Programmation Mathématique en variables entières: théorie et applications

Dominique Quadri

UNIVERSITÉ PARIS-SUD

HABILITATION À DIRIGER DES RECHERCHES

présentée par

Dominique QUADRI

Le 6 Juillet 2015

PROGRAMMATION MATHÉMATIQUE EN VARIABLES ENTIÈRES : THÉORIE ET APPLICATIONS

JURY

Rapporteurs :	Sourour ELLOUMI	MdC (HDR) à l'ENSIIE Paris, France
	Silvano MARTELLO	Pr. à l'Université de Bologne, Italie
	Frédéric ROUPIN	Pr. à l'Université Paris XIII, France
Examinateurs :	Alain DENISE	Pr. à l'Université Paris-Sud, France
	Dominique FEILLET	Pr. à l'Ecole des Mines de St Etienne (Site G. Charpak), France
	Steven MARTIN	Pr. à l'Université Paris-Sud, France

Table des matières

CHAPITRE 1

Introduction

Mes activités de recherche portent, d'une part, sur le développement de nouvelles formulations et techniques de résolution adaptées à des programmes mathématiques non linéaires en variables entières. D'autre part, elles concernent le traitement d'applications au moyen de la programmation mathématique en variables entières.

Plus précisemment, j'ai commencé par étudier, durant ma thèse de doctorat, un problème particulier de recherche opérationnelle, modélisé sous la forme d'un programme quadratique en variables entières : le problème du multi-sac-à-dos quadratique en variables entières noté $(QMKP)$. Ce dernier consiste en la maximisation d'une fonction quadratique concave en variables entières sous contraintes de type "sac à dos" (c'est à dire que les coefficients des variables dans les contraintes sont positifs ou nuls et les contraintes sont de la forme $\leq$). Il s'agit d'une des classes de problèmes au moins aussi difficiles que le problème classique de multi-sac-à-dos, pour lequel les méthodes de résolution pratiquement efficaces ne sont pas légion.

Le terme de "résolution en pratique" est la pierre angulaire de mes travaux de recherche réalisés et à venir. En effet, j'ai pris soin de porter une attention particulière aux techniques de résolution proposées dans un contexte théorique, de telle sorte à ce qu'elles donnent la possibilité par la suite de traiter en pratique des problèmes issus du monde réel. Des applications concernant les programmes quadratiques en variables entières seront exposées dans ce chapitre d'introduction.

Toutefois, pour le moment, en raison bien souvent parce que le cahier des charges de l'étude précisait que l'utilisation d'un *solveur* de programmation linéaire était attendu ou enfin parce que la structure du problème traité était fortement non linéaire (c'est à dire autre que quadratique, par exemple logarithmique, fractionnaire et logarithmique, exponentielle), les formulations mathématiques non linéaires et les méthodes de résolution exactes développées dans un contexte théorique n'ont pu être utilisées. Néanmoins, des formulations, sans doute moins précises (les formulations d'un même problème étant nombreuses mais inégales justement en terme d'efficacité de méthode de résolution en découlant) , linéaires en nombres entiers (ou mixtes) ont été établies ainsi que dans la majorité des cas, des heuristiques fournissant des solutions approchées de très bonne qualité (et ce en raison de la taille des instances réelles à traiter). Ainsi, je souhaiterais poursuivre cet effort, et à l'avenir développer des méthodes efficaces le plus possible en pratique.

Lors de mon intégration au Laboratoire d'Informatique d'Avignon, de l'université d'Avignon (où j'ai été maître de conférences de septembre 2007 à août 2013), j'ai développé de nouveaux travaux théoriques mais j'ai également étendu ces derniers à un contexte appliqué (combat contre la dengue, détection de cibles intelligentes, transport de personnes handicapées, e-commerce). J'ai poursuivi cette démarche à mon arrivée au Laboratoire de Recherche en Informatique de l'université Paris Sud XI en septembre 2013 (problèmes bi-niveaux stochastiques avec contraintes d'équilibres pour un problème de réseau de transport, efficacité

énergétique dans les réseaux hétérogènes, optimisation énergétique dans les réseaux sans fil multi-sauts (par agrégation de trafic et codage réseau)). Notons de surcroît que certaines recherches théoriques, nous le verrons dans les chapitres 4 et 5, proviennent de constats faits lors du traitement de certaines applications nécessitant d'aller plus loin dans la proposition de formulations et de méthodes de résolution que celles fournies par l'état de l'art concernant la programmation mathématique en variables entières.

1.1 CONTEXTE DE RECHERCHE

Depuis de nombreuses années la programmation mathématique s'est révélée être un outil précieux pour modéliser et résoudre de nombreux problèmes de la recherche opérationnelle, et plus généralement des problèmes issus du monde réel (transport de marchandises, de personnes, dimensionnement de réseaux) . La programmation mathématique constitue donc un moyen de formulation de problèmes réels mais aussi dans un second temps, guide la recherche et l'élaboration de méthodes d'optimisation combinatoire dédiées.

Ce travail s'inscrit dans ce cadre et concerne plus particulièrrement la programmation quadratique convexe en variables entières et le traitement d'applications issues du monde réel par la programmation linéaire en nombres entiers dans un premier temps, par la programmation non linéaire dans un second temps.
Pourquoi utiliser la programmation linéaire dans un premier temps ? La programmation linéaire en nombres entiers est la plus connue et la plus utilisée des formes de programmes mathématiques. En effet, aujourd'hui, même si les programmes linéaires en nombres entiers sont dits NP-difficiles, les *solveurs* d'optimisation linéaire sont extrêmement efficaces (en fournissant la solution optimale en un temps CPU très raisonnable, et ce pour des problèmes de grande taille). La programmation linéaire permet donc d'obtenir rapidement une solution, même si bien souvent nous constaterons qu'il est difficile de résoudre à l'optimum en pratique (en raison de la taille des instances).

Néanmoins, dans un contexte réel non seulement la formulation la plus adéquate nécessite l'introduction de termes non linéaires dans la fonction objectif et/ou dans les contraintes mais également la taille du programme mathématique résultant dépasse largement celle autorisée par les méthodes de résolution ou les *solveurs* actuellement à disposition. Bien entendu, afin de traiter au mieux les applications issues du monde réel, qui demandent bien souvent des modélisations très fines, il est nécessaire de développer, en amont de nouvelles approches dans un cadre théorique.

Les recherches présentées dans ce manuscrit suivent le raisonnement explicité précédemment. En effet, des résultats théoriques pour la programmation mathématique non linéaire en variables entières sont proposés d'une part. D'autre part, des modèles sous forme de programmes mathématiques et des techniques de résolution exactes ou approchées sont établis.

Enfin, comme nous l'avons déjà mentionné précédemment, de nombreuses applications requièrent une formulation fortement non linéaire pour laquelle les variables de décision doivent être mixtes (c'est à dire continues et entières) ou entières. Ce qui a pour conséquence d'aboutir à traiter un problème non seulement fortement non linéaire mais de surcroît non convexe. Ce cadre de travail constituera les perspectives de cette thèse d'habilitation.

1.2 PROBLÉMATIQUE

Dans cette section, nous présentons les différentes formulations des problèmes que nous étudions dans cette thèse puis nous établissons la complexité de ces derniers. Enfin, nous exposons l'intérêt de traiter certains problèmes théoriquement en amont.

Formulations et complexités des problèmes traités

Une fonction quadratique entière $f : \mathbb{N}^n \to \mathbb{R}$ est définie par $f(x) = \sum_{i=1}^{n} c_i x_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} q_{ik} x_i x_k$, où les coefficients c_i et q_{ik} sont tous des entiers ou encore des réels positifs ou nuls. La variable $x = (x_1, ..., x_i, ..., x_n)$ est à composantes entières.
Le problème du multi-sac-à-dos quadratique en variables entières convexe $(QMKP)$ consiste à maximiser une fonction quadratique entière à coefficients positifs ou nuls soumise à m contraintes de capacité linéaires. Ce problème s'énonce de la manière suivante :

$$(QMKP) \begin{cases} \max \ f(x) = \sum_{i=1}^{n} c_i x_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} q_{ik} x_i x_k \\ s.c \left| \begin{array}{lll} \sum_{i=1}^{n} a_{ji} x_i \leq & b_j & j = (1, 2, ..., m) \\ 0 \leq x_i \leq u_i & x_i \ entiers & i = (1, 2, ..., n) \end{array} \right. \end{cases}$$

où les coefficients $c_i \geq 0, q_{ik} \geq 0, a_{ji} \geq 0, b_j \geq 0$ sont tous des entiers ou des réels positifs ou nuls et les bornes $u_i \geq 0$ des variables x_i sont des entiers. De plus, les coefficients b_j sont tels que $0 < b_j < \sum_{i=1}^{n} a_{ji} u_i$ (sinon la contrainte est inutile).

Ainsi, le problème $(QMKP)$ est un **multi-sac-à-dos quadratique convexe en variables entières**.

Le problème $(QMKP)$ peut également se présenter sous la forme matricielle suivante :

$$(QMKP) \begin{cases} \max \ f(x) = c^t x - \frac{1}{2} x^t Q x \\ s.c \left| \begin{array}{ll} Ax \leq b & \\ 0 \leq x \leq u & x \ entier \end{array} \right. \end{cases}$$

où

- Le vecteur c est de dimension n et ses coefficients c_i sont positifs ou nuls.
- La matrice Q est symétrique (sans perte de généralité) semi-définie positive de dimension (n, n) (i.e. la fonction objectif est concave).
- La matrice des contraintes A est de dimension (m, n) et ses coefficients a_{ji} sont positifs ou nuls. On appelle ces contraintes des *contraintes de capacité*.
- Le vecteur b, de dimension m, est à coordonnées b_j positives ou nulles. Ce vecteur est appelé *membre de droite des contraintes*.
- Le vecteur x est de dimension n et ses coordonnées x_i sont entières et bornées par un entier u_i $(i = 1, .., n)$.

Remarque 1.1 *Selon le cadre d'étude nous utiliserons l'une ou l'autre des deux formulations présentées du problème* $(QMKP)$.

Remarque 1.2 *Le problème* $(QMKP)$ *est habituellement présenté comme un problème de maximisation, nous avons adopté cette formulation. Bien entendu, nous pouvons nous ramener à un problème de minimisation, en minimisant l'opposée de la fonction économique. Ces deux problèmes sont alors équivalents.*

Remarque 1.3 *Sans perte de généralité nous imposons aux variables* x_i *d'être telles que* $0 \leq x_i \leq u_i$. *Si les contraintes initiales sont du type* $l_i \leq x_i \leq u_i$, *il est aisé de se ramener au cas* $0 \leq x_i \leq u_i$ *à l'aide d'un changement de variables.*

Le problème $(QMKP)$ est donc un multi-sac-à-dos quadratique convexe en variables entières pour lequel on distingue de nombreux cas particuliers dont, par exemple, le multi-sac-à-dos quadratique non convexe en variables bivalentes (0-1), le multi-sac-à-dos quadratique convexe séparable en variables entières, le sac-à-dos quadratique en variables entières, etc.

Le multi-sac-à-dos quadratique convexe **séparable** en nombres entiers. Ce problème consiste à maximiser une fonction économique quadratique entière séparable c'est-à-dire que si l'on considère la forme matricielle du problème $(QMKP)$, cette propriété de la fonction économique se traduit par le fait que la matrice Q est diagonale, nous la notons alors D.

La formulation du **multi-sac-à-dos quadratique convexe séparable** est la suivante :

$$(QMKP) \left\{ \begin{array}{l|lll} \max\ f(x) = \sum_{i=1}^{n}(c_i x_i - d_i x_i^2) & & \\ s.c. & \sum_{i=1}^{n} a_{ji} x_i \leq & b_j & j = (1,2,...,m) \\ & 0 \leq x_i \leq u_i & x_i\ entiers & i = (1,2,...,n) \end{array} \right.$$

où $c_i \geq 0,\ d_i \geq 0, a_{ji} \geq 0, b_j \geq 0, u_i \leq (c_i/2d_i)$.

Remarque 1.4 *Nous imposons à toutes les bornes supérieures des variables* x_i *d'être telles que :* $u_i \leq (c_i/2d_i)$ *parce que qu'il n'existera pas de solution optimale pour laquelle* $x_i \geq \frac{c_i}{2d_i}$ *du fait que la fonction économique soit séparable.*

Le **sac-à-dos quadratique non convexe en variables bivalentes (0-1)** habituellement noté (QKP), se formule de la manière suivante :

$$(QKP) \left\{ \begin{array}{l|l} \max \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i x_j & \\ s.c. & \sum_{i=1}^{n} a_i x_i \leq b \\ & x_i \in \{0,1\}\ i = 1,...,n \end{array} \right.$$

où q_{ij} $(i = 1,...,n-1,\ j = i+1,...,n)$, c_i, a_i $(i = 1,...,n)$ et b sont des réels positifs. De plus, on suppose que $\forall\ i < j$, $q_{ji} = q_{ij}$.

Le problème du multi-sac-à-dos quadratique convexe en variables entières est **NP-difficile au sens fort**. En effet, lorsque tous les coefficients q_{ik} sont nuls et que $m = 1$, $(QMKP)$ correspond au sac-à-dos linéaire entier, bien connu, qui est un problème NP-complet. Ce résultat concernant le sac-à-dos classique est montré en 1975 par Lueker dans [1]. De plus, lorsque $m = 1$ et quand toutes les variables x_i sont bivalentes (0-1), le problème résultant est alors un sac-à-dos quadratique en variables bivalentes dont l'appartenance à la classe des problèmes NP-difficiles au sens fort est montrée par Chaillou, Hansen et Mahieu [2]. Or ces problèmes constituent des sous-problèmes de $(QMKP)$ donc $(QMKP)$ est également un problème **NP-difficile au sens fort**.

Par ailleurs, la partie "Applications" de ce manuscrit utilise en grande partie des **programmes linéaires en variables entières soumis à des contraintes linéaires**, dont la version matricielle est la suivante :

$$(PLNE) \begin{cases} \max\ f(x) = c^t x \\ s.c \left| \begin{array}{ll} Ax \leq b \\ 0 \leq x \leq u & x\ entier \end{array} \right. \end{cases}$$

où

- Le vecteur c est de dimension n et ses coefficients c_i sont positifs ou nuls.
- La matrice des contraintes A est de dimension (m, n) et ses coefficients a_{ji} sont positifs ou nuls. On appelle ces contraintes des *contraintes de capacité*.
- Le vecteur b, de dimension m, est à coordonnées b_j positives ou nulles. Ce vecteur est appelé *membre de droite des contraintes*.
- Le vecteur x est de dimension n et ses coordonnées x_i sont entières et bornées par un entier u_i $(i = 1, .., n)$.

Remarque 1.5 *Le programme $(PLNE)$ établi ci dessus est en fait un multi-sac-à-dos linéraire en variables entières (MKP). Bien entendu, c'est un cas particulier de programmes linéaires en nombres entiers. En effet, les contraintes ne sont pas nécessairement des contraintes de capacités. Néanmoins, il est bien souvent possible de ré-écrire des contraintes non linéaires initialement (logiques par exemple) sous la forme de contraintes linéaires de type $\leq$.*

Le problème du multi-sac-à-dos linéaire en variables entières est **NP-difficile au sens fort**. En effet, lorsque $m = 1$, (MKP) correspond au sac-à-dos linéaire entier (KP) (voir [3] pour un état de l'art), bien connu, qui est un problème NP-complet. Ce résultat concernant le sac-à-dos classique est montré en 1975 par Lueker dans [1]. Ainsi, (PQE) et (MPK) sont des généralisations de (KP) donc (PQE) et (MKP) sont **NP-complets**.

Enfin, la forme d'un programme mathématique en nombres entiers la plus générale est la suivante :

$$(PNLNE) \begin{cases} \max\ f(x) \\ s.c \left| \begin{array}{ll} g(x) \leq b \\ 0 \leq x \leq u & x\ entier \end{array} \right. \end{cases}$$

où

– $f(x)$ est une fonction non linéaire, concave ou non concave.

– $g(x)$ est une fonction non linéaire, concave ou convexe ou quelconque.

– Le vecteur de décision a ses composantes entières, mais le programme peut être en variables mixtes selon les applications traitées.

On est alors face à un **programme non linéaire non convexe (ou convexe, selon la nature de f et de l'ensemble des solutions admissibles) en variables entières** (ou mixtes).

Notations et conventions :

– La notation $(QMKP)$ représente un **multi-sac-à-dos quadratique convexe en variables entières** sans aucune hypothèse concernant la fonction objectif. Si une hypothèse est faite sur cette fonction c'est-à-dire si par exemple, elle est **séparable**, nous notons toujours le multi-sac-à-dos quadratique convexe en variables entières $(QMKP)$ mais nous précisons qu'il est, de plus, **séparable**.

– La notation (QKP) représente un **sac-à-dos quadratique non convexe en variables 0-1**.

– La notation $(PLNE)$ représente un **programme linéaire en variables entières**.

– La notation $(PL01)$ représente un **programme linéaire en variables 0-1**.

– Soit (P) un programme en variables entières, nous notons $(\overline{P})$ la relaxation continue de (P) (i.e. $x_i \in [0, u_i]$).

– La valeur optimale du problème (P) est notée $Z[P]$ et la valeur optimale du problème relâché continûment $(\overline{P})$ est notée $Z[\overline{P}]$.

Ces notations et conventions sont valables pour tout le rapport.

Intérêt des problèmes traités dans cette thèse

La programmation mathématique, comme nous l'avons mentionné dès le début de ce chapitre permet de traiter, à l'aide de sa formulation, de nombreux problèmes de recherche opérationnelle issus du monde réel. L'intérêt de son utilisation prend tout son sens à la lecture de la partie 2 de ce manuscrit. De surcroît les développements théoriques en lien aux programmes quadratiques en nombres entiers sont très utiles du fait donc des nombreuses applications possibles.

Du fait de sa généralité, $(QMKP)$ offre une large gamme d'applications. Comme il est mentionné dans [4], [5, 6] et [7], $(QMKP)$ a en particulier de nombreuses applications en finance.

Nous citons dans cette partie deux applications pratiques de $(QMKP)$ appartenant à ce domaine.

