybersécurité non linéaire $h_p(s_p)$.

Coût en cybersécurité. À l'inverse du travail de [Nagurney et al., 2017], nous bornons supérieurement le niveau de cybersécurité s_p par $\bar{s}_p$. [Nagurney et al., 2017] emploient une contrainte non linéaire $h_p(s_p) \leq B_p$, avec $h_p : [0, 1] \mapsto \mathbb{R}^+$ la fonction représentant le coût pour atteindre le niveau de cybersécurité voulu s_p ; ils supposent aussi que h_p est une fonction croissante, et que $h_p(0) = 0$, $h_p(1) = \infty$. En effet pour cette dernière hypothèse, peu importe le budget dépensé en cybersécurité, il est impossible en pratique de parer toutes les cyberattaques. Sous ces conditions, $h_p(s_p) \leq B_p \Leftrightarrow s_p \leq \bar{s}_p$ où $\bar{s}_p$ est défini de manière unique par $h_p(\bar{s}_p) = B_p$. Ainsi, la contrainte non linéaire $h_p(s_p) \leq B_p$ peut être remplacée par la contrainte linéaire $s_p \leq \bar{s}_p$, avec $\bar{s}_p < 1$ à cause des bornes sur h_p. Dans nos expériences, nous utilisons deux fonctions différentes : une fonction utilisant la fonction inverse, et une fonction utilisant le logarithme, qui sont détaillées dans les deux paragraphes suivants.

Fonction inverse. La fonction inverse que nous considérons est $h_p^{\text{inv}}(s_p) = \alpha_p \left(1/\sqrt{1 - s_p} - 1\right)$ avec α_p un paramètre positif.

Nous exprimons $h_p^{\text{inv}}(s_p)$ avec deux variables auxiliaires $s_{NL}, t_{NL} \geq 0$, deux contraintes quadratiques ${s_{NL}}^2 \leq 1 - s_p$ et $s_{NL} t_{NL} \geq 1$, et l'ajout d'un terme linéaire dans la fonction objectif $-\alpha_p(t_{NL} - 1)$. Nous montrons pourquoi cette modélisation représente bien $h_p^{\text{inv}}(s_p)$. Tout d'abord, le terme de fonction objectif force t_{NL} à être fini parce que c'est un problème de maximisation. Combiné à la contrainte $s_{NL} t_{NL} \geq 1$, cela force $s_{NL} > 0$ et $t_{NL} \geq 1/s_{NL}$. De plus, avec la contrainte ${s_{NL}}^2 \leq 1 - s_p$, nous avons aussi $t_{NL} \geq 1/\sqrt{1 - s_p}$. Finalement, le terme

de la fonction objectif $-\alpha_p(t_{NL}-1)$ représente bien $h_p^{\text{inv}}(s_p)$ car la maximisation force t_{NL} à être aussi petit que possible. Avec la formulation quadratique de h_p^{inv}, les meilleures réponses (6.28a)-(6.28c) sont les solutions optimales d'un *problème d'optimisation quadratique mixte en nombres entiers avec contraintes quadratiques*, noté MIQCQP pour mixed-integer quadratically constrained quadratic programming.

Fonction logarithmique. La fonction logarithmique que nous considérons est $h_p^{\log}(s_p) = -\alpha_p log(1-s_p)$.

Nous pouvons modéliser cette fonction avec le terme linéaire $\alpha_p t_{NL}$ dans la fonction objectif et la contrainte de cône exponentiel représentée par $(1-s_p, 1, t_{NL})$, qui est satisfaite par les vecteurs de $\mathbb{R}^3$ dans l'ensemble $\{(x_1,x_2,x_3)|x_1 \geq x_2 \exp(x_3/x_2), x_1, x_2 \geq 0\}$ [ApS, 2023]. La meilleure réponse (6.28a)-(6.28c) avec cette formulation de $h_p^{\log}$ est la solution optimale d'un problème d'optimisation dans un cône exponentiel (exponential cone optimization en anglais).

6.2.2.2 Expériences numériques

La méthodologie de [Nagurney et al., 2017] est basée sur la construction d'un problème MINLP dont un optimum local fournit un équilibre de Nash du JIC. Cette méthodologie fonctionne pour des problèmes sans variables entières, donc elle ne s'applique pas à notre cas à cause des variables binaires présentes dans le JIC. À la place, nous comparons nos deux procédures à SGM directement.

Instances. Nous générons 10 instances pour chaque triplet (m,n,h_p) où $m \in \{2,3,\ldots,7\}$, $n \in \{2,3,\ldots,10\}$ et $h_p = h_p^{\text{inv}}$ ou $h_p^{\log}$. Plus précisément, nous générons les paramètres de ces instances avec une distribution uniforme sur les ensembles donnés dans la Table 6.7. Ces ensembles ont été choisis pour obtenir des valeurs proches des instances de [Nagurney et al., 2017].

Au total, nous générons donc 1080 instances que l'on résout avec la *procédure d'approximation directe*, la *procédure d'approximation à deux niveaux* et *SGM*.

Lorsque nous utilisons SGM, nous distinguons les deux algorithmes de résolution *SGM-MIQCQP* et *SGM-ConeExp* suivant la fonction de coût en cybersécurité ; si nous utilisons h_p^{inv}, le solveur de meilleure réponse est Gurobi [Gurobi Optimization, LLC, 2023] et la méthode est SGM-MIQCQP ; si c'est $h_p^{\log}$, le solveur de meilleure réponse est MOSEK [ApS, 2023] et la méthode est appelée SGM-ConeExp. Quand nous employons l'une de nos deux procédures, le solveur de meilleure réponse est Gurobi car le problème à résoudre est un problème MILP. Dans les trois cas, les solveurs choisis sont parmi les meilleurs dans l'état de l'art pour le type de problème de la meilleure réponse. Nous catégorisons nos instances en quatre groupes pour mieux mettre en valeur les facteurs influençant les performances des trois méthodes. Pour cela,

Paramètre	domaine
q_j	$\{100, 101, ..., 200\}$
m_j	$\{0.5, 0.51, ..., 2\}$
r_j	$\{0.1, 0.11, ..., 0.5\}$
c_p^{prod}	$\{1, 2, ..., 10\}$
c_{pj}^{setup}	$\{500, 501, ..., 2000\}$
c_{pj}^{lin}	$\{1, 1.01, ..., 4\}$
c_{pj}^{quad}	$\{0.25, 0.26, ..., 1\}$
α_p	$\{1, 2, ..., 10\}$
D_p	$\{50, 51, ..., 100\}$
$\bar{Q}_{pj}$	$\{50, 51, ..., 200\}$
B_p	$\{0.5, 1, ..., 5\}$

TABLE 6.7: Paramètres des instances générées.

nous distinguons suivant la fonction de coût en cybersécurité (*inv* ou *log*) et le *petit* ou *grand* nombre de joueurs ($m = 2, 3, 4$ ou $m = 5, 6, 7$). Nous appelons les groupes résultants *inv234*, *inv567*, *log234* ou *log567*.

