UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Génie Industriel**

Arrêté ministériel : 7 août 2006

Présentée par

Manuel Ruiz

Thèse dirigée par **Bernard Penz**
et coencadrée par **Olivier Briant**

préparée au sein **du laboratoire G-SCOP**
et de **l'école doctorale I-MEP**^{2}

Une approche exacte de résolution de problèmes de pooling appliquée à la fabrication d'aliments

Thèse soutenue publiquement le ,
devant le jury composé de :

Philippe Mahey
Professeur, Université Blaise Pascal Clermont II, Président proposé
Leo Liberti
Professeur, École Polytechnique, Rapporteur
Frédéric Messine
Maître de Conférences Habilité à Diriger les Recherches, Institut de Recherche en Informatique de Toulouse, Rapporteur
Bernard Penz
Professeur, Institut Polytechnique de Grenoble, Directeur de thèse
Olivier Briant
Maitre de conférences, Institut Polytechnique de Grenoble, Co-Encadrant de thèse
Jean-Maurice Clochard
Directeur technique de A-Systems, Versailles , Co-Encadrant de thèse

Table des matières

Table des matières i

Table des figures vii

Liste des tableaux ix

Introduction 1

1 La fabrication d'aliments pour bétail 3

1.1 Le contexte métier 3

1.1.1 La formulation d'aliments 3

1.1.2 Les contraintes métiers 4

1.2 La formulation simple 5

1.2.1 Le problème de fabrication directe 5

1.2.2 Les notations 6

1.2.3 La modélisation mathématique 7

1.2.4 La résolution 8

1.3 La formulation avec pré-mélanges 9
1.3.1 Une fabrication à deux niveaux de mélange 9
1.3.2 Les notations 10
1.3.3 La modélisation 11
1.3.4 La résolution 13
1.4 Conclusion 13

2 Revue de la littérature sur les problèmes de *pooling* et les méthodes de résolution 15

2.1 Définitions et modélisation des problèmes de *pooling* 15
2.1.1 Exemple historique 16
2.1.2 Définition formelle du *pooling standard* et quelques extensions 17
2.1.3 Les deux principales formulations du *pooling standard* 18
2.1.4 Les applications des problèmes de *pooling* 20
2.2 Notations utilisées pour décrire les méthodes de la littérature 21
2.3 Les méthodes exactes 25
2.3.1 Les méthodes basées sur les *Branch-and-Bound* 25
2.3.2 Les méthodes de type décomposition de Benders 27
2.3.3 L'algorithme *Global OPtimization* (*GOP*) 29
2.4 Les méthodes de calcul de solutions réalisables 32
2.5 Les méthodes de calcul de bornes inférieures 33
2.6 Conclusion 38

3 Un approche de résolution par *Branch-and-Bound* 39

3.1 Description générale de l'algorithme 39

3.1.1 Notations allégées 40

3.1.2 Schéma de principe pour l'algorithme 41

3.2 Les techniques de contraction de bornes utilisées 43

3.2.1 Réduction du domaine réalisable 43

3.2.2 Réduction du domaine en utilisant une relaxation 46

3.3 Calcul de la borne inférieure 47

3.3.1 Amélioration de la convexification des termes bilinéaires 47

3.3.2 Génération de coupes 50

3.3.3 La linéarisation 51

3.4 *Alternate* pour la recherche de solutions réalisables 51

3.5 La stratégie de branchement 53

3.6 Deux formulations pour notre problème : *FP* et *RFP* 54

3.7 Conclusion 56

4 Implémentation et test 57

4.1 Implémentation 57

4.2 Génération d'instances à partir de données industrielles 58

4.2.1 Distribution des matières premières et des caractéristiques 59

4.2.2 Construction d'une solution de référence 59

4.2.3 Génération des contraintes 60

4.3 Résultats obtenus sur les instances industrielles 62

4.3.1 Comparaison du calcul des bornes inférieures pour *FP* et *RFP* 62

4.3.2 Calibrage d'*Alternate* 62

4.3.3 Comparaison des performances de notre approche avec *Couenne* pour les instances industrielles 64

4.4 Résultats obtenus sur des instances générales de fabrication d'aliment 65

4.5 Résultats obtenus sur les instances de pooling standard 67

4.6 Conclusion 69

5 Le problème de fabrication **71**

5.1 Définition du problème et génération d'instances 71

5.2 L'approche *Branch-and-Bound* générique 74

5.2.1 Résultats expérimentaux 75

5.2.2 La limite de la méthode 76

5.3 Une approche lagrangienne pour le calcul d'une borne inférieure 77

5.3.1 Définition de nouveaux ensembles 77

5.3.2 Une relaxation lagrangienne 79

5.3.3 Définition des générateurs 80

5.3.4 Simplification de la relaxation lagrangienne 81

5.4 Calcul de la borne lagrangienne 82

5.4.1 Les algorithmes utilisés 82

5.4.2 Initialisation de la procédure 84

5.5 Résultats expérimentaux 84

5.5.1 $\mathcal{L}(FP)$, $\mathcal{L}(RFP)$ et l'approche lagrangienne 84

5.6 Conclusion 85

Conclusion et perspectives **87**

Annexe 1 : Résultat sur les instances de fabrications d'aliments **89**

Annexe 2 : Résultat sur les instances de pooling standard **93**

Bibliographie **95**

Table des figures

1.2.1 Un exemple d'aliment 5

2.1.1 Le problème de *pooling* standard Haverly 1 16

3.3.1 Plus petit ensemble convexe contenant l'hyperboloïde 48

Liste des tableaux

1.1 Liens entre indices et contraintes 6

1.2 Lien entre indices et types de contraintes 10

2.1 Caractéristiques du problème de *pooling* Haverly 1 16

2.2 Caractéristiques des problèmes sous forme *BSBP* 23

4.1 Caractéristiques des 30 instances générées aléatoirement 61

4.2 Résultat de *Alternate* 63

4.3 Résultats moyens de *FP*, *RFP* et *Couenne* 65

4.4 Caractéristiques des instances 66

4.5 Résultats sur les instances générales 67

4.6 Caractéristiques des instances de pooling utilisées 67

4.7 Instances aléatoires 68

5.1 Caractéristiques de l'instance 75

5.2 Résultats obtenus avec les méthodes de type *Branch-and-Bound* 76

5.3 Comparaison des bornes inférieures 85

4 Résultat de RFP et Couenne 90

5 Résultat de FP et Couenne 91

6 Résultats de FP et RFP 92

7 Résolution des instances de petite taille 93

8 Résolution des instances de grandes tailles de *pooling standard* 94

Introduction

Cette thèse sous convention CIFRE est issue d'une collaboration entre la société A-Systems et le laboratoire G-SCOP.

A-Systems est une *TPE* qui s'est imposée sur le marché de la formulation d'aliments pour le bétail. Elle a été créée en 2001 dans la niche de la formulation d'aliments qui consiste à optimiser leur composition. A-Systems propose des outils d'aide à la formulation utilisés par tous les intervenants de la fabrication d'aliments pour bétail, c'est-à-dire autant les nutritionnistes élaborant les formules des aliments que les chefs d'usines qui les produisent. Les domaines d'activité de A-Systems se sont aussi diversifiés avec la commercialisation de modules d'édition d'étiquettes et de suivi de la qualité. Ils permettent un suivi accru des recettes élaborées par les outils de formulation.

Dans l'industrie de la fabrication d'aliment, le mode de production est généralement un mélange direct de matières premières et la suite de logiciels commercialisée par A-Systems répond entièrement à cette problématique. Depuis quelques années, les professionnels de la fabrication d'aliments souhaitent pouvoir composer leurs produits finis à partir de mélanges (les pré-mélanges) issus eux-mêmes d'un mélange de matières premières. Cette thèse a donc pour objectif d'explorer de nouveaux modèles de production qui intègrent la conception de pré-mélanges, la modélisation mathématique de ces nouveaux problèmes et la mise au point d'algorithmes d'optimisation efficaces pour les résoudre.

Dans le premier chapitre, nous présentons le contexte de la formulation d'aliments, décrivons le modèle classique correspondant au mélange direct des matières premières et introduisons le nouveau modèle qui sera étudié par la suite.

Dans le deuxième chapitre, nous faisons une revue de la littérature sur les problèmes de « pooling ». Le problème traité dans la thèse est effectivement un problème proche des problèmes de pooling déjà traités dans la littérature.

Nous décrivons dans le troisième chapitre l'outil que nous avons élaboré spécifiquement pour cette application. Il résout des problèmes d'optimisation bilinéaires disjoints et peut ainsi traiter les deux formulations mathématiques que nous avons proposées pour le problème de fabrication d'aliments. Dans le quatrième chapitre, nous rapportons et discutons les résultats des tests numériques effectués sur des instances générées par nos soins. Bien que générées aléatoirement, ces instances reprennent les caractéristiques des problèmes traités par les clients de A-Systems. Notre outil de résolution est comparé à *Couenne*, un outil open-source disponible dans la librairie *COIN-OR*. Le cinquième chapitre est consacré à un cas particulier du problème présenté dans le troisième chapitre. Il est intitulé « le problème de fabrication ». Pour ce dernier, nous appliquons une approche lagrangienne utilisée pour le pooling standard. Nous comparons cette approche avec celle proposée au troisième chapitre. Pour finir, des conclusions et des perspectives sont données.

Chapitre 1

La fabrication d'aliments pour bétail

Dans ce chapitre, nous commencerons par décrire le problème de la fabrication d'aliments pour bétail et les contraintes métiers. Le problème de formulation simple, quand les aliments sont fabriqués directement depuis les matières premières, sera ensuite présenté. Ce sera l'occasion d'introduire une partie des notations utilisées dans la thèse. Pour finir, nous présenterons le problème qui se pose aujourd'hui, la fabrication d'aliments à partir de pré-mélanges et de matières premières. D'autres notations seront introduites et un modèle mathématique décrivant le problème sera proposé.

1.1 Le contexte métier

L'optimisation des formulations d'aliments est un enjeu fort pour les fabricants d'aliments pour bétail. En effet, les marges sur ces produits sont faibles car le marché est très concurrentiel. De ce fait, l'optimisation du choix des matières premières à utiliser permet aux fabricants de réduire significativement les coûts d'achat de ces matières.

1.1.1 La formulation d'aliments

Tous les animaux ont des besoins nutritionnels spécifiques mais ont également des préférences quant à la présentation de la nourriture. L'odeur et l'aspect physique d'un aliment feront

qu'une oie le mangera ou ne le mangera pas dans sa totalité. La structure chimique de cet aliment favorise aussi l'absorption des nutriments et leurs effets sur l'animal. La composition de l'aliment n'est de plus pas la même aux différents stades de l'évolution de l'animal : un poulet fera d'abord des plumes avant de grossir en muscle, il faut donc favoriser dans un premier temps le développement des plumes puis le développement musculaire.

La formulation d'aliments consiste donc à définir l'ensemble des contraintes que doit respecter un produit fini pour être adapté à l'espèce et au stade de développement de l'animal.

Dans ce secteur économique, les différents intervenants industriels sont les formulateurs, les fabricants d'aliments et les éleveurs. Tout d'abord, les formulateurs élaborent les formules pour un animal donné à un stade d'évolution précis. Ces formules sont ensuite produites en usines par les fabricants. Les produits finis sont pour finir vendus aux éleveurs qui nourriront l'animal en question.

La formulation d'aliments s'étend de façon naturelle à la formulation d'autres types de mélanges. Par exemple, la conception d'une gamme de farines est élaborée par les meuniers, fabriquée, puis vendue ensuite aux boulangers. On retrouve ici des caractéristiques proches du problème de formulation d'aliments. D'autres exemples peuvent être rencontrés dans d'autres secteurs d'activité (engrais, salaison). Dans toute la suite, les termes *aliment*, *formule d'aliment*, *formule*, *produits finis* seront équivalents.

1.1.2 Les contraintes métiers

Le cahier des charges défini par les formulateurs est composé de contraintes de plusieurs types. Elles peuvent porter sur le taux d'incorporation d'un de ses composants. Par exemple, un produit doit posséder au moins 10% de blé. Un deuxième type de contraintes joue sur les caractéristiques nutritionnelles. Par exemple, les apports en glucide doivent être d'au moins 25%, mais ne doivent pas aller au delà de 40%. Ce type de contrainte peut aussi concerner des caractéristiques physiques du produit fini comme le taux d'humidité qui doit être compris entre deux bornes. Un troisième type de contrainte est le rapport de caractéristiques. Dans certains cas, le rapport entre deux nutriments (sodium et magnésium par exemple) est borné entre deux valeurs.

Au delà des contraintes sur le produit fini lui-même, d'autres contraintes apparaissent. Il s'agit de contraintes logistiques qui limitent l'utilisation d'une matière première (du fait d'une limitation des approvisionnements) ou au contraire d'un besoin d'écouler en totalité un lot de matières premières pour libérer un espace de stockage.

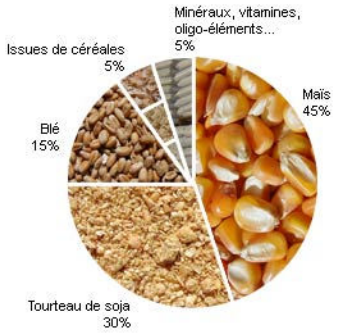
Enfin, le tonnage de chaque aliment (*i.e.* la quantité que l'on produit) est imposé par les utilisateurs. Cette quantité peut être fixée par la demande finale, ou par une taille de lot de fabrication standard.

1.2 La formulation simple

1.2.1 Le problème de fabrication directe

Actuellement, les formules sont élaborées de façon à ce que le produit fini soit directement issu du mélange de matières premières. Chaque matière première a des caractéristiques particulières, comme un certain taux de glucide, de lipide, etc. Ces caractéristiques sont transmises linéairement aux mélanges. Une recette est généralement exprimée en pourcentages qui représentent les taux d'incorporation des matières premières qui sont mélangées. Un exemple est donné dans la Figure 1.2.1.

FIGURE 1.2.1: Un exemple d'aliment



1.2.2 Les notations

Nous introduisons un ensemble de notations nécessaires à la formulation mathématique de notre problème et qui sera utilisé tout au long de la thèse. Nous commençons par les ensembles d'indices :

- M est l'ensemble des matières premières dont un élément sera noté m
- A est l'ensemble des aliments dont un élément sera noté a
- K est l'ensemble des caractéristiques dont un élément sera noté k
- $R \subset K \times K$ est l'ensemble des rapports de caractéristiques dont un élément sera noté r
- MA est l'ensemble des liens matière première - aliment
- KA est l'ensemble des couples matière première - caractéristique
- $MA_m \subset A$ est le sous ensemble des aliments liés à $m \in M$
- $MA_a \subset M$ est le sous ensemble des matières premières liées à $a \in A$
- $KA_a \subset K$ est l'ensemble des caractéristiques contraintes dans $a \in A$
- $RA_a \subset R \times A$ est l'ensemble des rapports de caractéristiques contraintes dans $a \in A$
- $\forall r \in R$, $n_r \in K$ est la caractéristique associée au numérateur de r
- $\forall r \in R$, $d_r \in K$ est la caractéristique associée au dénominateur de r

Pour chaque type de contraintes nous utilisons les lettres L et U pour représenter la borne inférieure et la borne supérieure associées à la contrainte. Par exemple L_{ka} représente la borne inférieure du taux de caractéristique k dans l'aliment a et U_{ra} représente la borne supérieure imposée au rapport de caractéristique r dans l'aliment a. C'est l'indice de la borne qui permet de déduire le type de contrainte, un résumé est disponible dans le tableau 1.1 :

Indice	Contrainte
ka	caractéristique
ra	rapport de caractéristiques
ma	taux d'incorporation
m	consommation

TABLE 1.1: Liens entre indices et contraintes

Nous explicitons maintenant les données associées à chaque instance :
- $\forall (m,k) \in M \times K$, S_{mk} est l'apport en caractéristique k de la matière première m
- $\forall (k,a) \in KA$, $[L_{ka}, U_{ka}]$ sont les bornes de la caractéristique k dans l'aliment a
- $\forall (r,a) \in RA$, $[L_{ra}, U_{ra}]$ sont les bornes du rapport r dans l'aliment a
- $\forall m \in M$, $[L_m, U_m]$ sont les bornes de la consommation totale en matière première m
- $\forall a \in A$, D_a est la demande à satisfaire en aliment a
- $\forall (m,a) \in MA$, C_{ma} est le prix unitaire d'utilisation de la matière première m dans l'aliment a

Chaque formule est produite dans une usine, les coûts d'acheminement sont à prendre en compte. C'est pourquoi le prix d'une matière première peut varier d'une formule à une autre. Enfin nous finissons par la définition des variables :

$\forall (m,a) \in MA$**,** $x_{ma} \in [L_{ma}, U_{ma}]$ est le pourcentage de la matière première m dans l'aliment a.

1.2.3 La modélisation mathématique

Le modèle de formulation simple est maintenant explicité avec les notations de la partie précédente. L'objectif est de minimiser le coût d'achat en matières premières :

$$\sum_{a \in A} D_a \left(\sum_{m \in MA_a} C_{ma} x_{ma} \right) \tag{1.2.1}$$

Les contraintes caractéristiques sur les aliments permettent de vérifier que la teneur dans une certaine caractéristique est bien comprise entre deux bornes données :

$$L_{ka} \leq \sum_{m \in MA_a} S_{mk} x_{ma} \leq U_{ka} \qquad \forall\, (k,a) \in KA \tag{1.2.2}$$

Nous supposons que les taux des caractéristiques utilisés en dénominateur de rapports sont positifs. Les contraintes rapports de caractéristiques sur les aliments se linéarisent donc en :

$$0 \leq \sum_{m \in MA_a} (S_{mn_r} - L_{ra} S_{md_r})\, x_{ma} \qquad \forall (r,a) \in RA \tag{1.2.3}$$

$$0 \geq \sum_{m \in MA_a} (S_{mn_r} - U_{ra} S_{md_r})\, x_{ma} \qquad \forall (r,a) \in RA \tag{1.2.4}$$

Les contraintes de stock sur une matière première permettent de vérifier qu'on n'utilise pas plus que le stock disponible, mais qu'on utilise au moins une quantité souhaitée :

$$L_m \leq \sum_{a \in MA_m} D_a x_{ma} \leq U_m \qquad \forall m \in M \tag{1.2.5}$$

Les contraintes imposant aux variables d'être des proportions et donc qu'au total, la somme des proportions est bien égale à 1.

$$\sum_{m \in MA_a} x_{ma} = 1 \qquad \forall a \in A \tag{1.2.6}$$

En pratique ces contraintes peuvent aussi être introduites comme des contraintes portant sur une caractéristique k_0 tel que $\forall m\ S_{mk_0} = 1$ et $\forall a\ L_{k_0 a} = U_{k_0 a} = 1$.

1.2.4 La résolution

Ce modèle est un programme linéaire qui est résolu efficacement à l'aide de l'algorithme du simplexe. Même dans le cas où le nombre de formules à élaborer est très important (plusieurs centaines) et que ces formules partagent un nombre important de matières premières (quelques milliers). Actuellement, c'est ce type d'outils qui est intégré dans la suite de logiciels de A-Systems.

1.3 La formulation avec pré-mélanges

1.3.1 Une fabrication à deux niveaux de mélange

Les problématiques évoluant, les industriels du secteur souhaitent maintenant avoir la possibilité de fabriquer leurs produits en deux étapes. Dans un premier temps, ils désirent mélanger des matières premières pour obtenir des pré-mélanges. Dans un second temps, ces pré-mélanges sont mélangés, éventuellement avec des matières premières spécifiques, pour obtenir les produits finis.

Ce mode de production peut être utilisé dans la conception de complémentaires, ce sont des produits qui apportent les vitamines et les sels minéraux qui ne sont pas suffisamment présents naturellement dans les matières premières. Les éleveurs utilisent ces compléments car une partie de leurs matières premières cultivées sur leur exploitation présentent des carences en certains nutriments. Elles doivent donc être associées à des complémentaires pour satisfaire les contraintes nutritives des aliments. Actuellement un complémentaire est élaboré pour chaque aliment, les éleveurs sont donc peut-être obligés d'acheter en petites quantités beaucoup de complémentaires (le prix d'achat est donc difficilement négociable) et ensuite les entreposer dans leur exploitation (dans des conteneurs spécifiques).

La conception à deux niveaux est aussi applicable à la fabrication de *prémix*, des mélanges de minéraux ou de vitamines dosés très finement. Les fabricants de *prémix* souhaitent élaborer un *prémix* entrant dans la composition de plusieurs aliments plutôt que des *prémix* à la carte et spécifiques à chaque aliment. Cela leur permet de réduire les coûts logistiques et marketing liés à la création des produits.

Enfin le *problème de fabrication* consiste à élaborer des pré-mélanges à partir d'un profil contenant des contraintes de fabrication (limitation en taux de graisse par exemple) et pouvant produire un ensemble d'aliments. Cela permet de réduire le nombre de composants mis en jeu lors de la fabrication des aliments, ce qui réduit le nombre de silos nécessaires dans les usines pour stocker les matières premières.