Gestion d'un portefeuille d'actifs

Une application possible de $(QMKP)$ est un problème d'investissements. Ce problème se modélise de la manière suivante :

$$\begin{cases} \max\ f(x) = \langle x - B|R\rangle - \lambda^t (x-B)^t W (x-B) \\ s.c \left| \begin{array}{l} Ax \leq b \\ 0 \leq x \leq u \quad x\ entier \end{array} \right. \end{cases}$$

où

- L'opération $\langle \cdot | \cdot \rangle$ représente le produit scalaire.
- x est un vecteur d'entiers de dimension n, c'est l'inconnue du problème.
- B est un vecteur de dimension n. Il représente l'allocation *Benchmark* fixée par le marché. C'est-à-dire que si l'achat d'un actif est envisagé, le client doit acheter une quantité minimum fixée par le marché.
- A est une matrice (m, n) dont les coefficients sont positifs ou nuls. A est appelée *matrice des contraintes de budget*.
- b est un vecteur de dimension m à coordonnées entières, appelé *vecteur de ressources*.
- W est une matrice (n, n) dite des *variances-covariances* ;
- R est un vecteur de dimension n traduisant le *rendement des actifs*.
- u est un vecteur à coordonnées entières de dimension n constituant une borne supérieure de la variable x.
- λ est une constante positive, c'est le coefficient d'*aversion ou de goût pour le risque*.

Le problème d'investissements est alors le suivant : un client a le choix parmi n investissements possibles i sur m périodes données j. Ces investissements sont des produits financiers particuliers appelés *options*. L'achat de ces options s'effectue obligatoirement sur un nombre de périodes données, ici m. Les coefficients a_{ji} représentent le coût de l'actif i pour le période j pour un budget b_j fixé. Les variables x_i représentent le nombre de placements de l'actif i dans le portefeuille. On peut alors effectuer l'achat de l'actif i une ou plusieurs fois ou encore l'ignorer (d'où l'intérêt de travailler avec des variables entières). De plus, chaque investissement i rapporte c_i unités monétaires. Ainsi, le but est de maximiser le profit $\langle x - B|R\rangle$ tout en minimisant le risque représenté par l'expression $\lambda^t (x-B)^t W (x-B)$. Si les actifs sont indépendants (respectivement dépendants) alors la matrice W est diagonale (respectivement non diagonale) et le problème de sac-à-dos quadratique correspondant est séparable (respectivement non séparable). Dans le cas où les actifs sont indépendants, la matrice W n'est plus la matrice des variances-covariances mais uniquement la matrice des variances. Toutefois, le modèle le plus approprié et réaliste est celui pour lequel les actifs sont dépendants les uns des autres.
Le modèle ci-dessus est une généralisation du modèle de Markowitz [8]. En effet, le modèle de Markowitz est le programme quadratique ci-dessus, soumis à une seule contrainte de budget, pour lequel la contrainte d'intégrité des variables x_i est relâchée.

Un problème de décision d'un comité d'investissements

Nous exposons ici un problème voisin au problème de Markowitz qui est un problème de décision de comité d'investissements. Il s'agit, initialement, d'un problème **multi-sac-à-dos quadratique continu**. Il s'agit du modèle classique de Markowitz. Le vecteur de rendement R est obtenu par le calcul suivant : $R = (z_{score} \times \mu)$ où le z_{score} est un vecteur de dimension n. Le vecteur $\mu = (\mu_1, ..., \mu_i, ..., \mu_n)$ est fixé par le marché et représente le risque lié à l'investissement de chacun des actifs i.
Le problème relâché présenté précédemment est donc résolu à l'optimum continu une première fois. Notons $x^{(0)}$ la solution optimale obtenue.

Le comité d'investissements se réunit et change le poids affecté à chaque actif. Ce changement se répercute directement sur la valeur du vecteur z_{score}. Le problème est alors le suivant : quelle répercussion ce changement engendre-t-il sur l'allocation optimale du portefeuille ? C'est-à-dire : l'allocation du portefeuille obtenue lors de la première optimisation du problème est-elle toujours optimale ? Afin de répondre à cette question, on réoptimise le problème initial en prenant en compte la variation des poids affectés aux actifs. Si la solution optimale obtenue est différente de la solution initiale, elle sera validée si la différence est inférieure, en valeur absolue, à un seuil fixé par le comité d'investissements. En effet, le comité d'investissements fixe un seuil d'autorisation de mouvements. Par exemple, si dans la première solution optimale $x_i^{(0)} = 2$ et que sa nouvelle valeur après réoptimisation vaut $x_i^{(1)} = 2.08$ et que le seuil de mouvements est égal à 0.01 alors cette nouvelle solution ne sera pas acceptée. Pour modéliser ce critère, on introduit dans le modèle une autre variable de décision qui transforme le programme continu initial en un programme en nombres entiers.
La modélisation de ce problème se traduit par deux étapes que nous appelons **réoptimisation classique** du problème et **réoptimisation révisée**.

1. **Réoptimisation classique** : Il s'agit donc de résoudre le **multi-sac-à-dos quadratique continu** suivant :

$$\begin{cases} \max \ f(x) = \langle x - B|R\rangle - \lambda^t (x-B)^t W (x-B) \\ s.c \left| \begin{array}{ll} Ax \leq b \\ 0 \leq x \leq u & x \ continu \end{array} \right. \end{cases}$$

   Nous notons $x^{(1)}$ la solution optimale du problème.

2. **Réoptimisation révisée** : Le comité d'investissements fixe, en plus du z_{score}, un seuil "d'autorisation de mouvements" noté δ c'est-à-dire une possibilité, par exemple, d'augmenter de $\delta = 0.01$ la quantité d'actifs placée dans le portefeuille du client. Ainsi, la variable de décision correspondant au problème prenant en compte ce seuil d'autorisation de mouvements n'est plus x mais H, où $H = (h_1, .., h_i, .., h_n)$ est telle que $h_i = n_i \times \delta$, où $\forall i \in \{1, .., n\},\ n_i \in \mathbb{N}$. La variable H est à composantes réelles mais les variables n_i sont entières, ce qui signifie que nous traitons, lors de cette étape appelée "réoptimisation révisée", un problème de **multi-sac-à-dos quadratique en variables entières** n_i.

1.3 ORGANISATION DU DOCUMENT

Cette thèse est organisée en deux parties, présentées ci-dessous. La première partie est composée de trois chapitres. La seconde en compte deux.

- **Partie I Résultats théoriques.** La première partie de ce manuscrit expose les résulats obtenus sur des travaux théoriques concernant la programmation quadratique en variables entières et 0-1. Plus précisemment, dans le **chapitre 2** nous proposons un tour d'horizon des méthodes de résolution exactes pour les programmes quadratiques en variables entières. Et ce, en fonction de la nature du problème (convexe ou non convexe), de la nature de la fonction objectif (séparable ou non séparable) et enfin de la nature des variables (entières ou 0-1). Cet état de l'art, présenté de façon succinte mais complète (à notre connaissance), permet de situer les travaux théoriques de cette premère partie. Le **chapitre 3** est consacré au problème du multi-sac-à-dos quadratique en variables entières $(QMKP)$ pour lequel la fonction objectif est concave et non séparable. Une méthode exacte est développée. Celle ci

est basée sur une reformulation équivalente de $(QMKP)$ en un problème séparable qui est lui-même, par la suite, linéarisé. Le **chapitre 4** traite le cas particulier du sac-à-dos quadratique en variables 0-1 (pour lequel la fonction économique est non séparable et le problème non convexe). Une méthode de résolution exacte est suggérée s'appuyant sur une linéarisation des termes quadratiques de la fonction objectif. Cette linéarisation ne nécessite que l'ajout d'une unique variable de décision ainsi que de contraintes linéaires (en nombre conséquent).

- **Partie II Applications.** Cette seconde partie est dédiée au traitement d'applications, issues du monde réel, par la programmation mathématique. Les problèmes traités sont regroupés en chapitre, selon la modélisation proposée (sous forme de programmes linéaires en nombres entiers ou programmes bi-niveaux et non linéaires). Dans le **chapitre 5** les problèmes sont modélisés sous forme d'un programme linéaire en nombres entiers puis une solution approchée est fournie par une heuristique adaptée aux applications considérées. Les applications présentées dans ce chapitre sont les suivantes : combat contre la dengue et détection d'une cible intelligente. Le **chapitre 6** traite les applications suivantes : un problème de tarification optimale d'un service de livraison et un problème d'efficacité énergétique dans les réseaux hétérogènes. La première est modélisée au moyen d'un programme bi-niveaux stochastique avec contraintes d'équilibre (MPSEC). La seconde est formulée au moyen d'un programme non linéraire non convexe.

- **Conclusion et perspectives**
  Dans le dernier chapitre, nous concluons sur le travail réalisé dans cette thèse et présentons plusieurs perspectives en lien avec les travaux développés.

Première partie

Résultats théoriques

Organisation de la partie I

Cette première partie est consacrée à l'exposé des résultats théoriques obtenus sur des programmes quadratiques en variables entières ou 0-1. Cette dernière est constituée de trois chapitres. Le **chapitre 2** propose un état de l'art succinct concernant les différrentes formulations de programmes quadratiques en variables entières, permettant ainsi de situer les travaux théoriques proposés dans ce manuscrit. Le **chapitre 3** traite de reformulations et techniques de résolution pour le problème du multi-sac-à-dos quadratique en variables entières $(QMKP)$ pour lequel la fonction objectif est non séparable. Le **chapitre 4** concerne le problème du sac-à-dos quadratique en variables 0-1 (QKP) pour lequel une linéarisation est proposée.

CHAPITRE 2

Un état de l'art sur la PQE

Nous présentons dans ce chapitre un état de l'art succint (mais complet à notre connaissance), concernant les différrentes formulations de programmes quadratiques en variables entières afin de positionner les recherches présentées dans les chapitres 3 et 4.

2.1 INTRODUCTION

Nous considérons dans ce chapitre la classe de problèmes quadratiques en variables entières, soumis à des contraintes linéaires, dont la formulation générale est la suivante :

$$(PQE) \begin{cases} \max\ f(x) = c^t x - x^t Q x \\ s.c. \left| \begin{array}{l} Ax \leq b \\ 0 \leq x \leq u \\ x \in \mathbb{N}^n \end{array} \right. \end{cases} \tag{2.1.1}$$

où $\mathbb{N}$ est l'ensemble des entiers naturels, $c = (c_1, ..., c_i, ..., c_n) \in \mathbb{N}^n$, $Q = (q_{ik})_{1 \leq j < k \leq n}$ est une matrice $n \times n$ d'entiers et symétrique, sans perte de généralité (si de plus elle est semi-définie positive alors (PQE) est convexe), $A = (a_{ji})_{1 \leq j \leq m, 1 \leq i \leq n}$ est une matrice $m \times n$ d'entiers, $b = (b_1, ..., b_j, ..., b_m) \in \mathbb{N}^m$ et $u = (u_1, ..., u_i, ..., u_n) \in \mathbb{N}^n$.

Les programmes quadratiques en nombres entiers ont beaucoup été étudié ces vingt dernières années. Ces programmes peuvent être classés selon les cinq catégories suivantes en lien avec la nature de la fonction objectif (non séparable ou séparable et convexe et/ou non convexe) et avec la nature des variables (purement entières ou 0-1) : (i) Le problème du sac-à-dos Quadratique 0-1 (QKP) qui consiste en la minimisation d'une fonction objectif non convexe non séparable soumise à des variables 0-1, (ii) Les programmes Quadratiques (PQE) soumis à une unique ou plusieurs contraintes de capacité et pour lequel la fonction objectif est concave et séparable avec des variables entières, (iii) Les programmes Quadratiques (PQE) soumis à une unique ou plusieurs contraintes de capacité et pour lequel la fonction objectif est concave non-séparable avec des variables entières, (iv) Les programmes Quadratiques (PQE) soumis à une unique ou plusieurs contraintes de capacité et pour lequel la fonction objectif est non concave séparable avec des variables entières, et (v) Les programmes Quadratiques (PQE) soumis à une unique ou plusieurs contraintes de capacité et pour lequel la fonction objectif est non concave non séparable avec des variables entières.
Chaque classe de programme quadratique en variables entières ou 0-1 trouve de nombreuses applications, notamment en finance [9], [7], allocation de ressources [10] et ordonnancement [11] et [12].
Nous présentons dans les sections suivantes l'état de l'art correspondant aux cinq classes de problèmes de façon succincte mais exhaustive, à notre connaissance .

2.2 SAC-À-DOS QUADRATIQUE 0-1 (QKP)

(QKP) est l'un des programmes quadratiques le plus connu et étudié. Pour résoudre (QKP) deux approches se distinguent (cf. l'état de l'art [13]) : Une approche dite quadratique qui vise à résoudre directement (QKP) (voir [14], [15], [16], [17], [18], [19]) utilisant des techniques de programmation semi-définie ou de relaxation par convexification et des approches par linéarisation (voir [20], [21], [22], [23], [24], [25]) consistant à transformer (QKP) en un programme linéaire équivalent. Les techniques de linéarisation peuvent être divisées en deux sous classes d'approches :

(i) linéarisation sans ajout de variables de décision mais avec un nombre exponentiel de contraintes (cf. [26], [27], [28] et [29] pour le cas où (QKP) est non contraint). Dans ce contexte, des procédures efficaces de

génération de contraintes ont été développées. Un problème de séparation est alors résolu afin de générer des contraintes violées. **C'est dans ce cadre que se situe le travail présenté lors du chapitre 4 ;**

(ii) linéarisation avec un nombre polynomial de variables de décision supplémentaires . L'une des toutes premières procédures, dite classique, proposée dans [30] consiste à remplacer chaque terme produit $x_i x_j$ avec une variable non négative z_{ij} et des contraintes supplémentaires du type $z_{ij} \leq x_i$, $z_{ij} \leq x_j$, $z_{ij} \geq x_i + x_j - 1$. Le schéma de linéarisation induit $O(n^2)$ variables et contraintes additionnelles. Même si de nombreuses améliorations ont été apportées pour le cas non contraint (linéarisation avec $O(n)$ nombre de variables et contraintes) [31], [32], [33], [23] et [20], le problème contraint reste difficile en raison du nombre de variable supplémentaires qui croît de manière significative par rapport au problème initial, voir [22], [17], [34].

2.3 (PQE) CONVEXE SÉPARABLE EN VARIABLES ENTIÈRES

Dans ce contexte, le problème a été étudié *via* des méthodes de résolution exactes et des reformulations (cf. [35], [36], [37] and [38]) ou en utilisant des heuristiques ([39] or [40]). Les méthodes exactes dans ce cadre sont de type *branch and bound*. Ces algorithmes sont basés sur le calcul de bornes inférieures et supérieures fines, de la valeur optimale. C'est le cas dans [35] où un majorant est obtenu par la relaxation continue du problème quadratique initial. Dans [36] un majorant est calculé *via* une reformulation linéraire du problème initial (PQE)). Une amélioration de ce travail est développée par Quadri et al. [37] permettant alors le traitement d'instances allant jusqu'à 2000 variables et contraintes. Bien entendu, les algorithmes de *branch and bound* sont bien souvent consommateurs en terme de temps CPU. Ainsi, des heuristiques sont suggérées afin de fournir des solutions approchées mais en des temps CPU très rapides. (PQE) soumis à une seule contrainte de sac à dos est considéré. Un algorithme d'approximation est développé dans [39], basé sur la programmation dynamique. Dans [40], l'heuristique est basée sur la résolution de la relaxation continue du problème initial fournissant un majorant de la valeur optimale du problème initial. La solution approchée est trouvée par approximation autour de la solution continue.

2.4 (PQE) CONVEXE NON SÉPARABLE EN VARIABLES ENTIÈRES

Cette classe de problèmes a reçu peu d'attention dans la littérature, bien qu'elle présente de nombreuses applications, telles que la gestion de portefeuilles d'actions [9], [7]. **Cette catégorie fait l'objet du chapitre 3**. Une transformation de la fonction objectif non séparable du problème initial, en un problème séparable est proposée par Djerdjour [41] utilisant la plus petite valeur propre de la matrice constituant les termes quadratiques du problème. Cette technique a été employée dans un cadre continu également (cf. Dassault et al. [42]).

2.5 (PQE) NON CONVEXE SÉPARABLE EN VARIABLES ENTIÈRES

La non convexité du problème induit une difficulté supplémentaire par rapport aux deux précédentes catégories : la relaxation continue du problème initial ne peut plus être calculée facilement. La littérature dédiée à ce cas est très peu fournie. Un exemple est donné par More and Vavasis [43], qui portent leur attention sur la recherche de solutions optimales locales au lieu de rechercher des solutions optimales globales. Des algorithmes de *branch and bound* ([44] et [45]) pour des problèmes d'optimisation non convexe, typiquement utilisent la notion d'enveloppe convexe qui sous-évalue la valeur de la fonction non convexe (voir [46]).

2.6 (PQE) NON CONVEXE NON SÉPARABLE EN VARIABLES ENTIÈRES

C'est le cadre le plus général qui n'a été que très récemment étudié. D'une part, les travaux portent sur l'aspect non convexe du problème. Une méthode de résolution générique basée sur une refomrulation quadratique convexe du problème a été développée par Billionnet et al. dans [47]. D'autre part, l'idée clé de l'autre méthode proposée concerne la nature non séparable de la fonction objectif du problème traité. Une approche de diagonalisation de la matrice est suggérée afin de calculer un majorant pour le problème non convexe de départ [48]. Enfin, des techniques de fixation de variables (à la valeur 0) sont établies dans [49] dans le but de réduire la taille du problème initial. Des conditions nécessaires et suffisantes pour identifier les termes dominés sont mentionnées.

Notons que la littérature mentionnée ci dessus, se réfère au contexte déterministe. C'est dans ce cadre que les travaux présentés dans ce manuscrit sont menés. Bien entendu, de nombreuses applications issues du monde réel nécessiteraient l'introduction de la prise en compte de l'incertitude ; qui se traduirait par une incertitude sur les coefficients de la fonction objectif, des contraintes et/ou du membre de droite des contraintes. Par exemple, nous pouvons citer ce travail [50] qui utilise la programmation quadratique stochastique.