Configuration expérimentale. Nous calculons des δ_f-équilibres de Nash avec $\delta_f = 10^{-4}$. Pour produire un δ_f-équilibre, nous modifions le critère d'arrêt de SGM pour prendre en compte l'erreur produite par le solveur de meilleure réponse. En effet, ce solveur utilise un gap relatif et un gap absolu. Nous fixons la valeur du gap relatif à 0 pour ne pas qu'il déclenche le critère d'arrêt, et le gap absolu à $\delta_{gap} = (1 - \mu)\frac{4\delta_f}{5}$, avec $\mu = 0.5$. Ainsi, le critère d'arrêt de SGM devient, pour un vecteur donné de profil de stratégies mixtes σ,

$$\max_{x^p \in \hat{X}^p} \hat{\Pi}^p(x; \sigma^{-p}) - \hat{\Pi}^p(\sigma^p; \sigma^{-p}) < \delta_f - \delta_{gap}, \qquad \forall p \in M,$$

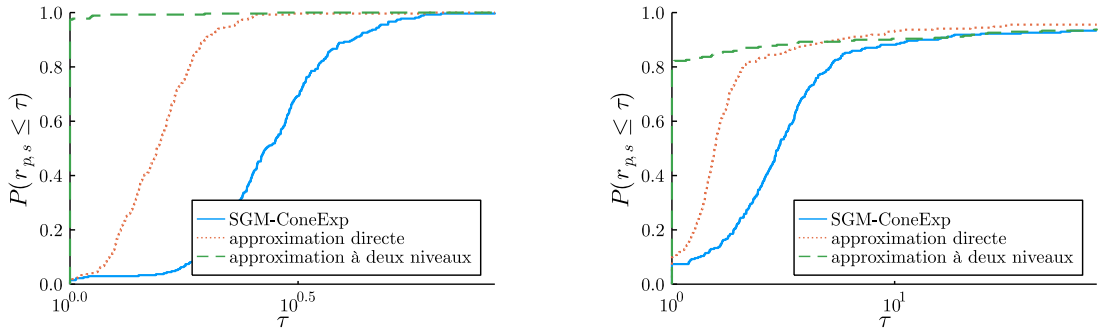
où $\hat{\Pi}^p$ et $\hat{X}^p$ sont la fonction d'utilité approchée et l'ensemble de stratégies du joueur p. Nous utilisons le modèle MILP pour calculer la solution du jeu sous forme normale, et nous le résolvons avec Gurobi. Aussi, les formulations des fonctions PWL dans les meilleures réponses sont calculées automatiquement par Gurobi grâce à la fonction *addGenConstrPWL*. À l'inverse de [Sandholm et al., 2005], nous remplaçons les contraintes utilisant des formulations big-M, et nous utilisons les contraintes indicatrices [Bonami et al., 2015]. En pratique, cela laisse le solveur décider de la meilleure stratégie pour reformuler ces conditions logiques.

Pour des raisons numériques, les paramètres *IntFeasTol* et *FeasibilityTol* de Gurobi sont fixés à 10^{-9}, et les paramètres *mioTolAbsRelaxInt* et *mioTolFeas* de MOSEK sont fixés à la même valeur. Pour approximer les fonctions de coût en cybersécurité avec des fonctions PWL,

nous utilisons la librairie `LinA` [Codsi et al., 2021] du langage Julia qui a une implémentation open-source efficace de [Codsi et al., 2021, Algorithme 5] [4]. Les meilleures réponses résolues avec Gurobi sont modélisées avec la librairie gurobipy tandis que les meilleures réponses résolues avec MOSEK sont modélisées avec la librairie `pyomo` 6.4.

La partie principale du code concernant les approximations PWL est écrite en langage Julia 1.6, tandis que SGM est implémenté en Python 3.8. Ainsi, il y a un délai chaque fois que SGM est appelé par le code Julia le temps de charger l'environnement Python utilisé. Ce temps de chargement n'est pas pris en compte dans le temps de calcul affiché car il suffirait de tout réimplémenter dans un même langage pour le supprimer.

Pour l'Algorithme 8 de la procédure d'approximation à deux niveaux, nous fixons le paramètre du niveau d'approximation initial δ_0 à 0.05, ce qui est grand par rapport à $\delta_f = 10^{-4}$. En effet, notre but est de montrer qu'utiliser une très grande valeur δ_0 par rapport à δ_f permet de trouver un δ_f-équilibre plus vite qu'avec la procédure d'approximation directe. Nous fixons un temps limite de 900 secondes pour SGM. Nous utilisons un processeur Intel i5-10310U de 1.7GHz avec 32 Go de RAM, Gurobi 9.1.1 et MOSEK 10.0 chacun avec 4 threads.



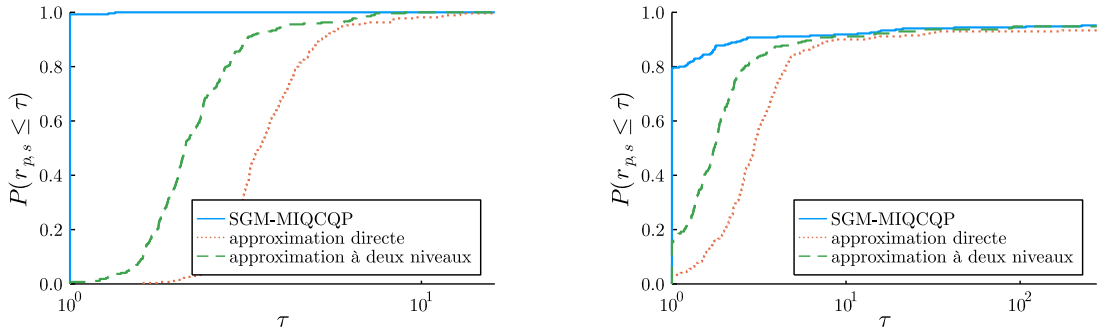
(a) Résultats pour les instances log234

(b) Résultats pour les instances log567

FIGURE 6.9: Profils de performances avec la fonction de coût en cybersécurité logarithmique

Nous présentons les résultats expérimentaux comme des *profils de performances* dans les Figures 6.9 et 6.10. Nous décrivons ici ce qu'est un profil de performance [Dolan et Moré, 2002]. Le *rapport de performance d'une instance i avec le solveur s pour l'ensemble de solveurs S* est le nombre supérieur ou égal à 1 calculé par la formule $t_{i,s} / \min_{s' \in S} t_{i,s'}$, où $t_{i,s}$ est le temps de calcul du solveur s pour l'instance i. Le profil de performance du solveur s montre la proportion d'instances résolues par s avec un rapport de performance pas plus grand que τ. Plus le profil de performance du solveur s est haut dans le graphe, plus s est performant par rapport aux autres solveurs. Finalement, nous affichons plusieurs profils de performance sur la même figure avec une échelle logarithmique en τ pour mieux montrer les différences.

4. https ://github.com/LICO-labs/LinA.jl



(a) Résultats pour les instances inv234

(b) Résultats pour les instances inv567

FIGURE 6.10: Profils de performances avec la fonction de coût en cybersécurité inverse

Type	Instances	SGM			approximation directe			approximation à deux niveaux		
		% résolues	temps (s)	iter.	% résolues	temps (s)	iter.	% résolues	temps (s)	iter.
Cône exp.	log234	100	1.04	15.61	100	0.59	16.29	100	0.38	16.32
	log567	93.3	5.06	38.44	95.6	3.22	41.5	94.1	2.19	41.17
MIQCQP	inv234	100	0.21	15.80	100	0.76	16.19	100	0.47	16.23
	inv567	95.2	1.63	39.12	93.7	3.87	39.98	94.8	2.55	40.53

TABLE 6.8: Résumé des résultats numériques

Analyse. Dans la Figure 6.9, le profil de performance de notre approximation à deux niveaux est globalement au-dessus des deux autres profils de performance, indiquant que cette méthode est globalement plus rapide que les autres sur les instances log234 et log567. Quant à la Figure 6.10, elle montre que le profil de performance de SGM-MIQCQP est toujours au-dessus des deux autres profils de performance, ce qui veut dire que cette méthode est globalement plus rapide que les autres sur les instances inv234 et inv567. Nous croisons ces informations avec les statistiques de la Table 6.8. Elle montre le pourcentage d'instances résolues, la moyenne géométrique du temps de calcul et le nombre moyen d'itérations de SGM, rassemblés par méthode et groupe d'instances. Nous utilisons une moyenne géométrique pour les temps de calcul pour diminuer l'influence sur la moyenne des instances résolues en un temps important. En effet, 91.0% des instances sont résolues en moins de 10 secondes.