1.3.2 Les notations

Les notations sont maintenant complétées pour pouvoir modéliser ce nouveau problème. Nous commençons par les ensembles d'indices :
- P est l'ensemble des pré-mélanges dont un élément sera noté p
- MP est ensemble des liens matière première - pré-mélange
- PA est ensemble des liens pré-mélange - aliment
- KP est ensemble des caractéristiques contraintes dans des pré-mélanges
- $PA_a \subset P$ est le sous-ensemble des pré-mélanges liés à $a \in A$
- $MP_m \subset P$ est le sous-ensemble des pré-mélanges liés à $m \in M$
- $PA_p \subset A$ est le sous-ensemble des aliments liés à $p \in P$
- $MP_p \subset M$ est le sous-ensemble des matières premières liées à $p \in P$
- $KP_p \subset K$ est l'ensemble des caractéristiques contraintes dans $p \in P$
- $RP_p \subset R$ est l'ensemble des rapports de caractéristiques contraints dans $p \in P$

Comme dans la partie précédente, les bornes des contraintes sont notées de façon générique comme L et U. Le type de la contrainte, ainsi que son expression mathématique, sont encore une fois guidés par l'indice. Le tableau 1.2 résume l'ensemble des notations de notre modèle :

Indice	Contrainte
ka	caractéristique
kp	caractéristique
ra	rapport de caractéristiques
rp	rapport de caractéristiques
ma	taux d'incorporation
pa	taux d'incorporation
mp	taux d'incorporation
m	consommation

TABLE 1.2: Lien entre indices et types de contraintes

Nous continuons par les données :
- $\forall (k,p) \in KP$, $[L_{kp}, U_{kp}]$ sont les bornes de la caractéristique k dans le pré-mélange p
- $\forall (r,p) \in RP$, $[L_{rp}, U_{rp}]$ sont les bornes du rapport r dans le pré-mélange p
- $\forall\, (m,a) \in MA$, C_{ma} est le coût unitaire de l'utilisation directe de la matière première m dans l'aliment a

– $\forall(m,p) \in MP$, C_{mp} est le prix unitaire d'utilisation de la matière première m dans le pré-mélange p
– $\forall(p,a) \in PA$, C_{pa} est le surcoût unitaire d'utilisation du pré-mélange p dans l'aliment a

Enfin, nous terminons par les variables :
– $\forall(m,p) \in MP$, $y_{mp} \in [L_{mp}, U_{mp}]$ est le pourcentage de la matière première m entrant dans la composition du pré-mélange p
– $\forall(p,a) \in PA$, $z_{pa} \in [L_{pa}, U_{pa}]$ est le pourcentage du pré-mélange p entrant dans la composition de l'aliment a
– $\forall(m,a) \in MA$, $x_{ma} \in [L_{ma}, U_{ma}]$ est le pourcentage de la matière première m entrant directement dans la composition de l'aliment a, sans passer par les pré-mélanges

1.3.3 La modélisation

Le modèle de fabrication à deux étapes peut maintenant être explicité, il reprend essentiellement celui de la formulation simple. L'objectif reste de minimiser les coûts d'achat en matières premières :

$$\sum_{a \in A} D_a \left(\sum_{m \in MA_a} C_{ma} x_{ma} + \sum_{p \in PA_a} C_{pa} \sum_{m \in MP_p} C_{mp} y_{mp} z_{pa} \right) \tag{1.3.1}$$

Nous retrouvons les contraintes caractéristiques sur les aliments :

$$L_{ka} \leq \sum_{m \in MA_a} S_{mk} x_{ma} + \sum_{p \in PA_a} z_{pa} \sum_{m \in MP_p} S_{mk} y_{mp} \leq U_{ka} \qquad \forall(k,a) \in KA \tag{1.3.2}$$

Les contraintes rapports de caractéristiques sur les aliments sont :

$$\forall (r,a) \in RA$$

$$\sum_{m \in MA_a} (S_{mn_r} - L_{rp} S_{md_r})\, x_{ma} + \sum_{p \in PA_a} z_{pa} \sum_{m \in MP_p} (S_{mn_r} - L_{rp} S_{md_r})\, y_{mp} \geq 0 \tag{1.3.3}$$

$$\sum_{m \in MA_a} (S_{mn_r} - U_{rp} S_{md_r})\, x_{ma} + \sum_{p \in PA_a} z_{pa} \sum_{m \in MP_p} (S_{mn_r} - U_{rp} S_{md_r})\, y_{mp} \leq 0 \tag{1.3.4}$$

Les contraintes de stock sur une matière première :

$$L_m \leq \sum_{a \in MA_m} D_a x_{ma} + \sum_{p \in MP_m} y_{mp} \sum_{a \in PA_p} D_a z_{pa} \leq U_m \qquad \forall m \in M \tag{1.3.5}$$

Nous ajoutons les contraintes caractéristiques sur les pré-mélanges :

$$L_{kp} \leq \sum_{m \in MP_p} S_{mk} y_{mp} \leq U_{kp} \qquad \forall (k,p) \in KP \tag{1.3.6}$$

Les contraintes rapports de caractéristiques sur les pré-mélanges sont :

$$\sum_{m \in MP_p} (S_{mn_r} - L_{rp} S_{md_r})\, y_{mp} \geq 0 \qquad \forall (r,p) \in RP \tag{1.3.7}$$

$$\sum_{m \in MP_p} (S_{mn_r} - U_{rp} S_{md_r})\, y_{mp} \leq 0 \qquad \forall (r,p) \in RP \tag{1.3.8}$$

Enfin, les contraintes imposant aux variables d'être des proportions sont aussi présentes. Comme précédemment, ces contraintes peuvent aussi être introduites comme des contraintes portant sur une caractéristique k_0 tel que $\forall m\ S_{mk_0} = 1$ et $\forall a \in A\ L_{k_0 a} = U_{k_0 a} = 1$, $\forall p \in P\ L_{k_0 p} = U_{k_0 p} = 1$. En pratique, ces contraintes peuvent aussi être considérées comme des contraintes caractéristiques où chaque matière première a un apport de 1 et dont le taux est contraint à être égal à 1.

$$\sum_{m \in MA_a} x_{ma} + \sum_{p \in PA_a} z_{pa} = 1 \qquad \forall a \in A \tag{1.3.9}$$

$$\sum_{m \in MP_p} y_{mp} = 1 \qquad \forall p \in P \tag{1.3.10}$$

1.3.4 La résolution

A la différence de la formulation simple, le modèle mathématique de cette section nous amène à traiter un programme mathématique qui n'est plus linéaire. L'objectif (1.3.1) et les contraintes (1.3.2)(1.3.8)(1.3.5) contiennent en effet des termes bilinéaires de la forme $y_{mp}z_{pa}$. L'outil actuel de A-Systems ne permet plus de traiter ce problème et il devient nécessaire de trouver une ou des solutions pour résoudre cette nouvelle problématique.

1.4 Conclusion

Ce chapitre nous a permis d'expliquer le problème de formulation simple ainsi que le nouveau problème de formulation à deux niveaux, faisant intervenir les pré-mélanges. Ce nouveau problème nécessite d'introduire des contraintes bilinéaires qui rendent sa résolution plus complexe.

La plupart des outils d'optimisation du marché traitent de problèmes génériques (non linéaires et en variables mixtes). Le prix de leur licence est très élevé et il est difficile de savoir exactement quels algorithmes sont utilisés dans ces boites noires. En accord avec notre partenaire industriel, nous avons décidé de créer un outil permettant de résoudre des problèmes d'optimisation associés au modèle de formulation d'aliments avec des pré-mélanges qui soit basé sur des composants open-source ou peu onéreux.

Le problème de formulation avec pré-mélange n'est pas sans rappeler le problème du *pooling*, déjà étudié dans la littérature. Dans le chapitre suivant, un état de l'art présentera le problème du pooling ainsi que certaines de ses variantes. Nous en profiterons pour positionner notre problème par rapport à l'existant. Ce chapitre sera aussi l'occasion de présenter les outils déjà développés pour résoudre ce type de problème.

Chapitre 2

Revue de la littérature sur les problèmes de *pooling* et les méthodes de résolution

Dans ce chapitre nous introduisons les problèmes de *pooling* à partir d'un exemple de la littérature. Les problèmes de *pooling standard*, *pooling généralisé* et *pooling étendu* sont ensuite décrits. Les formulations utilisées pour le *pooling standard* utilisent des problèmes d'optimisation bilinéaires structurés par bloc dont nous expliciterons une forme standard avant de décrire les différentes méthodes utilisées pour résoudre les problèmes de *pooling standard* que nous avons trouvées dans la littérature.

2.1 Définitions et modélisation des problèmes de *pooling*

Dans cette section nous présentons les problèmes de *pooling* en commençant par décrire l'un des plus célèbres : le problème de Haverly. Nous introduisons ensuite une description formelle du problème de *pooling standard* introduite dans [11] et quelques extensions. Le problème de *pooling standard* sera ensuite comparé à notre problème de fabrication d'aliments et nous verrons que ce dernier est une variante du *pooling standard*. Deux formulations sont ensuite introduites, elles utilisent des problèmes d'optimisation non linéaires faisant intervenir des termes bilinéaires.

2.1.1 Exemple historique

Les produits bruts extraits des gisements pétroliers ont besoin d'être purifiés avant d'être mélangés entre eux. Les procédés de purification ont un coût et produisent des matières premières qui ont des caractéristiques différentes et qui sont candidates à plusieurs mélanges. Le premier problème de *pooling* étudié par Haverly [27] consiste à déterminer quelle quantité obtenir de chaque matière première purifiée et comment les combiner afin de satisfaire la demande tout en maximisant les bénéfices. Haverly [27] a introduit trois exemples célèbres dans la littérature du *pooling*. L'un d'eux est composé de trois matières premières, deux produits finis et un mélange intermédiaire. Le graphe obtenu est le suivant :

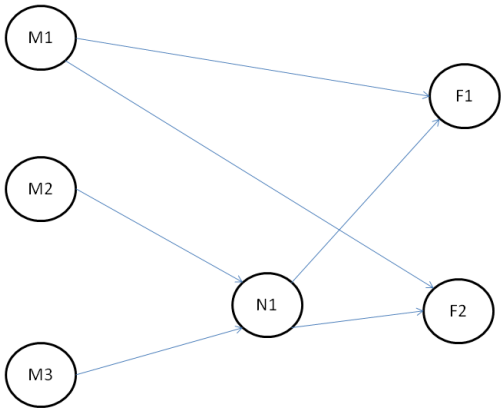


FIGURE 2.1.1: Le problème de *pooling* standard Haverly 1

TABLE 2.1: Caractéristiques du problème de *pooling* Haverly 1

	Q	Coût	Quantité
M1	2	10	∞
M2	3	6	∞
M3	1	16	∞
P1	$\leq$2.5	9	$\leq$100
P2	$\leq$1.5	15	$\leq$200

Une seule caractéristique K est contrainte, les données du problème sont indiquées dans le tableau (2.1). Le raffinement de produit pétrolier traité dans [27] a été l'occasion d'introduire le problème de *pooling standard* qui va maintenant être décrit formellement.

2.1.2 Définition formelle du *pooling standard* et quelques extensions

Comme défini dans [11], un problème de *pooling standard* est représenté sous forme d'un graphe tri-parti et orienté $G = (N, A)$. Pour chaque nœud $i \in N$, nous notons $N_i^+ = \{j \in N : (i, j) \in A\}$ et $N_i^- = \{j \in N : (j, i) \in A\}$. N est composé de *matières premières* $s \in S$ tels que $N_s^- = \emptyset$, de *produits finis* $t \in T$ tels que $N_t^+ = \emptyset$ et de nœuds *produits intermédiaires* $i \in I$. L'ensemble des arcs A est un sous-ensemble de $(S \times I) \cup (S \times T) \cup (I \times T)$, chaque arc (i, j) traduit le fait que i peut entrer dans la composition de j.

Les matières premières ont un prix d'achat unitaire et les mélanges un prix de vente unitaire. Nous noterons $C_{ij} \in \mathbb{R}$ le coût associé à l'arc $(i, j) \in A$. Si $(s, t) \in S \times T$, alors C_{st} est le coût d'achat unitaire de s moins le prix de vente unitaire de t. Si $(i, t) \in I \times T$, $C_{it} < 0$ est l'opposé du prix de vente unitaire de t. Enfin, si $(s, i) \in S \times I$ $C_{si} > 0$ est le coût d'achat unitaire de s.

Soit K un ensemble de caractéristiques, pour chaque $i \in S \cup T$, Q_i^k est le paramètre associé à la caractéristique k pour le nœud i. Si $i \in S$, Q_i^k est l'apport en caractéristique de la matière première i. Si $i \in T$, Q_i^k est la borne supérieure du *taux de la caractéristique* dans le produit fini i. Enfin, le *taux de caractéristique* d'un nœud $i \in I \times T$ est la somme des taux de caractéristiques des arcs entrants dans i divisée par la somme des flots des mêmes arcs entrants.

Le *pooling standard* consiste à déterminer un flot de G soumis à des contraintes de capacité et produisant des mélanges respectant des contraintes de caractéristiques qui maximise le profit. C'est un problème NP-difficile [11].

Dans le problème de *pooling*, les quantités de produits finis sont bornées supérieurement, c'est-à-dire que l'on produira uniquement les produits rentables. Dans les problèmes de fabrication d'aliments, des contraintes de rapport de caractéristiques ainsi que des contraintes spécifiques aux pré-mélanges sont présentes. Dans les problèmes de pooling standard, les contraintes sur les nœuds intermédiaires sont uniquement des contraintes de capacité.

Nous décrivons maintenant deux autres classes de *pooling* qui sont toutes basées sur le *pooling standard*. D'autres exemples peuvent être trouvés dans [37].

Dans le *pooling généralisé*, les connexions mettant en jeu des nœuds intermédiaires sont soumises à des variables de décision, c'est-à-dire que soit le flot ne passe pas par un arc, soit il y passe en satisfaisant les contraintes de bornes. Ce problème a été introduit par Audet et al. [15] et Meyer et al. [35], puis récemment étudié par [38].

Dans le *pooling étendu*, les contraintes sur les caractéristiques peuvent être linéaires par morceaux, rationnelles, exponentielles en fonction des caractéristiques du flot. Ce problème a été introduit par Misener et al. [36] et Furman et al. [26].

2.1.3 Les deux principales formulations du *pooling standard*

La première formulation proposée par [27] est la P-formulation. Dans cette formulation, les variables de décision sont les quantités transitant sur chaque arc ainsi que les teneurs en caractéristiques de chaque pré-mélange. Si nous notons F_{ij} la variable de décision associée au flot sur l'arc $(i,j) \in A$ et W_i^k le taux de caractéristique k du flot quittant i (si $i \in S$ $W_i^k = Q_i^k$), la *P-formulation* pour le *pooling standard* est :

$$[P] \qquad \min_{F,W} \sum_{(i,j)\in A} C_{ij}F_{ij}, \tag{2.1.1}$$

$$s.t. \qquad \sum_{j\in N_i^+} F_{ij} \leq B_i \qquad i \in N\backslash T \tag{2.1.2}$$

$$\sum_{j\in N_i^-} F_{jt} \leq B_t \qquad t \in T \tag{2.1.3}$$

$$\sum_{j\in N_i^-} F_{ij} - \sum_{j\in N_i^+} F_{ji} = 0 \qquad i \in I \tag{2.1.4}$$

$$\sum_{j\in N_i^-} W_j^k F_{ji} - W_i^k \sum_{j\in N_i^+} F_{ij} = 0 \quad i \in I, k \in K \tag{2.1.5}$$

$$\sum_{j\in N_i^-} W_j^k F_{jt} - Q_t^k \sum_{j\in N_i^-} F_{jt} \leq 0 \quad t \in T, k \in K \tag{2.1.6}$$

$$F_{ij} \geq 0 \qquad (i,j) \in A \tag{2.1.7}$$

Les contraintes (2.1.2)(2.1.3) expriment la contrainte de capacité à chaque nœud. La contrainte (2.1.4) impose la conservation du flot à chaque nœud intermédiaire. La contrainte (2.1.5) permet de définir le taux de caractéristique des nœuds intermédiaires. La contrainte (2.1.6) permet de satisfaire les contraintes imposées aux taux de caractéristiques des produits finis.

Ben-Tal et al. [20] proposent une formulation où des variables Y_i^s représentent le pourcentage du flot passant par $i \in I$ et provenant de $s \in S$. Autrement dit :

$$Y_i^s = \frac{F_{si}}{\sum_{t \in N_i^+} F_{it}}$$

La deuxième formulation classique est la *Q-formulation*. Elle s'exprime de la façon suivante :

$$[Q] \qquad \min_{F,Y} \sum_{i \in I} \sum_{s \in N_i^-} C_{si} Y_i^s \sum_{t \in N_i^+} F_{it} + \sum_{t \in T} \sum_{j \in N_t^-} C_{jt} F_{jt}, \tag{2.1.8}$$

$$s.t. \quad \sum_{i \in I \cap N_s^+} Y_s^i \sum_{t \in N_i^+} F_{it} + \sum_{t \in T \cap N_s^+} F_{st} \leq B_s \qquad s \in S \tag{2.1.9}$$

$$\sum_{t \in N_i^+} F_{it} \leq B_i \qquad i \in I \tag{2.1.10}$$

$$\sum_{j \in N_t^-} F_{jt} \leq B_t \qquad t \in T \tag{2.1.11}$$

$$\sum_{s \in S \cap N_t^-} Q_s^k F_{st} + \sum_{i \in I \cap N_t^-} F_{it} \sum_{s \in S \cap N_i^-} Q_s^k Y_i^s \leq Q_t^k \sum_{j \in N_t^-} F_{jt} \qquad t \in T, k \in K \tag{2.1.12}$$

$$\sum_{s \in N_i^-} Y_i^s = 1 \qquad i \in I \tag{2.1.13}$$

$$F_{jt} \geq 0 \qquad t \in T, j \in N_t^- \tag{2.1.14}$$

$$0 \leq Y_i^s \leq 1 \qquad i \in I, s \in N_i^- \tag{2.1.15}$$

Plusieurs variantes de ces formulations ont été proposées. Dans [44], des contraintes redondantes dans la *Q-formulation* sont introduites afin d'en renforcer les relaxations linéaires pour former la *PQ-formulation*. Dans Audet et al. [15] chaque équation de proportion de la *Q-formulation* est utilisée pour éliminer une variable et certains termes non linéaires. Dans Alfaki et al. [11]

de nouvelles variables liées à la répartition de chaque pré-mélange dans les aliments sont introduites.

2.1.4 Les applications des problèmes de *pooling*

Les problèmes de *pooling* ont initialement été introduits pour modéliser des problèmes issus de l'industrie pétrolière, nous présentons maintenant d'autres applications industrielles.

Fabrication de gaz avec contraintes environnementales

Les produits pétroliers produisent de la pollution due aux émanations de mercure, sulfure, etc. L'administration américaine impose depuis quelques années aux raffineries de fabriquer des produits dont les émanations sont limitées. Au modèle de *pooling standard* sont ajoutées des contraintes linéaires par morceaux, rationnelles ou exponentielles modélisant les émissions des particules polluantes. Ce problème est un problème de *pooling étendu* qui a été traité par Misener et al. [36].

L'acheminement d'eau dans des usines

L'eau est un composant très utilisé dans les industries de fabrication de métal, de textile, de produits pharmaceutiques. Dans ces usines, elle est utilisée sous forme liquide ou sous forme de vapeur. Chacune des utilisations nécessite une eau avec des caractéristiques physiques données qui évoluent au fur et à mesure que l'eau circule dans l'usine.

L'eau utilisée pour nettoyer les installations de raffinement de produit pétrolier en contient nécessairement des traces. Celle utilisée comme catalyseur doit avoir des caractéristiques précises pour que les réactions chimiques se passent comme prévu.

La législation imposent aux usines de traiter ces eaux usées avant de les rejeter ce qui engendre un coût directement lié à leur degré de pollution. Les problèmes d'acheminement d'eaux usées consistent à déterminer comment faire circuler l'eau dans les usines afin que les coûts de retraitement soient maitrisés. Ce problème est un problème de *pooling généralisé* étudié entre autres par Bagajewicz [17] et Jezowki [29].

Construction d'une chaine de production de gaz avec prise en compte de l'aléa

Lors de la découverte d'un gisement de gaz, il est nécessaire de construire des installations adéquates pour extraire le gaz et l'acheminer vers les usagers. Lors du transport, le gaz est soumis à des contraintes physiques qui garantissent sa stabilité et qui sont directement liées au gazoduc utilisé. Le gaz brut ne satisfait pas ces contraintes. Il est alors possible d'installer des purificateurs le long du gazoduc mais cela coûte très cher.

La solution retenue à la construction des installations est de mélanger les différents gaz avec éventuellement d'autres produits afin d'avoir un mélange stable pour le transport. C'est alors qu'un problème de *pooling standard* apparait, le lecteur pourra se référer au livre de Mokhatab et al. [39]. Afin d'être robuste, la conception de l'installation inclut la prise en compte d'un aléa portant sur la qualité et la quantité du gaz extrait. Cette version stochastique du problème de pooling standard a été étudiée par Li et al. [30] et Armagan et al. [14].

2.2 Notations utilisées pour décrire les méthodes de la littérature

Dans cette section, nous définissons formellement les problèmes traités dans cette thèse : les problèmes d'optimisation bilinéaires structurés par bloc ($BSBP$). Ces problèmes sont un cas particulier des problèmes quadratiques (QP) que nous allons maintenant introduire. Soit $\mathcal{V}$ un ensemble de n variables continues $v = (v_i)_{i=1..|\mathcal{V}|} \in \mathbb{R}^{|\mathcal{V}|}$ et bornées $\forall i = 1..|\mathcal{V}|\ l_i \leq v_i \leq u_i$. Un problème d'optimisation quadratique peut se représenter sous la forme :

$$\min \quad v_{obj} \tag{2.2.1}$$

$$Av = b \tag{2.2.2}$$

$$v_i = v_j v_k \quad \forall (i,j,k) \in \mathcal{B} \tag{2.2.3}$$

$$l_i \leq v_i \leq u_i \quad \forall i \in \langle 1, |\mathcal{V}| \rangle \tag{2.2.4}$$

où A est une matrice avec $|V|$ colonnes et $|C|$ lignes. $\mathcal{B} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{V}$ contient la définition des termes non linéaires du problème et v_{obj} est une variable représentant l'objectif à minimiser. Par exemple, le problème :

$$\begin{aligned} \min \quad & v_1^2 + v_2^2 \\ & v_1 + v_2 = 1 \\ & 0 \leq v_1, v_2 \leq 10 \end{aligned}$$

peut se représenter sous la forme :

$$\begin{aligned} \min \quad & v_3 \\ & v_1 + v_2 = 1 \\ & v_4 = v_1 \cdot v_1 \\ & v_5 = v_2 \cdot v_2 \\ & v_3 = v_4 + v_5 \\ & 0 \leq v_1, v_2 \leq 10 \\ & 0 \leq v_3, v_4, v_5 \leq 10^6 \end{aligned}$$

Les variables v_4 et v_5 sont introduites pour représenter les termes carrés v_1^2 et v_2^2, la variable v_3 représente l'objectif à minimiser $v_1^2 + v_2^2 = v_3 + v_4$. Des bornes triviales sont imposées sur v_3, v_4, v_6 car toutes les variables d'un problème (QP) sont supposées bornées.

Dans le cas des problèmes d'optimisation bilinéaires structurés par bloc, l'ensemble des variables est partitionné en trois sous-ensembles $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \mathcal{V}_3$ et l'ensemble des termes non-linéaires $\mathcal{B}$ est un sous-ensemble de $\mathcal{V}_1 \times \mathcal{V}_2 \times \mathcal{V}_3$. Si $\mathbb{P}$ est un $(BSBP)$ alors nous pouvons affirmer que
– $\mathbb{P}$ est linéaire si les variables de $\mathcal{V}_2$ sont fixées
– $\mathbb{P}$ est linéaire si les variables de $\mathcal{V}_3$ sont fixées

Tout problème de *pooling standard* est un $(BSBP)$: si la composition des pré-mélanges est fixée ou bien si leur intégration dans les produits finis est fixée alors le problème est linéaire. Pour les mêmes raisons, tout problème de formulation d'aliments décrit dans le chapitre précédent est un $(BSBP)$. Le tableau 2.2 indique les ensembles de variables ainsi que les termes bilinéaires utilisés pour modéliser le *pooling standard et* notre problème de fabrication d'aliments sous forme de $(BSBP)$.

Formulation	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{B}$
P-formulation	$\left(W_j^k\right)_{(j,k)\in(S\cup I)\times K}$	$(F_{it})_{i\in I,t\in N_i^+}$	$\left(W_i^k F_{it}\right)_{i\in I,t\in N_i^+,k\in K}$
Q-formulation	$(Y_i^s)_{i\in I,s\in N_i^-}$	$(F_{it})_{i\in I,t\in N_i^+}$	$(Y_i^s F_{it})_{i\in I,s\in N_i^-,t\in N_i^+}$
FAB	$(y_{mp})_{(m,p)\in MP}$	$(z_{pa})_{(p,a)\in PA}$	$(y_{mp}z_{pa})_{p\in P,m\in MP_p,a\in PA_a}$

TABLE 2.2: Caractéristiques des problèmes sous forme *BSBP*

Si $\mathbb{P}$ est un $(BSBP)$, pour tout $v \in V$, nous noterons :
- $v_{|1}$: le sous-ensemble de variables de v dans $\mathcal{V}_1$
- $v_{|2}$: le sous-ensemble de variables de v dans $\mathcal{V}_2$
- $v_{|3}$: le sous-ensemble de variables de v dans $\mathcal{V}_3$
- v_i ou $[v]_i$: la i-ème variable de v
- $\mathbb{P}_2\left(v_{|2}\right)$: le problème linéaire construit à partir de $\mathbb{P}$ en fixant les variables de $\mathcal{V}_2$ aux valeurs de $v_{|2}$
- $\mathbb{P}_3\left(v_{|3}\right)$: le problème linéaire construit à partir de $\mathbb{P}$ en fixant les variables de $\mathcal{V}_3$ aux valeurs de $v_{|3}$
- $lb(\mathbb{P})$ une borne inférieure de $\mathbb{P}$
- $ub(\mathbb{P})$ une borne supérieure de $\mathbb{P}$
- $opt\,(\mathbb{P})$ la valeur d'une solution optimale de $\mathbb{P}$
- $loc\,(\mathbb{P})$ la valeur d'une solution réalisable de $\mathbb{P}$ sans preuve d'optimalité
- $sol\,(\mathbb{P}) \in V$ une solution optimale de $\mathbb{P}$
- $val\,(\mathbb{P}, v)$ l'objectif de $\mathbb{P}$ évalué en $v \in \mathcal{V}$

Enfin, beaucoup de méthodes sont basées sur la dualité en programmation non linéaire ([20, 8, 12]). Elles utilisent un outil mathématique appelé lagrangien qui est défini sur un problème $\tilde{\mathbb{P}}$ équivalent à $\mathbb{P}$ mais non borné :

$$\begin{aligned} \min \quad & v_{obj} \\ & \tilde{A}v = b \\ & v_i = v_j v_k \quad \forall (i,j,k) \in \mathcal{B} \\ & v_i \in \mathbb{R} \quad \forall i \in \left\langle 1, \left|\tilde{\mathcal{V}}\right| \right\rangle \end{aligned}$$

$\tilde{A}$ contient les contraintes de bornes sur les variables et $\tilde{V} = (\mathcal{V}_1 \cup \mathcal{S}) \cup \mathcal{V}_2 \cup \mathcal{V}_3$ contient en plus les variables d'écarts s_i^+ et s_i^-. Par exemple la contrainte :

$$v_i \geq l_{v_i}$$

est reformulée en :

$$v_i - s_i^- = l_{v_i}$$

Nous associons à chaque contrainte de $\tilde{\mathbb{P}}$ une variable duale $\lambda_c \in \mathbb{R}$ et nous associons à chaque triplet de $\mathcal{B}$ une variable duale $\lambda_{ijk} \in \mathbb{R}$, le lagrangien de $\tilde{\mathbb{P}}$ est une fonction de $\tilde{\mathcal{V}} \times \mathbb{R}^{\tilde{C}} \times \mathbb{R}^{\mathcal{B}}$ défini par :

$$L(v, \lambda) = v_{obj} + \sum_{c \in \tilde{C}} \left(\tilde{A}v - b\right) \lambda_c + \sum_{(i,j,k) \in \mathcal{B}} (v_i - v_j v_k) \lambda_{ijk}$$

Dans le cas où l'on considère les composantes $v_{|1}$, $v_{|2}$ et $v_{|3}$ de v nous continuerons de noter par abus de langage ce lagrangien $L\left(v_{|1}, v_{|2}, v_{|3}, \lambda\right)$. Enfin, nous noterons $\tilde{\mathbb{P}}^{NR}$ le problème où l'on minimise les violations des contraintes :

$$\begin{aligned} \min \quad & \sum_{\left(s_i^+, s_i^-\right) \in \mathcal{S}} s_i^- + s_i^+ \\ & \tilde{A}v = b \\ & v_i = v_j v_k \qquad \forall (i,j,k) \in \mathcal{B} \\ & v_i \in \mathbb{R} \qquad \forall i \in \left\langle 1, \left|\tilde{\mathcal{V}}\right| \right\rangle \end{aligned}$$

et le lagrangien associé :

$$L^{NR}(v, \lambda) = \sum_{\left(s_i^+, s_i^-\right) \in \mathcal{S}} \left(s_i^- + s_i^+\right) + \sum_{c \in \tilde{C}} \left(\tilde{A}v - b\right) \lambda_c + \sum_{(i,j,k) \in \mathcal{B}} \left(v_i - v_j v_k\right) \lambda_{ijk}$$

Toutes ces notations nous permettent maintenant de présenter l'état de l'art sur les problèmes (QP) ou $(BSBP)$. Nous commencerons par des méthodes exactes, puis donnerons quelques heuristiques, des calculs de bornes inférieures et enfin des techniques de contraction de domaines réalisables.