CHAPITRE 3

Programme quadratique convexe en variables entières : reformulations et méthode de résolution exacte

Ce travail a été effectué en collaboration avec Eric Soutif (McF au CNAM jusqu'en septembre 2013, changement de nom en 2014 : Eric Soutil) lors de différentes visites au Laboratoire Informatique d'Avignon et au CEDRIC entre 2008 et 2012. Ce dernier a donné lieu à la publication de deux articles (ref. 2 et 8 de la section Liste des Publications). Cette étude a été financée en partie par le GDR RO.

Nous présentons dans ce chapitre une méthode visant à résoudre de manière exacte le problème $(QMKP)$ dont la fonction économique est concave et non séparable. Notre approche se déroule en deux étapes. La première repose sur une transformation du problème initial non séparable en un problème équivalent séparable $(QP_{x,y})$. Cette transformation est effectuée en utilisant la méthode de décomposition de Gauss appliquée à la matrice constituant le terme quadratique de la fonction objectif. Cette transformation nécessite l'ajout de variables de décision continues et de contraintes linéaires. Lors de la deuxième étape, le problème précédemment reformulé est linéarisé en approchant chaque terme carré par K fonctions linéaires correspondant à la tangente de l'hyperbole en K points de rupture. Le problème linéarisé est noté $(LP^K_{x,y})$. Nous établissons la preuve de la propriété intuitive suivante : lorsque K est très grand, alors la valeur optimale du problème linéarisé $(LP^K_{x,y})$ est extrêmement proche (voir égale) à la valeur optimale de $(QP_{x,y})$ (elle même égale à la valeur optimale de $(QMKP)$). Le reste du chapitre est dédié à l'implémentation d'un algorithme de *branch-and-bound* pour traiter le problème linéarisé $(LP^K_{x,y})$.
Il est important de souligner, qu'à notre connaissance, parmi les rares résultats concernant la résolution exacte du problème $(QMKP)$ non séparable, aucune approche ne propose la résolution de ce problème à l'aide d'un problème équivalent. Djerdjour [41], par exemple, propose un algorithme de résolution exacte basé sur le calcul d'un majorant. Ce dernier est obtenu par la résolution d'un problème de multi-sac-à-dos quadratique séparable en variables entières. L'auteur ne reporte aucun résultat numérique.

Nous consacrons la section 3.1 à l'énoncé du problème non séparable que nous traitons. La section suivante est dédiée à la description de la transformation de (QMKP) en un problème séparable. De plus, nous établissons une comparaison théorique de notre approche avec celle proposée par Djerdjour [41]. La section 3.3 détaille la technique de linéarisation qui fournit de surcroît une méthode exacte de résolution pour (QMKP). La section 3.4 expose l'implémentation de notre algorithme exact. La section 3.5 concerne les résultats numériques. Enfin nous concluons lors de la section 3.6.

3.1 INTRODUCTION

Nous traitons dans ce chapitre le problème de **multi-sac-à-dos quadratique convexe non séparable en variables entières** que nous notons $(QMKP)$ et qui se présente sous la forme matricielle suivante :

$$(QMKP) \begin{cases} \max\ f(x) = c^t x - x^t Q x \\ s.c \left| \begin{array}{l} Ax \leq b \\ 0 \leq x \leq u \quad x\ entier \end{array} \right. \end{cases}$$

où

– Le vecteur c est de dimension n et ses coefficients c_i sont positifs ou nuls.

– La matrice Q est *symétrique* définie positive de dimension (n, n).

– La matrice des contraintes A est de dimension (m, n) et ses coefficients a_{ji} sont positifs ou nuls. On appelle ces contraintes des *contraintes de capacité*.

– Le vecteur *second membre* b, de dimension m, est à coordonnées, b_j, positives ou nulles.

– Le vecteur x est de dimension n et ses coordonnées x_i sont entières et bornées supérieurement par des entiers u_i $(i = 1, .., n)$. On note X_i l'ensemble de valeurs entières que peut prendre chaque variable x_i et $|X_i|$ le cardinal de cet ensemble.

Compte tenu des résultats encourageants obtenus lors de ma thèse de doctorat concernant le problème du multi-sac-à-dos quadratique concave séparable en variables entières, il semble naturel d'adopter l'idée de notre approche dans un contexte où la fonction économique est non séparable. Ces résultats montraient qu'il est très efficace de traiter la formulation linéarisée équivalente au problème quadratique initial. Cette approche, qui ne pourra pas être suivie dans son intégralité était basée sur la linéarisation par morceaux du problème initial séparable. La linéarisation par morceaux ne peut être appliquée dans le contexte de ce chapitre parce que la transformation du non séparable vers le cas séparable nécessite l'introduction de variables réelles supplémentaires. En conséquence, il n'est plus possible d'utiliser le développement direct des variables entières visant à ré-écrire les variables entières en variables binaires. Développement des variables entières, qui constitue l'un des points clés de la linéarisation par morceaux. Toutefois, une autre linéarisation est proposée.
Pour ce faire, nous proposons de transformer le problème $(QMKP)$ non séparable en un problème séparable équivalent. Cette transformation est fréquemment envisagée en algèbre linéaire du fait que les formes quadratiques diagonales sont plus simples à analyser qu'une forme quadratique comportant des termes croisés.
Il existe, donc des méthodes de transformation bien connues qui transforment une forme quadratique quelconque en une forme quadratique dont la matrice représentative est diagonale. Bien entendu, nous ne pouvons pas appliquer n'importe quel changement de coordonnées à notre problème $(QMKP)$. En effet, notre forme quadratique initiale à matrice non diagonale admet des propriétés particulières que nous souhaitons conserver. D'autre part, cette forme quadratique est incluse dans un problème bien particulier dont nous devons tenir compte. C'est pourquoi, nous avons testé trois transformations (les deux premières sont bien connues, la troisième est adaptée à notre problème initial) du problème initial en un problème connu i.e. séparable.

Nous présentons, ci-dessous, les trois possibilités que nous avons envisagées pour transformer notre problème $(QMKP)$ non séparable en un problème séparable :

1. **Diagonaliser la matrice Q** de telle sorte que $Q = PDP^{-1}$ où D est une matrice diagonale constituée des **valeurs propres** de la matrice Q et P est une matrice de passage (c'est-à-dire constituée des vecteurs propres associés aux valeurs propres de Q) orthogonale (c'est-à-dire telle que $P^t = P^{-1}$ et telle que les vecteurs propres soient orthonormaux). Cette diagonalisation est réalisable en raison de l'hypothèse de symétrie de la matrice Q. Nous proposons alors d'appliquer le **changement de variables** $\mathbf{y = Px \Longleftrightarrow x = P^{-1}y}$. Une fois le changement de variables appliqué, nous traitons un problème séparable.
2. **Rendre diagonale la matrice Q** à l'aide de la **décomposition en carrés de Gauss** (méthode analytique) encore appelée décomposition en matrice diagonale à l'aide des **matrices d'éliminations de Gauss** (méthode matricielle). Supposons que la matrice Q soit symétrique et carrée de dimension n. Q est remplacée par le produit matriciel suivant : $E_1^{-1}E_2^{-1}...E_{n-1}^{-1}D(E_{n-1}^t)^{-1}...(E_2^t)^{-1}(E_1^t)^{-1}$ où les matrices E_i $\forall i = 1, ..., n-1$ sont les matrices d'éliminations de Gauss et la matrice D est une matrice diagonale dont les éléments diagonaux sont les pivots de Gauss obtenus lors de la

transformation. Le changement de variables serait le suivant : $\mathbf{y} = (\mathbf{E^t_{n-1}})^{-1}...(\mathbf{E^t_2})^{-1}(\mathbf{E^t_1})^{-1}\mathbf{x} \Longleftrightarrow \mathbf{x} = \mathbf{E^t_1}\ \mathbf{E^t_2}\ ...\ \mathbf{E^t_{n-1}}\ \mathbf{y}$. Le problème ainsi obtenu est un problème séparable.

3. La dernière transformation, que nous avons retenue en définitive, consiste à appliquer une méthode de **décomposition de Gauss à la matrice** Q mais en effectuant un **changement de variables partiel**, dérivé de celui évoqué dans la deuxième possibilité. Nous conservons les n variables initiales x et nous introduisons n variables supplémentaires y ainsi qu'au pire des cas n contraintes. Nous détaillons cette méthode dans la sous-section 3.2.2.

Notre choix s'est donc porté sur la dernière transformation. Nous justifions dans la section suivante notre choix.

3.2 DU NON SÉPARABLE AU SÉPARABLE

Cette section est dédiée à la première étape de notre approche. Nous convertissons le problème non séparable en un problème séparable. Comme nous l'avons mentionné dans l'introduction de ce chapitre, une technique classique consiste à diagonaliser la matrice Q en utilisant les valeurs propres de Q (voir [41]). Nous établissons dans la sous-section 3.2.1 les principaux inconvénients quant à l'utilisation de cette technique dans notre contexte. Pour ces raisons, nous proposons l'utilisation d'une décomposition de Gauss de la matrice Q dans la sous-section 3.2.2.

Afin de rendre la lecture de ce chapitre plus aisée, nous proposons d'appliquer les transformations 1 et 3 sur un exemple simple en deux dimensions. Le programme quadratique $(QMKP_{ex})$ correspondant est le suivant :

$$(QMKP_{ex}) \begin{cases} \max\ f(x) = 69x_1 + 71x_2 - (15x_1^2 + 2x_1x_2 + 17x_2^2) \\ s.c. \left| \begin{array}{l} 81x_1 + 50x_2 \leq 61 \\ 17x_1 + 2x_2 \leq 105 \\ 0 \leq x_1 \leq 3 \\ 0 \leq x_2 \leq 2 \\ x_1, x_2\ entiers \end{array} \right. \end{cases} \tag{3.2.1}$$

La formulation matricielle du problème est la suivante :

$$(QMKP_{ex}) \begin{cases} \max\ f(x) = c_{ex}^t x - x^t Q_{ex} x \\ s.c. \left| \begin{array}{l} A_{ex}x \leq b_{ex} \\ 0 \leq x \leq u_{ex} \\ x\ entier \end{array} \right. \end{cases} \tag{3.2.2}$$

avec
$Q_{ex} = \begin{pmatrix} 15 & 1 \\ 1 & 17 \end{pmatrix}$, $c_{ex} = \begin{pmatrix} 69 \\ 71 \end{pmatrix}$, $A_{ex} = \begin{pmatrix} 81 & 50 \\ 17 & 2 \end{pmatrix}$, $b_{ex} = \begin{pmatrix} 61 \\ 105 \end{pmatrix}$ et $u_{ex} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$.

La valeur optimale $Z[IQP_{ex}]$ est éagle à 54 et sa solution optimale (x_1^*, x_2^*) vaut $(0, 1)$. La valeur optimale du problème relaxé continûment (IQP_{ex}) est $Z[\overline{IQP_{ex}}]$ et est égale à 62.87 et sa solution optimale correspondante $(\overline{x_1^*}, \overline{x_2^*})$ est égale à $(0.17, 0.95)$. Les solutions et valeurs optimales sont obtenues par l'emploi de IBM ILOG CPLEX 12.2. En effet, puisque le problème est convexe, CPLEX est en mesure de calculer les valeurs et solutions optimales des problèmes en nombres entiers et en variables continues.

3.2.1 Diagonalisation à l'aide des valeurs propres

Tout d'abord, rappelons deux théorèmes, bien connus d'algèbre linéaire, nécessaires (pour plus de détails voir [51]) pour établir la transformation utilisant les valeurs propres.

Theorème 3.1 *Soit Q une matrice (n, n) symétrique. Alors il existe une matrice orthogonale P qui diagonalise Q i.e. $P^tQP = D$ où P est orthogonale.*

Theorème 3.2 *Soit $Q = (q_{ik})_{n\times n}$ une matrice symétrique de valeurs propres $\lambda_1, ..., \lambda_n$. Soit P une matrice orthogonale qui diagonalise Q. Alors le changement de coordonnées $x = Py$ transforme $\sum_{i,k}^{n} q_{ik}x_ix_k$ en $\sum_{i}^{n} \lambda_i y_i^2$.*

D'après le théorème 3.2, le changement de variables $x = Py$ (i.e $y = P^{-1}x$) transforme $(QMKP)$ en un programme séparable en variables discrètes comme suit :

$$(QP_{sepVP}) \begin{cases} \max\ g(y) = c^tP_{ex}y - y^tDy \\ s.c. \left| \begin{array}{l} APy \leq b \\ 0 \leq y \leq P^{-1}u \\ y \in Y \end{array} \right. \end{cases} \tag{3.2.3}$$

où Y est un ensemble de valeurs discrètes induit du produit de P^{-1} (qui peut contenir des valeurs négatives) par chaque variable entière x prenant leur valeur entre 0 et u.

Appliquons cette technique à $QMKP_{ex}$. La première étape consiste à diagonaliser Q_{ex}. Les valeurs propres correspondant à $QMKP_{ex}$ sont les deux valeurs réelles suivantes $\lambda_1 = 16 + \sqrt{2}$ et $\lambda_2 = 16 - \sqrt{2}$, et les vecteurs propres orthonormés sont $v_1^t = (\frac{1}{\sqrt{4+2\sqrt{2}}}, \frac{1+\sqrt{2}}{\sqrt{4+2\sqrt{2}}})$ et $v_2^t = (\frac{1}{\sqrt{4-2\sqrt{2}}}, \frac{1-\sqrt{2}}{\sqrt{4-2\sqrt{2}}})$. Alors la matrice de passage orthogonale associée P_{ex} est égale à $P_{ex} = \begin{pmatrix} \frac{1}{\sqrt{4+2\sqrt{2}}} & \frac{1}{\sqrt{4-2\sqrt{2}}} \\ \frac{1+\sqrt{2}}{\sqrt{4+2\sqrt{2}}} & \frac{1-\sqrt{2}}{\sqrt{4-2\sqrt{2}}} \end{pmatrix}$. $(QMKP_{ex})$ devient alors :

$$(QMKP_{exVP}) \begin{cases} \max\ g(y) = 92.0006036y_1 + 36.57771y_2 \\ -(17.4142136y_1^2 + 14.5857864y_2^2) \\ s.c. \left| \begin{array}{l} 77.1913346y_1 + 55.7000705y_2 \leq 61 \\ 8.35337742y_1 + 14.9405852y_2 \leq 105 \\ y_1 \in Y_1 \\ y_2 \in Y_2 \end{array} \right. \end{cases} \tag{3.2.4}$$

où $Y_1 = \{0, 0.092, 0.38, 0.76, 1.14, 1.30, 1.68, 1.84, 2.07, 2.61, 2.99, 3.23\}$ et $Y_2 = \{0, -0.38, -0.76, 0.15, 0.61, 0.92, 1.0$

Remarque 3.3 *Pour une bonne lisibilité des ensembles Y_1 et Y_2, nous avons tronqué les valeurs que peuvent prendre les variables y_1 et y_2, toutefois ces valeurs peuvent être irrationnelles.*

Concluons sur l'emploi de cette transformation au moyen des valeurs propres :

- Les problèmes $(QMKP)$ et (QP_{exVP}) devraient être équivalents (*i.e.* les valeurs optimales devraient être égales) mais ce n'est pas le cas en pratique. En effet, les valeurs optimales obtenues pour les deux problèmes en utilisant IBM ILOG CPLEX 12.2 sont différrentes du fait de l'irrationalité des valeurs propres.
- Les variables de décision y sont maintenant discrètes et non plus entières ce qui ajoute une difficulté supplémentaire pour résoudre (QP_{exVP}) à travers une méthode de *branch-and-bound*. En effet, même IBM ILOG CPLEX 12.2 ne gère pas ce type de variables.

Afin de contourner ces deux principaux obstacles, nous proposons dans la prochaine sous-section l'utilisation de la décomposition de Gauss avec changement de variables partiel pour rendre séparable $(QMKP)$.

3.2.2 Décomposition de Gauss

La décomposition de Gauss (voir [51]) consiste à factoriser la matrice Q en un produit de matrice $Q = UDL$ où L est une matrice triangulaire inférieure, U une matrice triangulaire supérieure et D une matrice diagonale. Cette technique a déjà été utlilisée par Zheng et al. dans [48] pour traiter $(QMKP)$ non convexe afin d'obtenir une relaxation intéressante pour le problème de départ.
Plus précisemment, les principales phases de cette technique consistent à rendre triangulaire la matrice Q en la multipliant par la gauche avec la matrice d'élimination de Gauss ($n-1$ matrices d'élimination si Q est une matrice $n \times n$ comme dans notre cadre d'étude).Par exemple, la matrices d'élimination E_1 à la première itération ($k = 1$) s'écrit de la façon suivante : $E_1 = \begin{pmatrix} 1 & ... & 0 & ... & 0 \\ -\frac{q_{21}^{(1)}}{q_{11}^{(1)}} & 1 & 0 & ... & 0 \\ ... & ... & ... & ... & \\ -\frac{q_{i1}^{(1)}}{q_{11}^{(1)}} & ... & 1 & ... & 0 \\ ... & ... & ... & ... & \\ -\frac{q_{n1}^{(1)}}{q_{11}^{(1)}} & ... & 0 & ... & 1 \end{pmatrix}$

L'algorithme de triangularisation est le suivant :

$$k = 1, ..., n-1 \begin{cases} q_{ij}^{(k+1)} = q_{ij}^{(k)} & i = 1, ..., k \; j = 1, ..., n \\ q_{ij}^{(k+1)} = 0 & i = k+1, ..., n \; j = 1, ..., k \\ q_{ij}^{(k+1)} = q_{ij}^{(k)} - \frac{q_{ik}^{(k)} q_{kj}^{(k)}}{q_{kk}^{(k)}} & i = k+1, ..., n \; j = k+1, ..., n \end{cases} \tag{3.2.5}$$

Afin d'obtenir une matrice diagonale, il suffit alors de multiplier Q par la droite par les transposées des matrices d'élimination successives. Plus généralement, à l'issue de l'algorithme de décomposition de Gauss on a :
$D = E_{n-1} \times .... \times E_1 \times Q \times E_1^t \times ... \times E_{n-1}^t$. Ainsi Q peut être remplacée par D de la façon suivante :
$Q = (E_{n-1} \times .... \times E_1)^{-1} \times D \times (E_1^t \times ... \times E_{n-1}^t)^{-1}$. En conséquence, le changement de variable suivant peut être effectué $y = Rx$
tel que $x^t Q x = \underbrace{x^t E_{n-1}^{-1} \times .... \times E_1^{-1}}_{=y^t} \times D \times \underbrace{(E_{n-1}^t)^{-1} \times ... \times (E_1^t)^{-1} x}_{=y}$,
où $R = (E_{n-1}^t)^{-1} \times ... \times (E_1^t)^{-1}$.