Premièrement, nous résolvons toutes les petites instances (log234 et inv234) en moins de 11 secondes, tandis que 5.6% des grosses instances (log567 et inv567) ne sont pas résolues en la limite de temps (5.7% pour log567, 5.4% pour inv567). Nous observons que le nombre d'instances non résolues en la limite de temps est similaire pour les trois méthodes et varient entre 4.4% et 6.7%. Suivant la fonction de coût en cybersécurité, les solveurs ne sont pas aussi performants : les instances log234 et log567 sont résolues plus rapidement avec la procédure d'approximation à deux niveaux car elle résout plus de 80% des instances en moins de temps que les autres méthodes. Aussi, pour les groupes d'instances log234 et log567 les rapports de performance de

l'approximation à deux niveaux sont strictement les meilleurs pour τ en-dessous de 3. Si nous comparons la procédure d'approximation directe et SGM-ConeExp sur log234 (respectivement log567), la procédure d'approximation directe a un rapport de performance meilleur que SGM-ConeExp pour τ légèrement plus grande que 1 (resp. $\tau \geq 1$).

Pour les groupes d'instances inv234 et inv567, SGM-MIQCQP est la meilleure méthode avec respectivement 100% et 81% des instances résolues en moins de temps qu'avec les autres méthodes. De plus, les rapports de performance de SGM-MIQCQP sont sensiblement meilleurs que pour les autres méthodes pour $\tau \leq 5$ et $\tau \leq 3$ respectivement.

En ce qui concerne le nombre d'itérations, les résultats pour la procédure d'approximation à deux niveaux montrent seulement le nombre d'itérations du premier appel à la fonction `AppelSGM`, parce que la deuxième itération demande pour toutes les instances 5 itérations ou moins. Cela s'explique par le fait que la deuxième itération part relativement près d'un δ_f-équilibre de Nash grâce à l'initialisation. Le nombre moyen d'itérations semble ne pas dépendre de la méthode, puisque les résultats sont relativement proches.

Pour conclure sur cette analyse, les méthodes basées sur l'approximation PWL donnent de meilleurs résultats que SGM sur des instances utilisant la fonction logarithmique de coût en cybersécurité. En revanche, les résultats sont inversés pour des instances utilisant la fonction inverse. Cela peut être causé par le fait que les solveurs de problèmes MIQCQP sont plus matures que les solveurs de problèmes de cônes exponentiels ; et que peu importe le type de fonction non linéaire, les méthodes utilisant des fonctions PWL demandent toujours de résoudre des problèmes MILP. Selon cette hypothèse, plus le type de non linéarité des fonctions est difficile à résoudre, plus nos procédures seraient efficaces par rapport à une utilisation de SGM.

6.2.3 Conclusion

Nous proposons un cadre pour calculer des δ-équilibres de Nash dans des IPG avec des fonctions d'utilité non linéaires. Nous montrons que la transposition aux IPG de la méthode d'approximation des MINLP grâce aux fonctions PWL permet d'obtenir un δ-équilibre d'un IPG. D'un point de vue pratique, nous avons introduit deux procédures basées sur les approximations PWL pour calculer ces δ-équilibres de Nash. Finalement, nous montrons l'intérêt de nos méthodes sur un jeu d'investissement en cybersécurité. Nos méthodes semblent les plus intéressantes lorsque le type de problèmes d'optimisation à résoudre pour obtenir les meilleures réponses n'est pas efficacement résolu par les solveurs de l'état de l'art. Ce travail mériterait plus d'expériences numériques pour mieux connaître dans quels cas utiliser une procédure utilisant des approximations PWL, et dans quels cas utiliser simplement SGM.

6.3 Conclusion

Ce chapitre décrit deux applications numériques d'approximations linéaires par morceaux à erreur d'approximation prédéfinie. La première est issue d'un problème de placement de faisceau d'un satellite pour desservir des stations sur Terre. Elle compare deux modèles utilisant soit des faisceaux en forme d'ellipses soit des faisceaux en forme de disques. Les résultats numériques indiquent que le modèle qui utilise des faisceaux en forme d'ellipses produit de meilleurs solutions dans le temps imparti lorsque les instances possèdent peu de stations. La deuxième application est un problème de théorie des jeux en investissement en cybersécurité qui utilise une fonction non linéaire pour calculer le budget nécessaire pour atteindre un certain niveau en cybersécurité. Nous développons une méthode pour calculer des équilibres de Nash approchés puis nous la comparons numériquement à la méthode de l'état de l'art. Les résultats semblent dépendre de la fonction non linéaire de coût en cybersécurité au travers des performances du solveur de problème MINLP utilisé dans la méthode de l'état de l'art.

Conclusion

Ce manuscrit traite de la résolution du problème d'approximation de fonction de deux variables par une fonction linéaire par morceaux (fonction PWL) respectant une borne sur l'erreur d'approximation et minimisant le nombre de pièces, que l'on appelle le $\mathbb{R}^2$-corridor fitting problem ($\mathbb{R}^2$-CFP). Il a pour but l'amélioration de méthodes de résolution de problèmes MINLP.

Le Chapitre 1 a présenté une revue de littérature qui montre l'intérêt de résoudre le $\mathbb{R}^2$-CFP : il participe à donner des garanties à la méthode de résolution de MINLP par approximation en un problème MILP via les fonctions linéaires par morceaux. Il introduit aussi les méthodes de l'état de l'art résolvant un problème proche, qui servent de point de comparaison pour nos méthodes.

Le Chapitre 2 a motivé la recherche d'une fonction PWL non nécessairement continue au lieu de continue comme solution du $\mathbb{R}^2$-CFP. De plus, il a introduit la notion de corridor qui est capable de représenter toute une classe d'erreurs d'approximation. Cela a permis de définir le problème étudié dans cette thèse : le $\mathbb{R}^2$-CFP. Finalement, il a présenté un vocabulaire spécifique autour de la notion de corridor qui est utilisé dans l'ensemble du manuscrit.

Le Chapitre 3 a proposé une trame pour construire des heuristiques du $\mathbb{R}^2$-CFP basée sur trois principes pour obtenir des fonctions PWL avec peu de pièces. Nous avons identifié ces principes notamment grâce aux défauts de méthodes de l'état de l'art. Les expériences numériques dans le cas d'une erreur d'approximation dite absolue ont validé les performances des heuristiques issues de la trame en terme de qualité des solutions, tout particulièrement dans le cas des fonctions non linéairement séparables puisque l'état de l'art est battu.

Pour compléter les résultats numériques des heuristiques, le Chapitre 4 a décrit le lien entre les contraintes d'un problème de coloration d'hypergraphe et une famille d'inégalités valides du $\mathbb{R}^2$-CFP. Parmi les quatre méthodes décrites pour obtenir des bornes inférieures à ce problème, celle qui a produit les meilleurs résultats résout un problème MILP alliant contraintes du problème de coloration d'hypergraphe et respect de l'erreur d'approximation en un nombre fini de points du domaine de la fonction à approximer. Les bornes inférieures calculées combinées aux bornes supérieures de l'état de l'art ont permis de prouver que des solutions optimales pour un tiers du jeu d'instances ont été trouvées.

Dans le Chapitre 5, nous avons prouvé qu'une méthode permettant d'approximer la norme euclidienne de deux variables dans les contraintes nécessite le nombre minimum de contraintes linéaires pour respecter une erreur d'approximation spécifique. Cela a permis de prouver des solutions optimales du $\mathbb{R}^2$-CFP dans le cas de la norme euclidienne de deux variables et des normes qui ont des lignes de niveaux elliptiques.

Finalement, le Chapitre 6 a présenté deux applications à la résolution de problèmes MINLP via les fonctions PWL. Dans la première sur un problème de placement de faisceaux satellites pour couvrir des stations sur Terre, la possibilité de couvrir une zone en forme d'ellipse au lieu de disque permet de produire de meilleures solutions pour les petites instances. Dans la deuxième application sur un jeu d'investissement en cybersécurité, des équilibres de Nash approchés sont calculés à partir d'approximations par des fonctions PWL. L'analyse des résultats numériques a permis d'identifier les instances où cette méthode est plus rapide que la méthode de l'état de l'art.