2.3 Les méthodes exactes

Dans cette partie, nous présentons trois classes de méthodes exactes utilisées pour résoudre les problèmes (QP) ou $(BSBP)$.

2.3.1 Les méthodes basées sur les *Branch-and-Bound*

La plupart de ces algorithmes sont des *Spacial Branch-and-Bound* (sBB, parcours par séparation et évaluation) qui décrivent généralement un arbre binaire. A chaque itération, un sous-problème est sélectionné, le domaine de faisabilité est affiné, une borne inférieure est calculée et une borne supérieure est si possible aussi calculée. A l'issue de ce traitement, le domaine de définition peut être dans certains cas réduit : si c'est le cas, le traitement du nœud peut reprendre.

L'algorithme de principe est illustré dans l'algorithme (1). Il est très largement inspiré de celui de l'article de Belotti et al. [19]. Les algorithmes de *Branch-and-Bound* spatiaux convergent à partir du moment où la méthode de calcul de la borne inférieure est consistante avec le partitionnement des domaines de définitions des variables. La preuve est disponible dans Hoang Tuy [47] page 4. Autrement dit, il faut que le gap d'optimalité tende vers 0 si le diamètre de la partition tend lui aussi vers 0. Nous verrons dans le chapitre suivant que c'est effectivement le cas pour les bornes calculées dans notre outil ainsi que celles de [19] et [16].

Algorithme 1 : Description de l'algorithme de *Couenne*

Données :
$\mathbb{P}$: un problème d'optimisation
$\mathcal{L}$: une liste de problèmes d'optimisation
début
initialiser $\mathcal{L}$ avec le seul élément $\mathbb{P}$
$ub(\mathbb{P}) \leftarrow +\infty$
tant que $\mathcal{L} \neq \emptyset$ **faire**
choisir $\mathbb{P}_0 \in \mathcal{L}$
$\mathcal{L} \leftarrow \mathcal{L} \backslash \mathbb{P}_0$
contracter le domaine de définition de $\mathbb{P}_0$
si $\mathbb{P}_0$ *n'est pas non réalisable* **alors**
générer une relaxation linéaire $\mathcal{L}(\mathbb{P}_0)$ de $\mathcal{P}_0$
répéter
résoudre $\mathcal{L}(\mathbb{P}_0)$
affiner la linéarisation $\mathcal{L}(\mathbb{P}_0)$
jusqu'à $sol(\mathcal{L}(\mathbb{P}_0))$ *est une solution de* $\mathcal{P}_0$ *ou* $opt(\mathcal{L}(\mathbb{P}_0))$ *n'évolue plus* ;
si $sol(\mathcal{L}(\mathbb{P}_0))$ *est une solution de* $\mathcal{P}_0$ **alors** $ub(\mathbb{P}) \leftarrow \min\{ub(\mathbb{P}), opt(\mathcal{L}(\mathbb{P}_0))\}$
(optionnel) Chercher une solution de $\mathcal{P}_0$, $ub(\mathbb{P}) \leftarrow \min\{ub(\mathbb{P}), loc(\mathbb{P}_0)\}$
si $sol(\mathcal{L}(\mathbb{P}_0)) \leq ub(\mathbb{P}) - \epsilon$ **alors**
choisir une variable v_i
choisir une valeur de branchement v_i^b
créer les sous-problèmes
$\mathbb{P}_{0-}$ (associé à $v_i \leq v_i^b$)
$\mathbb{P}_{0+}$ (associé à $v_i \geq v_i^b$)
$\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathbb{P}_{0-}, \mathbb{P}_{0+}\}$
fin
fin
fin
fin

2.3.2 Les méthodes de type décomposition de Benders

Les algorithmes de décomposition de Benders sont des méthodes itératives alternant entre la résolution d'un sous-problème primal et d'un problème maître. Le problème maître est en général plus simple à résoudre que le problème initial mais n'en fournit qu'une relaxation. Le sous-problème a donc pour rôle de générer des coupes qui renforcent le problème maître jusqu'à obtenir un problème équivalent au problème de départ.

Sous certaines conditions indiquées dans Hooker et al .[28], cette méthode converge vers l'optimum global (nous expliciterons ces conditions pour notre problème dans la suite). Le problème maître est cependant généralement résolu avec des méthodes de type *Branch-and-Bound* mais de façon plus efficace que le problème de départ.

Dans notre cas, le problème est plus simple si on fixe la valeur des variables de $\mathcal{V}_2$ ou de $\mathcal{V}_3$: il devient linéaire. Nous choisissons arbitrairement de considérer des problèmes où les variables de $\mathcal{V}_2$ sont fixées. Pour décrire le déroulement de cette méthode itérative, nous noterons $u_2^h \in \mathcal{V}_2$ les valeurs utilisées à l'itération h pour construire le problème linéaire $\mathbb{P}_2\left(u_2^h\right)$, $\mathcal{H}^R$ l'ensemble des itérations où $\mathbb{P}_2\left(u_2^h\right)$ était réalisable et $\mathcal{H}^{NR}$ les itérations où il ne l'était pas.

La décomposition de Benders nécessite d'être capable de générer pour chaque valeur de u_2^h une fonction $\beta_{u_2^h} : u_2 \in V_2 \longmapsto \beta_{u_2^h}(u_2)$ telle que
- si $h \in \mathcal{H}^{NR}$, la coupe $0 \geq \beta_{u_2^h}(u_2)$ exclut u_2^h mais aucune solution réalisable de notre problème
- si $h \in \mathcal{H}^R$, pour toute solution réalisable v^* de $\mathbb{P}$, $\beta_{u^h}\left(v^*_{|2}\right) \leq val\left(\mathbb{P}, v^*\right)$
- si $h \in \mathcal{H}^R$, $\beta_{u_2^h}\left(u_2^h\right) = opt\left(\mathbb{P}_2\left(u_2^h\right)\right)$

A chaque itération nous pouvons construire le problème $MASTER\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$ qui fournit une borne inférieure de notre problème comme indiqué dans [28].

$$\begin{aligned} \min_{(\alpha,u_2)} \quad & \alpha \\ & \alpha \geq \beta_{u_2^h}(u_2) \quad \forall h \in \mathcal{H}^R \\ & 0 \geq \beta_{u_2^h}(u_2) \quad \forall h \in \mathcal{H}^{NR} \\ & u_2 \in V_2, \alpha \in \mathbb{R} \end{aligned}$$

Algorithme 2 : Description de la décomposition de Benders

Données :
$h = 0$
$\alpha^h \leftarrow -\infty$
$\mathcal{H}^R = \emptyset$
$\mathcal{H}^{NR} = \emptyset$
Choisir une valeur initiale u_2^0 pour les variables
répéter
  Générer la fonction $\beta_{u_2^h}(u_2)$ qui nous permettra d'ajouter les coupes valides dans $MASTER\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$
  si $\mathbb{P}_2\left(u_2^h\right)$ *est réalisable* **alors**
    ajouter h à $\mathcal{H}^R$
  sinon
    ajouter h à $\mathcal{H}^{NR}$
  fin
  $h \leftarrow h + 1$
  Résoudre $MASTER\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$
  Stocker les nouvelles valeurs pour $\left(\alpha^h, u_2^h\right)$
jusqu'à $opt\left(\mathbb{P}_2\left(u^h\right)\right) - \alpha^h \leq \epsilon$;

L'algorithme général de la méthode de Benders est donné dans l'algorithme (2). Hooker a démontré que si l'algorithme se termine en un nombre fini d'itérations à la solution u^* alors
– si $MASTER\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$ est non réalisable, $\mathbb{P}$ est non réalisable
– si $\mathbb{P}_2(u^*)$ est non bornée, $\mathbb{P}$ est non bornée
– si $\mathbb{P}_2(u^*)$ est réalisable, $opt(\mathbb{P}_2(u^*)) = opt(\mathbb{P})$

Il reste à démontrer que l'algorithme converge en un nombre fini d'itérations dans notre cas. Supposons avoir résolu $MASTER\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$ pour obtenir u_2^* et $\mathbb{P}_2(u_2^*)$. Si $\tilde{\mathbb{P}}_2(u_2^*)$ est réalisable, nous noterons $\lambda(u_2^*)$ une de ses solutions duales. La fonction $\beta_{u_2^*}$ est alors définie à l'aide d'un lagrangien L de $\mathbb{P}_2$:

$$\beta_{u_2^*}(u_2) = \min_{(u_1,u_3)\in\mathcal{V}_1\times\mathcal{V}_3} L(u_1, u_2, u_3, \lambda(u_2^*))$$

Si $\tilde{\mathbb{P}}_2(u_2^*)$ n'est pas réalisable, nous noterons $\lambda^{NR}(u_2^*)$ une des solutions duales de $\tilde{\mathbb{P}}_2^{NR}(u_2^*)$ (qui lui l'est toujours), et nous utilisons L^{NR} à la place de L.

$MASTER\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$ est donc :

$$\begin{aligned} \min_{(\alpha,u_2)} \quad & \alpha \\ \alpha &\geq \min_{(u_1,u_3)\in\mathcal{V}_1\times\mathcal{V}_3} L\left(u_1,u_2,u_3,\lambda\left(u_2^h\right)\right) \qquad \forall h\in\mathcal{H}^R \\ 0 &\geq \min_{(u_1,u_3)\in\mathcal{V}_1\times\mathcal{V}_3} L^{NR}\left(u_1,u_2,u_3,\lambda^{NR}\left(u_2^h\right)\right) \quad \forall h\in\mathcal{H}^{NR} \\ & u_2\in\mathcal{V}_2, \alpha\in\mathbb{R} \end{aligned}$$

Même si la méthode converge théoriquement vers l'optimum global de notre problème [14], il est difficile de la mettre en œuvre sous cette forme.

$MASTER\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$ est toujours un problème d'optimisation non linéaire faisant intervenir des contraintes non linéaires : les coupes formées à l'aide des fonctions non linéaires β_{u^H} à chaque itération H.

Dans les travaux de Armagan [14], $MASTER\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$ est résolu à l'aide de Baron [43], un outil d'optimisation générique commercial. Les résultats obtenus suggèrent que la décomposition de Benders sous cette forme n'est pas adaptée pour le *pooling standard* car Baron n'arrive pas à résoudre $MASTER\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$ efficacement. La section suivante décrit ce qui peut être vue comme une variante de cette décomposition : l'algorithme GOP, proposé par Floudas et al.[25].

2.3.3 L'algorithme *Global OPtimization* (*GOP*)

Floudas et al. [24, 25] ont démontré que l'expression du lagrangien

$$L\left(u_1,u_2,u_3,\lambda\left(u_2^h\right)\right)$$

ne comprend que des termes en variables u_2 ou u_3. Nous le noterons alors $L_h\left(u_2,u_3\right)$. Ces termes sont ceux présents dans les contraintes non linéaires $v_i = v_j v_k$.

De plus, ils ont démontré que le gradient de $L_h\left(u_2,u_3\right)$ par rapport à u_3 est une fonction linéaire en u_2. Nous noterons $g_i^h\left(u_2\right)$ la composante du gradient par rapport à la variable $[u_3]_i$.

La base de GOP est que l'on peut substituer la résolution de $MASTER\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$ durant la décomposition de Benders à la résolution d'une série de problèmes linéaires. Nous notons V_3^B l'ensemble des points extrêmes de $\mathcal{V}_3$, c'est-à-dire l'ensemble des vecteurs u_3 ayant chaque variable $[u_3]_i$ à une borne.

A l'itération H, c'est-à-dire que l'on vient de traiter u_2^H et que $H \notin \mathcal{H}^R \cup \mathcal{H}^{NR}$, on définit $u_3^{B_h^H}$ pour tout $h \in \mathcal{H}^R, \mathcal{H}^{NR}$ comme suit :

$$g_i^h\left(u_2^H\right) \geq 0 \Rightarrow \left[u_3^{B_h^H}\right]_i = l_{[u_3]_i}$$
$$g_i^h\left(u_2^H\right) \leq 0 \Rightarrow \left[u_3^{B_h^H}\right]_i = u_{[u_3]_i}$$

Puis $\forall u_3^B \in \mathcal{V}_3^B$, nous notons $MASTER_{u_3^B}\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$:

$$
\begin{array}{ll}
\min & \alpha \\
s.t. & \\
& \left\{ \begin{array}{lll} \alpha \geq & L_h\left(u_2, u_3^{B_h^H}\right) & \\ & g_i^h\left(u_2\right) \geq 0 & si\, u_3^{B_h^H} = l_{[u_3]_i} \\ & g_i^h\left(u_2\right) \leq 0 & si\, u_3^{B_h^H} = u_{[u_3]_i} \end{array} \right. \qquad \forall h \in \mathcal{H}^R \\
& \left\{ \begin{array}{lll} 0 \geq & L_h\left(u_2, u_3^{B_h^H}\right) & \\ & g_i^h\left(u_2\right) \geq 0 & si\, \left[u_3^{B_h^H}\right]_i = l_{[u_3]_i} \\ & g_i^h\left(u_2\right) \leq 0 & si\, \left[u_3^{B_h^H}\right]_i = u_{[u_3]_i} \end{array} \right. \qquad \forall h \in \mathcal{H}^{NR} \\
& \left\{ \begin{array}{lll} \alpha \geq & L_h\left(u_2, u_3^{B}\right) & \\ & g_i^h\left(u_2\right) \geq 0 & si\, \left[u_3^{B}\right]_i = l_{[u_3]_i} \\ & g_i^h\left(u_2\right) \leq 0 & si\, \left[u_3^{B}\right]_i = u_{[u_3]_i} \end{array} \right. \\
& u_2 \in \mathcal{V}_2, \alpha \in \mathbb{R}
\end{array}
$$

Comme u_3 est fixé à u_3^B, $MASTER_{u_3^B}\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$ est un programme linéaire en α, u_2. Si on note $\alpha^H\left(u_3^B\right)$ la valeur optimale de $MASTER_{u_3^B}^H$, Floudas et al. [24, 25] ont démontré par récurrence que :

$$
opt\left(MASTER\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)\right) = \min_{u_3^B \in Z^B} \; opt\left(MASTER_{u_3^B}\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)\right)
$$

L'algorithme *GOP* est une décomposition de Benders où à chaque itération H tous les problèmes $MASTER_{u_3^B}\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$ sont énumérés au lieu de résoudre directement $MASTER\left(\mathcal{H}^R, \mathcal{H}^{NR}\right)$. La convergence de l'algorithme *GOP* a été démontrée pour le *pooling standard* mais aussi pour des problèmes d'optimisation non convexes plus généraux par Liu et al. [32, 33] qui deviennent convexes si un sous-ensemble de variables est fixé.

2.4 Les méthodes de calcul de solutions réalisables

Dans cette section, nous présentons quelques méthodes permettant de déterminer une solution réalisable de notre problème mais sans garantie que ce soit la solution optimale. La plupart de ces méthodes exploitent le fait que notre problème devient linéaire si les variables de $\mathcal{V}_2$ ou de $\mathcal{V}_3$ sont fixées.

La méthode *Alternate* est un processus itératif qui, à partir d'une solution initiale $v \in \mathcal{V}$, choisit arbitrairement de fixer les variables de $\mathcal{V}_2$ ou de $\mathcal{V}_3$ aux valeurs $v_{|2}$ ou $v_{|3}$ et résout le problème linéaire obtenu. Si le programme linéaire est réalisable et que la solution optimale améliore la solution de référence, on réitère le processus en fixant les variables libres la fois précédente. Cette méthode a été introduite par Haverly et al. [27] et utilisée par Audet et al. [15] comme élément d'une méta-heuristique de type recherche à voisinages variables.

La méthode de discrétisation consiste à discrétiser l'ensemble de définition des variables de $\mathcal{V}_2$ ou de $\mathcal{V}_3$. Le problème revient donc à choisir quelle valeur on affecte à chacune de ces variables en introduisant des variables booléennes. On passe donc d'un problème bilinéaire en variables continues à un problème bilinéaire où chacun des produits contient une variable continue et une variable entière. Ce type de problème est linéarisable en un problème linéaire en variable 0-1 et peut être résolu avec un outil de programmation linéaire en nombres entiers. Cette méthode a été utilisée pour le *pooling standard* par Pham et al. [40].

La méthode *Global Optimum Search* est basée sur la dualité en programmation non linéaire. C'est une méthode itérative. A l'itération h, on a une valeur u_2^h pour les variables u_2 et on résout le problème linéaire associé. On obtient la solution (u_1^h, u_3^h) et les variables duales $\left(\lambda^h\right)$ et une expression du lagrangien : $(u_1, u_2, u_3) \longmapsto L(u_1, u_2, u_3, \lambda^h)$. Ensuite, on résout un problème $MASTER_h$ formulé avec ce lagrangien et ceux des itérations précédentes :

$$
\begin{aligned}
\min \quad & \alpha \\
& \alpha \geq L\left(u_1^h, u_2, u_3^h, \lambda^h\right) \quad h \in H^R \\
& 0 \geq L\left(u_1^h, u_2, u_3^h, \lambda^h\right) \quad h \in H^{NR} \\
& u_2 \in \mathcal{V}_2 \\
& \alpha \in \mathbb{R}
\end{aligned}
$$

où H^R est l'ensemble des itérations h où $\mathbb{P}_2\left(u_2^h\right)$ était réalisable et H^{NR} celles où il ne l'était pas. $MASTER_h$ est linéaire et sa résolution nous fournit les prochaines valeurs auxquelles seront fixées les variables y. La procédure cesse lorsque la valeur α obtenue lors du dernier $MASTER_h$ est égale à la meilleure solution réalisable.

Enfin, les méthodes barrières consistent à convexifier le problème de départ en relaxant les contraintes non convexes et en introduisant un terme convexe dans l'objectif qui est strictement positif si ces contraintes ne sont pas satisfaites. Un problème d'optimisation convexe est alors à résoudre. Les algorithmes mis en place sont des méthodes de points intérieurs [48] et deux implémentations utilisées dans des contextes industriels sont disponibles : IpOpt [49] et Knitro [1].

2.5 Les méthodes de calcul de bornes inférieures

Les bornes inférieures servent dans les méthodes d'optimisation globale à estimer la valeur d'une solution optimale d'un problème. Elles sont généralement obtenues en relaxant des contraintes du problème initial afin d'avoir un problème plus simple à résoudre. Qualizza et al. [41] proposent une comparaison des différentes approches pour les problèmes (QP).

La relaxation de Mc Cormick

Dans la relaxation de *McCormick* [34], les contraintes de définition des termes bilinéaires sont relaxées. Dans ce cas particulier, c'est même une linéarisation car une fois les contraintes non linéaires retirées, le problème à résoudre est un problème linéaire. Cette relaxation est généralement faible car il n'y a plus de lien entre les variables des triplets de $\mathcal{B}$. Plusieurs variantes sont alors utilisées pour améliorer la borne obtenue, les plus courantes sont les relaxations *RLT* et *SDP* qui sont décrites dans les parties suivantes.

La relaxation RLT

RLT signifie *Reformulation Linearization Technique*, cette relaxation a été introduite par Sherali [46]. Cette technique est développée pour l'optimisation polynomiale générale. La relaxation

de *McCormick* enlève les contraintes non linéaires alors que les relaxations *RLT* les convexifient. C'est-à-dire que la contrainte (2.2.3) :

$$v_i = v_j v_k, \forall\, (i,j,k) \in \mathcal{B}$$

est convexifiée en imposant que (v_i, v_j, v_k) soit dans le plus petit ensemble convexe contenant l'hyperboloïde. Cette étape passe nécessairement par l'ajout de variables et contraintes artificielles mais la borne inférieure résultant est de meilleure qualité. Nous verrons dans le chapitre suivant quelles modifications apporter pour construire cette relaxation.

De plus, durant une phase de reformulation chaque contrainte est multipliée par chaque variable. Nous créons ainsi une contrainte redondante dans le problème initial mais qui renforce potentiellement la relaxation. Par contre, nous créons aussi de nouveaux termes non linéaires. Considérons le problème suivant :

$$\begin{aligned} \min \quad & v_3 \\ & v_1 + v_2 = 1 \\ & v_3 = v_4 + v_5 \\ & v_4 = v_1 \cdot v_1 \\ & v_5 = v_2 \cdot v_2 \\ & 0 \leq v_1, v_2 \leq 10 \\ & 0 \leq v_3, v_4, v_5 \leq 10^6 \end{aligned}$$

Nous pouvons alors créer la contrainte résultant de la multiplication de $v_1 + v_2 = 1$ par v_1 :

$$v_1 v_1 + v_1 v_2 = v_1$$

Cette multiplication nous définit un nouveau terme non-linéaire $v_6 = v_1 v_2$ et la contrainte à ajouter est alors

$$v_4 + v_6 = v_1$$

Nous pouvons faire de même en multipliant les contraintes de bornes sur les variables, par exemple en multipliant $0 \leq 10 - v_1$ et $v_3 \geq 0$ nous obtenons $0 \leq 10v_3 - v_1v_3$ qui elle aussi crée un nouveau terme non linéaire.