Remarque 3.4 *La méthode de décomposition de Gauss emploie des matrices de structures particulières (i.e. unidiagonales) ce qui simplifie aisément le calcul des matrices inverses.*

A ce stade du raisonnement, nous aurions pu choisir d'effectuer un changement de variables complet. C'est à dire que nous aurions pu choisir de remplacer le vecteur de décision x par y (cf. transformation 2 présentée dans l'introduction de ce chapitre). Néanmoins, nous aurions été confrontés à l'une des difficultés rencontrée à l'occasion de la transformation au moyen des valeurs propres : la nature des variables serait discrète et non réelle ou entière. En conséquence, nous avons décidé de conserver les contraintes linéaires initiales exprimées à l'aide des variables de décision x. Nous avons remplacé x uniquement au sein de la fonction objectif et nous avons introduit le changement de variables dans les contraintes (c'est la transformation 3 présentée dans l'introduction de ce chapitre).
$(QMKP)$ devient alors le programme quadratique séparable en variables mixtes suivant :

$$(QP_{x,y}) \begin{cases} \max\ g(y) = c'^t y - \frac{1}{2} y^t D y \\ s.c. \left| \begin{array}{l} Rx = y \\ Ax \leq b \\ 0 \leq x \leq u \\ y \in \mathbb{R}^n \\ x \in \mathbb{N}^n \end{array} \right. \end{cases} \tag{3.2.6}$$

où $c' = (c'_1, ..., c'_i, ..., c'_n) = c^t \times R^{-1}$ est le nouveau vecteur des coefficients de la fonction objectif.

Appliquons cette transformation à notre simple exemple. Le problème séperable résultant est le suivant :

$$(QP_{ex,x,y}) \begin{cases} \max\ g(x,y) = 69x_1 + 71x_2 - 15y_1^2 - \frac{254}{15} y_2^2 \\ s.c. \left| \begin{array}{l} 81x_1 + 50x_2 \leq 61 \\ 17x_1 + 2x_2 \leq 105 \\ x_1 + \frac{1}{15} x_2 - y_1 = 0 \\ 0 \leq x_1 \leq 3 \\ 0 \leq x_2 \leq 2 \\ 0 \leq y_1 \leq \frac{47}{7} \\ x_1, x_2 \in \mathbb{N} \\ y_1, y_2 \in \mathbb{R} \end{array} \right. \end{cases} \tag{3.2.7}$$

La valeur optimale de $(QP_{ex,x,y})$ est $Z[QP_{ex,x,y}] = Z[IQP_{ex}] = 54$ est la valeur optimale de la relaxation continue est $Z[\overline{QP_{ex,x,y}}] = Z[\overline{IQP_{ex}}] = 62.87$.

Conclusions sur cette procédure :
- Le problème transformé séparable $(QP_{x,y})$ reste convexe.
- $(QMKP)$ et $(QP_{x,y})$ sont équivalents , dans le sens où les fonctions objectifs ont les mêmes valeurs.
- $(QP_{x,y})$ est un programme en variables mixtes *i.e.* les variables ne prennent pas leurs valeurs parmi un ensemble discret de valeurs.
- La transformation peut être réalisée en temps polynomial.

3.3 UN SCHÉMA DE LINÉARISATION

Dans cette section nous proposons de résoudre exactement $(QMKP)$ *via* un programme linéarisé équivalent $(LP^K_{x,y})$. Ce dernier est paramétré par K. Nous motivons ce choix (de linéariser $(QMKP)$) car nous avons

constaté, lors des travaux menés au cours de ma thèse de doctorat, que la résolution du problème linéarisé équivalent au problème quadratique séparable était efficace.
Le problème linéarisé fournit un majorant pour le problème quadratique initial quand K est fixé. De plus, quad K est suffisamment grand alors le problème linéarisé est équivalent au problème quadratique initial. De surcroît, une méthode exacte peut être obtenue grâce au problème transformé. Nous décrivons dans cette section le schéma de linéarisation proposé ainsi que des propriétés relatives à ce présent schéma fournissant un majorant ou une solution optimale pour $(QMKP)$.

3.3.1 Linéarisation de $(QP_{x,y})$

Voici les principales étapes de la linéarisation de $(QP_{x,y})$.

Linéarisation de la fonction objectif. La fonction objectif de $(QP_{x,y})$ peut s'écrire comme $\sum_{i=1}^{n} g_i(y_i) = \sum_{i=1}^{n} c_i^{'} y_i - d_i y_i^2$, où $c_i^{'} = [cR^{-1}]_i = \sum_{i=1}^{n} c_i R_{ji}^{-1}$.
Tout d'abord, calculons les bornes supérieures et inférieures B_i^- et B_i^+ de chaque variable y_i en utilisant les contraintes $Rx = y$ et les bornes sur les variables x_i :

$$\forall i \in \{1, ..., n\}, B_i^- \leq y_i \leq B_i^+ \; avec \begin{cases} B_i^- = \sum_{j|R_{ij}<0} R_{ij} u_j \\ B_i^+ = \sum_{j|R_{ij}>0} R_{ij} u_j) \end{cases}$$

Puis, chaque fonction $g_i(y_i)$ est remplacée par une fonction linéraire par morceaux $h_i(y_i)$ telle que $h_i(y_i) \geq g_i(y_i)$ $\forall y_i \in [B_i^-, B_i^+]$. Nous construisons $h_i(y_i)$ en utilisant K segments tangents à la courbe G_i qui représente la fonction g_i. K est un paramètre de la procédure et son choix est évidemment une étape clef afin de calculer une sur-estimation de la valeur optimale de $(QP_{x,y})$ la plus fine possible. Ce point crucial est discuté dans la sous-section suivante. La figure 3.1 illustre la linéarisation proposée.

Transformation des variables y en z. Afin de résoudre le programme associé à la courbe H_i, nous associons une variable z_{ik} au k^{eme} segment de H_i. Nous remplaçons donc chaque variable y_i par K non-négatives variables réelles z_{ik} en posant :

$$y_i = B_i^- + \sum_{k=1}^{K} z_{ik}$$

Introduisons les notations suivantes pour formuler $(LP_{x,z}^K)$:
- $cst = \sum_{i=1}^{n} g_i(B_i^-)$;
- $step_i = \frac{B_i^+ - B_i^-}{K-1}$, $i = 1, ..., n$;

et pour chaque $i \in \{1, ..., n\}$ et $k \in \{1, ..., K\}$:
- $r_{ik} = B_i^- + (k-1) \times step_i$: r_{ik} est l'abscisse du k^{eme} point tangent entre H_i et G_i ;
- $\alpha_{ik} = c_i^{'} - d_i r_{ik}$: α_{ik} est le coefficient directeur du k^{eme} segment ;
- $\beta_{ik} = g(r_{ik}) - r_{ik}\alpha_{ik}$: β_{ik} est l'ordonnée à l'intersection du k^{eme} segment ;
- $t_{i1} = B_i^-, t_{i,K+1} = B_i^+, t_{ik} = \frac{\beta_{i,k-1} - \beta_{i,k}}{\alpha_{i,k} - \alpha i,k-1}$, $k = 2, ..., K$: t_{ik} est l'abscisse du point d'intersection entre les k^{th} et $(k+1)^{eme}$ segments ;
- $Z_{ik} = t_{i,k+1} - t_{ik}$: Z_{ik} est la borne supérieure de la variable z_{ik}.

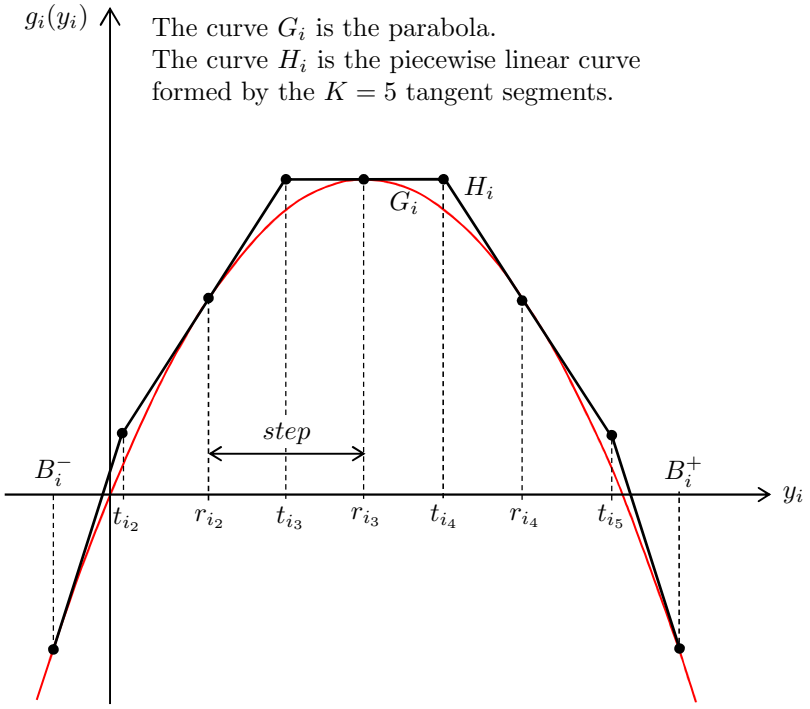

FIG. 3.1 – Couvrir la courbe G_i par une courbe linéaire par morceaux H_i

Nous pouvons maintenant établir le problème $(LP^K_{x,z})$ qui est un problème en variables mixtes comptant, n variables entières x_i, Kn non négatives et bornées variables réelles z_{ik} et $m + n$ contraintes linéaires :

$$(LP^K_{x,z}) \begin{cases} \max\ h(z) = cst + \sum_{i=1}^{n}\sum_{k=1}^{K}\alpha_{i,k}z_{i,k} & \\ s.c. \begin{array}{|l} Ax \leq b & \\ \sum_{j=1}^{n} R_{ij}x_j = B_i^- + \sum_{k=1}^{K} z_{ik} & i = 1,\dots,n \\ 0 \leq x_i \leq u_i & i = 1,\dots,n \\ x_i \in \mathbb{N} & i = 1,\dots,n \\ 0 \leq z_{ik} \leq Z_{ik} & i = 1,\dots,n, k = 1,\dots,K \end{array} \end{cases} \tag{3.3.1}$$

$(LP^K_{x,z})$ est une sur-estimation de la valeur optimale de $(QMKP)$ si K n'est pas assez grand. Cependant nous montrons dans la sous-section suivante comment déterminer la bonne valeur de K telle que $(LP^K_{x,z})$ fournisse la valeur optimale de $(QMKP)$.

3.3.2 Une propriété intuitive pour un problème linéarisé équivalent

Le théorème suivant indique que la linéarisation proposée peut être utilisée non seulement pour produire un majorant fin mais également pour résoudre de façon exacte le problème initial.

Introduisons les notations suivantes. Nous notons z^K la valeur optimale de $(LP^K_{x,z})$ pour K fixé et z^* la valeur optimale de $(QMKP)$.

Theorème 3.5

$$\lim_{K \to +\infty} z^K = z^*$$

Preuve 3.6 *L'intérieur de l'enveloppe constituée par l'ensemble des tangentes devient intuitivement très proche de la courbe représentant chaque fonction $g_i(y)$ quand K devient grand. De plus quand K tend vers $+\infty$ l'enveloppe se confond avec la courbe. Montrons formellement ce résultat. Nous démontrons que :*

$$\forall \epsilon > 0, \exists K > 0 \text{ tel que } |z^K - z^*| = z^K - z^* < \epsilon. \tag{3.3.2}$$

Soeint $\epsilon > 0$ et K fixé. Nous considérons la solution optimale (x^K, z^K) de $(LP^K_{x,z})$. x^K est une solution réalisable de $(QMKP)$. Nous posons ensuite $y_i^K = B_i^- + \sum_{k=1}^K z_{ik}^K$. Nous obtenons

$$z^K = h(x^K, z^K) = \sum_{i=1}^{n} \left[g_i(y_i^K) + \epsilon_i^K \right] = f(x^*) + \sum_{i=1}^{n} \epsilon_i^K$$

où $\epsilon_i^K = h_i(y_i^K) - g_i(y_i^K) \geq 0$ (cf. Figure. 3.2).

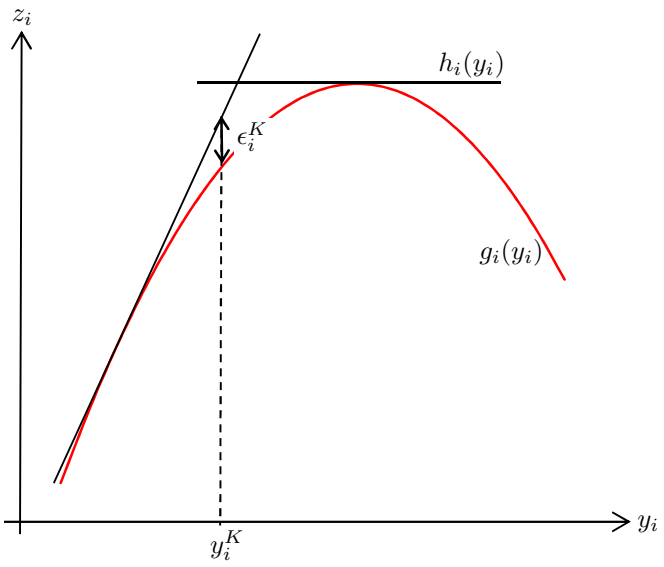

FIG. 3.2 – Définition d' ϵ_i^K

Puisque x_k est réalisable pour $(QMKP)$ (pas nécessairement optimale), nous prenons

$$z^k \leq z^* + \sum_{i=1}^{n} \epsilon_i^K, \textit{ i.e. } z^K - z^* \leq \sum_{i=1}^{n} \epsilon_i^K$$

Pour démontrer (3.3.2), il suffit maintenant de montrer que, pour K assez grand

$$\sum_{i=1}^{n} \epsilon_i^K < \epsilon$$

Nous montrons que pour K assez grand, $\epsilon_i^K < \frac{\epsilon}{n}$, le résultat est obtenu. L'idée réside dans le fait que l'écart maximal entre l'enveloppe constituée par les tangentes et la courbe doit réduire selon la précision choisie (ici $\frac{\epsilon}{n}$) et ce, en ajoutant des tangentes autant que nécessaire. Considérons deux tangentes successives aux points A et B ayant respectivement pour abscisse y_1 et y_2, tels que $y_1 \leq y_2$. Sans perte de généralité, nous pouvons considérer que A et B sont tous les deux situés soit sur la droite de la parabole, soit sur la gauche, puisque l'on peut toujours ajouter un point tangent au sommet de la parabole. Supposons que A et B sont situés, tous les deux, à gauche du sommet de la parabole. Notons h l'écart maximal entre h_i et g_i dans l'intervalle $[y_1, y_2]$ (cf. Figure 3.3). Puisque la fonction g_i est croissante, nous avons que : $h \leq H$ avec $H = g(y_2) - g(y_1) = c_i^{'} y_2 - \frac{1}{2} d_i y_2^2 - c_i^{'} y_1 + \frac{1}{2} d_i y_1^2$.

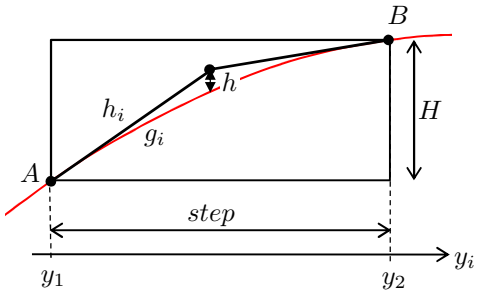

FIG. 3.3 – Sur-estimation de ϵ_i^K

Notons $step$ l'espace entre les abscisses de A et B ($step = y_2 - y_1$). En utilisant des notions de base d'algèbre, on obtient facilement que :

$$h \leq step \left(c_i^{'} - \frac{1}{2} d_i (y_1 + y_2) \right),$$

et comme $y_1 \geq B_i^-$ et $y_2 \geq B_i^-$ nous avons :

$$h \leq step \ (c_i^{'} - d_i B_i^-)$$

Supposons maintenant que A et B se situent tous les deux à droite du sommet de la parabole, la précédente sur-estimation h devient :

$$h \leq step \ (d_i B_i^+ - c_i^{'})$$

Comme $step = \frac{B_i^+ - B_i^-}{K}$ alors nous pouvons sur-estimer ϵ_i^K :

$$\epsilon_i^K \leq \frac{B_i^+ - B_i^-}{K} \ \max(c_i^{'} - d_i B_i^-, d_i B_i^+ - c_i^{'})$$

Pour s'assurer que $\epsilon_i^K < \frac{\epsilon}{n}$, il est maintenant suffisant de choisir K tel que :

$$K > \frac{n \ \max(c_i^{'} - d_i B_i^-, d_i B_i^+ - c_i^{'})}{\epsilon \ (B_i^+ - B_i^-)}$$

Notons que la preuve du Théorème 3.5 fournit une borne supérieure théorique pour le paramètre K. Nous pouvons supposer sans perte de généralité que les coefficients du problème initial $(QMKP)$ sont entiers. Ainsi, lorsque l'écart (entre le majorant z^K et la valeur de la solution réalisable $\tilde{z}$) est strictement plus petit que 1, nous pouvons considérer que la solution réalisable est optimale et que nous connaissons alors la valeur optimale de $(QMKP)$.