Développements informatiques et publications

Cette thèse a nécessité un travail important d'implémentation informatique, notamment le travail sur la trame de génération d'heuristiques du $\mathbb{R}^2$-CFP dans le langage julia. Également, les différentes expériences numériques nous ont fait manipuler plusieurs langages de modélisation pour utiliser les solveurs MILP/MINLP : la librairie JuMP en julia, la librairie pyomo en python, les solveurs commerciaux gurobi et CPLEX, et l'appel au serveur NEOS pour le solveur commercial BARON. Finalement, les méthodes implémentées ont demandé une grande quantité de tests pour résoudre les différentes instances ou analyser l'impact des différents paramètres.

Le travail de cette thèse a donné lieu à différentes publications, notamment : un article publié dans un volume de Lecture Notes in Computer Science [Duguet et Ngueveu, 2022], un article publié dans la revue internationale Journal of Optimization Theory and Applications [Duguet et al., 2022], une présentation en conférence internationale avec actes ISCO 2022, et trois présentations en conférences nationales sans actes ROADEF 2021-2022-2023. De plus, deux rapports techniques sont en cours d'écriture à partir du Chapitre 4 et de la Section 6.2.

Perspectives

Nous présentons ci-après quatre exemples de perspectives découlant de nos travaux

Branch-and-Bound pour un problème avec des fonctions PWL. [Hübner et al., 2023] propose un Branch-and-Bound exploitant l'enveloppe convexe des fonctions PWL d'une variable pour obtenir une borne inférieure. Il serait intéressant d'étendre ces résultats en développant un algorithme efficace pour calculer l'enveloppe convexe d'une fonction PWL de plusieurs variables.

Amélioration de la trame de génération d'heuristiques. La trame fonctionne grâce à l'utilisation de différents composants et à la résolution de plusieurs sous-problèmes de natures différentes. Il serait intéressant de tester de nouveaux composants pour obtenir des fonctions PWL avec moins de morceaux. De plus, une étude plus approfondie des sous-problèmes de la trame devrait permettre de diminuer le temps de calcul, particulièrement celui de pièce maximale dans

une direction qui peut se modéliser comme un problème d'optimisation semi-infinie généralisé.

Approximation de fonctions de plus de deux variables. La théorie derrière la trame d'heuristiques et les bornes inférieures est valide pour des fonctions de plus de deux variables. L'implémentation informatique de ces méthodes pour n variables permettrait d'augmenter leur intérêt en pratique.

Analyse de cas particuliers. La norme euclidienne de deux variables et les normes de lignes de niveaux en forme d'ellipses sont des fonctions avec de fortes propriétés de structure, comme l'homogénéité absolue. Il serait intéressant d'étudier le cas d'autres fonctions avec des propriétés de structure pour tenter de calculer analytiquement une solution exacte du $\mathbb{R}^2$-CFP dans ces cas.

Bibliographie

M. ApS. *MOSEK Optimization Toolbox for MATLAB 10.0.39*, 2023. URL `https://docs.mosek.com/10.0/toolbox/index.html`. (Cité en page 148.)

P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, et A. Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22 :1–131, 2013. doi : 10.1017/S0962492913000032. (Cité en pages 12 et 21.)

J. Bezanson, A. Edelman, S. Karpinski, et V. B. Shah. Julia : A fresh approach to numerical computing. *SIAM Review*, 59(1) :65–98, 2017. doi : 10.1137/141000671. URL `https://epubs.siam.org/doi/10.1137/141000671`. (Cité en page 20.)

L. T. Biegler. *Nonlinear programming : Concepts, algorithms, and applications to chemical processes*. SIAM, 2010. (Cité en page 12.)

P. Bonami, A. Lodi, A. Tramontani, et S. Wiese. On mathematical programming with indicator constraints. *Mathematical Programming*, 151 :191–223, 2015. (Cité en page 149.)

A. Borghetti, C. D'Ambrosio, A. Lodi, et S. Martello. An MILP approach for short-term hydro scheduling and unit commitment with head-dependent reservoir. *IEEE Transactions on Power Systems*, 23(3) :1115–1124, 2008. doi : 10.1109/TPWRS.2008.926704. (Cité en page 12.)

F. Boukouvala, R. Misener, et C. A. Floudas. Global optimization advances in mixed-integer nonlinear programming, MINLP, and constrained derivative-free optimization, CDFO. *European Journal of Operational Research*, 252(3) :701–727, 2016. doi : 10.1016/j.ejor.2015.12.018. (Cité en page 12.)

R. Burlacu. *Adaptive Mixed-Integer Refinements for Solving Nonlinear Problems with Discrete Decisions*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnber, 2019. URL `https://opus4.kobv.de/opus4-fau/files/13178/phd_thesis_robert_burlacu.pdf`. (Cité en page 25.)

R. Burlacu. On refinement strategies for solving MINLPs by piecewise linear relaxations : a generalized red refinement. *Optimization Letters*, 2021. doi : 10.1007/s11590-021-01740-1. (Cité en page 25.)

R. Burlacu, B. Geißler, et L. Schewe. Solving mixed-integer nonlinear programmes using adaptively refined mixed-integer linear programmes. *Optimization Methods and Software*, 35(1) : 37–64, 2020. doi : 10.1080/10556788.2018.1556661. (Cité en page 25.)

A. Bärmann, R. Burlacu, L. Hager, et K. Kutzer. An approximation algorithm for optimal piecewise linear interpolations of bounded variable products. *Journal of Optimization Theory and Applications*, 2023. doi : 10.1007/s10957-023-02292-3. URL `https://doi.org/10.1007/s10957-023-02292-3`. (Cité en pages 17, 28 et 71.)

J.-T. Camino, S. Mourgues, C. Artigues, et L. Houssin. A greedy approach combined with graph coloring for non-uniform beam layouts under antenna constraints in multibeam satellite systems. In *2014 7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, pages 374–381, 2014. doi : 10.1109/ASMS-SPSC.2014.6934570. (Cité en pages 106, 125 et 127.)

J.-T. Camino, C. Artigues, L. Houssin, et S. Mourgues. Mixed-integer linear programming for multibeam satellite systems design : Application to the beam layout optimization. In *2016 Annual IEEE Systems Conference (SysCon)*, pages 1–6, 2016. doi : 10.1109/SYSCON.2016.7490613. (Cité en page 125.)

J.-T. Camino, C. Artigues, L. Houssin, et S. Mourgues. Linearisation of euclidean norm dependent inequalities applied to multibeam satellites design. *Computational Optimization and Applications*, 2019. doi : 10.1007/s10589-019-00083-z. (Cité en pages viii, 2, 105, 106, 107, 123, 125, 126 et 131.)

M. Carvalho, G. Dragotto, A. Lodi, et S. Sankaranarayanan. The cut and play algorithm : Computing nash equilibria via outer approximations. *ARXIV*, 2021. URL `https://arxiv.org/abs/2111.05726`. (Cité en page 140.)

M. Carvalho, A. Lodi, et J. Pedroso. Computing equilibria for integer programming games. *European Journal of Operational Research*, 2022. ISSN 0377-2217. doi : 10.1016/j.ejor.2022.03.048. URL `https://www.sciencedirect.com/science/article/pii/S0377221722002727`. (Cité en pages 140 et 142.)

M. Carvalho, G. Dragotto, A. Lodi, et S. Sankaranarayanan. Integer programming games : a gentle computational overview. *INFORMS Tutorials in Operations Research*, (forthcoming), 2023. URL `https://arxiv.org/abs/2306.02817`. (Cité en page 141.)

D. Z. Chen et H. Wang. Approximating points by a piecewise linear function. *Algorithmica*, 2013. doi : 10.1007/s00453-012-9658. (Cité en page 26.)