Parmi toutes les opérations qu'il est possible de faire, certaines peuvent créer des contraintes qui nous permettront d'éliminer des termes non linéaires. En multipliant $v_1 + v_2 = 1$ par v_2, on obtient :

$$v_5 + v_6 = v_2$$

Or, nous avions déjà l'équation suivante :

$$v_4 + v_6 = v_1$$

Autrement dit, on vient d'exprimer les termes non linéaire v_4 et v_5, en fonction des autres variables du problème. Sherali et al. [45] ont démontré que si toutes les contraintes $v_i = v_j \cdot v_k$, hormis celles définissant un terme qui a été éliminé, sont satisfaites, alors la contrainte associée au terme éliminé est aussi satisfaite. Dans notre exemple, cela revient à dire que le problème est équivalent à

$$
\begin{aligned}
\min \quad & v_3 \\
& v_1 + v_2 = 1 \\
& v_3 = v_4 + v_5 \\
& v_5 + v_6 = v_2 \\
& v_4 + v_6 = v_1 \\
& v_6 = v_1 \cdot v_2 \\
& 0 \leq v_1, v_2 \leq 10 \\
& 0 \leq v_3, v_4, v_5 \leq 10^6
\end{aligned}
$$

En effet, si $v_6 = v_1 \cdot v_2$ alors

$$
\begin{aligned}
v_4 = v_1 - v_6 = v_1 (1 - v_2) = v_1^2 \\
v_5 = v_2 - v_6 = v_2 (1 - v_1) = v_2^2
\end{aligned}
$$

La relaxation SDP

Un problème SDP est un problème d'optimisation où l'objectif est une forme linéaire et les contraintes imposent à des formes quadratiques d'être semie-définie positive (SDP). Soit $c \in \mathbb{R}^{|V|}$ un vecteur de coût et $(A_i)_{i=0..|V|}$ des matrices réelles carrées de tailles $|\mathcal{V}|$. Nous pouvons définir le programme SDP suivant :

$$
\begin{aligned}
\max \quad & \sum_i c_i v_i && (2.5.1) \\
& A_0 - \sum_i A_i v_i \succeq 0 && (2.5.2)
\end{aligned}
$$

La contrainte (2.5.2) impose que la matrice $A_0 - \sum_i A_i v_i$ soit définie positive, c'est-à-dire :

$\forall u \in \mathbb{R}^{|V|}$, $u^T \left(-A_0 + \sum_i A_i v_i \right) u \geq 0$. Les problèmes SDP sont des problèmes d'optimisation convexes qui peuvent être vus comme une généralisation de la programmation linéaire. La plupart des algorithmes mis en œuvre pour résoudre ce type de problèmes sont basés sur des méthodes de points intérieurs ou des algorithmes de coupes.

Dans notre cas, pour tout $(i, j, k) \in \mathcal{B}$, on considère la matrice B définie par le terme générique $B_{jk} = x_i - x_j x_k$. Les contraintes (2.2.3) sont équivalentes à $B = 0$. Dans les relaxations (SDP) de (QP), la contrainte $B = 0$ est relaxée en B est semi-définie positive ($B \succcurlyeq 0$).

Cette relaxation nous amène à résoudre un problème SDP. Généralement, on commence par résoudre la relaxation de Mc Cormick et on cherche si la matrice $\bar{B}$ construite avec la solution de la relaxation est semi-définie positive. Si ce n'est pas le cas, c'est que l'on a trouvé $\tilde{u}$ tel que $\tilde{u}^T \bar{B} \tilde{u} < 0$. On ajoute alors la contrainte $\tilde{u}^T \bar{B} \tilde{u} \geq 0$ au problème relaxé, ce qui coupe la solution actuelle. Le processus s'arrête si la matrice construite à l'aide de la relaxation de Mc Cormick est semi-définie positive. Comme indiqué dans [45], les relaxations SDP et RLT sont complémentaires.

Les relaxations lagrangiennes

Les bornes lagrangiennes ont été utilisées pour le *pooling* standard dans [8, 12]. Elles sont calculées par un procéde itératif grâce au lagrangien défini dans la section 2.2. A l'itération H, $\forall h < H$, nous notons λ^h un ensemble de valeurs pour λ, $\theta^h = \min_v L(v, \lambda^h)$ et (v^h) tel que $\theta^h = L(v^h, \lambda^h)$. Nous pouvons alors définir $MASTER_H$ le problème maître de l'itération H :

$$\begin{aligned} &\max_{(\theta,\lambda)} \theta \\ &\theta \leq L(v^h, \lambda) \quad \forall h \leq H-1 \\ &\theta \in \mathbb{R}, \lambda \in \Lambda \end{aligned}$$

L'algorithme de la relaxation lagrangienne est indiqué dans l'algorithme 3. Dans les deux approches [8, 12], le calcul de $\theta(\lambda) = \min_v L(v, \lambda)$ est un problème linéaire en nombres entiers.

Algorithme 3 : Description d'une relaxation lagrangienne

Données :
$H = 0$
λ^0 initialiser arbitrairement
Choisir une valeur initiale pour les variables y^0
ϵ la précision
répéter
  $H \leftarrow H + 1$
  Résoudre $MASTER_H$, stocker θ^H et (x^H, y^H, z^H)
jusqu'à $\theta^{H+1} - \theta^H \leq \epsilon$;

2.6 Conclusion

Dans cet état de l'art, nous avons positionné notre problème de formulation avec pré-mélanges parmi les problèmes de *pooling*. Ceci nous a conduits à étudier les différentes techniques utilisées classiquement pour résoudre ces types de problème.

Les marges dans l'industrie de fabrication d'aliment étant trés faible, l'objectif de A-Systems est de proposer à ses clients un outil performant permettant de leur fournir les meilleures solutions. Nous avons opté pour le développement d'une méthode exacte. Le choix s'est porté sur une méthode de type *Branch-and-Bound*. La méthode proposée sera décrite dans le chapitre suivant.

Chapitre 3

Un approche de résolution par *Branch-and-Bound*

La méthode que nous avons développée [42] s'inspire des techniques utilisées dans l'outil *Couenne* [19] et de celles présentées dans l'article de Audet et al. [16]. Comme dans ces deux méthodes, notre problème est convexifié puis une borne inférieure est calculée en résolvant un problème linéaire. Notre outil est destiné à résoudre des problèmes ($BSBP$). Nous avons fait évoluer les notations par rapport à celles présentes dans l'article publié sur le sujet [42] et nous avons opté pour un formalisme plus général dans cette thèse.

Dans ce chapitre, nous décrivons le schéma général de notre algorithme et tous les blocs qui le composent. Nous terminons par la description de deux formulations utilisées pour modéliser notre problème sous la forme d'un ($BSBP$) et qui modifient le comportement de notre algorithme.

3.1 Description générale de l'algorithme

Nous présentons une forme standard représentant des problèmes d'optimisation bilinéaires avec contraintes bilinéaires. L'algorithme permettant de résoudre de tels problèmes est alors intro-

duit. Les sections suivantes expliciteront chacun des blocs de notre méthode et le chapitre se terminera sur la présentation de deux formulations de notre problème industriel.

3.1.1 Notations allégées

Dans cette partie, nous noterons $l_i \leq v_i \leq u_i$ les contraintes de bornes sur la variable v_i au lieu de $l_{v_i} \leq v_i \leq u_{v_i}$. De plus, $\forall\,(i,j,k) \in \mathcal{B} \subseteq \mathcal{V}_1 \times \mathcal{V}_2 \times \mathcal{V}_3$ nous supposons que $l_i = \min\{l_j l_k, u_j u_k, u_j l_k, l_j u_k\}$ et $u_i = \max\{l_j l_k, u_j u_k, u_j l_k, l_j u_k\}$. Si elles existent, les contraintes de bornes sur des termes bilinéaires doivent être introduites à l'aide de variables d'écart et de nouvelles contraintes. Par exemple, si $v_{i_0} = v_{j_0} v_{k_0}$ et que l'on veut imposer $v_{i_0} \geq \bar{l}_{i_0} > \min\{l_{j_0} l_{k_0}, u_{j_0} u_{k_0}, u_{j_0} l_{k_0}, l_{j_0} u_{k_0}\}$ nous devons ajouter la contrainte définie à l'aide de la variable d'écart $s \geq 0$:

$$v_{i_o} - s = l_{i_0}$$

Les problèmes $(BSBP)$ définis dans le chapitre précédent sont des problèmes d'optimisation de la forme :

$$\begin{aligned}
\min \quad & v_{obj} && && (3.1.1)\\
& Av = b && && (3.1.2)\\
& v_i = v_j v_k && \forall\,(i,j,k) \in \mathcal{B} && (3.1.3)\\
& l_i \leq v_i \leq u_i && \forall i \in \langle 1, |\mathcal{V}| \rangle && (3.1.4)
\end{aligned}$$

$\forall i \in \langle 1, |\mathcal{V}| \rangle$, $I_i = [l_i; u_i]$ sont les bornes inférieures et supérieures imposées à la variable v_i. La variable $v_{obj} \in \mathcal{V}$ représente l'objectif à minimiser. L'ensemble des contraintes est noté $\mathcal{C}$. $\forall c \in \mathcal{C}$, b_c est le second membre de la contrainte c et, $\forall i \in \langle 1, |\mathcal{V}| \rangle$ $A_{ci} \in \mathbb{R}$ est le coefficient de v_i dans la contrainte c.

3.1.2 Schéma de principe pour l'algorithme

La méthode globale que nous avons implémentée est un parcours de recherche par séparation et évaluation (*Branch-and-Bound*) qui construit une suite d'estimations de la solution optimale de notre problème. Cette suite d'estimations est aussi une suite de bornes inférieures qui sont obtenues en convexifiant les termes non linéaires et affinées par la séparation sur le domaine d'une variable du problème non linéaire. Les estimations sont associées aux nœuds de l'arbre contenant l'ensemble des décisions prises aux nœuds parents et celles prises lors du branchement. A partir de chaque estimation, nous utilisons une heuristique simple consistant à fixer alternativement les variables de $\mathcal{V}_2$ ou $\mathcal{V}_3$. Enfin, le domaine de définition du problème non linéaire est réduit en utilisant les opérateurs de l'arithmétique d'intervalles. Comme nous le verrons par la suite, cela permet d'améliorer la qualité des estimations et l'efficacité de l'heuristique permettant de chercher une solution réalisable.

Une estimation sera $\varepsilon - \textit{réalisable}$ si l'erreur maximale sur tous les termes non linéaires est inférieure ou égale à $\varepsilon > 0$. Une solution du problème non linéaire ayant une valeur U est $\varepsilon - \textit{optimale}$ s'il existe une borne inférieure L valide telle que le gap d'optimalité $\frac{U-L}{U}$ est inférieur à $\varepsilon > 0$. Nous utilisons une stratégie de meilleur d'abord pour explorer l'arbre de *Branch-and-Bound* : le nœud sélectionné à chaque itération est celui avec la plus petite borne inférieure.

Le principe général du *Branch-and-Bound* est donné dans l'algorithme 4

Algorithme 4 : Description du $Branch - and - Bound$

Données :

$noeud, noeud^+, noeud^-$: notations pour des nœuds
$\mathcal{N}$: conteneur des nœuds triés par valeur de borne inférieure
gap($\mathcal{N}$) : retourne le gap relatif d'optimalité sur l'ensemble des nœuds $\mathcal{N}$
sélection($\mathcal{N}$) : une procédure qui sélectionne et retire un nœud de $\mathcal{N}$
séparation(n) : une procédure de séparation binaire
estimation(n) : une procédure qui calcule une estimation à partir des décisions prises à un nœud
heuristique(n) : notre procédure qui essaie de calculer une solution réalisable en fixant alternativement les variables
ajout(n,$\mathcal{N}$) : une procédure qui ajoute le nœud n à $\mathcal{N}$
miseAjour : une procédure qui met à jour la meilleure solution

début
  $\mathcal{N} \leftarrow$ problème initial
  tant que $gap(\mathcal{N}) \geq gap\_max$ **faire**
    $noeud \leftarrow$sélection($\mathcal{N}$)
    $noeud^+, noeud^- \leftarrow$ séparation(n)
    pour chaque $noeud \in \{noeud^+; noeud^-\}$ **faire**
      réduction(noeud)
      estimation(noeud)
      heuristique(noeud)
      ajout(noeud,$\mathcal{N}$)
      miseAjour()
    fin
  fin
fin

3.2 Les techniques de contraction de bornes utilisées

3.2.1 Réduction du domaine réalisable

Les méthodes de contractions de bornes utilisant l'arithmétique d'intervalles sont utilisées pour réduire le domaine de définition d'un problème non linéaire afin de détecter qu'il est non réalisable, ou pour renforcer la borne obtenue par une relaxation. Le lecteur trouvera un état de l'art et une description théorique de la méthode utilisée dans cette partie dans Belotti et al. [18]. Nous ne détaillons pas ces techniques pour ne pas alourdir inutilement la lecture.

Le domaine de définition des variables de notre problème est réduit en considérant les opérateurs de l'arithmétique d'intervalles. Soit $I_1 = [l_1; u_1] \subset \mathbb{R}$ et $I_2 = [l_2; u_2] \subset \mathbb{R}$ deux intervalles ; la somme, la différence, le produit et la division sont définis en étendant simplement les définitions de ces opérateurs pour les nombres réels comme indiqué dans 3.2.1.

$$
\begin{aligned}
I_1 + I_2 &= [l_1 + l_2; u_1 + u_2] \\
I_1 - I_2 &= [l_1 - u_2; u_1 - l_2] \\
I \times J &= [\min\{l_1 l_2, u_1 u_2, l_1 u_2, l_2 u_1\}; \max\{l_1 l_2, u_1 u_2, l_1 u_2, l_2 u_1\}] \\
[l_i; u_i]^{-1} &= \begin{cases} \mathbb{R} & si\ 0 \in]l_i, u_i[\\ \left[\min\left\{\frac{1}{u_i}, \frac{1}{l_i}\right\}; \max\left\{\frac{1}{u_i}, \frac{1}{l_i}\right\}\right] & si\ 0 \notin [l_i, u_i] \\ \mathbb{R}^+ & si\ 0 = l_i \\ \mathbb{R}^- & si\ 0 = u_i \end{cases} \\
I \div J &= I \times J^{-1}
\end{aligned}
\tag{3.2.1}
$$

Nous considèrerons le système de contraintes suivant pour illustrer notre méthode :

$$v_1 = v_2 v_3 \tag{3.2.2}$$

$$v_1 + v_4 = 1 \tag{3.2.3}$$

$$0 \leq v_1, v_2, v_3 \leq 1 \tag{3.2.4}$$

$$0.2 \leq v_4 \leq 0.5 \tag{3.2.5}$$

L'algorithme *contraction(I,J)* (5) permet de mettre à jour un intervalle I à partir d'un intervalle J. I est mis à jour si $I \cap J$ est strictement inclus dans I. Par exemple, le domaine de v_1 sera strictement réduit si l'on démontre qu'il passe de $[0, 1]$ à $[0.5, 1]$.

Algorithme 5 : contraction(I,J)

si $I \cap J \subsetneq I$ **alors**
  $I \leftarrow I \cap J$
  retourner VRAI
sinon
  retourner FAUX
fin

L'algorithme *propagation(c)* (6) indique comment sont traitées les contraintes. Nous allons expliciter le traitement de l'équation 3.2.3 de notre exemple. Nous pouvons exprimer v_1 en fonction de v_4 : $v_1 = 1 - v_4$. Transposé aux intervalles, cela nous donne $[1, 1] - I_4 = [0.5; 0.8]$. I_1 est alors réduit car $I_1 \cap [0.5; 0.8] = [0; 1] \cap [0.5; 0.8] \subsetneq [0; 1]$.

L'algorithme (7) indique comment sont traités les termes bilinéaires. Dans notre exemple, nous allons voir comment propager la contraction de I_1 en $[0.5; 0.8]$. Par exemple pour v_2 :

$$I_1 \div I_3 = I_1 \times I_3^{-1} = [0.5; 0.8] \times [0; 1]^{-1} = [0.5; 0.8] \times [1; +\infty] = [0.5; +\infty]$$

I_2 est donc contracté en $[0.5; 1]$, c'est aussi le cas de I_3.

Enfin, l'algorithme *Feasibility Based Bound Tightenning* (8) présente le fonctionnement général de la procédure servant à réduire l'espace de définition avant le traitement de chaque nœud. Cette méthode très générale est notamment décrite dans Belotti et al. [18] où l'ensemble de ces algorithmes est remplacé par la résolution d'un programme linéaire.

Algorithme 6 : propagation(c)

Données :
$c \in \mathcal{C}$: une contrainte de la forme $\sum_{i \in \mathcal{N}} A_{ci} v_i = b_c$

début
$SUCCES \leftarrow FAUX$
pour chaque $i_0 \in \mathcal{N}$ **faire**
$SUCCES \leftarrow SUCCES \vee contraction\left(I_{i_0}, \frac{1}{A_{ci_0}}\left[[b_c, b_c] - \sum_{i \in \mathcal{N} - \{i_0\}} A_{ci} I_i)\right]\right)$
fin
retourner $SUCCES$
fin

Algorithme 7 : propagation(b)

Données :
$b = (i, j, k) \in \mathcal{B}$

début
$SUCCES \leftarrow FAUX$
$\delta \leftarrow VRAI$
tant que δ **faire**
$\delta \leftarrow FAUX$
$\delta \leftarrow \delta \vee contraction(I_i, I_j \times I_k)$
$\delta \leftarrow \delta \vee contraction(I_j, I_i \div I_k)$
$\delta \leftarrow \delta \vee contraction(I_k, I_i \div I_j)$
$SUCCES \leftarrow SUCCES \vee \delta$
fin
retourner SUCCES
fin

Algorithme 8 : L'algorithme FBBT

$SUCCES \leftarrow VRAI$
tant que $SUCCES$ **faire**
$SUCCES \leftarrow FAUX$
pour chaque $c \in \mathcal{C}$ **faire** $SUCCES \leftarrow SUCCES \vee propagation(c)$
pour chaque $b \in \mathcal{B}$ **faire** $SUCCES \leftarrow SUCCES \vee propagation(b)$
fin

Comme on l'a vu dans notre exemple, durant la contraction de borne, il est possible d'avoir $(i_0, j_0, k_0) \in \mathcal{B}$ tel que $I_i \subsetneq I_j \times I_k$. En effet $I_2 \times I_3 = [0.5; 1] \times [0.5; 1] = [0.25; 1]$, et $I_1 = [0.5; 0.8]$ est strictement inclu dans $[0.25; 1]$. Cela voudrait dire qu'il faut rajouter une contrainte portant sur le terme non linéaire v_{i_0}. De telles contraintes peuvent être pénalisantes pour les méthodes de recherche de solutions réalisables [44]. A la fin de *FBBT*, nous imposons $\forall (i, j, k) \in \mathcal{B}$ $I_i = I_j \times I_k$, $I_j$et I_k étant bien entendu les bornes obtenues à l'issue de *FBBT*. Autrement dit, à l'issue de *FBBT* même si nous disposons de bornes plus fines pour v_i, nous les remplaçons par le produit des bornes contractées des variables v_j et v_k.

3.2.2 Réduction du domaine en utilisant une relaxation

Comme indiqué par exemple dans [15, 19, 18, 11], les relaxations construites dans les méthodes de type *Branch-and-Bound* nous permettent de réduire les bornes de nos variables. En effet, si nous changeons l'objectif pour minimiser v_{i_0}, la borne inférieure ainsi calculée est peut-être plus élevée que l_{i_0} : nous pouvons alors réduire le domaine de la variable v_{i_0}. Comme indiqué dans l'algorithme *OBBT* (*Optimality Based Bound Tightenning*) (9), cette méthode requiert à chaque itération deux appels à la procédure de calcul de la borne inférieure par variable, ce qui est coûteux. Elle n'est donc pas préconisée en pratique pour les très grosses instances et son efficacité en terme d'amélioration de la résolution n'est pas garantie [17].

Algorithme 9 : L'algorithme OBBT

Données :
borneInf(i) et borneSup(i) : une procédure permettant d'obtenir une borne inférieure et supérieure pour la variable $v_i, i \in \mathcal{N}$

$SUCCES \leftarrow VRAI$
tant que *SUCCES* **faire**
  pour chaque $i \in \mathcal{N}$ **faire**
  $SUCCES \leftarrow SUCCES \vee contraction\,(I_i, [borneInf(i), borneSup(i)])$
fin

Pour les mêmes raisons que *FBBT*, à la fin de *OBBT*, nous imposons $\forall (i, j, k) \in \mathcal{B}$ $I_i = I_j \times I_k$.

3.3 Calcul de la borne inférieure

Une technique très utilisée dans la littérature pour isoler la non-linéarité consiste à relaxer les contraintes (3.1.3) :

$$v_i = v_j v_k, \forall (i,j,k) \in \mathcal{B}$$

La borne inférieure calculée par cette estimation est généralement faible car les variables associées aux termes non-linéaires ont beaucoup plus de liberté (dans notre cas, elles ne sont plus contraintes de décrire un hyperboloïde). Nous allons maintenant introduire l'ensemble des méthodes utilisées dans notre approche pour améliorer cette estimation.

3.3.1 Amélioration de la convexification des termes bilinéaires

Remplacer les termes bilinéaires par des variables artificielles fournit de mauvaises bornes car « on relaxe trop le problème ». Dans cette partie, la relaxation est construite pour relaxer le moins possible les contraintes non-linéaires tout en gardant un problème simple à résoudre (*i.e.* un programme linéaire).

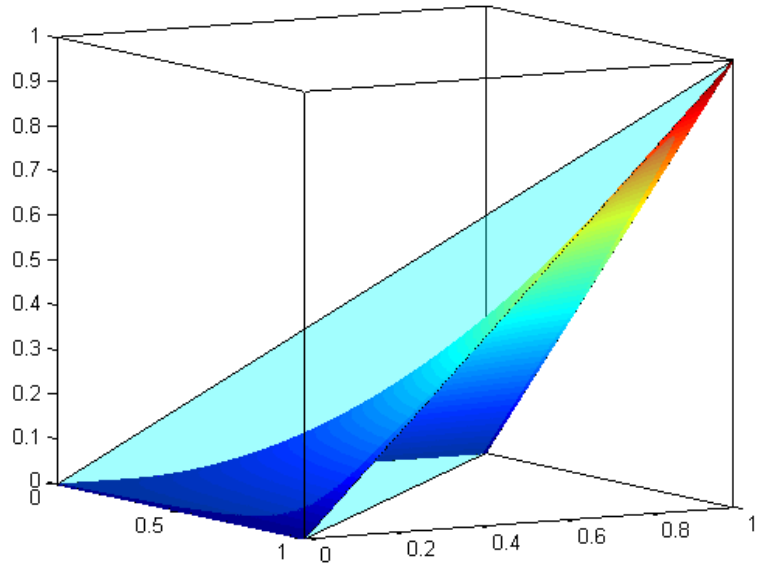
L'enveloppe convexe d'une fonction f sur un domaine D est définie en $v \in D$ par $\max_{\phi \in Vex} \{\phi(v)\}$, Vex étant l'ensemble des fonctions convexes qui sont inférieures à f sur D. De façon similaire, l'enveloppe concave d'une fonction f sur un domaine D est définie en $v \in D$ par $\min_{\phi \in Cav} \{\phi(v)\}$, Cav étant l'ensemble des fonctions concaves qui sont supérieures à f sur D. L'ensemble convexe d'une fonction f sur un domaine D est l'intersection des épigraphes de son enveloppe convexe sur D et des hypographes de son enveloppe concave sur D.

Al-Khayal and Falk [9] ont démontré que le plus petit ensemble convexe contenant l'hyperboloïde est un tétraèdre. Nous l'appellerons ensemble convexe du terme bilinéaire associé, il est représenté dans la figure 3.3.1. La relaxation est renforcée car le volume du domaine de définition de (v_i, v_j, v_k) est réduit, (v_i, v_j, v_k) sont maintenant contraints à évoluer dans *l'ensemble convexe*. Androulakis et al. [13] ont montré que cet ensemble est un tétraèdre, et que de plus

l'erreur $|v_i - v_j v_k|$ commise sur ce terme non linéaire est au maximum $\frac{1}{4}(u_j - l_j)(u_k - l_k)$ si (v_i, v_j, v_k) évolue dans l'ensemble convexe .

La convergence de l'algorithme tient alors au fait que l'erreur diminue au fur et à mesure que l'on partitionne le domaine de définition des variables.