En considérant la preuve du Théorème 3.5, et en posant ϵ égal à 1, nous savons qu'en fixant le paramètre K à la valeur $\frac{n \max(c_i^{'}-d_iB_i^-, d_iB_i^+-c_i^{'})}{(B_i^+-B_i^-)}$ nous obtiendrons la valeur optimale de $(QMKP)$. Bien entendu, cette borne supérieure sur K n'est pas très pertinente en pratique puisqu'elle nécessite une valeur très grande. Voyons comment mettre en oeuvre cette méthode.

3.4 MISE EN ŒUVRE DE LA MÉTHODE EXACTE

La résolution de $(QMKP)$ repose sur les trois étapes suivantes :

(i) calcul d'une solution admissible (borne inférieure) de bonne qualité (cf. sous-section 3.4.1) ;
(ii) une étape de pré-traitement afin de borner les variables de décision du problème (cf. sous-section 3.4.2) ;
(iii) un calcul itératif du majorant z^K nécessitant de choisir les points de ruptures et le nombre de segments tangents à chaque courbe g_i (cf. sous-section 3.4.3).

De plus, nous mentionnons dans la Section 3.5 comment chaque paramètre de la méthode de résolution exacte est déterminé dans le but de fournir un algorithme le plus rapide possible en terme de temps CPU.

3.4.1 Solution admissible

Nous proposons deux voies possibles pour calculer une solution réalisable. La première est obtenue par amélioration successive au fur et à mesure de la méthode de résolution exacte de $(QMKP)$. Cette solution exacte est obtenue *via* l' $Heuristique1$ et est dérivée de la relaxation continue de $(QP_{x,y})$.
La deuxième solution admissible (fournie par $Heuristique2$) provient de la solution de $(LP_{x,z})$.
Nous précisons lors de l'écriture de l'algorithme 1 quand chacune des heuristiques est employée.

Heuristique 1. Cette heuristique est utilisée une fois au début de la méthode exacte (voir Algorithme 1).
Cette borne inférieure est fournie par une heuristique développée durant ma thèse de doctorat et publiée dans [52] basée sur l'idée principale suivante. Considérons la solution optimale $\overline{x} = (\overline{x_1}, ..., \overline{x_i}, ..., \overline{x_n})$ de la relation continue de $(QP_{x,y})$.Nous restreignons l'ensemble des solutions admissibles autour de la solution trouvée afin de résoudre $(LP_{x,z}^K)$. Plus précisemment, nous restreignons les bornes sur chaque variable aux parties entières inférieures et supérieures en partant de sa valeur dans $\overline{x}$. On obtient alors un problème $0-1$ non équivalent au problème initial $(QMKP)$. La solution entière ainsi calculée est réalisable pour $(QMKP)$. Sa valeur $f(x)$ est alors un minorant pour $(QMKP)$ et, en pratique est optimale dans la plupart des cas (cf. Section 3.5).

Heuristique 2. Cette heuristique est employée plusieurs fois au cours de la méthode de résolution exacte : après chaque calcul des bornes z^K. Nous partons de la solution de $(LP_{x,z}^K)$ écrite à l'aide des variables x. Elle est admissible pour $(QMKP)$. Alors, sa valeur objectif correspondante $f(x)$ est un minorant pour $(QMKP)$.

3.4.2 Prétraitement

Une étape clef concernant les méthodes de résolution exacte d'un programme en nombres entiers consiste à restreindre les bornes (inférieures et supérieures) sur les variables entières, de telle sorte à réduire le plus possible, en amont, l'aspect combinatoire du problème. Le prétraiement suivant a été appliqué sur chaque variable de décision du problème initial.

Réduction des bornes supérieures des variables x_i. Pour chaque variable entière x_i, nous résolvons d'abord un programme linéaire consistant en la maximisation de x_i soumise à l'ensemble des contraintes de $(QMKP)$. Bien entendu, ce prétraitement est consommateur en temps CPU. En effet, pour chaque variable de décision un programme linéaire de grande taille doit être résolu (même si ces programmes sont moins difficiles à résoudre que le problème initial). La borne supérieure de départ de x_i est remplacée par la valeur optimale du programme linéaire.

Augmentation de la valeur des bornes inférieures des variables x_i. Nous résolvons ici un second type de programme mathématique pour lequel x_i est minimisée soumise à l'ensemble des contraintes de $(QMKP)$. La borne inférieure initiale de x_i est remplacée par la valeur obtenue.
Ce prétraitement impacte directement les bornes B_i^+ et B_i^- sur chaque variable y_i. Comme cela est mentionné dans la Section 3.5 , ce pré-traitement peut constituer une part très importante du temps de calcul total de la solution exacte : pour les instances de taille moyenne (n=40), il représente même la quasi totalité du temps de calcul global et pour les plus grosses instances considérées, il peut représenter jusqu'à 25% du temps de calcul global de la solution exacte. Les nouvelles bornes sur chaque variable ainsi calculées permettent naturellement de resserrer pour chaque variable y_i l'amplitude de l'intervalle $[B_i^-, B_i^+]$.

3.4.3 Intensification autour du voisinage des solutions admissibles

Le calcul de z^K est très consommateur en temps car il compte la résolution du programme linéaire en variables mixtes $(LP_{x,z}^K)$. Ainsi, il est important de limiter le nombre de problèmes à résoudre pour ce calcul. Si l'on prend une valeur de K trop grande alors le calcul de z^K devient très lent puisque le nombre de variables est alors égal à $(K+1) \times n$. La figure 3.4 illustre ce phénomène et établit les valeurs des bornes inférieure et supérieure pour un problème comptant 80 variables et 30 contraintes (densité = 30%). Ce problème n'est pas résolu à l'optimum, étant donné qu'il est présenté uniquement pour montrer l'influence du paramètre K, $\tilde{z}^{max}$ est la valeur fournie par $Heuristique_1$, $Z[\overline{QMKP}]$ représente la valeur optimale de la relaxation continue de $(QMKP)$, et z^K (obtenue après s secondes) correspond à la valeur de la borne supérieure z^K après s secondes (où $s \in \{5, 10, 20, 60, 180\}$). Notons que pour une même valeur de K plus du temps est alloué pour résoudre le problème, meilleure est la qualité de z^K. De plus une remarque importante est la suivante : passé un certain seuil (autours de $K = 40$ pour l'instance présentée ici), il n'est plus nécessaire d'augmenter la valeur de K en pratique. En effet, même si nous savons que la meilleure borne supérieure n'est pas encore atteinte, le temps CPU requis pour l'obtenir, devrait augmenter considérablement.
Un bon compromis consiste à ne pas fixer les segments tangents uniformément sur la courbe. Pour cela, on s'appuie sur le fait que les solutions renvoyées par $Heuristique_1$ et $Heuristique_2$ s'avèrent en pratique très souvent optimales : c'est autour des solutions renvoyées par ces heuristiques que l'on ajoute un grand nombre de segments tangents, pour affiner la sur-estimation dans ce voisinage. Nous appelons intensification ce procédé. Il s'avère en pratique extrêmement important pour la résolution exacte, en limitant (à 2) le nombre de fois où l'on doit calculer une borne z^K après avoir augmenté la valeur de K. Nous détaillons dans

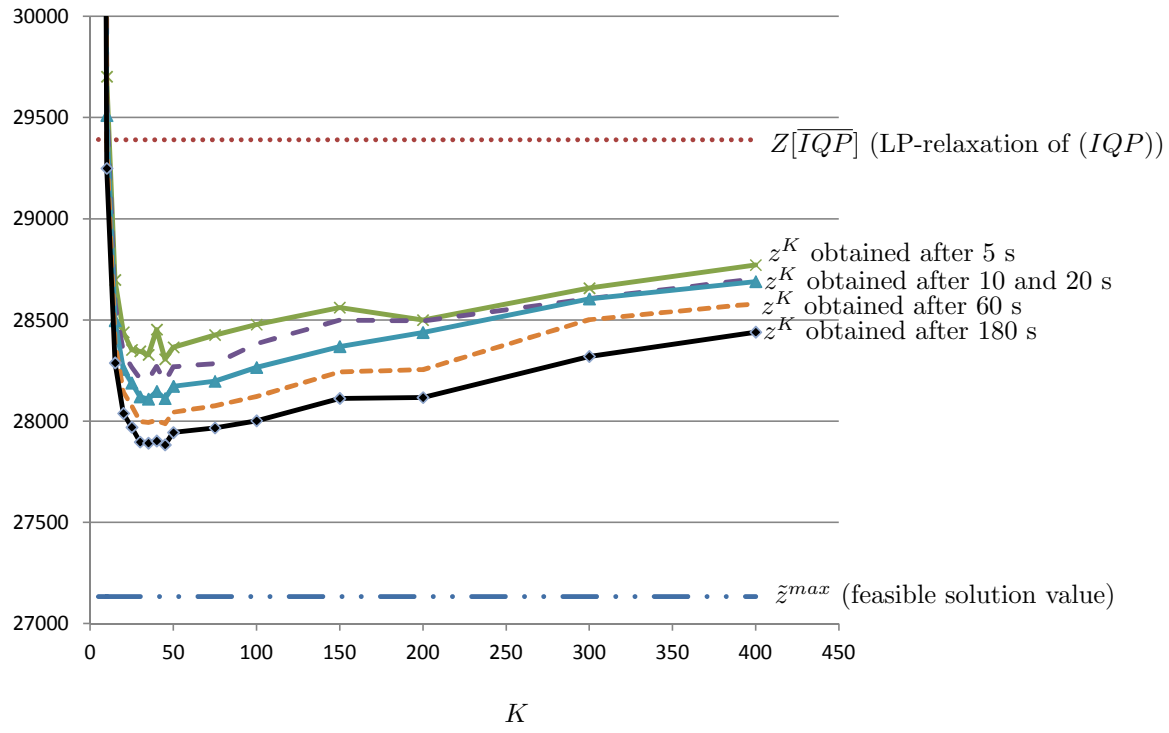

FIG. 3.4 – Influence du paramètre K sur le temps de calcul et la qualité de la borne z^K (n=80)

l'Algorithme 1 les deux étapes successives : trouver une solution admissible (cf. heuristiques) et raffiner la valeur de la borne supérieure. Dans cet algorithme $NbIntens_1$ et $NbIntens_2$ représentent le nombre de segments que nous ajoutons au voisinage des solutions heuristiques (c'est donc ce raffinement que nous nommons "intensification").

L'algorithme 1 est employé pour résoudre les instances présentées dans la section 3.5. Notons que si $z^K = \tilde{z}^{max}$ alors nous savons que z^K est valeur optimale pour $(LP^K_{x,z})$ et en conséquence pour $(QMKP)$.

3.5 RÉSULTATS NUMÉRIQUES

Nous avons choisi de tester notre méthode sur des problèmes de multi-sac-à-dos quadratiques. Toutefois, notre méthode pourrait s'appliquer tout aussi bien sur des problèmes avec des contraintes linéaires quelconques. Nous avons généré des instances de la façon suivante. Chu et Beaslsey ont proposé dans [53] (fichiers mknpcb1, mknpcb2, mkpcb4, mkpcb5, mkpcb7, mkpcb8)[1] des instances difficiles pour le problème du multi-sac-à-dos linéaire en variables entières. On a $n \in \{40, 50, 60\}$, $m = 30$, et la densité ($\alpha = b_j / \sum_{j=1}^{n} a_{ji} \in \{0.25, 0.5, 0.75\}$). Les coefficients entiers a_{ji} prennent aléatoirement leur valeur dans $\{0...1000\}$.
Ces instances sont réputées difficiles principalement en raison du caractère corrélé des coefficients de la fonction objectif d'une part et des contraintes d'autres part. Le problème que nous considérons n'est pas

[1] instances disposinibles sur le site Internet `http ://people.brunel.ac.uk/ mastjjb/jeb/orlib/mknapinfo.html`

Data: La matrice R définie dans la section 3.2.2
Result: La valeur optimale du problème $(QMKP)$
begin
On attribue une valeur au paramètre K;
$step \longleftarrow \frac{1}{K} \max_{i \in 1,\ldots,n}(B_i^+ - B_i^-)$; $NbIntens_1 \longleftarrow 0$; $NbIntens_2 \longleftarrow 0$;
foreach $i \in 1, \ldots, n$ **do** $K_i \longleftarrow \frac{(B_i^+ - B_i^-)}{step}$;
Calcul d'une solution admissible $\tilde{x}^1$ de valeur $\tilde{z}^1$ pour $(QMKP)$ par $Heuristique_1$;
$\tilde{z}^{max} \longleftarrow \tilde{z}^1$; $\tilde{y}^1 \longleftarrow R\tilde{x}^1$;
Calcul de z^K, valeur optimale de $(LP_{x,z}^K)$ (Section 3.3.1) ;
Dérivée du calcul de z^K, une nouvelle solution $\tilde{x}^2$ de valeur $\tilde{z}^2$ est obtenue pour $(QMKP)$ *via* $Heuristique_2$;
$\tilde{y}^2 \longleftarrow R\tilde{x}^2$;
$\tilde{z}^{max} \longleftarrow \max(\tilde{z}^1, \tilde{z}^2)$;
while $z^K - \tilde{z}^{max} > 1$ **do**
Augmenter K, $NbIntens_1$ et $NbIntens_2$;
$Intensification$;
for $i \in 1, \ldots, n$ **do**
for $j = 1, 2$ **do**
Ajouter $NbIntens_j$ segments autour de $\tilde{y}_i^j$ dans l'intervalle $\left[\tilde{y}_i^1 - 1; \tilde{y}_i^1 + 1\right]$
end
end
Calcul d'une nouvelle borne z^K, valeur optimale du nouveau problème $(LP_{x,z}^K)$
Mise à jour de $\tilde{x}^2$ et $\tilde{z}^2$ *via* $Heuristique_2$, ainsi que z^{max}
end
return $\lfloor z^K \rfloor$
end

Algorithm 1: Résolution exacte

linéaire, mais quadratique (dans la fonction objectif seulement, les contraintes des instances considérées sont toutes des contraintes de sac-à-dos) et la fonction objectif est non séparable. Nous avons donc ajouté à la fonction objectif des instances de Chu et Beasley une partie quadratique non séparable, en nous assurant de la concavité de la fonction objectif. Nous avons aussi généré les bornes u_j sur les variables entières comme prenant leurs valeurs dans l'ensemble $\{0...10\}$.

Les instances que nous avons pu résoudre sont de grande taille si l'on considère le caractère très combinatoire du problème (les variables sont entières et bornées et à cette difficulté s'ajoute le caractère quadratique de la fonction objectif).
Toutes les expérimentations ont été menées sur une machine PC Intel Dual Core i5 CPU (2.53 GHz) avec 8 Gb de RAM, OpenSUSE 11.4. Le langage utilisé est le C pour notre méthode exacte et les heuristiques, et nous avons utilisé la version 12.2 de IBM ILOG-CPLEX pour la résolution des programmes linéaires et la version quadratique initiale du problème.
Nous présentons dans un premier temps la façon dont les principaux paramètres de la méhode ont été choisis (après de nombreux essais) afin de garantir les meilleurs temps de calculs. Puis nous présentons des comparaisons entre notre méthode d'une part, et l'utilisation "brute" d'un *solveur* commercial (ILOG-CPLEX) pour résoudre soit le problème initial directement (il est convexe), soit la version quadratique rendue séparable du problème.

3.5.1 Ajustement des paramètres

On note en pratique que pour résoudre des instances de grande taille notamment, il est parfois nécessaire pour garantir l'optimalité de calculer plusieurs fois la borne z^K, en augmentant à chaque fois le nombre de segments afin d'affiner la borne et de bénéficier d'une amélioration éventuelle par l'$Heuristique_2$ de la valeur de la meilleure solution admissible connue. Le tableau 3.1 précise les valeurs des paramètres K (nombre de segments initiaux), $NbIntens_1$ et $NbIntens_2$ segments ajoutés pendant l'ajustement (cf. sous-section 3.4.3) que nous avons utilisés pour résoudre les instances à l'optimum, après de nombreux essais correspondant aux différentes combinaisons des différents paramètres. Le tableau 3.1 ne précise que la valeur des paramètres utilisés. Les résultats obtenus sont exposés dans la sous section suivante 3.5.2.
Chaque ligne du tableau représente la moyenne de 10 instances pour une valeur fixée de n et de la densité d du problème. La colonne "happens" indique, pour chaque essai (i.e. chaque calcul de la borne z^k), sa fréquence : par exemple, 20% des 10 instances résolues avec $n = 50$ et $d = 0.5$ requière un second essai (un autre calcul de z^K) pour trouver la solution optimale. Il est remarquable qu'au plus deux tentatives suffisent dans tous les cas à trouver l'optimum, et que la deuxième tentative est rarement nécessaire (dans moins de 20% des cas). Nous avons constaté après de nombreux essais que lors de la deuxième tentative pour poruver l'optimalité, il n'est pas judicieux d'augmenter K mais bel et bien d'intensifier le nombre de segments dans la région supposée optimale.
Notons qu'il n'est pas surprenant que la valeur trouvée par nos heuristiques soit systématiquement optimale puisque l'on sait que pour une valeur de K suffisamment grande, la borne z^K est optimale et que la solution admissible fournie par l'heuristique 2 n'est autre que la solution en x trouvée lors du calcul de z^K.

3.5.2 Résolution d'instances de grandes tailles

On ne présente ici que les résultats obtenus pour des instances de taille moyenne à grande, c'est à dire $n = 40, 50, 60$ et $m = 30$: en dessous de n=40, les 3 méthodes présentées et comparées résolvent quasi instantanément toutes les instances de façon exacte, et au delà de n=60, aucune des méthodes ne parvient plus à résoudre les instances de façon exacte dans un temps limite raisonnable (fixé à 10800 secondes).