X. Chen et X. Deng. Settling the complexity of two-player Nash equilibrium. In *Foundations of Computer Science, 2006. FOCS '06. 47th Annual IEEE Symposium on*, pages 261–272, Oct 2006. doi : 10.1109/FOCS.2006.69. (Cité en page 142.)

J. Codsi, S. U. Ngueveu, et B. Gendron. Lina : A faster approach to piecewise linear approximations using corridors and its application to mixed-integer optimization. Technical report, hal-03336003, 2021. (Cité en pages 28, 38, 39, 40, 68 et 150.)

C. Contardo et S. U. Ngueveu. On the approximation of separable non-convex optimization programs to an arbitrary numerical precision. Technical report, School of Business, University of Québec in Montreal, Canada Université de Toulouse, CNRS, INP, LAAS, Toulouse, France, 2021. (Cité en pages 23 et 25.)

T. Crönert et S. Minner. Equilibrium identification and selection in finite games. *Operations Research*, 0(0) :null, 2022. doi : 10.1287/opre.2022.2413. URL `https://doi.org/10.1287/opre.2022.2413`. (Cité en page 140.)

J. Czyzyk, M. P. Mesnier, et J. J. Moré. The neos server. *IEEE Journal on Computational Science and Engineering*, 5(3) :68–75, 1998. doi : 10.1109/99.714603. (Cité en page 132.)

J. Desrosiers et M. E. Lübbecke. *A Primer in Column Generation*, pages 1–32. Springer US, Boston, MA, 2005. ISBN 978-0-387-25486-9. doi : 10.1007/0-387-25486-2_1. URL `https://doi.org/10.1007/0-387-25486-2_1`. (Cité en page 142.)

E. D. Dolan. The neos server 4.0 administrative guide. Technical Memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory, 2001. (Cité en page 132.)

E. D. Dolan et J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 2002. doi : 10.1007/s101070100263. (Cité en page 150.)

G. Dragotto et R. Scatamacchia. The zero regrets algorithm : Optimizing over pure Nash equilibria via integer programming. *INFORMS Journal on Computing (forthcoming)*, 2023. URL `https://arxiv.org/abs/2111.06382`. (Cité en page 140.)

A. Duguet et S. U. Ngueveu. Piecewise linearization of bivariate nonlinear functions : Minimizing the number of pieces under a bounded approximation error. In I. Ljubić, F. Barahona, S. S. Dey, et A. R. Mahjoub, editors, *Combinatorial Optimization*, pages 117–129, Cham, 2022. Springer International Publishing. ISBN 978-3-031-18530-4. (Cité en pages 28 et 156.)

A. Duguet, C. Artigues, L. Houssin, et S. U. Ngueveu. Properties, extensions and application of piecewise linearization for euclidean norm optimization in $\mathbb{R}^2$. *Journal of Optimization Theory and Applications*, 2022. doi : 10.1007/s10957-022-02083-2. URL `https://doi.org/10.1007/s10957-022-02083-2`. (Cité en pages 136 et 156.)

J. G. Dunham. Optimum uniform piecewise linear approximation of planar curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(1) :67–75, 1986. doi : 10.1109/TPAMI.1986.4767753. (Cité en page 27.)

C. D'Ambrosio, A. Lodi, et S. Martello. Piecewise linear approximation of functions of two variables in MILP models. *Operations Research Letters*, 38(1) :39–46, 2010. doi : 10.1016/j.orl.2009.09.005. (Cité en page 17.)

C. Frenzen, T. Sasao, et J. T. Butler. On the number of segments needed in a piecewise linear approximation. *Journal of Computational and Applied Mathematics*, 234(2) :437–446, 2010. doi : 10.1016/j.cam.2009.12.035. (Cité en page 60.)

M. R. Garey et D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. 1979. (Cité en page 76.)

B. Geißler, A. Martin, A. Morsi, et L. Schewe. Using piecewise linear functions for solving MINLPs. In J. Lee et S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, pages 287–314, New York, NY, 2012. Springer New York. ISBN 978-1-4614-1927-3. doi : 10.1007/978-1-4614-1927-3_10. (Cité en pages 1, 13 et 23.)

K. Ghedira. *Constraint satisfaction problems : CSP formalisms and techniques*. John Wiley & Sons, 2013. (Cité en page 92.)

W. Gropp et J. J. Moré. Optimization environments and the NEOS server. In M. D. Buhman et A. Iserles, editors, *Approximation Theory and Optimization*, pages 167–182. Cambridge University Press, 1997. URL `https://www.osti.gov/biblio/563264`. (Cité en page 132.)

F. Guerra Vázquez, J.-J. Rückmann, O. Stein, et G. Still. Generalized semi-infinite programming : A tutorial. *Journal of Computational and Applied Mathematics*, 217(2) : 394–419, 2008. ISSN 0377-0427. doi : https://doi.org/10.1016/j.cam.2007.02.012. URL `https://www.sciencedirect.com/science/article/pii/S0377042707000982`. Special Issue : Semi-infinite Programming (SIP). (Cité en page 49.)

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL `https://www.gurobi.com`. (Cité en page 148.)

S. Hakimi et E. Schmeichel. Fitting polygonal functions to a set of points in the plane. *CVGIP : Graphical Models and Image Processing*, 53(2) :132–136, 1991. doi : 10.1016/1049-9652(91)90056-P. (Cité en page 27.)

F. M. Hante et M. Schmidt. Gas transport network optimization : Mixed-integer nonlinear models. Technical report, 2023. (Cité en page 12.)

E. Hebrard et G. Katsirelos. Constraint and satisfiability reasoning for graph coloring. *Journal of Artificial Intelligence Research*, 2020. doi : 10.1613/jair.1.11313. (Cité en page 92.)

J. Huchette et J. P. Vielma. Nonconvex piecewise linear functions : Advanced formulations and simple modeling tools. *Operations Research*, 2022. doi : 10.1287/opre.2019.1973. URL `https://doi.org/10.1287/opre.2019.1973`. (Cité en pages 17, 20 et 21.)

R. B. Hughes et M. R. Anderson. Simplexity of the cube. *Discrete Mathematics*, 158(1) :99–150, 1996. doi : 10.1016/0012-365X(95)00075-8. (Cité en page 16.)

T. Hübner, A. Gupte, et S. Rebennack. Spatial branch-and-bound for non-convex separable piecewise-linear optimization. Technical report, 2023. URL `https://optimization-online.org/2023/04/spatial-branch-and-bound-for-non-convex-separable-piecewise-linear-optimization/`. (Cité en pages 21 et 156.)

R. G. Jeroslow. The polynomial hierarchy and a simple model for competitive analysis. *Mathematical Programming*, 32 :146–164, 1985. doi : 10.1007/BF01586088. (Cité en page 14.)

J. Kallrath et S. Rebennack. Cutting ellipses from area-minimizing rectangles. *Journal of Global Optimization*, 59 :405–437, 2014. doi : 10.1007/s10898-013-0125-3. (Cité en page 129.)

R. Kannan et C. L. Monma. On the computational complexity of integer programming problems. In R. Henn, B. Korte, et W. Oettli, editors, *Optimization and Operations Research*, pages 161–172, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg. ISBN 978-3-642-95322-4. doi : 10.1007/978-3-642-95322-4_17. (Cité en page 13.)

R. M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. ISBN 978-1-4684-2001-2. doi : 10.1007/978-1-4684-2001-2_9. URL `https://doi.org/10.1007/978-1-4684-2001-2_9`. (Cité en page 85.)

K. Kazda et X. Li. Nonconvex multivariate piecewise-linear fitting using the difference-of-convex representation. *Computers & Chemical Engineering*, 150 :107310, 2021. doi : 10.1016/j.compchemeng.2021.107310. (Cité en pages 28, 30 et 80.)