FIGURE 3.3.1: Plus petit ensemble convexe contenant l'hyperboloïde



Ce tétraèdre peut être décrit de plusieurs façons. La première pourrait être d'exprimer chacun de ses points comme une combinaison convexe de ses sommets. La deuxième serait de décrire explicitement chaque face du tétraèdre. Costa et al. [22] ont comparé ces deux approches pour des problèmes comprenant des termes multi-linéaires. Les résultats pour le cas bilinéaire suggèrent qu'aucune d'elle ne surpasse l'autre. Pour chaque triplet $(i, j, k) \in \mathcal{B}$, nous optons donc pour la deuxième solution et ajoutons les contraintes suivantes :

$$convexSet\,(i,j,k) = \left\{ \begin{array}{c} v_i - u_j v_k - l_k v_j + u_j l_k \leq 0 \\ v_i - l_j v_k - u_k v_j + l_j u_k \leq 0 \\ v_i - u_j v_k - u_k v_j + u_j u_k \geq 0 \\ v_i - l_j v_k - l_k v_j + l_j l_k \geq 0 \end{array} \right\}$$

Ces contraintes sont obtenues en multipliant les contraintes de bornes sur les variables v_j et v_k. Par exemples, les contraintes suivantes sont de signes constants :

$$0 \leq u_j - v_j$$
$$0 \leq u_k - v_k$$

Nous pouvons donc les multiplier pour obtenir la nouvelle contrainte ;

$$\begin{array}{rc} & (u_j - v_j)\,(u_k - v_k) \geq 0 \\ \Leftrightarrow & v_j v_k - u_j v_k - u_k v_j + u_j u_k \geq 0 \end{array}$$

Enfin, en remplaçant $v_j v_k$ par v_i nous obtenons la contrainte de l'ensemble convexe du terme $(i,j,k) \in \mathcal{B}$:

$$v_i - u_j v_k - u_k v_j + u_j u_k \geq 0$$

Les ensembles convexes des termes bilinéaires ont une propriété bien connue : si l'on a une solution réalisable où l'une des variables v_j ou v_k est à une de ses bornes alors nécessairement $v_i = v_j v_k$. En effet, si par exemple $v_j = l_j$ alors :

$$\left\{ \begin{array}{c} v_i - l_j v_k - u_k v_j + l_j u_k \leq 0 \Longrightarrow v_i \leq l_j v_k \\ v_i - l_j v_k - l_k v_j + l_j l_k \geq 0 \Longrightarrow v_i \geq l_j v_k \end{array} \right\} \Longrightarrow v_i = l_j v_k = v_j v_k$$

Enfin, si l'on ajoute les contraintes décrivant l'ensemble convexe pour chaque terme bilinéaire $(i,j,k) \in \mathcal{B}$, les contraintes de bornes sur les variables associées sont redondantes. En effet, ces

contraintes définissent des tétraèdres dont les sommets satisfont les contraintes de bornes sur les variables. On peut le voir sur la figure (3.3.1).

3.3.2 Génération de coupes

Une autre façon d'améliorer la borne inférieure est de générer des contraintes redondantes dans le modèle non linéaire, mais qui renforcent la relaxation. Cette technique est connue sous le nom de *Reformulation Linearization Technique* (RLT) [45, 46]. L'idée principale est de multiplier des contraintes de $\mathcal{C}$ par une variable afin d'obtenir une nouvelle contrainte redondante dans le problème non linéaire mais qui renforce les relaxations utilisées dans des *Branch-and-Bound*.

Le principal défaut de cette méthode est que ces produits créent de nouveaux termes bilinéaires, ce qui ajoute plus de contraintes de convexification. Nous avons choisi de ne faire que des multiplications qui ne créent pas de nouveaux termes bilinéaires, comme dans Liberti et al. [31]. Nous appellerons $RRLT$ ces coupes, $Reduced - RLT$, car nous ne faisons qu'un sous-ensemble des multiplications.

Un terme bilinéaire $v_j v_k$ existe dans le modèle si $\exists i \in \langle 1, |\mathcal{V}| \rangle$ tel que $(i,k,j) \in \mathcal{B}$ ou bien $(i,j,k) \in \mathcal{B}$, dans la suite nous noterons $i \equiv bil\,(k,j) \equiv bil\,(j,k)$. Pour $c \in \mathcal{C}$, nous noterons $nonZero\,(c) = \{i \in \langle 1, |\mathcal{V}| \rangle \,:\, A_{ci} \neq 0\}$ l'ensemble des coefficients non nuls de la contrainte c. L'algorithme $isValidRRLT\,(c,i)$ (10) décrit si la multiplication de la contrainte c par la variable v_i est valide, c'est-à-dire qu'elle ne crée pas de nouveaux termes bilinéaires.

Algorithme 10 : L'algorithme isValidRRLT

si $v_i \in \mathcal{V}_1$ **alors**
  retourner $FAUX$
sinon
  pour chaque $j \in nonZero\,(c)$ **faire**
    si *le terme bilinéaire* $v_i v_j$ *n'existe pas dans le modèle* **alors** retourner $FAUX$
  fin
fin
retourner $VRAI$

$\forall c \in \mathcal{C}$, on note $RRLT\,(c) \subset \langle 1, |\mathcal{V}| \rangle$ l'ensemble des variables v_i telles que la multiplication de c par v_i est valide. La contrainte résultant de cette multiplication est notée $rrlt\,(c,i)$:

$$\sum_{j \in nonZero(c)} A_{cv_j} v_{bil(j,i)} = b_c v_i$$

3.3.3 La linéarisation

En utilisant l'ensemble des techniques présentées précédemment, nous convexifions, voire linéarisons, le modèle $\mathbb{P}$ en $\mathcal{L}(\mathbb{P})$. Nous n'ajoutons que les contraintes associées aux tétraèdres des termes bilinéaires présents dans $\mathcal{B}$ lors de la phase de convexification. Lors de la génération des coupes $RRLT$, nous faisons le choix de ne faire que des produits contraintes-variables ne créant pas de nouveaux termes bilinéaires. Malgré toutes ces précautions, le programme linéaire à résoudre a beaucoup de variables et de contraintes. $\mathcal{L}(P)$ contient $4|\mathcal{B}|$ contraintes de convexification et $\sum_{c \in \mathcal{C}} |RRLT(c)|$ coupes $RRLT$.

$$\begin{array}{lcl}
\min & v_{obj} & \\
& Av = b & \\
& l_i \leq v_i \leq u_i & \forall i \in \langle 1, |\mathcal{V}| \rangle \\
& convexSet(i,j,k) & \forall (i,j,k) \in \mathcal{B} \\
& rrlt(c,i) & \forall c \in \mathcal{C}, i \in RRLT(c)
\end{array}$$

3.4 *Alternate* pour la recherche de solutions réalisables

Nous essayons de construire une solution réalisable à chaque nœud à partir de la solution fournie par la relaxation. Cela permet d'améliorer la meilleure solution connue, ou bien tout simplement d'en déterminer une s'il n'y en a pas de trouvée précédemment. Comme nous l'avons vu avant, si l'on fixe les variables de $\mathcal{V}_2$ ou les variables de $\mathcal{V}_3$ notre problème devient linéaire. Nous pouvons ainsi définir un processus itératif qui a déjà été étudié dans [15, 27], l'heuristique *Alternate*, permettant d'améliorer une solution réalisable.

Algorithme 11 : L'algorithme *Alternate*

Données :
δ : VRAI si l'on fixe les variables $\mathcal{V}_2$, FAUX si on fixe $\mathcal{V}_3$
$(v_i)_{i\in\mathcal{N}}$: une solution courante
début
  $\delta \leftarrow$ choisi au hasard
  tant que *il y a une amélioration dans la fonction objectif* **faire**
    si δ **alors**
      $(v_i)_{i\in\mathcal{V}_1\cup\mathcal{V}_3} \leftarrow$ la solution du problème à $\mathcal{V}_2$ fixé
      $\delta \leftarrow FAUX$
    sinon
      $(v_i)_{i\in\mathcal{V}_1\cup\mathcal{V}_2} \leftarrow$ la solution du problème à $\mathcal{V}_3$ fixé
      $\delta \leftarrow VRAI$
    fin
  fin
fin

Dans le cas où la solution issue de la relaxation n'est pas réalisable, nous ajoutons des variables d'écart sur les contraintes de bornes des variables qui ne sont pas des termes bilinéaires. En effet, comme indiqué dans la section contraction de bornes, à la fin de l'algorithme $FBBT$, nous imposons que :

$$\forall\,(i,j,k)\in\mathcal{B},\; I_i = I_j \times I_k$$

Ainsi, en notant $\mathcal{V}^{\mathcal{B}} = \{i \in \mathcal{V}_1 : \exists\,(j,k) \in \mathcal{V}_2 \times \mathcal{V}_3\,(i,j,k) \in \mathcal{B}\}$, toute solution satisfaisant $v_i = v_j v_k\ \forall\,(i,j,k) \in \mathcal{B}$, satisfait nécessairement les contraintes de bornes sur les variables $v_i \in \mathcal{V}^{\mathcal{B}}$.

Le problème résolu dans cette phase de recherche de solution réalisable est :

$$
\begin{aligned}
\min \quad & \sum_{v_i \in \mathcal{V}\setminus\mathcal{V}^{\mathcal{B}}} \varepsilon_i^+ + \varepsilon_i^- & \\
& Av = b & \\
& v_i = v_j v_k & \forall\,(i,j,k) \in \mathcal{B} \\
& l_i \leq v_i + \varepsilon_i^- - \varepsilon_i^+ \leq u_i & \forall i \in \mathcal{V}\setminus\mathcal{V}^{\mathcal{B}} \\
& l_i \leq v_i \leq u_i & \forall i \in \mathcal{V}^{\mathcal{B}} \\
& \varepsilon_i^-, \varepsilon_i^+ \geq 0 & \forall i \in \mathcal{V}\setminus\mathcal{V}^{\mathcal{B}}
\end{aligned}
$$

Nous minimisons la somme de ces variables d'écarts. Si le minimum obtenu est zéro alors une solution réalisable a été trouvée. L'algorithme est alors relancé à partir de cette solution réalisable mais cette fois sans variable d'écart. Des stratégies d'utilisation de cette technique ont été testées sur notre problème. Elles seront détaillées dans le chapitre suivant.

3.5 La stratégie de branchement

La séparation est nécessaire pour affiner l'estimation mais aussi pour garantir la convergence du *Branch-and-Bound*. Elle améliore l'estimation car elle réduit le volume des tétraèdres associés aux termes bilinéaires. Pour un terme bilinéaire dont une variable est la variable de branchement, l'union des tétraèdres des nœuds fils est strictement incluse dans le tétraèdre du nœud père. Nous noterons $(\bar{v}_i)_{i\in\mathcal{N}}$ la solution obtenue lors de la résolution de $\mathcal{L}(P)$.

Cette règle a été proposée dans Audet et al [15]. Nous choisissons le terme bilinéaire $(i^*, j^*, k^*) = arg \max_{(i,j,k)\in\mathcal{B}} |\bar{v}_i - \bar{v}_j\bar{v}_k|$. Ensuite, nous calculons pour chaque variable le point de séparation qui « éloigne » le plus (i^*, j^*, k^*) des ensembles convexes des fils. Pour finir, nous branchons sur la variable dont la valeur définie par la règle précédente est la plus près du milieu de son intervalle.

3.6 Deux formulations pour notre problème : *FP* et *RFP*

Jusqu'à présent, nous avons présenté une boite noire permettant de résoudre des problèmes ($BSBP$). Nous pouvons maintenant nous poser la question de savoir quelle formulation mathématique proposer à cette boite noire.

Nous allons introduire deux formulations de notre problème de fabrication d'aliments qui induisent deux comportements bien différents de l'algorithme décrit dans les parties précédentes. La principale différence entre ces deux formulations est leur ensemble de termes non linéaires. Comme dans la P-formulation du *pooling standard*, nous introduisons des variables directement liées aux caractéristiques des pré-mélanges. Pour $p \in P$ et $k \in K$, $t_{kp} \in \left[L^K_{kp}, U^K_{kp}\right]$ est la teneur en caractéristique k du pré-mélange p. Nous introduisons aussi des variables artificielles liées aux coûts des pré-mélanges, pour $p \in P$ c_p est le coût unitaire du pré-mélange p. Si l'on ne considère aucune contrainte de rapport de caractéristiques ou de consommations de matières premières, notre modèle (FP) est :

$$\min \quad v_{obj} \tag{3.6.1}$$

$$v_{obj} = \sum_a D_a \left(\sum_m C_{ma} x_{ma} + \sum_p \sum_m C_{mpa} w_{mpa} \right)$$

$$t_{ka} = \sum_{m \in MP_a} S_{mk} x_{ma} + \sum_{p \in P} \sum_{m \in MP_p} S_{mk} w_{mpa} \tag{3.6.2}$$

$$t_{kp} = \sum_{m \in MP_p} S_{mk} y_{mp} \tag{3.6.3}$$

$$w_{mpa} = y_{mp} z_{pa} \tag{3.6.4}$$

$$L_{ka} \leq t_{ka} \leq U_{ka} \tag{3.6.5}$$

$$L_{kp} \leq t_{kp} \leq U_{kp} \tag{3.6.6}$$

$$L_{ma} \leq x_{ma} \leq U_{ma} \tag{3.6.7}$$

$$L_{mp} \leq y_{mp} \leq U_{mp} \tag{3.6.8}$$

$$L_{pa} \leq z_{pa} \leq U_{pa} \tag{3.6.9}$$

Les variables x_{ma}, w_{mpa}, t_{ka} forment l'ensemble $\mathcal{V}_1$, les variables y_{mp}, t_{kp} forment l'ensemble $\mathcal{V}_2$ et les variables z_{pa} forment l'ensemble $\mathcal{V}_3$.

Le modèle (RFP) s'écrit quant à lui :

$$\min \quad v_{obj} \tag{3.6.10}$$

$$v_{obj} = \sum_a D_a \left(\sum_m C_{ma} x_{ma} + \sum_p w_{pa} \right)$$

$$t_{ka} = \sum_{m \in MP_a} S_{mk} x_{ma} + \sum_{p \in P} w_{kpa}$$

$$c_{pa} = \sum_m C_{mpa} w_{mpa}$$

$$w_{mpa} = y_{mp} z_{pa}$$

$$w_{kpa} = t_{kp} z_{pa}$$

$$w_{pa} = c_{pa} z_{pa}$$

$$(3.6.3)$$

$$(3.6.5)(3.6.6)(3.6.8)(3.6.9)(3.6.7)$$

Les variables x_{ma}, w_{mpa}, w_{kpa}, w_{pa}, t_{ka} forment l'ensemble $\mathcal{V}_1$, les variables y_{mp}, t_{kp} forment l'ensemble $\mathcal{V}_2$ et les variables z_{pa} forment l'ensemble $\mathcal{V}_3$.

Même si (FP) et (RFP) sont équivalents, $\mathcal{L}(FP)$ et $\mathcal{L}(RFP)$ ne le sont pas. $\mathcal{L}(RFP)$ contient en plus les contraintes des ensembles convexes des termes non linéaires $t_{kp} z_{pa}$ et $z_{pa} c_{pa}$.

$\mathcal{L}(RFP)$ fournit de meilleures bornes que $\mathcal{L}(FP)$ car il y a plus de contraintes de convexification. Les stratégies de branchement peuvent nous amener naturellement à brancher sur les variables t_{kp} ou c_{pa}, les taux caractéristiques des pré-mélanges ou les coûts des pré-mélanges. Le prix à payer est que la taille des programmes linéaires augmente car il y a beaucoup plus de termes non linéaires, donc beaucoup plus de contraintes de convexification. FP ne contient que $\sum_{p \in P} |MP_p| \,.\, |PA_p|$ termes bilinéaires alors que RFP en contient $\sum_{p \in P} (|MP_p| + |KP|) \cdot |PA_p|$. Il n'est donc pas évident de savoir a priori quelle est la meilleure formulation, celle qui permettra à la boite noire de converger le plus efficacement.

3.7 Conclusion

Nous avons présenté une méthode de type *Branch-and-Bound* permettant de déterminer une solution optimale d'un problème ($BSBP$). Notre méthode inclut une heuristique spécifique pour ces problèmes, *Alternate,* ainsi que les briques utilisées dans l'outil de résolution de problèmes généraux *Couenne*. Dans le chapitre suivant, nous allons générer des instances afin de tester notre outil. Nous soumettrons aussi ces instances à *Couenne*.

Chapitre 4

Implémentation et test

Ce chapitre présente les expérimentations menées pour tester la performance de la méthode décrite dans le chapitre précédent. Dans un premier temps, nous présentons la génération des instances sur lesquelles nous avons effectué ces tests. Nous donnons ensuite des détails sur l'implémentation de l'algorithme. Enfin, nous décrivons les tests effectués pour paramétrer les différentes briques utilisées de notre algorithme. Nous reportons aussi les résultats obtenus par *Couenne* afin de comparer les deux approches.

4.1 Implémentation

La méthode a été implémentée en C++11. Nous avons utilisé la bibliothèque *Coin-OR* [2] pour interfacer les outils de programmation linéaire via le projet *OSI* [3] ainsi que l'outil de résolution de programmation linéaire *CLP* [4]. A tout moment, un utilisateur peut cependant choisir un autre outil pour résoudre les programmes linéaires (*CPLEX* [5], *XPRESS-MP* [6], etc.).

Pour notre modèle d'optimisation bilinéaire disjoint, il est important de bien définir les fonctions linéaires et les fonctions bilinéaires. En effet, l'heuristique et la construction de la linéarisation nécessitent de parcourir l'ensemble des éléments non nuls de chaque fonction du modèle. Nous avons utilisé une structure d'arbre équilibré disponible dans les classes *std : :map* du *C++*.

L'ajout et la suppression de termes se font en temps logarithmique mais le parcours des éléments non nuls est en temps constant.

La construction du modèle standard du chapitre 3 consiste à remplacer chaque terme non linéaire par une variable artificielle et à transformer le problème en un problème ne comprenant que des contraintes linéaires égalités. Nous ajoutons donc des variables d'écart liées à toutes les contraintes.

Lors de l'exécution de notre *Branch-and-Bound*, les seules informations qui sont modifiées sont les bornes sur les variables initiales ou bien celles des variables d'écart. Les informations stockées à chaque nœud sont donc réduites à ces intervalles, ce qui permet de limiter la taille mémoire utilisée par notre outil.

Une interface homme-machine a aussi été développée pour que les utilisateurs puissent définir de façon intuitive des problèmes d'optimisation de mélanges utilisant des pré-mélanges.

4.2 Génération d'instances à partir de données industrielles

Nous avons généré des instances à partir d'une base de données d'un des clients de A-Systems. Pour des raisons de confidentialité, nous n'avons gardé que les informations sur les matières premières et généré aléatoirement les formules ainsi que leurs contraintes. Les formules utilisées n'ont donc pas de liens avec un quelconque produit commercialisé. Ces instances sont disponibles sur le site [7].

Ce premier jeu d'instances ne comporte ni de contraintes portant sur les rapports de caractéristiques ni de contraintes portant sur les consommations de matières premières. La structure particulière de ces instances a été élaborée avec notre partenaire industriel. Pour les construire, nous répartissons les matières premières et les caractéristiques puis nous construisons une solution de référence qui servira de base pour ajouter les contraintes.

Au final, nous avons conservé 30 instances difficiles à résoudre pour l'ensemble des expérimentations de cette section.

4.2.1 Distribution des matières premières et des caractéristiques

Nous commençons par distribuer les matières premières et les caractéristiques potentiellement contraintes dans chaque aliment et chaque pré-mélange. Les matières premières peuvent être proposées soit aux aliments, soit aux pré-mélanges soit aux deux. Étant donnés deux paramètres $0 \leq \alpha_1 \leq \alpha_2 \leq 1$, pour chaque matière première, un nombre $p \in [0, 1]$ est généré aléatoirement.
- $p < \alpha_1$: la matière première ne peut être utilisée que pour les produits pré-mélanges
- $\alpha_1 \leq p < \alpha_2$: la matière première peut être utilisée pour les produits pré-mélanges et les aliments
- $p \geq \alpha_2$: la matière première ne peut être utilisée que pour les aliments

Nous procédons de la même façon pour les caractéristiques.

4.2.2 Construction d'une solution de référence

Pour générer une solution réalisable, nous fixons tout d'abord le nombre de pré-mélanges et le nombre d'aliments. Chaque recette (aliment ou pré-mélange) est générée en déterminant la quantité de chaque composant (pré-mélange ou matière première) y étant présent. Étant donnés deux paramètres $0 \leq \beta_1 \leq \beta_2 \leq 1$, pour chaque formule et pour chaque composant pouvant intervenir dans cette formule, un nombre $p \in [0, 1]$ est généré aléatoirement :
- $p < \beta_1$: le composant apparait avec une valeur strictement positive dans la formule et sa quantité est tirée aléatoirement dans $[0, 100]$
- $\beta_1 \leq p < \beta_2$: le composant apparait avec une valeur nulle dans le produit, il est proposé à la formule mais ne figure pas dans la recette.
- $p \geq \beta_2$: le composant ne peut pas être dans la formule

A la fin de cette étape, nous pouvons définir la demande en chaque aliment et les proportions associées à chaque composant. C'est uniquement à ce stade que nous pouvons calculer le nombre de matières premières effectivement utilisées dans l'instance. Il est en effet possible qu'une matière première ne soit utilisée dans aucune formule.

4.2.3 Génération des contraintes

Une fois la solution de référence construite, nous devons faire en sorte qu'elle soit réalisable lorsque nous ajoutons les contraintes. Pour cela, nous ajoutons des contraintes respectées par cette solution : chaque instance possède donc une solution réalisable.

Nous commençons par les contraintes de bornes sur les taux d'incorporation de matières premières qui sont générées avec le paramètre γ. Par exemple, nous allons expliciter la génération des bornes de la variable x_{ma} associée à la proportion de la matière première m dans l'aliment a. Étant donnés deux paramètres $0 \leq l \leq u$ et $\bar{x}_{ma}$ la valeur de cette variable dans la solution de référence, nous tirons un nombre aléatoire $p \in [0, 1]$ et si $p \leq \gamma$ nous contraignons cette variable en choisissant aléatoirement la borne inférieure dans $[max\{0, \bar{x}_{ma} - l\}, \bar{x}_{ma}]$ et la borne supérieure dans $[\bar{x}_{ma}, min\{1, \bar{x}_{ma} + u\}]$. Si $p > \gamma$, aucune contrainte n'est ajoutée. Nous faisons de même pour les variables y_{mp} et z_{pa}.

Les taux de caractéristiques sont contraints de façon analogue. Nous commençons par calculer la valeur du taux de chaque caractéristique pour la solution de référence. Ensuite, nous contraignons aléatoirement des caractéristiques dans des formules en imposant un intervalle contenant la valeur de la solution de référence.

Le tableau 4.1 présente les propriétés de chaque instance ainsi que le nombre de termes bilinéaires contenus dans nos deux formulations.

ID	$\|M\|$	$\|P\|$	$\|A\|$	$\|E\|$	$\|K\|$	BIL_FP	BIL_RFP
1	48	1	5	99	58	55	340
2	38	1	10	92	48	290	760
3	49	1	25	509	76	100	1175
4	40	1	45	617	71	495	2925
5	52	3	5	114	28	123	285
6	55	3	5	127	40	58	248
7	55	3	5	125	61	165	606
8	56	3	5	132	66	61	451
9	55	4	6	138	70	60	409
10	60	5	25	581	72	469	3390
11	59	5	35	819	80	524	4278
12	58	5	45	781	80	1301	6002
13	60	5	50	1154	79	828	7177
14	60	7	10	286	59	230	1139
15	58	7	20	366	74	1383	5400
16	59	7	25	450	77	1393	5978
17	59	7	30	430	80	1785	7378
18	59	7	35	689	79	1279	6982
19	59	9	10	290	47	1016	2524
20	59	9	15	320	66	1157	3454
21	60	9	20	406	80	1434	6198
22	60	9	25	456	81	2394	8758
23	59	9	30	517	72	1994	7524
24	60	9	30	463	81	2543	9897
25	57	9	35	514	78	3152	12555
26	60	11	20	482	72	1559	5302
27	60	11	25	464	75	2155	8567
28	60	11	30	560	80	2416	9898
29	60	11	35	699	80	3063	12124
30	60	11	45	723	82	4376	17269

TABLE 4.1: Caractéristiques des 30 instances générées aléatoirement

4.3 Résultats obtenus sur les instances industrielles

4.3.1 Comparaison du calcul des bornes inférieures pour *FP* et *RFP*

Dans un premier temps, nous nous intéressons à la forme du modèle mathématique que nous allons soumettre à la boite noire que représente notre algorithme. Pour cela, nous avons testé les meilleures stratégies pour résoudre la linéarisation dans le cas où l'on soumet la formulation (FP) ou (RFP).