TAB. 3.1 – Ajustement des paramètres

n	d	attempt 1				attempt 2			
		happens	K	$intens_1$	$intens_2$	happens	K	$intens_1$	$intens_2$
40	25	100%	5	10	0	0%	5	30	0
	50	100%	5	10	0	0%	5	30	0
	75	100%	5	10	0	10%	10	30	0
50	25	100%	5	16	0	0%	5	100	0
	50	100%	5	16	0	20%	5	100	0
	75	100%	5	16	0	10%	5	75	0
60	25	100%	10	30	4	10%	10	60	4
	50	100%	10	30	4	0%	10	60	4
	75	100%	10	30	4	20%	10	100	4

Le tableau 3.2 présente les gains de temps apportés par notre méthode par rapport à l'utilisation directe d'un solveur commercial (ici IBM ILOG-CPLEX 12.2), possible pour ce type de problème (quadratique concave $(QMKP)$ et $(QP_{x,y})$) dans les colonnes 2 et 1 respectivement.
La colonne 3 présente les temps de calcul pour la résolution exacte utilisant notre méthode, et enfin la colonne 4 indique le gain de temps apporté par notre méhode.

TAB. 3.2 – Comparaison des 3 formulations : $(QP_{x,y})$, $(QMKP)$ et notre méthode

n	d	1. Tps (s) opt (QP_{xy}) (séparable) avec Cplex 12.2	2. Tps (s) opt $(QMKP)$ (non séparable) avec Cplex 12.2	3. Tps (s) $(LP^K_{x,z})$ incluant prétraitement	4. Amélioration
40	25	2.71	1.83	4.45	tps mult. par 2.4
	50	3.07	1.92	2.54	tps mult. par 1.3
	75	0.86	0.57	1.73	tps mult. par 3
50	25	40.39	22.68	8.89	tps div. par 2.6
	50	10.84	6.10	4.90	tps div. par 1.2
	75	9.97	6.12	3.76	tps div. par 1.6
60	25	1 454.16	896.94	484.29	tps div. par 1.9
	50	304.19	2 077.26	15.79	tps div. par 131.6
	75	231.98	1 161.23	13.40	tps div. par 86.7

Le tableau 3.2 montre l'intérêt de notre méthode en terme de gain de temps de calcul spécialement pour les instances difficiles de grande taille quand $n = 50$ et $n = 60$, le temps de calcul est divisé par 131.6 et 86.7, respectivement. Pour les plus grandes instances résolues, notre méthode peut améliorer d'un facteur 100 la résolution de ces instances. Il montre également qu'il n'est pas efficace de rendre séparable le problème $(QP_{x,y})$ afin de le résoudre par le *solveur* (mieux vaut attaquer directement le problème non séparable $(QMKP)$ si l'on désire utiliser cplex sans notre méthode).
Les instances moyennes $(n = 40)$ sont comparablement résolues par les trois méthodes (en moyenne en 5 secondes).

Enfin le tableau 3.3 fait le point sur notre méthode en détaillant les temps CPU mentionnés dans le tableau 3.2. Chauqe ligne du trableau représente une moyenne de 10 instancespour un n donné (nombre de variables) et d (densité du problème).

TAB. 3.3 – Temps de calcul complet de la méthode exacte proposée

n	d	Gap asymptotique linéarisation	Tps (s) optimum incluant prétraitement et borne inf.	Tps (s) asympt. linéarisation	Tps (s) Borne Inf.	Tps pré-traitement (s)	# attemps
40	25%	0.001%	4.45	0.47	0.49	3.49	1
	50%	0.001%	2.54	0.37	0.16	2.01	1
	75%	0.001%	1.73	0.22	0.12	1.38	1.1
50	25%	0.001%	8.89	5.36	0.58	2.95	1
	50%	0.001%	4.90	2.21	0.17	2.51	1.2
	75%	0.001%	3.76	1.52	0.18	2.06	1.1
60	25%	0.002%	484.29	475.25	1.67	7.37	1.1
	50%	0.000%	15.79	11.30	0.34	4.14	1
	75%	0.000%	13.40	11.88	0.30	1.22	1.2

Dans la plupart des cas, un seul calcul du majorant z^K est nécessaire. Le temps CPU requis par les heuristiques pour fournir une solution admissible est négligeable en comparaison avec le temps CPU total. Au contraire, le pré-traitement (cf. Section 3.4.2) est très consommateur en temps. Par exemple, quand $n = 40$, le temps CPU correspondant représente pratiquement la totalité du temps CPU de la méthode dans son intégralité. Quand $n = 60$ il représente 25% du temps total.

3.6 CONCLUSION

Nous avons proposé une méthode de résolution exacte pour un programme quadratique non séparable convexe en variables entières. L'algorithme développé se déroule en deux phases dans le but de transformer le programme quadratique initial en un problème linéaire en variables mixtes équivalent, dépendant d'un paramètre K. Une étude théorique est alors menée afin d'attester de l'équivalence des problèmes. Les expérimentations numériques ont été menées sur des instanses de multi-sac-à-dos quadratique non séparable en variables entières. Toutefois, notre algorithme peut être employé dans un cadre plus large. Les résultats numériques menés sur des instances de grandes tailles montrent que notre approche permet de diviser par 1.2 jusqu'à 131.6 fois le temps nécessaire à leur résolution en comparaison avec CPLEX.

CHAPITRE 4

Programme quadratique non convexe en variables 0-1 : linéarisation

Ce chapitre fait l'objet d'une partie de la thèse de Carlos Diego Rodrigues (actuellement maître de conférences à l'université de Fortaleza au Brésil), que j'ai co-encadré avec P. Michelon (Pr. à l'université d'Avignon). Ce travail a donné lieu à la publication d'un article (ref. 6 dans le chapitre "Liste de publications") et d'une conférence internationale avec actes (ref. 6 dans le chapitre "Liste de publications").

Ce travail concerne le problème du sac-à-dos quadratique en variables 0-1 (QKP) qui consiste en la maximisation d'une fonction pseudo-booléenne dont les coefficients sont non négatifs, soumis à une contrainte de capacité (linéaire).
De nombreux travaux ont été menés afin de résoudre ce problème. Parmi ces méthodes deux approches se distinguent : une approche quadratique, consistant à résoudre directement (QKP) (voir [54], [55]) ; une approche par linéarisation (voir [22]), consistant à transformer (QKP) en un programme linéaire équivalent. Notons que chacune des approches se focalisent sur la recherche d'un bon majorant pour (QKP), dans un contexte de maximisation.
Ce travail se situe dans un contexte de linéarisation. Contrairement aux linéarisations classiques, notre approche ajoute au problème initial uniquement une nouvelle variable réelle t. En effet, nous transformons tout d'abord (QKP) en un problème équivalent $(LP(Q))$, en utilisant une variable supplémentaire et une contrainte quadratique. Nous remplaçons ensuite la contrainte quadratique par un ensemble de contraintes linéaires, dérivé de la caractérisation de l'enveloppe convexe du problème quadratique 0-1.
Le nombre de contraintes étant trop important, nous suggérons alors une technique de génération de certaines contraintes : une borne est alors obtenue. La linéarisation proposée offre trois possibilités de calculer un majorant pour (QKP). Nous avons donc mené dans un premier temps une étude expérimentale afin de sélectionner la meilleure des trois possibilités (en terme de valeur de borne).
Cette borne est ensuite incorporée dans une méthode arborescente exacte. Par ailleurs, une solution admissible est calculée en utilisant l'heuristique proposée par Billionnet et Calmels [56].
Les expérimentations numériques sont menées afin de valider la qualité de la borne proposée ainsi que l'efficacité de la méthode exacte. Notre borne est comparée à celle suggérée par Billionnet et al. (1999) [54], qui est à ce jour la meilleure borne connue. Enfin, notre méthode exacte est comparée à celle développée par Pisinger et al. [55] (meilleure méthode exacte connue) qui incorpore en chaque noeud de l'arborescence la borne calculée par [17]. Notre borne est compétitive avec celle de [54] pour (QKP) (moins de 1% par rapport à l'optimum). Aussi, notre *branch-and-bound* améliore clairement les résultats obtenus par [55] pour les instances de faible densité (25% et50%) pour les problèmes allant jusqu'à 400 variables.

4.1 INTRODUCTION

(QKP) est le problème le plus connu et traité des programmes quadratiques. Nous ne reprenons pas dans ce chapitre l'état de l'art s'y rapportant, établi au chapitre 2. Le travail présenté ici se situe dans la classe de méthodes de résolution que nous avons notée (i) dans le chapitre 2 section 2.2. C'est à dire que nous proposons une linéarisation pour (QKP) reposant sur l'ajout d'une unique variable supplémentaire et sur un nombre exponentiel de contraintes linéaires (dérivées de la caractérisation de l'enveloppe convexe du problème quadratique 0-1. Nous appelons la linéarisation : t-linéarisation.

Ce chapitre s'organise comme suit. La prochaine section est dédiée à la t-linéarisation. Celle-ci engendre trois majorants possibles pour (QKP) que nous détaillons dans la section 4.2. La section 4.4 propose une étude expérimentale pour déterminer quel est le plus adapté des majorants à incorporer par la suite dans l'algorithme de recherche arborescente. La section 4.5 propose une voie d'amélioration du calcul des coefficients des inégalités valides générées (cf. algorithme de génération de contraintes). La section 4.6 est dédiée à

notre algorithme de *branch-and-bound*. les expérimentations numériques sont menées à la section 4.7. Nous concluons à la section 4.8.

Nous utilisons les notations établies au chapitre 1 auxquelles nous ajoutons la notation suivante : soit un ensemble S, $Co(S)$ représente l'enveloppe convexe de S.

4.2 UN CADRE DE t-LINÉARIZATION

Le cadre de t-linéearisation, que nous proposons, se déroule en deux étapes principales. Tout d'abord, nous remplaçons les termes quadratiques de la fonction objectif de (QKP) à l'aide d'une variable réelle t et nous ajoutons une contrainte quadratique. La seconde étape consiste en la ré-écriture de la contrainte additionnelle quadratique en un ensemble de contraintes linéaires. Ces contraintes proviennent de la caractérisation de l'enveloppe convexe du problème quadratique ainsi obtenu. Cette caractérisation est l'un des résultats principaux de ce travail.

Commençons par ré-écrire (QKP) tel que

$$(LP(Q)) \begin{cases} \max\ t + \sum_{i=1}^{n} c_i x_i \\ s.c. \left| \begin{array}{l} \sum_{i=1}^{n} a_i x_i \leq b \\ t \leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i x_j \\ t \in R,\ x_i \in \{0,1\}\ \ i = 1, ..., n \end{array} \right. \end{cases}$$

Les problèmes $(LP(Q))$ et (QKP) sont équivalents. La fonction objectif de $(LP(Q))$ est maintenant linéaire mais soumise à une contrainte de capacité à laquelle s'ajoute la contrainte quadratique.

La seconde étape traite donc de la linéarisation de la contrainte quadratique. Cette dernière peut être effectuée selon les deux possibilités suivantes : (i) en ne considérant que la contrainte quadratique (cf. sous-section 4.2.1) ; (ii) en considérant l'ensemble des solutions admissibles de $(LP(Q))$ dans son intégralité (cf. sous-section 4.2.2).

4.2.1 Linéarisation de la contrainte quadratique (uniquement)

Nous linéarisons ici la contrainte quadratique de (QKP) sans tenir compte de la contrainte de sac-à-dos. Introduisons l'ensemble suivant

$$X = \left\{ (x,t) \in R^{n+1} \mid\ t \leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i x_j,\ x \in \{0,1\}^n \right\} \tag{4.2.1}$$

En supposant que $Co(X)$ peut être entièrement décrit et en imposant que $x \in \{0,1\}^n$, le programme linéaire suivant est alors équivalent à (QKP)

$$(LP(X)) \begin{cases} \max\ t + \sum_{i=1}^{n} c_i x_i \\ s.c. \left| \begin{array}{l} \sum_{i=1}^{n} a_i x_i \leq b \\ (x,t) \in Co(X) \\ t \in R,\ x \in \{0,1\}^n \end{array} \right. \end{cases}$$

Le théorème 4.1 décrit l'enveloppe convexe de X, $Co(X)$, avec des contraintes linéaires.

Theorème 4.1 *Quadric Polytope*

Considérons l'ensemble suivant

$$QP_n = \left\{ (x,t) \in R^{n+1} \mid t \leq \sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{\pi(j)\pi(i)} x_{\pi(i)},\ x \in [0,1]^n \ \forall\ \pi\ \in\ S_n \right\}$$

où S_n est l'ensemble de toutes les permutations de $\{1, ..., n\}$.

Nous avons $QP_n = Co(X)$.

Preuve 4.2 *Montrons tout d'abord que $QP_n \subset Co(X)$.*

Soit $(x,t) \in QP_n$, nous devons montrer que

$$\exists\, p \in \mathbf{N},\ (x^i,t^i) \in X\ ,\ \beta_i \geq 0\ (0 \leq i \leq p) \mid \sum_{i=0}^{p} \beta_i (x^i,t^i) = (x,t)\ ,\ \sum_{i=0}^{p} \beta_i = 1.$$

Sans perte de généralité, nous supposons que les composantes de x sont triées par ordre décroissant. Soient $\alpha_1, \alpha_2, ..., \alpha_m$ les valeurs (également triées par ordre décroissant) prises par les composantes de x. Alors

$$x = (\alpha_1, \alpha_1, ..., \alpha_1, \alpha_2, \alpha_2, ..., \alpha_2, ..., \alpha_m, \alpha_m, ..., \alpha_m).$$

Plus précisemment, soit k_j le plus grand indice et l un indice tel que $x_l = \alpha_j$ ($l \in \{1, ..., n\}$ et $j \in \{1, ..., m\}$), on a

$$x_l = \alpha_j \quad , \quad k_{j-1} < l \leq k_j$$

avec $k_0 = 0$ et $\alpha_0 = 1 - \alpha_1$.

Puis, notons x^0 le vecteur de $\mathbf{R}^n$ dont toutes les composantes sont nulles et notons x^i ($1 \leq i \leq m$) le point de $\mathbf{R}^n$ defini par

$$x_l^i = \begin{cases} 1, & \textit{if } l \leq k_i \\ 0, & \textit{sinon} \end{cases}$$

Nous avons ensuite $x = \sum_{i=1}^{m-1} (\alpha_i - \alpha_{i+1}) x^i + \alpha_m x^m + \alpha_0 x^0$. Ainsi, en considérant $p = m$ et

$$\beta_i = \begin{cases} \alpha_0 = 1 - \alpha_1, & \textit{if } i = 0 \\ \alpha_i - \alpha_{i+1}, & \textit{if } 1 \leq i \leq p-1 \\ \alpha_m, & \textit{if } i = p \end{cases}$$

ce qui implique que $x = \sum_{i=0}^{p} \beta_i x^i$ avec $\beta_i \geq 0$ (comme $1 \geq \alpha_1 > ... > \alpha_m \geq 0$) et $\sum_{i=0}^{p} \beta_i = \beta_0 + (\alpha_1 - \alpha_2) + (\alpha_2 - \alpha_3) + ... + \alpha_m = \beta_0 + \alpha_1 = 1$.

Reste à démontrer que $\exists\, t^i\,, (i = 1, \ldots, p)$ *tel que* $\left(x^i, t^i\right) \in X$ *et* $t = \sum_{i=0}^{p} \beta_i t^i$.

Comme $(x, t) \in QP_n$ *nous avons*

$$t \leq \sum_{k=2}^{n} \left(\sum_{j=1}^{k-1} q_{\pi(j)\pi(k)} \right) x_{\pi(k)} \; \forall \pi \in S_n$$

Pour la permutation identité on a

$$t \leq \sum_{k=2}^{n} \left(\sum_{j=1}^{k-1} q_{jk} \right) x_k$$

Définissons δ, *tel que* $\delta = \sum_{k=2}^{n} \left(\sum_{j=1}^{k-1} q_{jk} \right) x_k - t$. *Alors* $\delta \geq 0$ *et*

$$t = -\delta + \sum_{k=2}^{n} \left(\sum_{j=1}^{k-1} q_{jk} \right) x_k$$

Donc

$$\begin{aligned} t &= -\left(\sum_{i=0}^{p} \beta_i\right) \delta + \sum_{k=2}^{n} \left(\sum_{j=1}^{k-1} q_{jk} \right) \left(\sum_{i=0}^{p} \beta_i x_k^i \right) \\ &= \sum_{i=0}^{p} \beta_i \left(-\delta + \sum_{k=2}^{n} \left(\sum_{j=1}^{k-1} q_{jk} \right) x_k^i \right) \end{aligned}$$

En définissant $t^i = -\delta + \sum_{k=2}^{n} \left(\sum_{j=1}^{k-1} q_{jk} \right) x_k^i$ *nous avons* $t = \sum_{i=0}^{p} \beta_i t^i$. *Ainsi,* $QP_n \subset Co(X)$.

Dans l'autre sens, nous devons montrer que $Co(X) \subset QP_n$. *En ce sens, il est suffisant d'établir que les inégalités définies pour* QP_n *sont valides pour* X, *ou plus formellement nous devons établir que*

$$\forall (x, t) \in X, \; \forall \pi \in S_n \quad t \leq \sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{\pi(j)\pi(i)} x_{\pi(i)}.$$

Comme $(x, t) \in X$, *nous avons* $t \leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i x_j$ *et alors il est suffisant de montrer que*

$$\forall (x, t) \in X, \; \forall \pi \in S_n \; \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i x_j \leq \sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{\pi(j)\pi(i)} x_{\pi(i)}.$$

Ce qui est facilement démontrer en établissant que

$$\begin{aligned} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i x_j &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{\pi(i)\pi(j)} x_{\pi(i)} x_{\pi(j)} \\ &= \sum_{i=2}^{n} x_{\pi(i)} \left(\sum_{j=1}^{i-1} q_{\pi(i)\pi(j)} x_{\pi(j)} \right) \end{aligned}$$

pour chaque permutation π *et en prenant en compte que* $x_{\pi(j)} \in \{0, 1\}$ *et* $q_{\pi(j)\pi(i)} \geq 0$ *(tel que* $q_{\pi(j)\pi(i)} x_{\pi(j)} \leq q_{\pi(j)\pi(i)}$*).*

Nous pouvons maintenant remplacer $Co(X)$ par l'expression fournie par le Theorème 4.1, nous pouvons alors transformer $(LP(Q))$ en un programme linéaire en variables $0-1$ (LP), équivalent à (QKP)

$$(LP) \begin{cases} \max\ t + \sum_{i=1}^{n} c_i x_i \\ s.c. \left| \begin{array}{l} \sum_{i=1}^{n} a_i x_i \leq b \\ t \leq \sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{\pi(j)\pi(i)} x_{\pi(i)},\ \forall \pi \in S_n \\ t \in \mathbf{R},\ x_i \in \{0,1\}\ ,\ i = 1, ..., n \end{array} \right. \end{cases}$$

Ainsi, le problème quadratique initial a été converti en un programme linéaire équivalent en utilisant une unique variable de décision supplémentaire mais avec l'ajout de $n!$ contraintes.