K. Kazda et X. Li. A linear programming approach to difference-of-convex piecewise linear approximation. *European Journal of Operational Research*, 2023. ISSN 0377-2217. doi : https://doi.org/10.1016/j.ejor.2023.07.026. URL `https://www.sciencedirect.com/science/article/pii/S0377221723005647`. (Cité en pages 28, 32 et 33.)

A. B. Keha, I. R. de Farias, et G. L. Nemhauser. Models for representing piecewise linear cost functions. *Operations Research Letters*, 32(1) :44–48, 2004. doi : 10.1016/S0167-6377(03)00059-2. (Cité en page 17.)

A. B. Keha, J. Ismael R. de Farias, et G. L. Nemhauser. A branch-and-cut algorithm without binary variables for nonconvex piecewise linear optimization. *Operations Research*, 2006. doi : 10.1287/opre.1060.0277. (Cité en page 21.)

V. Klee et G. J. Minty. How good is the simplex algorithm. *Inequalities*, pages 159–175, 1972. (Cité en page 13.)

L. Kong et C. T. Maravelias. On the derivation of continuous piecewise linear approximating functions. *INFORMS Journal on Computing*, 32(3) :531–546, 2020. doi : 10.1287/ijoc.2019.0949. (Cité en pages 27, 28, 38 et 80.)

B. Lyu, I. V. Hicks, et J. Huchette. Building formulations for piecewise linear relaxations of nonlinear functions. *Arxiv*, 2023. URL `https://arxiv.org/pdf/2304.14542.pdf`. (Cité en page 21.)

C. D. Maranas et C. A. Floudas. Global minimum potential energy conformations of small molecules. *Journal of Global Optimization*, 4 :135–170, 1994. doi : 10.1007/BF01096720. (Cité en page 12.)

A. G. Medeiros et L. C. Resendo. Mixed integer linear programming approaches to optimising oil production in offshore petroleum fields. *International Journal of Mathematics in Operational Research*, 22(4) :468–495, 2022. doi : 10.1504/IJMOR.2022.126045. URL `https://www.inderscienceonline.com/doi/abs/10.1504/IJMOR.2022.126045`. (Cité en page 12.)

R. Misener et C. A. Floudas. Piecewise-linear approximations of multidimensional functions. *Journal of Optimization Theory and Applications*, 145 :120–147, 2010. doi : 10.1007/s10957-009-9626-0. (Cité en page 13.)

R. Misener, C. E. Gounaris, et C. A. Floudas. Global optimization of gas lifting operations : A comparative study of piecewise linear formulations. *Industrial & Engineering Chemistry Research*, 48(13) :6098–6104, 2009. doi : 10.1021/ie8012117. (Cité en page 12.)

A. Nagurney, P. Daniele, et S. Shukla. A supply chain network game theory model of cybersecurity investments with nonlinear budget constraints. *Annals of Operations Research*, 2017. doi : 10.1007/s10479-016-2209-1. (Cité en pages 145, 146, 147 et 148.)

J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2) :286–295, 1951. doi : 10.2307/1969529. (Cité en page 140.)

S. U. Ngueveu. Piecewise linear bounding of univariate nonlinear functions and resulting mixed integer linear programming-based solution methods. *European Journal of Operational Research*, 275(3) :1058–1071, 2019. doi : 10.1016/j.ejor.2018.11.021. (Cité en page 27.)

J. O'Rourke. An on-line algorithm for fitting straight lines between data ranges. *Programming Techniques and Data Structures*, 24(9) :574–578, 1981. doi : 10.1145/358746.358758. (Cité en pages 27, 28, 82 et 95.)

M. Padberg. Approximating separable nonlinear functions via mixed zero-one programs. *Operations Research Letters*, 27(1) :1–5, 2000. doi : 10.1016/S0167-6377(00)00028-6. (Cité en pages 17 et 20.)

R. Porter, E. Nudelman, et Y. Shoham. Simple search methods for finding a Nash equilibrium. *Games and Economic Behavior*, 63(2) :642 – 662, 2008. ISSN 0899-8256. Second World Congress of the Game Theory Society. (Cité en page 145.)

F. A. Potra et S. J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1) :281–302, 2000. ISSN 0377-0427. doi : https://doi.org/10.1016/S0377-0427(00)00433-7. URL `https://www.sciencedirect.com/science/article/pii/S0377042700004337`. Numerical Analysis 2000. Vol. IV : Optimization and Nonlinear Equations. (Cité en page 13.)

S. Rao, M. Tang, et C.-C. Hsu. Multiple beam antenna technology for satellite communications payloads. *ACES Journal*, 21(3) :1054–4887, 2006. doi : 10.2514/6.2007-3179. (Cité en page 126.)

S. Rebennack. Computing tight bounds via piecewise linear functions through the example of circle cutting problems. *Mathematical Methods of Operations Research*, 2016. doi : 10.1007/s00186-016-0546-0. (Cité en page 21.)

S. Rebennack et J. Kallrath. Continuous piecewise linear delta-approximations for bivariate and multivariate functions. *Journal of Optimization Theory and Applications*, 167 :102–117, 2015a. doi : 10.1007/s10957-014-0688-2. (Cité en pages vii, 28, 29, 32, 62, 63, 71, 94, 101 et 104.)

S. Rebennack et J. Kallrath. Continuous piecewise linear delta-approximations for univariate functions : Computing minimal breakpoint systems. *Journal of Optimization Theory and Applications*, 167 :617–643, 2015b. doi : 10.1007/s10957-014-0687-3. (Cité en page 29.)

S. Rebennack et V. Krasko. Piecewise linear function fitting via mixed-integer linear programming. *INFORMS Journal on Computing*, 32(2) :507–530, 2020. doi : 10.1287/ijoc.2019.0890. (Cité en pages 26, 27, 28, 38 et 80.)

S. Sagratella. Computing all solutions of Nash equilibrium problems with discrete strategy sets. *SIAM Journal on Optimization*, 26(4) :2190–2218, 2016. ISSN 1052-6234, 1095-7189. doi : 10.1137/15M1052445. (Cité en page 140.)

T. Sandholm, A. Gilpin, et V. Conitzer. Mixed-integer programming methods for finding Nash equilibria. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*, AAAI'05, pages 495–501. AAAI Press, 2005. ISBN 1-57735-236-x. (Cité en pages 145 et 149.)

A. Santoyo-González et C. Cervelló-Pastor. Latency-aware cost optimization of the service infrastructure placement in 5g networks. *Journal of Network and Computer Applications*, 114 :29–37, 2018. doi : 10.1016/j.jnca.2018.04.007. (Cité en page 106.)

S. Schwarze et O. Stein. A branch-and-prune algorithm for discrete Nash equilibrium problems. *Optimization Online*, Preprint ID 2022-03-8836 :27, 2022. doi : 10.1007/s10589-023-00500-4. (Cité en page 140.)

S. Sridhar, J. Linderoth, et J. Luedtke. Locally ideal formulations for piecewise linear functions with indicator variables. *Operations Research Letters*, 41(6) :627–632, 2013. doi : 10.1016/j.orl.2013.08.010. (Cité en page 17.)

P. Van Hentenryck. *Principles and Practice of Constraint Programming-CP 2002 : 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, volume 2470. Springer, 2003. (Cité en page 92.)

J. P. Vielma. Mixed integer linear programming formulation techniques. *SIAM Review*, 57(1) : 3–57, 2015. doi : 10.1137/130915303. URL `https://doi.org/10.1137/130915303`. (Cité en pages 17 et 20.)

J. P. Vielma et G. L. Nemhauser. Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. *Mathematical Programming Series A*, 128 :49–72, 2011. doi : 10.1007/s10107-009-0295-4. (Cité en pages 17, 20 et 21.)

J. P. Vielma, S. Ahmed, et G. Nemhauser. Mixed-integer models for nonseparable piecewise-linear optimization : Unifying framework and extensions. *Operations Research*, 58(2), 2010. doi : 10.1287/opre.1090.0721. (Cité en pages 17, 19 et 20.)