Il est en effet possible d'ajouter directement toutes les contraintes de convexification ou bien d'ajouter itérativement uniquement celles qui sont violées. Quatre stratégies ont été testées :
- toutes les contraintes sont ajoutées au programme linéaire pour $\mathcal{L}(FP)$
- toutes les contraintes sont ajoutées au programme linéaire pour $\mathcal{L}(RFP)$
- les contraintes sont ajoutées dynamiquement au programme linéaire pour $\mathcal{L}(FP)$
- les contraintes sont ajoutées dynamiquement au programme linéaire pour $\mathcal{L}(RFP)$

Pour comparer les approches, le temps CPU (en secondes) nécessaire pour résoudre le programme linéaire à la racine du *Branch-and-Bound* a été mesuré. Il s'avère que pour $\mathcal{L}(FP)$ l'ajout de toutes les contraintes au départ était 45% plus rapide que l'ajout dynamique de celles-ci. En revanche, pour $\mathcal{L}(RFP)$, l'ajout dynamique de contraintes est 40% plus rapide que l'ajout de toutes les contraintes dès le début. Ces tests suggèrent que la génération dynamique est efficace pour $\mathcal{L}(RFP)$ mais pas pour $\mathcal{L}(FP)$.

Dans toute la suite, nous résoudrons $\mathcal{L}(FP)$ en ajoutant toutes les contraintes de convexification dès le début, alors que pour $\mathcal{L}(RFP)$ nous choisirons d'ajouter dynamiquement uniquement celles qui sont violées.

4.3.2 Calibrage d'*Alternate*

Alternate est une méthode dont l'utilisation permet beaucoup de variation et qui est appelée très souvent dans notre algorithme. C'est pourquoi nous avons cherché une bonne stratégie d'utilisation dans le *Branch-and-Bound*. Trois paramètres sont importants pour tester *Alternate* :

– la solution de départ
– le type de variables que nous fixons en premier
– utiliser ou non des variables d'écart dans le but de converger vers une solution réalisable

Huit configurations sont donc étudiées, la table 4.2 présente une synthèse des résultats obtenus sur nos 30 instances. Ces tests montrent que l'utilisation des variables d'écarts augmentent les taux de succès de l'heuristique, c'est-à-dire les chances de converger vers une solution réalisable.

On pourrait aussi penser que fixer les variables z en premier est la meilleure stratégie. Cependant, les tests sur l'instance 23 reportent un taux de succès de moins de 2% pour FP et de moins de 6% pour RFP si les variables z sont fixées en premier. De plus, les tests sur l'instance 28 reportent un taux de succès de moins de 1% pour FP et de moins de 2% pour RFP si les variables y, t sont fixées en premier. Nous concluons qu'il n'y a pas de choix préférable pour le type de variables fixées lors de la première itération.

	Premier type de variables fixées	(FP) slack	(FP) no slack	(RFP) slack	(RFP) no slack
Taux de succès (%)	$\mathcal{V}_2\left(y_{mp}, t_{kp}^{KP}\right)$	54	21	52	22
avec départ au hasard	$\mathcal{V}_3\left(z_{pa}\right)$	73	17	72	20
Taux de succès (%)	$\mathcal{V}_2\left(y_{mp}, t_{kp}^{KP}\right)$	63	51	69	48
avec départ à la solution de $\mathcal{L}(.)$	$\mathcal{V}_3\left(z_{pa}\right)$	86	66	88	67

TABLE 4.2: Résultat de *Alternate*

D'après les résultats des tests précédents, nous avons opté pour la stratégie suivante : à chaque nœud, *Alternate* est lancé en utilisant les variables d'écarts à partir de la solution de la relaxation. Le premier type de variables est choisi aléatoirement. Si aucune solution réalisable n'a été trouvée, nous générons un point de départ aléatoirement et utilisons ou non les variables d'écarts. Nous avons remarqué que partir d'un point aléatoire avec les variables d'écart augmentait le temps d'exécution de l'heuristique.

4.3.3 Comparaison des performances de notre approche avec *Couenne* pour les instances industrielles

Nous comparons maintenant les performances de notre outil auquel nous avons soumis les deux formulations (FP) et (RFP) à celles obtenues avec *Couenne*. Pour réaliser ces tests, nous avons utilisé un ordinateur Intel(R) Core(TM) 2 Duo CPU E8400 sous Linux équipé d'un processeur de 3 GHz et 3 *Go* de RAM . *Couenne* a été généré avec $CPLEX$ 12.2 [5] et nous lui soumettons la formulation (FP).

Vu la taille des instances, nous désactivons l'option $OBBT$ ce qui évite d'utiliser la méthode $OBBT$ décrite dans le chapitre 3 mais nous conservons l'heuristique basée sur IpOpt sinon *Couenne* a très peu de chance de trouver une solution réalisable (cf. manuel d'utilisation de *Couenne*).

Le but est de résoudre les instances à l'optimalité. Nous comparons donc le gap d'optimalité (ou le temps) et non la meilleure solution trouvée.

Nous remarquons que les valeurs des solutions optimales ne sont pas exactement les mêmes. Cela est du à la précision relative fixée à 10^{-4} pour la faisabilité et l'optimalité. Pour notre code, $a < b$ si et seulement si $\frac{b-a}{|a|} < 10^{-4}$ (si $a \geq 0$) et aucune opération d'arrondi n'est effectuée.

Le tableau 4.3 présente les résultats moyens obtenus sur les 30 instances de la section précédente. La première ligne indique l'approche qui a été utilisée (*FP*, *RFP* ou *Couenne*). La seconde (resp. la troisième) ligne indique combien de solutions réalisables (resp. optimales) ont été déterminées. La quatrième ligne indique le temps moyen d'exécution en secondes sur les instances résolues jusqu'à l'optimalité et la cinquième ligne indique le gap d'optimalité obtenu au bout d'une heure de calculs. Enfin, la dernière ligne indique le nombre moyen de nœuds explorés par les *Branch-and-bound*. Les résultats instance par instance sont disponibles en annexe 1.

TABLE 4.3: Résultats moyens de *FP*, *RFP* et *Couenne*

Approche	*FP*	*RFP*	*Couenne*
Solutions réalisables trouvées	29	29	29
Solutions optimales trouvées	15	15	12
Temps moyen des instances résolues (s)	424.41	236.87	369.93
Gap moyen (%) des instances non résolues	3.17	3.59	9.22
Nombre de nœuds moyen	4430	370	22720

4.4 Résultats obtenus sur des instances générales de fabrication d'aliment

Nous avons testé notre outil sur des instances générées à partir de données métier ayant une structure particulière. Par exemple, pour une caractéristique il est possible d'avoir un apport de 0.001 pour une matière et 2300 pour une autre. Nous testons maintenant notre outil sur des instances plus générales, nous avons généré quelques instances ; nous les avons soumises à notre outil et à *Couenne*.

Les instances sont générées à l'aide de plusieurs paramètres dont la taille des ensembles de nœuds :
- $|M|$: le nombre de matières premières
- $|P|$: le nombre de pré-mélanges
- $|A|$: le nombre d'aliments

Nous générons les coûts des matières premières ainsi que les demandes en chaque produit. Nous construisons une solution de référence où le flot sur chaque arc est non nul avec une probabilité de 0.3. Nous n'ajoutons pas de bornes supplémentaires sur les arcs de la solution de référence. Si le flot d'un arc est nul dans la solution de référence, l'arc est conservé dans le graphe final avec une probabilité de 0.5.

Pour ajouter une contrainte caractéristique, nous générons un apport pour chaque matière première. Nous sélectionnons un pré-mélange et/ou un aliment et nous ajoutons une contrainte bornant le taux de cette caractéristique dans le pré-mélange et/ou l'aliment. Les paramètres suivants indiquent combien de contraintes sont générées uniquement sur les pré-mélanges, uniquement sur les aliments et pour les deux :

– $|KP|$: le nombre de contraintes caractéristiques dans un pré-mélange
– $|KA|$: le nombre de contraintes caractéristiques dans un aliment
– $|KPA|$: le nombre de contraintes caractéristiques dans un aliment et un pré-mélange

Pour ajouter une contrainte de rapport, nous sélectionnons deux caractéristiques générées lors de l'ajout de contraintes caractéristiques pour former un rapport. Ensuite nous procédons comme pour les caractéristiques à la sélection d'un pré-mélange et/ou d'un aliment où sera ajoutée la contrainte. Les paramètres liés à ces contraintes dans le générateur sont :
– $|RP|$: le nombre de contraintes rapports dans un pré-mélange
– $|RA|$: le nombre de contraintes rapports dans un aliment
– $|RPA|$: le nombre de contraintes rapports dans un pré-mélange et un aliment

Pour les contraintes portant sur les consommations de matière première, nous sélectionnons une matière première et calculons sa consommation dans la solution de référence. La contrainte est ajoutée en imposant une consommation inférieure ou égale à celle de la solution de référence. $|CONSO|$ est le paramètre lié à ce type de contraintes, il représente le nombre de contraintes consommation de matières premières.

Le tableau 4.4 regroupe les caractéristiques des instances que nous avons générées.

TABLE 4.4: Caractéristiques des instances

Instances	$\|M\|$	$\|P\|$	$\|A\|$	$\|KP\|$	$\|KA\|$	$\|KPA\|$	$\|RP\|$	$\|RA\|$	$\|RPA\|$	$\|CONSO\|$
G1	30	2	10	10	10	10	5	5	5	0
G2	30	2	10	10	10	10	0	0	0	15
G3	30	2	10	10	10	10	5	5	5	5
G4	30	4	10	10	10	10	0	0	0	10
G5	30	4	10	10	10	10	5	5	5	0
G6	30	4	10	10	10	10	5	5	5	5

Nous avons résolu ces instances avec notre algorithme et la formulation *RFP,* et avec *Couenne.* L'outil de résolution de programmes linéaires que nous avons utilisé est Clp. Le tableau 4.5 présente les résultats obtenus pour *Couenne* et notre outil. Nous avons arrêté *Couenne* lorsque le temps d'exécution était de l'ordre de *100* fois supérieur à notre outil.

Nous pouvons voir que notre outil obtient les mêmes solutions que *Couenne* en terme de valeur de l'objectif. Notre outil est cette fois-ci bien plus rapide.

Table 4.5: Résultats sur les instances générales

	RFP		*Couenne*	
Instances	CPU(s)	Valeurs	CPU(s)	Valeurs
G1	<1	10341.30	72	10341.30
G2	<1	8171.91	>150	8171.91
G3	3.7	8586.71	45.6	8586.69
G4	2	9238.34	>3600	9238.34
G5	1.6	1409.48	>3600	1409.48
G6	3.6	9307.76	>3600	9309.19

4.5 Résultats obtenus sur les instances de pooling standard

Nous avons soumis les instances de *pooling standard* publiées par Alfaki et al. [11] à l'adresse `http://www.ii.uib.no/~mohammeda/spooling/`. Les instances classiques de la littérature ainsi que leurs caractéristiques sont décrites dans le tableau 4.6.

Table 4.6: Caractéristiques des instances de pooling utilisées

Instances	|S|	|I|	|T|	|K|
Adhya1	5	2	4	4
Adhya2	5	2	4	6
Adhya3	8	3	4	6
Adhya4	8	2	5	4
BenTal4	4	1	2	1
BenTal5	5	3	5	2
Foulds2	6	2	4	1
Foulds3	11	8	16	1
Foulds4	11	8	16	1
Foulds5	11	4	16	1
Haverly1	3	1	2	1
Haverly2	3	1	2	1
Haverly3	3	1	2	1
RT2	3	2	3	8

Alfaki et al. [11] ont aussi généré aléatoirement des instances de *pooling standard* de grande taille. Elles sont classées en trois catégories ayant un nombre de nœuds et de caractéristiques différents. Le tableau 4.7 résume les caractéristiques des instances, le paramètre de densité est

la probabilité qu'un arc $(i, j) \in (S \cup T) \cup (I \cup T) \cup (S \cup I)$ soit présent dans le graphe du problème.

TABLE 4.7: Instances aléatoires

		Tailles de l'instance				Intervalle du paramètre
Groupe	#instance	|S|	|I|	|T|	|K|	de densité
A	10	20	10	15	24	0.30 - 0.80
B	6	35	17	21	34	0.30 - 0.80
C	4	60	15	50	40	0.20 - 0.35

Dans la thèse de doctorat de Alfaki [10], le papier E de la section 2 illustrant la méthode présentée dans la section 4.2.3 décrit une heuristique constructive pour obtenir de bonne solution pour les instances de *pooling standard* de grande taille. Nous avons comparé les meilleures solutions obtenues par notre *Branch-and-Bound* limité à *10* itérations aux meilleures solutions rapportées dans cet article.

Pour ces expérimentations, nous avons choisi d'utiliser Clp comme outil de résolution de programmes linéaires, *Alternate* est lancée à chaque nœud une seule fois en choisissant aléatoirement et avec la même probabilité, le premier type de variable fixée. Nous avons soumis à notre outil la *Q-formulation* du pooling standard. Notre outil génère automatiquement les coupes associées à la PQ-formulation.

Sur les instances de type A, même si nous améliorons les valeurs pour 5 instances, en moyenne l'amélioration est de -3.08%. Pour les instances de type B, nous améliorons toutes les meilleures solutions. La moyenne est de 41.86%, la plus petite étant de 22.22% et la plus grande de 63.31%. Pour les instances de type C, nous améliorons toutes les meilleures solutions. La moyenne est de 35.12%, la plus petite étant de 8.30% et la plus grande de 69.53%.

Les tests de Alfaki et al. ont été effectués sur un ordinateur de bureau équipé d'un processeur Intel(R) 3.00GHz et 8Go de RAM. Sur les 20 instances proposées, nous améliorons donc les meilleures solutions connues de 15 d'entre elles. Globalement, nous améliorons de 18% la valeur de l'objectif tout en diminuant de moitié (54%) les temps d'exécution par rapport à leur heuristique, sachant que notre poste de travail est moins puissant. Les résultats détaillés instance par instance sont en annexe 2.

4.6 Conclusion

Dans ce chapitre, nous avons présenté des tests effectués pour valider notre outil de résolution. Nous avons généré des instances iundustrielles issues des bases de données d'un des clients de A-Systems. Nous avons également testé notre outil sur des instances plus générales ainsi que sur les instances de la littérature du *pooling standard*.

Les résultats obtenus valident notre approche. En effet, comparé à un solveur générique utilisé avec le paramétrage décrit plus haut, notre *Branch-and-Bound* donne de très bons résultats. Sur les instances industrielles, notre méthode trouve plus de solutions optimales et le gap moyen pour les instances non résolues est meilleur, pour des temps de calcul comparables. Sur les instances générées aléatoirement, les valeurs des solutions obtenues par notre méthode avec la formulation *RFP* sont les mêmes que celles obtenues par *Couenne*, mais dans des temps de calcul beaucoup plus rapides. Ces résultats sont aussi confirmés lors de la résolution des instances de *pooling standard*.

Chapitre 5

Le problème de fabrication

Ce chapitre présente un cas particulier du problème de *formulation d'aliments* : le problème de *fabrication*. Une fois ce nouveau problème introduit, nous essaierons de le résoudre à l'aide de notre outil et de *Couenne*. Nous proposerons ensuite une approche lagrangienne permettant de calculer une autre borne inférieure.

5.1 Définition du problème et génération d'instances

Le problème de *fabrication* est un problème de *formulation d'aliments* avec des données particulières. Industriellement, le but est de réduire le nombre de composants entrant en jeu dans la fabrication d'un aliment. Par exemple, un industriel disposant de 100 matières premières pour produire 10 aliments préfèrera utiliser 20 pré-mélanges plutôt que les 100 matières premières. Cela lui permettra de stocker moins de composants dans son usine mais aussi de produire en gros volumes, et donc en continu, les pré-mélanges.

Dans ce problème, toutes les matières premières sont proposées à tous les pré-mélanges, tous les pré-mélanges sont proposés à tous les aliments. Aucune matière première n'est de plus directement proposée à un aliment. Autrement dit, en reprenant les notations du chapitre 2 :

- $MA = \emptyset$
- $PA = P \times A$
- $MP = M \times P$

Les contraintes des aliments ne sont pas liées aux contraintes des pré-mélanges. Nous notons K^P et K^A les caractéristiques contraintes dans les pré-mélanges et les aliments tels que $K = K^P \cup K^A$ et $K^P \cap K^A = \emptyset$. Nous notons de la même façon R^P et R^A. Enfin, toutes les données associées aux pré-mélanges sont identiques, c'est-à-dire :

- $\forall a \in A : \forall p \in P,\ L_{pa} = \mathbf{L}_a, U_{pa} = \mathbf{U}_a$
- $\forall m \in M : \forall p \in P,\ L_{mp} = \mathbf{L}_m, U_{mp} = \mathbf{U}_m$
- $\forall k \in K^P : \forall p \in P,\ L_{kp} = \mathbf{L}_k, U_{kp} = \mathbf{U}_k$
- $\forall r \in R^P : \forall p \in P,\ L_{rp} = \mathbf{L}_r, U_{rp} = \mathbf{U}_r$
- $(m, a) \in M \times A : \forall p \in P\ C_{mp} = \mathbf{C}_m,\ C_{pa} = \mathbf{C}_a = 1$

Une modélisation du problème de *fabrication* est maintenant introduite :

$$\min \quad \sum_{m\in M}\sum_{p\in P}\sum_{a} D_a \mathbf{C}_m y_{mp} z_{pa} \tag{5.1.1}$$

$$L_{ka} \leq \sum_{m\in M}\sum_{p\in P} S_{mk} y_{mp} z_{pa} \leq U_{ka} \qquad \forall\,(k,a) \in K^A \times A \tag{5.1.2}$$

$$\mathbf{L}_k \leq \sum_{m\in M} S_{mk} y_{mp} \leq \mathbf{U}_k \qquad \forall\,(k,p) \in K^P \times P \tag{5.1.3}$$

$$\sum_{m\in M}\sum_{p\in P} \left(S_{mn_r} - L_{ra} S_{md_r}\right) y_{mp} z_{pa} \geq 0 \quad \forall\,(r,a) \in R^A \times A \tag{5.1.4}$$

$$\sum_{m\in M}\sum_{p\in P} \left(S_{mn_r} - U_{ra} S_{md_r}\right) y_{mp} z_{pa} \leq 0 \quad \forall\,(r,a) \in R^A \times A \tag{5.1.5}$$

$$\sum_{m\in M} \left(S_{mn_r} - \mathbf{L}_r S_{md_r}\right) y_{mp} \geq 0 \qquad \forall\,(r,p) \in R^P \times P \tag{5.1.6}$$

$$\sum_{m\in M} \left(S_{mn_r} - \mathbf{U}_r S_{md_r}\right) y_{mp} \leq 0 \qquad \forall\,(r,p) \in R^P \times P \tag{5.1.7}$$

$$\mathbf{L}_m \leq y_{mp} \leq \boldsymbol{U}_m \qquad \forall\,(m,p) \in M \times P \tag{5.1.8}$$

$$\mathbf{L}_a \leq z_{pa} \leq \mathbf{U}_a \qquad \forall\,(p,a) \in P \times A \tag{5.1.9}$$

Dans le problème de fabrication, nous choisissons d'exprimer les contraintes de proportions comme des contraintes caractéristiques. Les contraintes suivantes sont dans (5.1.3) et (5.1.2) :

$$\sum_{m\in M} S_{mk} y_{mp} = 1 \quad \forall p \in P$$

$$\sum_{m\in M}\sum_{p\in P} y_{mp} z_{pa} = 1 \quad \forall a \in A$$

Le but de ce chapitre est de mettre en avant les limites de l'approche par *Branch-and-Bound* sur des cas simples. Pour simplifier l'énoncé, les problèmes que nous considérerons ne comprennent pas de contraintes de rapports de caractéristiques mais notre démarche se généralise facilement.

Nous allons maintenant présenter la procédure utilisée pour générer aléatoirement des instances de problème de *fabrication*. Les caractéristiques contraintes dans les aliments et les pré-mélanges

sont différentes, nous n'avons plus besoin de les répartir comme dans le chapitre précédent. Les paramètres de notre générateur sont :

- $|M|$: le nombre de matières premières
- $|A|$: le nombre d'aliments
- $\left|K^P\right|$: le nombre de contraintes caractéristiques dans les pré-mélanges
- $\left|K^A\right|$: le nombre de contraintes caractéristiques dans les aliments

Tout d'abord, les coûts des matières premières ainsi que les demandes en chaque aliment sont des nombres entiers tirés aléatoirement dans $\langle 0, 100\rangle$. Nous construisons ensuite une solution de référence en fixant les valeurs des variables y et z.

Pour ajouter une contrainte caractéristique dans les pré-mélanges, nous générons des coefficients entiers S_{mk} associés à cette caractéristique dans $\langle 0, 100\rangle$. Pour chaque pré-mélange de la solution de référence nous calculons la valeur du taux de cette caractéristique.Nous obtenons un intervalle minimal pour que la solution de référence soit réalisable, que nous retenons comme contrainte de bornes pour cette caractéristique.

Pour ajouter une contrainte caractéristique dans les aliments, nous générons des coefficients entiers S_{mk} comme dans le paragraphe précédent. Pour chaque aliment, nous calculons la valeur du taux de caractéristiques dans la solution de référence. Ensuite nous tirons un nombre aléatoire α dans $[0, 1]$ et la contrainte est ajoutée ou non dans l'aliment suivant les valeurs de α :

- $\alpha < 0.25$: la caractéristique est contrainte inférieurement
- $0.25 \leq \alpha < 0.5$: la caractéristique est contrainte supérieurement
- $\alpha \geq 0.5$: la caractéristique n'est pas contrainte.

5.2 L'approche *Branch-and-Bound* générique

Nous avons soumis des exemples simples de problèmes de *fabrication* à *Couenne* et à notre outil. Nous reporterons les performances des outils puis nous donnerons des remarques sur la reformulation en mettant en avant les inconvénients de cette approche.

5.2.1 Résultats expérimentaux

Voici les résultats d'une instance où l'on fait varier le nombre de pré-mélanges. Cette instance contient 10 matières premières et 5 aliments. 5 caractéristiques sont contraintes dans les aliments et 5 caractéristiques sont contraintes dans les pré-mélanges. L'instance a été générée à partir d'une solution à 3 pré-mélanges. Le tableau 5.1 indique les caractéristiques de l'instance.

TABLE 5.1: Caractéristiques de l'instance

	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}
a_1					$\geq$42					
a_2	$\leq$36	$\leq$51								
a_3		$\leq$50	$\leq$51							
a_4	$\leq$27	$\geq$51		$\geq$63						
a_5	$\geq$24	$\leq$50		$\geq$59	$\leq$43					
P						$\leq$68	$\geq$42	$\leq$41	$\geq$46	$\leq$74
m_1	0	52	19	83	75	60	75	19	91	89
m_2	68	45	92	91	55	2	37	7	5	79
m_3	59	31	17	13	40	37	67	53	72	66
m_4	23	31	80	21	6	39	53	48	36	46
m_5	14	35	73	23	3	65	86	12	36	91
m_6	77	26	9	23	5	33	30	35	85	55
m_7	8	86	40	95	77	79	19	52	74	36
m_8	56	39	7	54	74	60	62	93	77	15
m_9	20	63	84	51	69	85	88	35	40	50
m_{10}	67	7	51	30	0	37	71	72	72	91

Nous avons soumis cette instance à *Couenne* et à notre outil avec la formulation *(FP)* et *(RFP)*. Le tableau 5.2 présente les résultats obtenus avec un temps limite de deux minutes.

La valeur de l'optimal est décroissante avec $|P|$, c'est normal car augmenter le nombre de pré-mélanges revient à relaxer le problème. Il y a un nombre de pré-mélanges au dessous duquel l'instance ne sera plus réalisable.