Bien heureusement, il n'est pas nécessaire de générer toutes les contraintes pour résoudre (LP). En effet, seul un sous ensemble de contraintes adéquates va être généré itérativement, et ce au moyen d'un algorithme de génération de contraintes. Ce type d'algorithme, résout (LP) (ou $(\overline{LP})$), de manière itérative, en recherchant une contrainte linéaire valide pour une formulation du problème original et violée par la solution trouvée pour la relaxation courante $(\overline{x}, \overline{t})$, puis le cas échéant va ajouter une telle contrainte (voire plusieurs) dans la relaxation courante. Ce processus est réeitéré jusqu'à ce qu'aucune contrainte violée ne soit trouvée pour la solution de la relaxation courante, celle-ci constituant alors une solution optimale du problème original (QKP).

Le problème consistant à trouver une telle inégalité est connu sous le nom de Problème de Séparation. Le theorème 4.3 établit que ceci peut être fait en temps polynomial pour $Co(X)$.

Theorème 4.3 *Problème de séparation*

Soit $(\overline{x}, \overline{t}) \in [0,1]^n \times R$, trouver une inégalité qui sépare $(\overline{x}, \overline{t})$ de $Co(X)$ ou prouver que $(\overline{x}, \overline{t}) \in Co(X)$ peut être effectué en $O(m + n \log n)$ où m est le nombre de coefficients positifs q_{ij}.

Preuve 4.4 *Soit $\overline{S}_n = \{\pi \in S_n \mid \overline{x}_{\pi(1)} \geq \overline{x}_{\pi(2)} \geq ... \geq \overline{x}_{\pi(n)}\}$. Nous montrons que*

$$\sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{\overline{\pi}(j)\overline{\pi}(i)} \overline{x}_{\overline{\pi}(i)} = \min_{\pi \in S_n} \sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{\pi(j)\pi(i)} \overline{x}_{\pi(i)}\ ,\ \forall\, \overline{\pi} \in \overline{S}_n \tag{4.2.2}$$

Démontrer cette propriété équivaut à montrer que les éléments de $\overline{S}_n$ minimisent le membre de droite des contraintes (RHS) de (QP_n) pour la solution $(\overline{x}, \overline{t})$. Nous en déduisons qu'aucune autre permutation, séparant la solution de $Co(X)$, ne peut être trouvée. Deux étapes sont nécessaires pour compléter la preuve.

(i) Montrons d'abord que

$$\forall\, \pi \notin \overline{S}_n,\ \exists\, \overline{\pi} \in \overline{S}_n \mid \sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{\overline{\pi}(j)\overline{\pi}(i)} \overline{x}_{\overline{\pi}(i)} \leq \sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{\pi(j)\pi(i)} \overline{x}_{\pi(i)}. \tag{4.2.3}$$

Chaque permutation dans $\overline{S}_n$ vérifie $\overline{x}_{\pi(i)} \geq \overline{x}_{\pi(i+1)}$, $i = 1, 2, ..., n-1$.

Ainsi, $\pi \notin S_n \Rightarrow \exists\, k \in \{1, ..., n-1\} \mid \overline{x}_{\pi(k)} < \overline{x}_{\pi(k+1)}$. En conséquence, considérons la permutation élémentaire $\pi^{'}$, obtenue par π, en échangeant deux valeurs consécutives $\pi(k)$ et $\pi(k+1)$. Nous avons

$$\pi^{'}(k) = \pi(k+1), \pi^{'}(k+1) = \pi(k), \pi^{'}(i) = \pi(i), i \neq k, k+1$$

En écrivant en détails (RHS), nous obtenons

$$\begin{aligned}\sum_{i=2}^{n}\sum_{j=1}^{i-1} q_{\pi'(j)\pi'(i)}\overline{x}_{\pi'(i)} = & \sum_{i=2}^{k-1}\sum_{j=1}^{i-1} q_{\pi'(j)\pi'(i)}\overline{x}_{\pi'(i)} + \sum_{j=1}^{k-1} q_{\pi'(j)\pi'(k)}\overline{x}_{\pi'(k)} + \\ & \sum_{j=1}^{k-1} q_{\pi'(j)\pi'(k+1)}\overline{x}_{\pi'(k+1)} + q_{\pi'(k)\pi'(k+1)}\overline{x}_{\pi'(k+1)} + \\ & \sum_{i=k+2}^{n}\Big[\sum_{\substack{j=1\\ j\neq k,k+1}}^{k-1} q_{\pi'(j)\pi'(i)}\overline{x}_{\pi'(i)} + q_{\pi'(k)\pi'(i)}\overline{x}_{\pi'(i)} \\ & + q_{\pi'(k+1)\pi'(i)}\overline{x}_{\pi'(i)}\Big]\end{aligned}$$

Par définition de $\pi^{'}$, nous avons

$$\begin{aligned}\sum_{i=2}^{n}\sum_{j=1}^{i-1} q_{\pi'(j)\pi'(i)}\overline{x}_{\pi'(i)} = & \sum_{i=2}^{k-1}\sum_{j=1}^{i-1} q_{\pi(j)\pi(i)}\overline{x}_{\pi(i)} + \sum_{j=1}^{k-1} q_{\pi(j)\pi(k+1)}\overline{x}_{\pi(k+1)} + \\ & \sum_{j=1}^{k-1} q_{\pi(j)\pi(k)}\overline{x}_{\pi(k)} + q_{\pi(k+1)\pi(k)}\overline{x}_{\pi(k)} + \\ & \sum_{i=k+2}^{n}\Big[\sum_{\substack{j=1\\ j\neq k,k+1}}^{k-1} q_{\pi(j)\pi(i)}\overline{x}_{\pi(i)} + q_{\pi(k+1)\pi(i)}\overline{x}_{\pi(i)} \\ & + q_{\pi(k)\pi(i)}\overline{x}_{\pi(i)}\Big]\end{aligned}$$

En utilisant la symétrie, le fait que q soit non négatif et que $\overline{x}_{\pi(k)} < \overline{x}_{\pi(k+1)}$, il suit que
$[\sum_{i=2}^{n}\sum_{j=1}^{i-1} q_{\pi'(j)\pi'(i)}\overline{x}_{\pi'(i)} - \sum_{i=2}^{n}\sum_{j=1}^{i-1} q_{\pi(j)\pi(i)}\overline{x}_{\pi(i)}] = q_{\pi(k+1)\pi(k)}[\overline{x}_{\pi(k)} - \overline{x}_{\pi(k+1)}] \leq 0$

Si $\pi^{'} \in \overline{S}_n$ alors le résultat (4.2.3) est montré en prenant $\overline{\pi} = \pi^{'}$. Sinon, on réitère sur $\pi^{'}$ une permutation élémentaire (comme décrite ci dessus), autant de fois que nécessaire, jusqu'à atteindre une permutation $\overline{\pi} \in \overline{S}_n$. A chaque échange, RHS décroît, et le résulat est démontré. Observons que soit les valeurs décroissent strictement, si $q_{\pi(k+1)\pi(k)} > 0$, ou bien restent identiques.

(ii) Maintenant, montrons que

$$\sum_{i=2}^{n}\sum_{j=1}^{i-1} q_{\pi(j)\pi(i)}\overline{x}_{\pi(i)} = \sum_{i=2}^{n}\sum_{j=1}^{i-1} q_{\pi'(j)\pi'(i)}\overline{x}_{\pi'(i)} \;\; \forall\, \pi, \pi^{'} \in \overline{S}_n, \pi \neq \pi^{'}$$

Notons $\alpha_1, \alpha_2, ..., \alpha_m$, les m valeurs distinctes, triées par ordre décroissant, prises par les composantes de $\overline{x}$. Par définition de π et $\pi^{'}$, et si nous notons k_j le plus grand indice l tel que $\overline{x}_{\pi(j)} = \alpha_j$ ($l \in \{1, ..., n\}$ et $j \in \{1, ..., m\}$), alors nous avons

$$\overline{x}_{\pi(l)} = \overline{x}_{\pi'(l)} = \alpha_j \quad , \quad k_{j-1} < l \leq k_j, \; avec\, k_0 = 0.$$

Notons $\pi_{|k_j}$ (resp. $\pi'_{|k_j}$) la restriction de π (resp. π') à l'ensemble $\{k_{j-1}+1, ..., k_j\}$.

$$i.e\ \ \pi_{|k_j} : \ \begin{array}{rcl} \{k_{j-1}+1, ..., k_j\} & \rightarrow & \{1, ..., n\} \\ l & \mapsto & \pi(l) \end{array}$$

Comme $\pi, \pi' \in \overline{S}_n \Rightarrow Im(\pi_{|k_j}) = Im(\pi'_{|k_j})$ où $Im(...)$ correspondent aux images de ...

En utilisant les mêmes arguments de calculs que dans (i), nous savons que pour chaque $j = 1, 2, ..., m$ et pour chaque $i \in \{k_{j-1}+1, ..., k_j - 1\}$, la permutation élémentaire $\pi(i)$ et $\pi(i+1)$ laisse identique l'expression de $\sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{\pi(j)\pi(i)} \overline{x}_{\pi(i)}$, car $\overline{x}_{\pi(i)} = \overline{x}_{\pi(i+1)}$.

En utilisant des résultats de base d'algèbre sur les permutations de groupes S_n, on peut voir que pour chaque j on a $\pi'_{|k_j}$ à partir de $\pi_{|k_j}$ en appliquant, on π, une composition de permutations élémentaires d'indices consécutifs. Comme chacune d'entre elles laisse inchangée l'expression de le résultat en découle.

Ainsi on obtient la permutation dans $\overline{S}_n$ de même valeur RHS. De plus, cette valeur est plus petite ou égale (cf. point (i)) à chaque autre permutation obtenue en dehors de $\overline{S}_n$. Les résultats (4.2.2) sont alors montrés. Puisque trier $\overline{x}$ peut être fait en $O(n \log n)$ et calculer les coefficients peut être fait en au plus m opérations, la complexité du problème de séparation est en $O(m + n \log n)$.

4.2.2 Linéarisation de l'ensemble des solutions admissibles de $(LP(Q))$ dans son intégralité

Nous présentons maintenant la linéarisation de l'ensemble des contraintes (QKP) comptant la contrainte quadratique et la contrainte de capacité. Introduisons l'ensemble Y de points (x, t) comprenant les deux contraintes.

$$Y = \left\{ (x,t) \in R^{n+1} \mid \ t \leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i x_j, \ \sum_{i=1}^{n} a_i x_i \leq b, \ x \in \{0,1\}^n \right\} \tag{4.2.4}$$

Le lemme suivant établit la relation stricte entre l'ensemble des solutions admissibles de (LP) et Y. De plus, comme (LP) et (QKP) sont équivalents, alors (Y) est ensemble de solutions admissibles pour (QKP).

Lemme 4.5 *Y constitue un ensemble de solutions admissibles pour (LP).*

Preuve 4.6 *Soient X_{LP} l'ensemble de solutions admissibles de (LP) et $(\bar{x}, \bar{t}) \in \mathbb{R}^{n+1}$. Pour montrer que $X_{LP} = Y$ il suffit de montrer que*

$$\bar{t} \leq \sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{ij} \bar{x}_i \bar{x}_j \Leftrightarrow \bar{t} \leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{\pi(j)\pi(i)} \bar{x}_{\pi(j)}, \ \forall \pi \in S_n$$

Soit $\bar{\pi}$ tel que les composantes de $\bar{x}$ soient triées par ordre décroissant. Notons que

$$\sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{ij} \bar{x}_i \bar{x}_j = \sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{\bar{\pi}(j)\bar{\pi}(i)} \bar{x}_{\bar{\pi}(i)}$$

et, $\bar{\pi}$ est une permutation qui minimise

$$\sum_{i=2}^{n}\sum_{j=1}^{i-1} q_{\pi(j)\pi(i)}\bar{x}_{\pi(i)}.$$

Ainsi, le résultat est démontré.

Corollaire 4.7 *Co(Y) constitue l'enveloppe convexe de l'ensemble des solutions réalisables de (LP).*

Preuve 4.8 *La preuve est une conséquence immédiate du Lemme 4.5.*

Définissons maintenant $(LP(Y))$ tel que

$$(LP(Y)) \begin{cases} \max\ t + \sum\limits_{i=1}^{n} c_i x_i \\ s.c. \big| \quad (x,t) \in Co(Y) \end{cases}$$

nous pouvons alors établir le corollaire suivant 4.9.

Corollaire 4.9 $V(QKP) = V(LP) = V(LP(X)) = V(LP(Y))$.

où nous rappelons que $V(P)$ représente la valeur optimale du problème (P) (cf. notations établies au **chapitre 1**).

Comme Y constitue l'ensemble des solutions réalisables du problème initial traité, il est donc intéressant d'établir sa description linéaire. Nous proposons en ce sens, une procédure de génération d'inégalités valides pour $Co(Y)$ basée sur la même approche que celle concernant $Co(X)$.

Considérons les inégalités suivantes qui décrivent $Co(X)$

$$t \leq \sum_{i=1}^{n} \alpha_{\pi(i)} x_{\pi(i)}, \forall \pi \in S_n, \tag{4.2.5}$$

où $\alpha_{\pi(i)}$ peut être défini comme

$$\alpha_{\pi(i)} = max \left\{ \sum_{j=1}^{i-1} q_{\pi(j)\pi(i)} x_{\pi(j)} \ :\ x_{\pi(j)} \in \{0,1\}, j < i,\ x_{\pi(i)} = 1 \right\} \tag{4.2.6}$$

Remarque 4.10 *$\alpha_{\pi(i)}$ n'est à prendre en compte dans (4.2.5) que si $x_{\pi(i)} = 1\ \forall i = 1, ..., n$.*

De façon analogue nous définissons les inégalités qui approximent $Co(Y)$

$$t \leq \sum_{i=1}^{n} \beta_{\pi(i)} x_{\pi(i)}, \forall \pi \in S_n, \tag{4.2.7}$$

où $\beta_{\pi(i)}$ peut être défini comme

$$\beta_{\pi(i)} = \max \left\{ \sum_{j=1}^{i-1} q_{\pi(j)\pi(i)} x_{\pi(j)} \ : \ \sum_{j=1}^{i-1} a_{\pi(j)} x_{\pi(j)} \leq b - a_{\pi(i)},\ x_{\pi(j)} \in \{0,1\},\ j < i \right\}. \tag{4.2.8}$$

Le lemme 4.11 montre que les inégalités (4.2.7) sont valides pour Y.

Lemme 4.11 *Soit $\pi \in S_n$ une permutation et $\beta_\pi \in R^n$ tels que*

$$\beta_{\pi(i)} = \max \left\{ \sum_{j=1}^{i-1} q_{\pi(j)\pi(i)} x_{\pi(j)} \ : \ \sum_{j=1}^{i-1} a_{\pi(j)} x_{\pi(j)} \leq b - a_{\pi(i)},\ x_{\pi(j)} \in \{0,1\},\ j < i \right\}.$$

Si $(\bar{x}, \bar{t}) \in Y$ alors $\bar{t} \leq \sum_{i=1}^{n} \beta_{\pi(i)} \bar{x}_{\pi(i)}$.

Preuve 4.12 *Soit $(\bar{x}, \bar{t}) \in Y$. Pour tous sous ensembles d'indices $I \subseteq \{1, \cdots, n\}$, nous avons*

$$\sum_{j \in I} a_j \bar{x}_j \leq b.$$

Ainsi, $\forall\ i = 1, \cdots, n$ nous pouvons établir la relation suivante

$$\sum_{j=1}^{i-1} q_{\pi(j)\pi(i)} \bar{x}_{\pi(j)} \leq \max \left\{ \sum_{j=1}^{i-1} q_{\pi(j)\pi(j)} x_{\pi(j)} \ : \ \sum_{j=1}^{i} a_{\pi(j)} x_{\pi(j)} \leq b,\ x \in \{0,1\}^i \right\}.$$

En particulier si $\bar{x}_{\pi(i)} = 1$, nous avons

$$\sum_{j=1}^{i-1} q_{\pi(j)\pi(i)} \bar{x}_{\pi(j)} \leq \beta_{\pi(i)}$$

D'autre part, $(\bar{x}, \bar{t}) \in Y$ implique que

$$\bar{t} \leq \sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{ji} \bar{x}_i \bar{x}_j = \sum_{i=2}^{n} \sum_{j=1}^{i-1} q_{\pi(j)\pi(i)} \bar{x}_{\pi(j)} \bar{x}_{\pi(i)} \leq \sum_{i=1}^{n} \beta_{\pi(i)} \bar{x}_{\pi(i)}$$

où la second inégalité est satisfaite puisque la remarque 4.10 est vérifiée pour les coefficients $\beta_{\pi(i)}$ $\forall i = 1, ..., n$.

Le calcul des coefficients β nécessite la résolution d'un problème de sac-à-dos classique, qui est un problème NP-difficile. Ainsi, par souci de gain de temps CPU lors de la mise en oeuvre de la méthode, nous avons considéré la version relâchée ($\bar{\beta}$) où les variables x_j prennent leurs valeurs dans $[0, 1]$ (cf. (4.2.8)). Ainsi, il est évident que comme (4.2.7) est valide pour $Co(Y)$ alors

$$t \leq \sum_{i=1}^{n} \bar{\beta}_{\pi(i)} x_{\pi(i)}, \forall \pi \in S_n, \tag{4.2.9}$$

sont aussi valides pour $Co(Y)$ (il est aisé de montrer que chaque coefficient dans β_π est inférieur ou égal à ceux dans $\bar{\beta}_\pi$ pour toute permutation π).