D. L. Wilson. *Polyhedral methods for piecewise-linear functions.* PhD thesis, University of Kentucky, 1998. (Cité en page 20.)

D. Yue et F. You. Game-theoretic modeling and optimization of multi-echelon supply chain design and operation under Stackelberg game and market equilibrium. *Computers & Chemical Engineering*, 71 :347–361, 2014. ISSN 0098-1354. doi : https://doi.org/10.1016/j.compchemeng.2014.08.010. URL `https://www.sciencedirect.com/science/article/pii/S0098135414002567`. (Cité en page 25.)

H. Zhang et S. Wang. Linearly constrained global optimization via piecewise-linear approximation. *Journal of Computational and Applied Mathematics*, 214(1) :111–120, 2008. doi : 10.1016/j.cam.2007.02.006. (Cité en pages 16 et 22.)

C. Zhou, A. Mazumder, A. Das, K. Basu, N. Matin-Moghaddam, S. Mehrani, et A. Sen. Relay node placement under budget constraint. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, pages 1–11, 2018. doi : 10.1145/3154273.3154302. (Cité en page 106.)

Annexe A

Modèle MILP de la méthode KL21

Le modèle MILP utilisé dans la méthode KL21 est (A.1)-(A.11).

$$\Phi_{qt} = \min \quad \sum_{l \in \{1,...,|D_q|\}} (s_l^+ + s_l^-) \tag{A.1}$$

$$s.c. \quad f_l^{PWL} \leq F_l^{NL,sup,\delta} + s_l^+ \qquad \forall l \in \{1,...,|D_q|\}, \tag{A.2}$$

$$f_l^{PWL} \geq F_l^{NL,inf,\delta} - s_l^- \qquad \forall l \in \{1,...,|D_q|\}, \tag{A.3}$$

$$f_l^{PWL} = f_l^{PWL,+} - f_l^{PWL,-} \qquad \forall l \in \{1,...,|D_q|\}, \tag{A.4}$$

$$f_l^{PWL,c} \geq m_{iq}^c X_l + c_{iq}^c \qquad \forall l \in \{1,...,|D_q|\}, c \in \{+,-\}, i \in \{1,...,p_{qt}^c + 1\}, \tag{A.5}$$

$$f_l^{PWL,c} \leq m_{iq}^c X_l + c_{iq}^c + M^{Big,c}(1 - z_{il}^c) \qquad \forall l \in \{1,...,|D_q|\}, c \in \{+,-\}, i \in \{1,...,p_{qt}^c + 1\}, \tag{A.6}$$

$$\sum_{i=1}^{p_{qt}^c+1} z_{il}^c = 1 \qquad \forall l \in \{1,...,|D_q|\}, c \in \{+,-\}, \tag{A.7}$$

$$z_{il}^c \leq r_t^c \qquad \forall l \in \{1,...,|D_q|\}, c \in \{+,-\}, i = p_{qt}^c + 1, \tag{A.8}$$

$$r_t^+ + r_t^- = 1 \qquad \forall l \in \{1,...,|D_q|\}, c \in \{+,-\}, \tag{A.9}$$

$$s_l^+, s_l^- \geq 0 \qquad \forall l \in \{1,...,|D_q|\}, \tag{A.10}$$

$$r_t^+, r_t^-, z_{il}^c \in \{0,1\} \qquad \forall l \in \{1,...,|D_q|\}, c \in \{+,-\}, i \in \{1,...,p_{qt}^c + 1\}. \tag{A.11}$$

Les paramètres $F_l^{NL,sup,\delta}$ et $F_l^{NL,inf,\delta}$ sont les bornes supérieures et inférieures à respecter pour satisfaire l'erreur d'approximation absolue de δ au point X_l de D_q. Les variables donnant les valeurs des composants convexe et concave au point X_l sont $f_l^{PWL,+}$ et $f_l^{PWL,-}$; f_l^{PWL} est la variable donnant la valeur de la fonction DC CPWL en X_l. Les contraintes (A.2) et (A.3) forcent la fonction DC CPWL à respecter l'erreur d'approximation au point X_l de D_q avec des marges positives de s_l^+ et s_l^-. Ces marges sont comptabilisées dans la fonction objectif (A.1) pour qu'une valeur optimale de 0 indique que la fonction DC CPWL respecte l'erreur d'approximation. Les contraintes (A.4) permettent de construire la fonction DC CPWL à partir des composants convexe et concave. Les contraintes (A.5) et (A.6) assurent que les composants convexe et concave valent le maximum sur les fonctions linéaires $m_{iq}^c X_l + c_{iq}^c$ en chaque point X_l. C'est la

variable binaire z_{il}^c qui indique si le point X_l est couvert par la fonction linéaire $m_{iq}^c X_l + c_{iq}^c$ grâce au big-M $M^{big,c}$. Les contraintes (A.7) assurent que chaque point X_l est couvert par une seule fonction linéaire de chaque composante. La somme se fait jusqu'à la valeur $p_{qt}^c + 1$ pour ajouter une fonction linéaire au composant convexe ou concave grâce aux variables r_t^+ et r_t^- et aux contraintes (A.8) et (A.9). Cet ajout de fonction linéaire à un des deux composants permet de sélectionner le composant pour lequel ajouter une fonction linéaire diminue le plus l'objectif. Finalement, les contraintes (A.10) donnent la positivité des marges s_l^+ et s_l^-, et les contraintes (A.11) assurent que les variables z_{il}^c soient binaires.

Quand ce modèle MILP est résolu, si la valeur optimale est 0, la méthode KL21 passe à l'étape de vérification du respect de l'erreur d'approximation grâce aux problèmes NLP. Sinon, une fonction linéaire est ajoutée au composant dont la variable r_t^c vaut 1, et le modèle (A.1)-(A.11) est résolu à nouveau. Cette nouvelle résolution aura une valeur optimale inférieure puisqu'il y a plus de fonctions linéaires.

Annexe B

Annexes sur les heuristiques pour la résolution du $\mathbb{R}^2$-CFP

B.1 Approximation de la variation totale des dérivées partielles

L'approximation de la variation totale des dérivées partielles des fonctions u et l définissant un corridor $Corridor(u, l, D)$ est faite en deux étapes. D'abord, le polygone convexe D est échantillonné, puis les dérivées partielles de u et l évaluées sur les échantillons sont utilisées pour calculer l'approximation.

Nous cherchons au moins N_{VaT} points dans le domaine du corridor D. Pour trouver ces points, nous calculons le plus petit rectangle dont la base est parallèle à l'axe des abscisses et qui contient le domaine de la pièce. Nous échantillonnons ensuite ce rectangle sur une grille régulière et enfin nous supprimons les échantillons qui ne sont pas dans le domaine du corridor. Si le nombre de points restants est strictement inférieur à N_{VaT}, nous recommençons l'échantillonnage avec 4 fois plus de points dans la grille régulière jusqu'à atteindre au moins N_{VaT} échantillons. Après cela, nous calculons pour chacun de ces échantillons la somme des normes euclidiennes des gradients des dérivées partielles de u et de l. En effet, ce sont les termes à intégrer pour calculer la variation totale des dérivées partielles de u et de l. L'approximation de la variation totale est alors la moyenne de ces valeurs multipliée par l'aire du domaine de la pièce.