Nous pouvons remarquer que le temps moyen de résolution par nœuds augmente avec $|P|$. Cela est en partie du à la taille des relaxations et aux nombre de termes w_{mpa}.

TABLE 5.2: Résultats obtenus avec les méthodes de type *Branch-and-Bound*

	$\|P\|$	2	3	4	5
Couenne	Nœuds	4916	38600	29400	23300
	CPU / déviation	29 s	7.6%	14%	18%
	LB	**7114.48**	6449.49	6018.62	5738.75
	UB	**7114.48**	6999.16	6955.05	6955.05
(FP)	Nœuds	1818	9616	19464	15960
	CPU / déviation	20 s	3.8%	7.1%	8.1%
	LB	**7114.48**	6734.23	6460.90	6378.62
	UB	**7114.48**	6999.61	6955.05	**6940.15**
(RFP)	Nœuds	262	759	671	78
	CPU / déviation	46 s	0.35%	0.27%	28 s
	LB	**7114.48**	6979.08	6941.45	**6940.15**
	UB	**7114.48**	7003.54	6960.13	**6940.15**

5.2.2 La limite de la méthode

Toutes les matières premières sont proposées à tous les pré-mélanges et tous les pré-mélanges sont proposés à tous les aliments. De ce fait, la formulation *(FP)* du problème de fabrication conduit à un modèle comprenant $|P|\,|M|\,|A|$ termes bilinéaires. Les problèmes linéaires à résoudre lors du calcul d'une borne inférieure sont de tailles plus importants que ceux du chapitre 3. De plus, les bornes inférieures obtenues sont très mauvaises, *(RFP)* étant bien supérieure à *(FP)* cette fois-ci en terme de qualité de borne mais aussi en temps de résolution, même si l'on génère les coupes dynamiquement.

Pour toutes solutions du problème de fabrication, nous pouvons échanger la composition et l'intégration de plusieurs pré-mélanges, c'est-à-dire les valeurs des variables $y_{.p}$ et $z_{p.}$. Nous obtenons alors une autre solution de même qualité. Cette symétrie ralentit la progression des *Branch-and-Bound* car sans le savoir, on traite un nœud une nouvelle fois. Dans les sections suivantes, nous allons essayer de prendre en compte cette symétrie pour calculer une meilleure borne inférieure.

5.3 Une approche lagrangienne pour le calcul d'une borne inférieure

Nous nous sommes inspirés des travaux de Adhya et al. [8] et de Almutairi et al. [12] pour construire une relaxation lagrangienne en dualisant les contraintes non linéaires (5.1.2) de notre problème : dans notre cas simple, ce sont les contraintes de caractéristiques sur chaque aliment. Dans cette partie, si B est un polyèdre borné, nous noterons $Vert\,(B)$ l'ensemble de ses points extrêmes.

5.3.1 Définition de nouveaux ensembles

Pour exploiter la structure du problème de *fabrication*, nous définissons de nouveaux ensembles :

$$Y = \left\{ \hat{y} \in \mathbb{R}^{|M|} \;\middle|\; \begin{array}{ll} \mathbf{L}_m \leq \hat{y}_m \leq \mathbf{U}_m & \forall m \in M \\ \mathbf{L}_k \leq \sum_m S_{mk} \hat{y}_m \leq \mathbf{U}_k & \forall k \in K^P \end{array} \right\}$$

$$Z = \left\{ \hat{z} \in \mathbb{R}^{|A|} \;\middle|\; \mathbf{L}_a \leq \hat{z}_a \leq \mathbf{U}_a \quad \forall a \in A \right\}$$

$$\mathcal{N}^{EXACT} = \left\{ (\hat{w}, \hat{y}, \hat{z}) \in \mathbb{R}^{|M||A|} \times Y \times Z \;\middle|\; \hat{w}_{ma} = \hat{y}_m \hat{z}_a \quad \forall\, (m,a) \in MA \right\}$$

Les variables $\hat{y}_m$, $\hat{z}_a$ et $\hat{w}_{ma}$ représentent les variables y_{mp}, z_{pa} et w_{mpa} à p fixé. Le problème de fabrication est équivalent au problème $\mathcal{P}_{\mathcal{N}^{EXACT}}$ suivant :

$$\min \quad \sum_m \sum_p \sum_a D_a C_m w_{mpa} \tag{5.3.1}$$

$$s.t. \quad L_{ka} \leq \sum_m \sum_p S_{mk} w_{mpa} \leq U_{ka} \quad \forall\,(k,a) \in K^A \times A \tag{5.3.2}$$

$$(w_{.p.}, y_{.p}, z_{p.}) \in \mathcal{N}^{EXACT} \qquad \forall p \in P \tag{5.3.3}$$

Dans la suite, la contrainte (5.3.3) est généralisée à $(w_{.p.}, y_{.p}, z_{p.}) \in \mathcal{N}$, $\forall p \in P$, où $\mathcal{N} \subset \mathbb{R}^{|M||A|} \times Y \times Z$ est un ensemble borné. Le problème $\mathcal{P}_{\mathcal{N}}$ nous servira à décrire notre relaxation lagrangienne. $\mathcal{P}_{\mathcal{N}}$ est intéressant car nous démontrerons que la borne inférieure calculée en résolvant la linéarisations $\mathcal{L}(FP)$ est égale à la valeur d'une solution optimale de $\mathcal{P}_{\mathcal{N}^{FP}}$, où $\mathcal{N}^{FP}$ est le polyèdre défini par :

$$\mathcal{N}^{FP} = \left\{ (\hat{w}, \hat{y}, \hat{z}) \in \mathbb{R}^{|M||A|} \times Y \times Z \;\middle|\; \begin{array}{ll} \sum_m \hat{w}_{ma} = \hat{z}_a & \forall a \in A \\ \hat{w}_{ma} - L_a \hat{y}_m - L_m \hat{z}_a + L_a L_m \geq 0 & \forall\,(m,a) \in MA \\ \hat{w}_{ma} - U_a \hat{y}_m - U_m \hat{z}_a + U_a U_m \geq 0 & \forall\,(m,a) \in MA \\ \hat{w}_{ma} - U_a \hat{y}_m - L_m \hat{z}_a + U_a L_m \leq 0 & \forall\,(m,a) \in MA \\ \hat{w}_{ma} - L_a \hat{y}_m - U_m \hat{z}_a + L_a U_m \leq 0 & \forall\,(m,a) \in MA \end{array} \right\}$$

Le théorème suivant nous permet de comparer $\mathcal{N}^{FP}$ et $\mathcal{N}^{EXACT}$.

Théorème. $\mathcal{N}^{EXACT} \subseteq \mathcal{N}^{FP}$.

Démonstration. Soit $(\hat{w}^*, \hat{y}^*, \hat{z}^*) \in \mathcal{N}^{EXACT}$, $(\hat{w}^*, \hat{y}^*, \hat{z}^*) \in \mathbb{R}^{|M||A|} \times Y \times Z$ et $\hat{w}^*_{ma} = \hat{y}^*_m \hat{z}^*_a$ $\forall\,(m,a) \in MA$. Comme Y contient la contrainte $\sum_m \hat{y}_m = 1$, alors pour tout $a \in A$:

$$\sum_m \hat{w}^*_{ma} = \hat{z}_a \left(\sum_m \hat{y}^*_m \right) = \hat{z}^*_a$$

Enfin, pour tout m et a, les inégalités sont satisfaites car ce sont les contraintes associées à l'ensemble convexe du terme bilinéaire $\hat{y}_m \hat{z}_a$. □

Nous allons maintenant présenter une relaxation lagrangienne sur le problème $\mathcal{P}_{\mathcal{N}}$, où $\mathcal{N} \in \left\{\mathcal{N}^{FP}, \mathcal{N}^{EXACT}\right\}$.

5.3.2 Une relaxation lagrangienne

Nous notons μ^+_{ka}, μ^-_{ka} les variables duales associées aux contraintes de caractéristiques (5.3.2) sur les aliments, μ^+_{ka}est associée à la contrainte $\leq$ et μ^-_{ka} à la contrainte $\geq$. Nous considérons une relaxation lagrangienne de notre problème dualisant les contraintes (5.3.2) de $\mathcal{P}_{\mathcal{N}}$. En regroupant les termes, nous obtenons le lagrangien suivant L :

$$\begin{array}{rcl} \mathcal{N}^{|P|} \times \left(\mathbb{R}^+\right)^{|K||A|} \times \left(\mathbb{R}^+\right)^{|K||A|} & \longmapsto & \mathbb{R} \\ (w_{.p.}, y_{.p}, z_{p.}, \mu^+, \mu^-) & \longmapsto & L_0\left(\mu^+, \mu^-\right) + \sum_m \sum_p \sum_a L_{ma}\left(\mu^+, \mu^-\right) w_{mpa} \end{array}$$

où

$$\begin{aligned} L_{ma}\left(\mu^+, \mu^-\right) &= D_a C_m + \sum_k S_{mk}\left(\mu^+_{ka} - \mu^-_{ka}\right) \\ L_0\left(\mu^+, \mu^-\right) &= \sum_k \sum_a \left(-U_{ka}\mu^+_{ka} + L_{ka}\mu^-_{ka}\right) \end{aligned}$$

Comme indiqué dans Adhya et al. [8] et Almutairi et al. [12], la borne lagrangienne $\theta^*_{\mathcal{N}}$ est une borne inférieure valide de notre problème. Nous généralisons la définition et le calcul de ces relaxations lagrangiennes au problème $\mathcal{P}_{\mathcal{N}}$ paramétré par $\mathcal{N}$.

$$\theta^*_{\mathcal{N}} = \max_{\mu^+_{ka} \geq 0, \mu^-_{ka} \geq 0} \left\{ L_0\left(\mu^+, \mu^-\right) + \sum_{p \in P} \min_{(w_{.p.}, y_{.p}, z_{p.}) \in \mathcal{N}} \sum_{(m,a) \in M \times A} L_{ma}(\mu^+, \mu^-) w_{mpa} \right\} \quad (5.3.4)$$

$\theta^*_{\mathcal{N}}$ est la valeur d'une solution optimale du problème $D_{\mathcal{N}}$:

$$\begin{aligned} \theta^*_{\mathcal{N}} = \max \quad & L_0\left(\mu^+, \mu^-\right) + |P|\,\mu \\ s.t. \quad & \\ & \mu \leq \sum_{m,a} L_{ma}(\mu)\hat{w}_{ma} \quad \forall\left(\hat{w}, \hat{y}, \hat{z}\right) \in \mathcal{N} \\ & \mu \in \mathbb{R} \\ & \mu^+_{ka}, \mu^-_{ka} \geq 0 \qquad \forall k, a \end{aligned}$$

Le programme linéaire $D_{\mathcal{N}}$ contient une infinité de contraintes. Nous allons introduire la notion de générateurs qui va nous permettre de simplifier la résolution de $\mathcal{D}_{\mathcal{N}}$ en ne gardant qu'un nombre fini de contraintes.

5.3.3 Définition des générateurs

Définition. Si $\mathcal{N}$ est un ensemble borné, nous définissons quand c'est possible $\mathcal{G}\left(\mathcal{N}\right) \subseteq \mathcal{N}$, un ensemble fini, tel que pour chaque fonction linéaire f définie sur $\mathcal{N}$:

$$\forall u \in \mathcal{N}, \exists e \in \mathcal{G}\left(\mathcal{N}\right), \quad f\left(e\right) \leq \quad f\left(u\right)$$

Théorème. *$\mathcal{G}\left(\mathcal{N}^{FP}\right)$ existe, c'est l'ensemble $Vert\left(\mathcal{N}^{FP}\right)$ des points extrêmes du polyèdre $\mathcal{N}^{FP}$*

Démonstration. C'est un résultat bien connu de l'analyse convexe. □

Théorème. *La définition du générateur a un sens pour $\mathcal{N}^{EXACT}$ et*

$$\mathcal{G}\left(\mathcal{N}^{EXACT}\right) = \left\{\left(\hat{w}, \hat{y}, \hat{z}\right) \in \mathbb{R}^{|M||A|} \times Vert\left(Y\right) \times Vert\left(Z\right) | \hat{w}_{ma} = \hat{y}_m \hat{z}_a \forall m, a\right\}$$

Démonstration. On se ramène à un programme bilinéaire, les points extrêmes sont obtenus en résolvant des programmes linéaires à $\hat{y}$ ou $\hat{z}$ fixé. □

Théorème. $\mathcal{G}\left(\mathcal{N}^{EXACT}\right) \subseteq \mathcal{G}\left(\mathcal{N}^{FP}\right)$.

Démonstration. Nous allons démontrer que les éléments de $\mathcal{G}\left(\mathcal{N}^{EXACT}\right)$ sont des points extrêmes de $\mathcal{N}^{FP}$. Si nous supposons avoir $(\hat{w}^0, \hat{y}^0, \hat{z}^0) \in \mathcal{G}\left(\mathcal{N}^{EXACT}\right)$ et que ce point n'est pas dans $Vert\left(\mathcal{N}^{FP}\right)$ alors :

$$\exists\left(\hat{w}^1, \hat{y}^1, \hat{z}^1\right), (\hat{w}^2, \hat{y}^2, \hat{z}^2) \in \mathcal{N}^{FP}, (\hat{w}^1, \hat{y}^1, \hat{z}^1) \neq (w^2, y^2, \hat{z}^2) \text{et} \begin{bmatrix} \hat{w}^0 \\ \hat{y}^0 \\ \hat{z}^0 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} \hat{w}^1 \\ \hat{y}^1 \\ \hat{z}^1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} \hat{w}^2 \\ \hat{y}^2 \\ \hat{z}^2 \end{bmatrix}.$$

En effet, si $(\hat{w}^0, \hat{y}^0, t^0, \hat{z}^0)$ n'est pas un point extrême de $\mathcal{N}^{FP}$, il est le milieu de deux éléments de $\mathcal{N}^{FP}$. Il est important de remarqué que $\hat{y}^1, \hat{y}^2 \in Y$ et que $\hat{z}^1, \hat{z}^2 \in Z$.

On sait que $\hat{z}^0 = \frac{1}{2}\hat{z}^1 + \frac{1}{2}\hat{z}^2$ avec $\hat{z}^1, \hat{z}^2 \in Z$ or $\hat{z}^0 \in Vert(Z)$ donc nécessairement $\hat{z}^0 = \hat{z}^1 = \hat{z}^2$. De même, comme $\hat{y}^0 \in Vert\left(Y\right)$ on a $\hat{y}^0 = \hat{y}^1 = \hat{y}^2$.

$\forall a \in A\ \hat{z}^1_a = \hat{z}^2_a = \hat{z}^0_a \in \{L_a, U_a\}$ car Z ne contient que des contraintes de bornes sur les variables $\hat{z}$ et que $\hat{z}^0 \in Vert\left(Z\right)$. Par définition de $\mathcal{N}^{FP}$, s'il existe a_0 tel que $\hat{z}_{a_0}$ est à une de ses bornes alors forcément $\hat{w}_{ma_0} = \hat{y}_m \hat{z}_{a_0}\ \forall m$. On a donc $\hat{w}^0_{ma} = \hat{w}^1_{ma} = \hat{w}^2_{ma}\ \forall m, a$. Ce qui rend impossible l'hypothèse de départ $(\hat{w}^1, \hat{y}^1, \hat{z}^1) \neq (\hat{w}^2, \hat{y}^2, \hat{z}^2)$, $(\hat{w}^0, \hat{y}^0, \hat{z}^0)$ est donc un point extrême de $\mathcal{N}^{FP}$. □

5.3.4 Simplification de la relaxation lagrangienne

En utilisant la définition du générateur $\mathcal{G}\left(\mathcal{N}\right)$, la borne $\theta^*_{\mathcal{N}}$ peut être calculée en résolvant le programme linéaire suivant :

$$\begin{aligned}
\theta^*_{\mathcal{N}} = \max \quad & L_0(\mu^+, \mu^-) + |P|\,\mu \\
s.t. \quad & \\
& \mu \leq \sum_{m,a} L_{ma}(\mu^+, \mu^-)\hat{w}_{ma} \quad \forall\left(\hat{w}, \hat{y}, \hat{z}\right) \in \mathcal{G}\left(\mathcal{N}\right) \\
& \mu \in \mathbb{R} \\
& \mu^+_{ka}, \mu^-_{ka} \geq 0 \qquad \forall k, a
\end{aligned}$$

Théorème. $\theta^*_{\mathcal{N}^{EXACT}} \geq \theta^*_{\mathcal{N}^{FP}}$.

Preuve directement déduite de $\mathcal{G}\left(\mathcal{N}^{EXACT}\right) \subseteq \mathcal{G}\left(\mathcal{N}^{FP}\right)$.

Ce résultat a déjà été démontré par Adhya et al. [8] pour la P-formulation du *pooling standard*. Nous montrons ici que la hiérarchie des bornes inférieures est directement liée à celle des générateurs.

Dans la suite nous noterons $e \in \mathcal{G}(\mathcal{N})$ les éléments du générateur et $(\hat{w}^e, \hat{y}^e, \hat{z}^e)$ leurs coordonnées. La forme duale de $\mathcal{D}_{\mathcal{N}}$ nous conduit au problème $\mathcal{DW}_{\mathcal{N}}$ suivant

$$\begin{aligned}
\theta^*_{\mathcal{N}} = \min & \sum_{e \in \mathcal{G}(\mathcal{N})} \left(\sum_m \sum_a D_a C_m \hat{w}^e_{ma} \right) \lambda^e \\
L_{ka} \leq & \sum_{e \in \mathcal{G}(\mathcal{N})} \left(\sum_m S_{mk} \hat{w}^e_{ma} \right) \lambda^e \leq U_{ka} \quad \forall (k,a) \in KA \\
& \sum_{e \in \mathcal{G}(\mathcal{N})} \lambda^e = |P| \\
& \lambda^e \geq 0 \qquad\qquad \forall e \in \mathcal{G}(\mathcal{N})
\end{aligned}$$

Cette forme est très proche de la relaxation linéaire utilisée dans la décomposition de Dantzig-Wolfe [23]. Cette méthode est très utilisée dans les problèmes en nombres entiers, plus de détails sur cette méthode sont disponibles dans Desrosiers [23].

5.4 Calcul de la borne lagrangienne

Dans cette section, nous décrivons une méthode itérative pour résoudre $\mathcal{DW}_{\mathcal{N}}$, ce qui permet de calculer la borne lagrangienne qui vient d'être introduite.

5.4.1 Les algorithmes utilisés

Comme nous l'avons vu dans les sections précédentes, $\mathcal{G}\left(\mathcal{N}^{EXACT}\right)$ est un ensemble construit à partir des points extrêmes de Y et de Z. Il contient $|Vert(Y)| \times |Vert(Z)|$ éléments, soit

de l'ordre de $2^{|M|}2^{|A|}$. Le programme $\mathcal{DW}_{\mathcal{N}^{EXACT}}$ ne peut pas être résolu explicitement car ce programme linéaire contiendrait beaucoup trop de variables.

Nous choisissons donc de résoudre $\mathcal{DW}_{\mathcal{N}^{EXACT}}$ itérativement en considérant un sous ensemble $\mathcal{E}$ de $\mathcal{G}\left(\mathcal{N}^{EXACT}\right)$. Une fois le problème maître restreint résolu, nous résolvons un sous-problème de minimisation des coûts réduits calculés à l'aide des valeurs optimales duales (μ^{+*}, μ^{-*}) de $\mathcal{DW}_{\mathcal{E}}$. Si un élément du générateur a un coût réduit strictement négatif, nous l'ajoutons à $\mathcal{E}$ et réitérons le processus. Sinon, les valeurs des solutions optimales de $\mathcal{DW}_{\mathcal{N}^{EXACT}}$ et $\mathcal{DW}_{\mathcal{E}}$ sont identiques et valent $\theta^*_{\mathcal{N}^{EXACT}}$.

Le sous-problème permettant de générer une colonne e à coût réduit minimum est :

$$\begin{aligned} \min \quad & \sum_{m,a} L_{ma}(\mu^{+*}, \mu^{-^*})\hat{y}_m\hat{z}_a \\ s.t. \quad & \hat{y} \in Y \\ & \hat{z} \in Z = [\mathbf{L}_a, \mathbf{U}_a] \end{aligned}$$

Ce problème possède une solution optimale ayant chaque variable $\hat{z}_a$ à l'une de ses bornes (cf. Adhya et al. [8]). Nous proposons donc de résoudre ce programme bilinéaire en résolvant une problème d'optimisation linéaire en nombres entiers. Nous exploitons une propriété des éléments $(\hat{w}, \hat{y}, \hat{z})$ de $\mathcal{N}^{FP}$: si $\hat{z}_a \in \{L_a, U_a\}$ pour tout a, $\hat{w}_{ma} = \hat{y}_m\hat{z}_a$ pour tout m et pour tout a.

$$\begin{aligned} \min \quad & \sum_{m,a} L_{ma}(\mu^{+*}, \mu^{-*})\hat{w}_{ma} \\ s.t. \quad & \hat{z}_a = \mathbf{L}_a + \delta_a\left(\mathbf{U}_a - \mathbf{L}_a\right) \\ & \delta_a \in \{0, 1\} \\ & (\hat{w}, \hat{y}, \hat{z}) \in \mathcal{N}^{FP} \end{aligned}$$

5.4.2 Initialisation de la procédure

Le calcul des coûts réduits nécessite de disposer d'une base réalisable, c'est-à-dire d'un sous ensemble $\mathcal{E}$ tel que $\mathcal{DW}_{\mathcal{E}}$ est réalisable. Nous introduisons des variables artificielles pour obtenir une solution réalisable et dans un premier temps nous cherchons à minimiser les violations, comme dans une phase 1 du simplexe :

$$\begin{aligned}
&\min \sum_{(k,a)\in KA} \varepsilon^+_{ka} + \varepsilon^-_{ka} \\
L_{ka} \leq -\varepsilon^+_{ka} + \varepsilon^-_{ka} + \sum_{e\in\mathcal{E}} \left(\sum_m S_{mk} \hat{w}^e_{ma} \right) \lambda^e \leq U_{ka} \quad & \forall\,(k,a) \in KA \\
\sum_{e\in\mathcal{E}} \lambda^e = |P| & \\
\lambda^e \geq 0 \quad & \forall e \in \mathcal{E} \\
\varepsilon^+_{ka},\, \varepsilon^-_{ka} \geq 0 \quad & \forall\,(k,a) \in KA
\end{aligned}$$

Nous aurions aussi pu considérer l'objectif suivant, comme indiqué dans Briant et al. [21] :

$$\sum_{(k,a)\in KA} C^+_{ka}\varepsilon^+_{ka} + C^-_{ka}\varepsilon^-_{ka} + \sum_{e\in\mathcal{DW}_{\mathcal{E}}} \left(\sum_m \sum_a D_a C_m \hat{w}^e_{ma} \right) \lambda^e$$

En augmentant itérativement les coûts C^+_{ka} et C^-_{ka} des variables artificielles, l'algorithme est guidé vers une solution réalisable tout en tenant compte de l'objectif initial.

5.5 Résultats expérimentaux

5.5.1 $\mathcal{L}(FP)$, $\mathcal{L}(RFP)$ et l'approche lagrangienne

Nous avons démontré que $\theta^*_{\mathcal{N}^{EXACT}} \geq \theta^*_{\mathcal{N}^{FP}}$. C'est-à-dire que la borne lagrangienne est au moins aussi bonne que celle obtenue avec $\mathcal{L}(FP)$. C'est aussi le cas pour la borne $\mathcal{L}(RFP)$, pour les mêmes raisons.

Nous avons calculé la borne lagrangienne pour notre instance et différentes valeurs pour $|P|$.

Dans tous les cas, nous générons moins d'une centaine d'éléments de $\mathcal{G}\left(\mathcal{N}^{EXACT}\right)$ et le calcul de la borne lagrangienne prend moins d'une seconde. Nous calculons la borne lagrangienne car nous arrêtons le processus lorsque l'on ne génère plus de d'élément à coût réduit négatif, c'est-à-dire une fois la preuve d'optimalité acquise.