B.2 Temps des méthodes de l'état de l'art pour le $\mathbb{R}^2$-CFP

La Table B.1 montre les temps de calcul en secondes utilisés par les différentes méthodes de l'état de l'art pour résoudre une instance de $\mathbb{R}^2$-CFP. Les astérisques dans les titres des colonnes indiquent que les temps affichés ne sont pas les temps totaux, mais seulement la somme des temps de résolution des problèmes LP de l'instance. Donc, il manque le temps de résolution des problèmes NLP. La mention "TL" indique qu'aucune solution n'a été trouvée car la limite de temps a été atteinte.

ref	δ	DN95	DN99	RK2D	LinA1D	KL21	KL23-LP*	KL23LPrelaxed*
	1.5	11.6	17.4	30.8	0.014	87.1	0.97	0.29
	1.0	20.6	25.2	84.4	0.004	148.9	17.0	0.8
L1	0.5	33.0	48.0	150.4	0.004	TL	51.0	3.0
	0.25	50.6	88.9	272.6	0.004	TL	2197.0	23.0
	0.1	141.9	215.8	380.6	0.006	TL	8454.0	156.0
	1.5	9.7	20.8	26.8	0.003	TL	6.0	30.0
	1.0	19.0	23.9	7.4	0.003	1495.5	392.0	1.7
L2	0.5	24.0	38.3	38.0	0.003	TL	11449.0	45.0
	0.25	45.8	69.2	35.8	0.004	TL	TL	361.0
	0.1	106.1	184.3	171.7	0.004	TL	TL	TL
	1.0	5.2	11.6	0.8	0.003	18.5	0.2	0.1
	0.5	11.8	17.5	72.4	0.004	416.8	8.3	1.6
N1	0.25	25.6	41.5	4.7	0.004	14762.5	152.0	6.0
	0.1	51.3	105.2	59.3	0.006	TL	1912.0	46.0
	0.05	145.9	291.7	45.3	0.006	TL	30824.0	88.0
	0.1	0.552	8.1	0.3	0.004	6.0	0.03	0.04
	0.05	2.9	7.4	18.7	0.006	13.4	0.06	0.14
N2	0.03	5.9	13.9	12.7	0.008	39.0	0.34	0.43
	0.01	26.0	82.3	54.6	0.005	TL	244.0	6.2
	0.001	344.5	1212.7	652.6	0.008	TL	TL	8418.0
	1.0	4.1	11.0	1.0	0.004	TL	0.05	0.05
	0.5	6.4	11.2	13.1	0.007	TL	3.9	0.11
N3	0.25	11.8	19.4	30.0	0.007	342.6	11.0	0.31
	0.1	29.7	66.3	74.6	0.008	TL	45.0	8.0
	0.05	76.3	181.2	141.9	0.013	TL	3514.0	37.0
	0.5	2.7	10.7	1.8	0.004	13.3	0.03	0.04
	0.25	5.4	10.3	1.0	0.007	19.6	0.08	0.12
N4	0.1	16.4	43.0	25.8	0.007	66.9	2.8	0.67
	0.05	41.0	116.1	14.4	0.006	7246.5	106.0	3.4
	0.03	85.6	205.2	161.4	0.006	TL	267.0	12.0
	1.0	2.9	6.5	7.7	0.006	6.3	0.02	0.03
	0.5	10.2	19.7	1.3	0.007	86.9	0.46	0.52
N5	0.25	18.9	48.8	30.8	0.007	2091.2	37.0	1.1
	0.1	39.4	106.1	73.0	0.01	TL	2031.0	44.0
	0.05	108.1	206.9	305.5	0.012	TL	35999.0	130.0
	1.0	4.3	12.4	22.8	0.267	23.3	0.22	0.29
	0.5	6.2	12.3	15.6	0.004	127.2	1.2	0.43
N6	0.25	11.1	31.1	22.9	0.005	305.5	62.0	11.0
	0.1	41.5	65.5	202.8	0.006	TL	115.0	17.0
	0.05	107.5	183.8	174.1	0.006	TL	9448.0	80.0
	1.0	1.1	8.6	0.8	0.003	2.9	0.01	0.004
	0.5	5.6	35.5	66.6	0.003	9.5	0.05	0.07
N7	0.25	10.2	33.6	4.4	0.004	35.5	40.0	2.9
	0.1	44.0	143.9	231.5	0.004	377.7	285.0	1.8
	0.05	67.6	578.8	57.8	0.004	TL	178.0	8.0

TABLE B.1: Comparaison des temps de calcul des heuristiques de l'état de l'art utilisant une erreur d'approximation absolue

Annexe C

Approximation à erreur relative d'un IPG

Le but de cette partie est d'établir un résultat similaire à la Proposition 60 sur le calcul de δ-équilibres avec l'erreur relative au lieu de l'erreur absolue.

De manière similaire à un δ-équilibre, un ϵ-équilibre relatif est une relaxation d'un équilibre de Nash, mais définie pour des fonctions d'utilité strictement positives. Nous disons qu'un profil de stratégie $\hat{\sigma}$ est un ϵ-équilibre relatif d'un jeu avec ϵ dans $[0, 1[$ si aucun joueur n'a d'intérêt à dévier de $\hat{\sigma}$ pour augmenter son utilité $\Pi(\hat{\sigma})$ de plus de $\epsilon\Pi(\hat{\sigma})$.

Proposition 61. *Soit G un IPG dans lequel le problème d'optimisation de chaque joueur p est de la forme* (6.22)*. Nous supposons que $\hat{G}$ est un IPG pour chaque joueur p qui résout le problème d'optimisation*

$$\max_{x^p}\{\hat{\Pi}^p(x^p; x^{-p}) := \hat{f}^p(x^p) + g^p(x^p; x^{-p}) | x^p \in X^p\},$$

où $\hat{f}^p$ est une approximation relative de f^p à ϵ près dans X^p. Alors, (i.) un équilibre de Nash $\hat{\sigma}$ de $\hat{G}$ est un $\frac{2\epsilon}{1-\epsilon}$-équilibre relatif de G, et (ii.) un ϵ_M-équilibre relatif de $\hat{G}$ est un $\frac{2\epsilon+\epsilon_M+\epsilon\epsilon_M}{1-\epsilon}$-équilibre de G.

Démonstration. Soit p n'importe quel joueur du jeu G. Nous savons que :
Fait 1 : $\hat{\Pi}^p = \hat{f}^p(x^p) + g^p(x^p, x^{-p})$ est une approximation relative de $\Pi^p = f^p(x^p) + g^p(x^p, x^{-p})$ à ϵ près pour tout x^{-p} parce que $\hat{f}^p$ en est une aussi.
Fait 2 : $\hat{\sigma}$ est un équilibre de Nash de $\hat{G}$.

Nous avons successivement :

$$\begin{aligned}(1+\epsilon)\Pi^p(\hat{\sigma}^p, \hat{\sigma}^{-p}) &\geq \hat{\Pi}^p(\hat{\sigma}^p, \hat{\sigma}^{-p}) && \text{(C.1)}\\ &\geq \hat{\Pi}^p(x^p, \hat{\sigma}^{-p}) \ \ \forall x^p \in X^p && \text{(C.2)}\\ &\geq (1-\epsilon)\Pi^p(x^p, \hat{\sigma}^{-p}) \ \ \forall x^p \in X^p, && \text{(C.3)}\end{aligned}$$

où, le *fait 1* mène aux inégalités (C.1) et (C.3), et le *fait 2* mène à l'inégalité (C.2). Alors, une division par $1-\epsilon$ des deux côtés donne l'inégalité

$$\frac{1+\epsilon}{1-\epsilon}\Pi_p(\hat{\sigma}^p, \hat{\sigma}^{-p}) \geq \Pi^p(x^p, \hat{\sigma}^{-p}) \ \ \forall x^p \in X^p. \quad \text{(C.4)}$$

Finalement,

$$\frac{1+\epsilon}{1-\epsilon} - 1 = \frac{2\epsilon}{1-\epsilon} \tag{C.5}$$

et nous avons montré que $\hat{\sigma}$ est un $\frac{2\epsilon}{1-\epsilon}$-équilibre de G. De plus, si le *fait 2* est remplacé par le *fait 3* : $\hat{\sigma}$ est un ϵ_M-équilibre relatif de $\hat{G}$; alors d'une manière similaire $\hat{\sigma}$ est un $\frac{2\epsilon+\epsilon_M+\epsilon\epsilon_M}{1-\epsilon}$-équilibre relatif de G. □