D'un autre coté, résoudre une relaxation *RLT* de notre problème est encore plus rapide mais la borne obtenue est naturellement de moins bonne qualité.

Dans le tableau 5.3, nous reportons les bornes inférieures des méthodes *Branch-and-bound* à la racine et après 120 secondes de *Branch-and-Bound* que nous avions obtenues lors des expérimentations décrites dans le tableau 5.2 .

Table 5.3: Comparaison des bornes inférieures

	Couenne		*(FP)*		*(RFP)*		*Relaxation*
$\|P\|$	Init	B&B (120s)	Init	B&B (120s)	Init	B&B (120s)	*Lagrangienne*
2	6174.33	**7114.48**	6367.80	**7114.48**	6961.46	**7114.48**	6996.44
3	5577.54	6449.49	6334.84	6734.23	6940.15	6979.08	**6956.19**
4	5483.00	6018.62	6334.84	6460.90	6940.15	6941.45	**6942.59**
5	5483.00	5738.75	6334.84	6378.62	**6940.15**	**6940.15**	**6940.15**

Ces résultats suggèrent que même si la borne lagrangienne est plus longue à obtenir que les bornes *RLT*, pour le problème de fabrication, elle est bien meilleure que celle obtenue après une exploration partielle de l'arbre d'un *Branch-and-Bound* basé sur des bornes *RLT*.

5.6 Conclusion

Cette partie met en avant le problème de fabrication. Les approches classiques fournissent des bornes inférieures très faibles. Nous avons proposé une approche lagrangienne qui exploite la symétrie de ce problème et fournit des bornes au moins aussi bonnes que les approches *Branch-and-Bound* basées sur des relaxations *RLT*.

La borne lagrangienne calculée sur des instances générées aléatoirement est parfois meilleure que celle obtenue en explorant un arbre partiel d'un *Branch-and-Bound* basé sur des relaxations

RLT. Cependant, nous n'avons pas trouvé le moyen d'intégrer cette borne dans une méthode exate : toutes les stratégies de branchement garantissant la convergence de cette borne lagrangienne vers la solution optimale cassent la symétrie de notre problème. Nous aurons donc à considérer plusieurs sous problèmes, ce qui ralentira le calcul de la borne lagrangienne et sera pénalisant.

Conclusion et perspectives

Nous avons étudié dans cette thèse un nouveau modèle mathématique pour le problème de formulation d'aliments : la conception de pré-mélanges. L'intégration d'une étape de fabrication intermédiaire dans le processus de fabrication induit une modélisation mathématique non linéaire. Le but de cette thèse était de proposer une approche exacte spécifique pour la nouvelle classe de problème à traiter par A-Systems ainsi qu'une implémentation rivalisant avec les outils génériques, et une interface utilisateurs.

Cette thèse a conduit à l'implémentation d'un outil de résolution ainsi que d'une interface permettant aux utilisateurs de créer, modifier et résoudre des problèmes de formulation avec conception de pré-mélanges. Les tests comparatifs effectués avec Couenne ont permis de valider le module d'optimisation. L'interface graphique pouvant interroger directement les bases des clients de A-Systems, ce nouveau module devrait rapidement être diffusé et utilisé dans l'industrie de formulation d'aliments.

Nous avons aussi mis en évidence des problèmes pour lesquels l'approche proposée semble insuffisante en terme d'efficacité : le problème de fabrication et la prise en compte des contraintes pourtant sur des pourcentages minimum. Le problème de fabrication, où les bornes inférieures de mauvaises qualités et les symétries présentes entre les solutions perturbent le déroulement des algorithmes de *Branch-and-Bound*. Les contraintes de pourcentage minimum font nécessairement intervenir des modèles contenant des nombres entiers, et ce même dans le cas de la formulation simple.

Les perspectives à ce travail sont nombreuses.

Tout d'abord, le programme linéaire issu de la linéarisation est résolu à chaque noeud sans

tenir compte de la solution du noeud parent. Il est possible que transmettre la base optimale du noeud parent aux noeuds fils améliore la résolution de la relaxation du noeud fils.

Dans une nouvelle version de l'algorithme présenté dans Audet et al. [16], les termes bilinéaires considérés lors du branchement sont ceux intervenant dans des contraintes du problème quadratique violées par la solution du problème relaxé. En plus d'utiliser cette même idée, nous pourrions l'étendre et ajouter uniquement des variables d'écart aux contraintes violées lorsque nous utilisons *Alternate* avec des variables d'écart.

Dans les problèmes de fabrication d'aliments, un ensemble de contraintes est composé de fonctions linéaires en variable y. Comme dans les récents travaux de Sherali et al. [45], nous pourrions gérer une base de ce système de contraintes, et dans le *Branch-and-Bound*, nous n'aurions plus qu'à travailler sur les termes bilinéaires ne comprenant pas ces variables en base.

Nous avons trouvé une autre stratégie de branchement qui sélectionne la variable dont la somme des erreurs sur tous les termes bilinéaires auxquels elle participe est maximale. L'intervalle de cette variable est séparé en son milieu si la valeur de la relaxation est trop près des bornes ou bien à la valeur fournie par la relaxation. Cette stratégie a été utilisée avec succès pour les problèmes d'optimisation polynomiaux (Sherali et al. [45]). Nous pourrions également tester cette approche.

Enfin, cette approche pourrait s'étendre à la résolution de problèmes *BSBP* issus d'autre contextes industriels (raffinage de produits pétroliers, acheminement de gaz, etc.).

Annexe 1 : Résultat sur les instances de fabrications d'aliments

Dans les tableaux qui suivent, pour chaque approche, la première colonne représente le temps CPU en secondes quand la solution optimale est trouvée. La seconde colonne est le gap d'optimalité relatif obtenu après une heure si l'instance n'a pas été résolue à l'optimalité. La troisième et la quatrième colonne sont les bornes inférieures et supérieures obtenues à l'issue du test. Enfin, la cinquième colonne indique le nombre de noeuds générés. Pour les instances résolues, le meilleur temps CPU est en gras, le meilleur gap relatif d'optimalité est en gras pour chaque instance non résolue.

Couenne résout optimalement 13 instances jusqu'à l'optimalité alors que (FP) et (RFP) en résolvent 16. Couenne est plus rapide sur deux instances : 2 et 13. Pour toutes les autres instances (FP) ou (RFP) sont les plus rapides. (FP) a résolu (18) instances alors que même après une heure Couenne et (RFP) non. (RFP) a résolu l'instance 20 alors que Couenne ou FP n'y sont pas arrivés.

Pour les instances de très grandes tailles, $(15 \rightarrow 30)$, (FP) et (RFP) surpasse Couenne. Aucune solution réalisable n'est trouvée pour l'instance 16, la meilleure borne inférieure est fournie par (FP). Nous remarquons que lorsque aucune méthode n'a trouvé de solution optimale en une heure, la meilleure solution a été déterminée par (FP) ou (RFP) pour toutes les instances sauf pour la 29 et la 30.

TABLE 4: Résultat de RFP et Couenne

	(RFP)				*COUENNE*			
	time (s)	gap (%)	LB	UB	NB	time (s)	gap (%)	LB	UB	NB
1	0.89		493489	493489	15	1.82		493472	493472	218
2	19.06		509221	509221	71	**18.24**		509220	509220	1013
3	0.35		3947097	3947097	1	3705.51		3946966	3947032	100356
4	116.83		7383214	7383860	97	417.12		7383362	7383362	6375
5	0.44		503341	503356	7	1.04		503340	503340	119
6	0.17		1275745	1275745	5	3.92		1275745	1275745	485
7	**3.68**		809874	809874	21	153.31		809873	809873	6335
8	**0.91**		1000192	1000192	11	1.38		1000192	1000192	132
9	**7.07**		583443	583449	61	12.24		583449	583449	967
10		**0.12**	3865557	3870290	3211		0.30	3857890	3869473	57360
11	8.01		6270118	6270608	9	986.06		6270169	6270169	19140
12	**526.43**		6268239	6268862	177		0.10	6262059	6268420	14414
13	1210.4		8140638	8141124	421	**46.35**		8140727	8140727	501
14	**0.19**		1367191	1367191	1	802.83		1367191	1367191	38305
15		3.06	1694554	1748049	657		3.05	1669111	1721707	62580
16			2099840		551			1941502		16276
17		3.28	2547497	2633922	325		5.91	2436127	2589142	19736
18		0.01	5420903	5421489	1003		2.77	5287877	5438423	39240
19	**1157.92**		376084	376084	767		10.32	340082	379235	59029
20	**500.7**		780092	780169	311		1.41	770580	781571	50732
21		**0.80**	2148092	2165328	767		5.64	2045451	2167638	22838
22		3.95	1602244	1668112	151		14.74	1461944	1714688	17246
23		**1.60**	2105990	2140228	513		9.17	1951463	2148538	18912
24		2.70	1336219	1373203	283		24.09	1069472	1408876	11542
25		3.22	3294296	3403928	121		12.51	3026944	3459595	13698
26		**0.62**	2327410	2341866	765		12.67	2064642	2363972	40571
27		3.21	2221361	2294916	381		15.16	1966052	2317277	20250
28		5.37	3649349	3856299	147		11.1	3432187	3860828	23119
29		5.08	3733322	3932969	217		7.71	3538451	3834242	12391
30		17.20	3982046	4809147	31		20.14	3548760	4443923	7727

Table 5: Résultat de FP et Couenne

	(FP)					$COUENNE$				
	time (s)	gap (%)	LB	UB	NB	time (s)	gap (%)	LB	UB	NB
1	**0.54**		493476	493476	17	1.82		493472	493472	218
2	28.31		509171	509222	367	**18.24**		509220	509220	1013
3	**0.19**		3947097	3947097	1	3705.51		3946966	3947032	100356
4	**5.7**		7383115	7383639	27	417.12		7383362	7383362	6375
5	**0.4**		503373	503373	17	1.04		503340	503340	119
6	**0.09**		1275819	1275819	9	3.92		1275745	1275745	485
7	2823.21		809820	809895	53071	153.31		809873	809873	6335
8	2.4		1000192	1000192	69	1.38		1000192	1000192	132
9	65.81		583415	583471	1219	12.24		583449	583449	967
10		0.12	3864706	3869225	14811		0.30	3857890	3869473	57360
11	**1.85**		6270310	6270310	7	986.06		6270169	6270169	19140
12	1988.44		6268043	6268607	3641		0.10	6262059	6268420	14414
13	719.93		8140642	8141297	1589	**46.35**		8140727	8140727	501
14	0.34		1367190	1367190	5	802.83		1367191	1367191	38305
15		**1.97**	1692772	1726860	5359		3.05	1669111	1721707	62580
16			**2106436**		2751			1941502		16276
17		**2.22**	2545296	2603149	2809		5.91	2436127	2589142	19736
18	**186.37**		5420935	5421350	461		2.77	5287877	5438423	39240
19		0.61	374725	377043	14411		10.32	340082	379235	59029
20	542.54		780141	780219	2177		1.41	770580	781571	50732
21		1.34	2137388	2166418	5135		5.64	2045451	2167638	22838
22		**3.93**	1599653	1665048	1619		14.74	1461944	1714688	17246
23		2.84	2095477	2156631	4241		9.17	1951463	2148538	18912
24		**2.42**	1328622	1361584	2007		24.09	1069472	1408876	11542
25		**2.91**	3289610	3388305	1813		12.51	3026944	3459595	13698
26		1.1	2325689	2351567	6633		12.67	2064642	2363972	40571
27		**2.78**	2224886	2288488	3863		15.16	1966052	2317277	20250
28		**5.00**	3649089	3841060	1861		11.1	3432187	3860828	23119
29		**4.33**	3710908	3878860	2357		7.71	3538451	3834242	12391
30		**12.74**	3914243	4485839	545		20.14	3548760	4443923	7727

TABLE 6: Résultats de FP et RFP

	(FP)					(RFP)				
	time (s)	gap (%)	LB	UB	NB	time (s)	gap (%)	LB	UB	NB
1	**0.54**		493476	493476	17	0.89		493489	493489	15
2	28.31		509171	509222	367	19.06		509221	509221	71
3	**0.19**		3947097	3947097	1	0.35		3947097	3947097	1
4	**5.7**		7383115	7383639	27	116.83		7383214	7383860	97
5	**0.4**		503373	503373	17	0.44		503341	503356	7
6	**0.09**		1275819	1275819	9	0.17		1275745	1275745	5
7	2823.21		809820	809895	53071	**3.68**		809874	809874	21
8	2.4		1000192	1000192	69	**0.91**		1000192	1000192	11
9	65.81		583415	583471	1219	**7.07**		583443	583449	61
10		0.12	3864706	3869225	14811		**0.12**	3865557	3870290	3211
11	**1.85**		6270310	6270310	7	8.01		6270118	6270608	9
12	1988.44		6268043	6268607	3641	**526.43**		6268239	6268862	177
13	719.93		8140642	8141297	1589	1210.4		8140638	8141124	421
14	0.34		1367190	1367190	5	**0.19**		1367191	1367191	1
15		**1.97**	1692772	1726860	5359		3.06	1694554	1748049	657
16			**2106436**		2751			2099840		551
17		**2.22**	2545296	2603149	2809		3.28	2547497	2633922	325
18	**186.37**		5420935	5421350	461		0.01	5420903	5421489	1003
19		0.61	374725	377043	14411	**1157.92**		376084	376084	767
20	542.54		780141	780219	2177	**500.7**		780092	780169	311
21		1.34	2137388	2166418	5135		**0.80**	2148092	2165328	767
22		**3.93**	1599653	1665048	1619		3.95	1602244	1668112	151
23		2.84	2095477	2156631	4241		**1.60**	2105990	2140228	513
24		**2.42**	1328622	1361584	2007		2.70	1336219	1373203	283
25		**2.91**	3289610	3388305	1813		3.22	3294296	3403928	121
26		1.1	2325689	2351567	6633		**0.62**	2327410	2341866	765
27		**2.78**	2224886	2288488	3863		3.21	2221361	2294916	381
28		**5.00**	3649089	3841060	1861		5.37	3649349	3856299	147
29		**4.33**	3710908	3878860	2357		5.08	3733322	3932969	217
30		**12.74**	3914243	4485839	545		17.20	3982046	4809147	31

Annexe 2 : Résultat sur les instances de pooling standard

Le tableau suivant contient les résultats obtenus par notre outils sur les instances de *pooling standard* de petite taille de la littérature.

TABLE 7: Résolution des instances de petite taille

Instances	Noeuds	Value
haverly1	2	-400
haverly2	24	-600
haverly3	40	-750
adhya1	1474	-549.804
adhya2	524	-549.803
adhya3	240	-561.045
adhya4	62	-877.647
foulds2	0	-1100
foulds3	6	-8
foulds4	0	-8
foulds5	4	-8
benTal4	2	-450
benTal5	0	-3500
rt2	94	-4391.85

Le tableau suivant contient les résultats obtenus par notre outils sur les instances de *pooling standard* de grande taille de la littérature.

TABLE 8: Résolution des instances de grandes tailles de *pooling standard*

Instances		UB (10 itérations)
sppA0	-35812,33	-33299.12639
sppA1	-29276,56	-23678.37877
sppA2	-23042,04	-19497.98288
sppA3	-39446,54	-36151.49470
sppA4	-37310,68	-38034.11176
sppA5	-24015,54	-26731.76310
sppA6	-41281,11	-41780.96927
sppA7	-41239,94	-41982.51346
sppA8	-28795,26	-30004.50429
sppA9	-21912,35	-21602.81018
sppB0	-32158,57	-40293.69063
sppB1	-50055,21	-61175.34773
sppB2	-33388,34	-50892.16092
sppB3	-55964,56	-73068.19219
sppB4	-37413,97	-58882,84486
sppB5	-36333,67	-59335,93517
sppC0	-66213,11	-79299.73814
sppC1	-80219,43	-86879.70279
sppC2	-68571,44	-116250.53398
sppC3	-79446,89	-113517.88974

Bibliographie

[1] http ://www.ziena.com/knitro.htm.

[2] http ://www.coin-or.org.

[3] https ://projects.coin-or.org/Osi.

[4] https ://projects.coin-or.org/Clp.

[5] http ://www.ibm.com/fr/fr.

[6] http ://www.fico.com.

[7] http ://www.g-scop.fr/ penzb.

[8] N. Adhya, M. Tawarmalani, and N.V. Sahinidis, *A Lagrangian approach to the pooling problem*, Industrial and Engineering Chemistry Research **38** (1999), no. 5, 1956–1972.

[9] F.A. Al-Khayyal and J.E. Falk, *Jointly constrained biconvex programming*, Mathematics of Operations Research **8** (1983), 273–286.

[10] M. Alfaki, *Models and solution methods for the pooling problem*, Doctoral thesis, BORA-UiB, The University of Bergen, 18 June 2012, http ://hdl.handle.net/1956/5847.

[11] M. Alfaki and D. Haugland, *Strong formulations for the pooling problem*, TOGO 2010, 2010, Toulouse, France, pp. 15–18.

[12] H. Almutairi and S. Elhedhli, *A new Lagrangean approach to the pooling problem*, Journal of Global Optimization **45** (2009), no. 2, 237–257.

[13] I. P. Androulakis, C.D. Maranas, and C.A. Floudas, αBB *: A global optimization method for general constrained nonconvex problems*, Journal of Global Optimization **7** (1995), no. 4, 337–363.

[14] E. Armagan, *Decomposition algorithms for global solution of deterministic and stochastic pooling problems in natural gas value chains*, Master's thesis, Massachusetts Institue of Technology, http ://yoric.mit.edu/sites/default/files/ArmaganThesis.pdf, 2009.

[15] C. Audet, J. Brimberg, P. Hansen, S. Le Digabel, and N. Mladenovic, *Pooling problem : alternate formulations and solution methods*, Management Science **50** (2004), no. 6, 761–776.

[16] C. Audet, P. Hansen, B. Jaumard, and G. Savard, *A branch and cut algorithm for non-convex quadratically constrained quadratic programming*, Mathematical Programming **87** (2000), no. 1, 131–152.

[17] M. Bagajewicz, *A review of recent design procedures for water networks in refineries and process plants*, Computers and Chemical Engineering **24** (2000), no. 9-10, 2093–2113.

[18] P. Belotti, S. Cafieri, J. Lee, and L. Liberti, *Feasibility-based bounds tightening via fixed points*, Combinatorial Optimization and Applications (O. Wu, W.and Daescu, ed.), Lecture Notes in Computer Science, vol. 6508, Springer Berlin / Heidelberg, 2010, pp. 65–76.

[19] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter, *Branching and bounds tightening techniques for non-convex MINLP*, Optimization Methods and Software **24** (2009), no. 4, 597–634.

[20] A. Ben-Tal, G. Eiger, and V. Gershovitz, *Global minimization by reducing the duality gap*, Mathematical Programming **63** (1994), no. 1-3, 193–212.

[21] O. Briant, C. Lemaréchal, Ph. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck, *Comparison of bundle and classical column generation*, Mathematical Programming **113** (2008), 299–344.

[22] A. Costa and L. Liberti, *Relaxations of multilinear convex envelopes : Dual is better than primal*, Experimental Algorithms (Ralf Klasing, ed.), Lecture Notes in Computer Science, vol. 7276, Springer Berlin Heidelberg, 2012, pp. 87–98.

[23] J. Desrosiers and M. E. Lubbecke, *A primer in column generation*, Column Generation (G. Desaulniers, J. Desrosiers, and M. Solomon, eds.), Springer US, 2005, pp. 1–32.

[24] C. A. Floudas and V. Visweswaran, *Primal-relaxed dual global optimization approach*, Journal of Optimization Theory and Applications **78** (1993), no. 2, 187–225.

[25] C.A. Floudas and V. Visweswaran, *A global optimization algorithm (GOP) for certain classes of nonconvex NLPs – i. theory*, Computers and Chemical Engineering **14** (1990), no. 12, 1397–1417.

[26] K.C. Furman and P.A. Ioannis, *A novel MINLP-based representation of the original complex model for predicting gasoline emissions*, Computers and Chemical Engineering **32** (2008), no. 12, 2857–2876.

[27] C.A. Haverly, *Studies of the behavior of recursion for the pooling problem*, ACM SIGMAP Bulletin **25** (1978), 19–28.

[28] J.N. Hooker and G. Ottosson, *Logic-based benders decomposition*, Mathematical Programming **96** (2003), 33–60.

[29] J. Jezowski, *Review and analysis of approaches for designing optimum industrial water networks*, Computers and Chemical Engineering **29** (2008), 663–681.

[30] X. Li, E. Armagan, A. Tomasgard, and P.I. Barton, *Stochastic pooling problem for natural gas production network design and operation under uncertainty*, AIChE Journal **57** (2011), no. 8, 2120–2135.

[31] L. Liberti and C.C. Pantelides, *An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms*, Journal of Global Optimization **36** (2006), no. 2, 161–189.

[32] W. B. Liu and C.A. Floudas, *Generalized primal-relaxed dual approach for global optimization*, Journal of Optimization Theory and Applications **90** (1996), no. 2, 417–434.

[33] W.B. Liu and C.A. Floudas, *Convergence of the (GOP) algorithm for a large class of smooth optimization problems*, Journal of Global Optimization **6** (1995), no. 2, 207–211.

[34] G.P. McCormick, *Computability of global solutions to factorable nonconvex programs : Part I – convex underestimating problems*, Mathematical Programming **10** (1976), no. 1, 147–175.

[35] C.A. Meyer and C.A. Floudas, *Global optimization of a combinatorially complex generalized pooling problem*, AIChE Journal **52** (2006), no. 3, 1027–1037.

[36] R. Misener and C. A. Floudas, *Global optimization of large-scale generalized pooling problems : Quadratically constrained minlp models*, Industrial & Engineering Chemistry Research **49** (2010), no. 11, 5424–5438.

[37] R. Misener and C.A. Floudas, *Advances for the pooling problem : modeling, global optimization, and computational studies (survey)*, Applied and Computational Mathematics **8** (2009), no. 1, 3–22.

[38] R. Misener, C.E. Gounaris, and C.A. Floudas, *Mathematical modeling and global optimization of large-scale extended pooling problems with the (EPA) complex emissions constraints*, Computers and Chemical Engineering **34** (2010), no. 9, 1432–1456.

[39] S. Mokhatab, W.A. Poe, and J.G. Speight, *Handbook of natural gas transmission and processing*, Chemical, Petrochemical & Process, Gulf Professional Pub., 2006.

[40] V. Pham, C. Laird, and M. El-Halwagi, *Convex hull discretization approach to the global optimization of pooling problems*, Industrial and Engineering Chemistry Research **48** (2009), no. 4, 1973–1979.

[41] A. Qualizza, P. Belotti, and F. Margot, *Linear programming relaxations of quadratically constrained quadratic programs*, Mixed Integer Nonlinear Programming (J. Lee and S. Leyffer, eds.), The IMA Volumes in Mathematics and its Applications, vol. 154, Springer New York, 2012, pp. 407–426.

[42] M. Ruiz, O. Briant, J.M. Clochard, and B. Penz, *Large-scale standard pooling problems with constrained pools and fixed demands*, Journal of Global Optimization (2012), 1–18.

[43] N.V. Sahinidis, *Baron : A general purpose global optimization software package*, Journal of Global Optimization **8** (1996), no. 2, 201–205.

[44] N.V. Sahinidis and M. Tawarmalani, *Accelerating branch-and-bound through a modeling language construct for relaxation-specific constraints*, Journal of Global Optimization **32** (2005), no. 2, 259–280.

[45] H. Sherali, E. Dalkiran, and L. Liberti, *Reduced rlt representations for nonconvex polynomial programming problems*, Journal of Global Optimization **52** (2012), 447–469.

[46] H.D. Sherali and C.H. Tuncbilek, *New reformulation linearization/convexification relaxations for univariate and multivariate polynomial programming problems*, Operations Research Letters **21** (1997), no. 1, 1–9.

[47] H. Tuy, *D(C)-optimization and robust global optimization*, Journal of Global Optimization **47** (2010), no. 3, 485–501.

[48] A. Waechter, *An interior point algorithm for large-scale nonlinear optimization with applications in process engineering*, Master's thesis, Carnegie Mellon University, 2002.

[49] A. Waechter and L.T. Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Mathematical Programming **106** (2006), 25–57.