D'après (4.2.5), (4.2.7) et (4.2.9) nous pouvons définir les problèmes associés (LP_α), (LP_β) et $(LP_{\bar{\beta}})$ qui peuvent être réécrits comme (LP) où les contraintes associées à t sont remplacées par (4.2.5), (4.2.7) et (4.2.9) respectivement.

Lemme 4.13 *Soient X_α, X_β et $X_{\bar{\beta}}$ les ensembles des solutions admissibles des problèmes (LP_α), (LP_β) et $(LP_{\bar{\beta}})$ respectivement. Alors $X_\beta \subseteq X_{\bar{\beta}} \subseteq X_\alpha$.*

Preuve 4.14 *Pour montrer ce résultat il suffit de montrer que pour une permutation $\pi \in S_n$, pour tout $x \in [0,1]^n$ nous avons*

$$\sum_{i=1}^{n} \beta_{\pi(i)} x_{\pi(i)} \leq \sum_{i=1}^{n} \bar{\beta}_{\pi(i)} x_{\pi(i)} \leq \sum_{i=1}^{n} \alpha_{\pi(i)} x_{\pi(i)}. \tag{4.2.10}$$

Montrons que

$$\beta_{\pi(i)} \leq \bar{\beta}_{\pi(i)} \leq \alpha_{\pi(i)}, \ \forall i = 1, \cdots, n \tag{4.2.11}$$

est suffisant pour démontrer les inégalités (4.2.10).

Pour montrer (4.2.11), étudions le problème de maximisation suivant pour chaque indice i. Par définition, nous savons que$\beta_{\pi(i)} \leq \bar{\beta}_{\pi(i)}$. Prouvons ensuite que $\bar{\beta}_{\pi(i)} \leq \alpha_{\pi(i)}$. Notons que, si nous relâchons l'intégrité des variables dans (4.2.6), le résultat serait le même. Alors, la seule différence entre les problèmes relatifs à $\bar{\beta}_{\pi(i)}$ ou à $\alpha_{\pi(i)}$ est l'ajout ou non de la contrainte de sac à dos. Donc, $\alpha_{\pi(i)}$ est calculée via la relaxation du problème qui définit que $\bar{\beta}_{\pi(i)}$ et le résultat est démontré.

Corollaire 4.15 $V(LP_\alpha) = V(LP_{\bar{\beta}}) = V(LP_\beta) = V(QKP)$.

Preuve 4.16 *Comme le Lemme 4.5 montre que Y est ensemble admissible pour (LP) et*

$$Co(Y) \subseteq \left\{ (x,t) \in R^{n+1} \mid t \leq \sum_{i=1}^{n} \beta_{\pi(i)} x_{\pi(i)}, \forall \pi \in S_n, \ \sum_{i=1}^{n} a_i x_i \leq b, \ x \in \{0,1\}^n \right\}$$

nous avons que $V(LP) \leq V(LP_\beta)$. Aussi, d'après le lemme 4.13 :

$$V(LP) \leq V(LP_\beta) \leq V(LP_{\bar{\beta}}) \leq V(LP_\alpha) = V(LP).$$

D'après le corollaire 4.9, $V(LP) = V(QKP)$et el résultat est démontré.

Corollaire 4.17 *Soient $\overline{X_\alpha}$, $\overline{X_\beta}$ et $\overline{X_{\bar{\beta}}}$ les ensembles des solutions admissibles des problèmes $(\overline{LP_\alpha})$, $(\overline{LP_\beta})$ et $(\overline{LP_{\bar{\beta}}})$ respectivement. Alors $\overline{X_\beta} \subseteq \overline{X_{\bar{\beta}}} \subseteq \overline{X_\alpha}$.*

Preuve 4.18 *L'argument du Lemme 4.13 est vérifié ici.*

Corollaire 4.19 $V(\overline{LP_\alpha}) \geq V(\overline{LP_{\bar{\beta}}}) \geq V(\overline{LP_\beta}) \geq V(QKP)$

Les résultats théoriques présentés dans cette section induisent plusieurs possibilités pour établir une méthode de résolution pour (QKP) *via* la t-linéarisation. Ces différentes voies consistent en la ré-écriture du problème initial sous la forme (LP_α) ou (LP_β) ou $(LP_{\bar{\beta}})$. Ces problèmes sont NP-difficiles et difficiles à résoudre en pratique puisqu'ils comptent un nombre exponentiel de variables de décision.

4.3 RÉSOUDRE (QKP) *via* LA t-LINÉARISATION.

Le problème (LP) résultant est linéaire mais le nombre de contraintes croît exponentiellement en fonction de la taille du problème. Pour traiter cet inconvénient, nous développons donc un algorithme de génération de contraintes. Ce dernier débute avec un ensemble de contraintes restreint, résout le problème avec cet ensemble de contraintes et ajoute itérativement les contraintes nécessaires (2.3) au problème.

Afin de résoudre les différentes formulations présentées dans la section 4.2, nous utilisons l'algorithme 2. Il est établi ici dans le contexte (LP_α). Cette procédure fournit la solution optimale de (QKP) quand elle est appliquée à (LP_α) (cf. Théorème 4.3), et un majorant pour (QKP) quand elle est appliquée à (LP_β) ou à $(LP_{\bar{\beta}})$.

1. Soit x_H une solution admissible fournie par l'heuristique de Billionnet et Calmels [56];
2. $\pi_1 = (\pi_1(1)), ..., \pi_1(i), ..., \pi_1(n))$ est une permutation telle que : $x^H_{\pi_1(1)} \geq x^H_{\pi_1(2)} \geq ... \geq x^H_{\pi_1(n)}$;
La contrainte correspondante est ajoutée à (LP_α), i.e. $t \leq \alpha_\pi x$;
3. Résoudre (LP_α) résultant;
Une nouvelle permutation est obtenue à travers la solution optimale du précédent (LP_α);
3. Déterminer la contrainte basée sur la précédente permutation;
4. Cette contrainte est elle violée ?;
if la contrainte est violée **then**
  ajout de la contrainte au problème et retour à l'étape 2 ;
else
  arrêt ;
end

Algorithm 2: Un algorithme de génération de contraintes pour (QKP)

L'algorithme 2 peut également être appliqué à la relaxation de (LP), comme le montre la Figure 4.3. Appelons cette procédure de relaxation, t-relaxation. Le principal avantage de prendre en considération la relaxation de (LP) appropriée est que le problème peut être résolu en temps linéaire à chaque itération.

Même si les résultats théoriques (voir le corollaire 4.19) montrent une stricte relation entre les bornes proposées pour (QKP) (en terme de valeur objective), ceci n'apparaît pas si clairement en pratique.

Nous avons mené une étude expérimentale préliminaire dans la section suivante (Section 4.4) afin de déterminer quelle procédure est la plus adéquate pour calculer un majorant fin pour (QKP) ; majorant qui sera ensuite incorporé dans un algorithme de *branch-and-bound* (cf. Section 4.6).

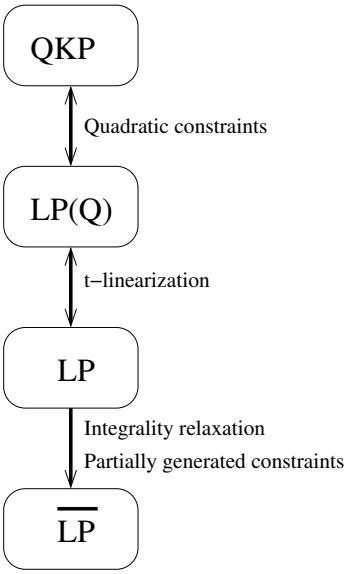

FIG. 4.1 – Transformations appliquées à QKP pour le calcul du majorant

4.4 EXPÉRIMENTATIONS NUMÉRIQUES PRÉLIMINAIRES

Dans cette section nous proposons de comparer expérimentalement les trois possibilités, détaillées à la section 4.3, pour obtenir un majorant fin pour (QKP), c'est-à-dire : (i) $V(\overline{LP_\alpha})$, (ii) $V(\overline{LP_\beta})$ et (iii) $V(\overline{LP_{\overline{\beta}}})$. De plus, puisque notre objectif est de résoudre de façon exacte (QKP), au moyen d'un *branch-and-bound*, nous avons besoin du majorant de meilleure qualité et obtenu en un temps CPU le plus rapide possible.
Les tests ont été menés sur 10 instances générées de la même manière que dans [17], [16], [57]. et [19]. Les coefficients c_i et q_{ij} de la fonction objectif sont des entiers uniformément générés entre 0 et 100, les poids a_i des contraintes sont des entiers uniformément générés entre 0 et 50. Le membre de droite des contraintes (rhs), b, est généré aléatoirement entre 50 et $max\{50, \sum_{i=1}^{n} a_i\}$. Chacune des 10 instances testées compte 100 variables et a 25% de densité au niveau des membres quadratiques.
Les trois méthodes de calcul de bornes ont été codées en langage C++ et les tests ont été réalisés sur une machine Pentium 4 2.66 GHz Intel et 1024 MB de RAM.
Le tableau 4.1 est composé de trois sous-tableaux pour chacune des bornes $V(\overline{LP_\alpha})$, $V(\overline{LP_\beta})$ et$V(\overline{LP_{\overline{\beta}}})$. Chaque sous tableau comprend : le numéro de l'instance, la valeur optimale, le temps CPU (en secondes) requis pour le calcul de la borne, la valeur de la borne et l'écart relatif ($\frac{UB-Opt}{Opt}$) en pourcentage. La qualité des trois majorants est vraiment proche ainsi que leurs temps CPU de calcul. Cependant, en moyenne le majorant $V(\overline{LP_{\overline{\beta}}})$ se comporte mieux que les deux autres $V(\overline{LP_\alpha})$ et $V(\overline{LP_\beta})$. De plus, nous savons que $V(\overline{LP_{\overline{\beta}}})$ correspond à un problème polynomial ce qui nous incite à employer $V(\overline{LP_{\overline{\beta}}})$ dans l'algorithme de *branch-and-bound* pour résoudre exactement (QKP).

4.5 AMÉLIORATION DU CALCUL DE MAJORANT SÉLECTIONNÉ

Notons que les contraintes fournies par $V(LP_\beta)$ et $V(LP_{\overline{\beta}})$ (cf. Section 4.2) ne sont pas suffisantes pour décrire complètement le domaine des solutions admissibles correspondant. Dans cette section nous proposons une façon de renforcer les inégalités valides de telle sorte à fournir une description plus fine de $Co(Y)$.

TAB. 4.1 – Comparaison de la qualité et du temps CPU de calcul des bornes

Borne Sup. calculée avec contraintes type α				
Instance	Valeur Optimale	Tps CPU (s)	Valeur Borne	GAP (%)
GHS100_25_1	18558	45.78	19124.28	3.05
GHS100_25_2	56525	14.06	56576.1	0,09
GHS100_25_3	3752	53.33	4063.68	8.31
GHS100_25_4	50382	77.5	51064.51	1.35
GHS100_25_5	61494	0.45	61621.96	0.21
GHS100_25_6	36360	188.59	36654.88	0.81
GHS100_25_7	14657	68.08	14854.07	1.34
GHS100_25_8	20452	18.95	20528,52	0.37
GHS100_25_9	35438	306.09	35487.01	0.14
GHS100_25_10	24930	42.47	25496.9	2.27
Moyenne		**81.53**		**1.80**
Borne Sup. calculée avec contraintes type β				
Instance	Valeur Optimale	Tps CPU (s)	Valeur Borne	GAP (%)
GHS100_25_1	18558	43.08	19124.07	3.05
GHS100_25_2	56525	14.5	56576.1	0.09
GHS100_25_3	3752	19.56	3904.35	4.06
GHS100_25_4	50382	74.36	51064.51	1.35
GHS100_25_5	61494	0.48	61621.96	0.21
GHS100_25_6	36360	188.83	36654.88	0.81
GHS100_25_7	14657	96.95	14853.89	1.34
GHS100_25_8	20452	18.17	20528.52	0.37
GHS100_25_9	35438	317.66	35487.01	0.14
GHS100_25_10	24930	42.7	25496.9	2,27
Moyenne		**81.6**		**1.37**
Borne Sup. calculée avec contraintes type $\bar{\beta}$				
Instance	Valeur Optimale	Tps CPU (s)	Valeur Borne	GAP (%)
GHS100_25_1	18558	41.91	19124.12	3.05
GHS100_25_2	56525	14.47	56576.1	0.09
GHS100_25_3	3752	12.38	3887.14	3.60
GHS100_25_4	50382	74.64	51064.51	1.35
GHS100_25_5	61494	0.48	61621.96	0.21
GHS100_25_6	36360	188.73	36654.88	0.81
GHS100_25_7	14657	88	14853.92	1.34
GHS100_25_8	20452	20.8	20528.52	0.37
GHS100_25_9	35438	310.95	35487.01	0.14
GHS100_25_10	24930	41.98	25496.9	2.27
Moyenne@		**79.43**		**1.32**

Rappelons les inégalités concernées :

$$t \leq \sum_{i=2}^{n} \overline{\beta}_{\pi(i)} x_{\pi(i)} \ \forall \pi \in S_n \tag{4.5.1}$$

Bien entendu, pour renforcer les inégalités valides (4.5.1) il est suffisant de calculer des coefficients plus fins $\overline{\beta}$. Notons $\beta^\star$ css nouveaux coefficients, tels que $\beta^\star_{\pi(i)} \leq \overline{\beta}_{\pi(i)}$, $\forall \pi \ i = 1, ..., n$.

Considérons ensuite le nouveau problème $(LP_{\beta^\star})$ pour lequel les coefficients des contraintes $\overline{\beta}$ sont remplacées par $\beta^\star$ et pour lequel les propriétés suivantes sont vérifiées selon le Lemme 4.13

1. $V(LP_{\beta^\star}) = V(QKP)$
2. $V(\overline{LP_{\beta^\star}}) \leq V(\overline{LP_{\bar{\beta}}})$

Sans perte de généralité, considérons la permutation identité pour présenter l'algorithme pour calculer $\beta^\star$.

Soit (x^*, t^*) solution optimale de (LP), alors $t^* = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i^* x_j^*$ qui peut se ré-écrire comme

$$t^* = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (q_{ij} - \delta_{ij}) x_i^* x_j^* + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \delta_{ij} x_i^* x_j^*, \text{ où}$$

où $\delta_{ij} \in \mathbb{R}$ $(1 \leq i < j \leq n)$.

En utilisant une transformation de base d'algèbre linéaire nous avons

$$t^* = \sum_{i=1}^{n} [\sum_{j=1}^{i-1} (q_{ji} - \delta_{ji}) x_j^* + \sum_{j=i+1}^{n} \delta_{ij} x_j^*] x_i^*. \tag{4.5.2}$$

Notre but est d'obtenir les coefficients $\beta^\star \in \mathbb{R}^n$ donnant la meilleure approximation des coefficients de x_i^* dans (4.5.2).
Définissons un problème par coefficient $\beta_i^\star$. En tenant compte de la remarque 4.10 traitons le problème suivant

$$(P_i(\delta)) \begin{cases} \max \sum_{j=1}^{i-1} (q_{ji} - \delta_{ji}) x_j + \sum_{j=i+1}^{n} \delta_{ij} x_j \\ s.c. \left| \begin{array}{l} \sum_{j=1}^{n} a_j x_j \leq b \\ x_i = 1 \\ x_j \in \{0,1\} \ i \neq j, \ j = 1, ..., n \end{array} \right. \end{cases}$$

Conséquence du lemme 4.11 : si on a $\beta_i(\delta) = V(P_i(\delta))$ alors

$$t \leq \sum_{i=1}^{n} \beta_i(\delta) x_i \ , \ \forall \ \delta \in \mathbb{R}^{\frac{n(n-1)}{2}}$$

sont des inégalités valides pour$Co(Y)$. Pour la cas particulier où $\delta = \mathbf{0}$ nous avons $\beta(\delta) = \beta$ et $V(\overline{P_i(\delta)}) = \bar{\beta}(\delta) = \bar{\beta}$.

Nous suggérons un algorithme pour améliorer δ, afin d'obtenir des meilleurs coefficients que $\bar{\beta}$. Le point clef de l'algorithme provient de l'observation que, comme $\overline{P_i(\delta)}$ est linéaire, la remarque suivante est vérifiée.

Remarque 4.20 *Pour tout programme linéaire, si on ajoute un coefficient à la fonction objectif, de chaque variable hors base, d'une quantité inférieure ou égale au coût réduit de chaque variable, alors la base optimale reste inchangée.*

Le theorème 4.21 montre comment δ peut être itérativement calculé pour chaque problème $(\overline{P_i(\delta)})$, en utlisant les coûts réduits provenant des problèmes précédents $(\overline{P_j(\delta)})$ pour $j < i$, $\forall i = 2, ..., n$ et $\forall j = 1, ..., n-1$.

Theorème 4.21 *Soit $\overline{c}_j^i(\delta)$ $(i,j=1,2...,n)$ coût réduit de x_j dans $\overline{P_i(\delta)}$ base optimale. Nous définissons la série de vecteurs $\delta^k=(\delta_{ij}^k)_{1\leq i<j\leq n}\in R^{\frac{n(n-1)}{2}}$, $k=1,2,...,n-1$, comme suit*

$$\delta_{ij}^k=\begin{cases}0, & & k=0\\ -\overline{c}_j^i(\delta^{k-1}), & i=k, & k=1,2,...,n-1\\ \delta_{ij}^{k-1}, & i\neq k, & k=1,2,...,n-1\end{cases}$$

Nous avons $\bar{\beta}_i(\delta^{n-1})\leq\bar{\beta}_i(\mathbf{0})$, $i=1,2,...,n$.

Preuve 4.22 *Considérons le problème*

$$(\overline{P_i(\delta^{n-1})})\begin{cases}\max \sum\limits_{j=1}^{i-1}(q_{ji}-\delta_{ji}^{n-1})x_j+\sum\limits_{j=i+1}^{n}\delta_{ij}^{n-1}x_j\\ s.c.\begin{vmatrix}\sum\limits_{j=1}^{n}a_jx_j\leq b\\ x_j\leq 1\ i\neq j,\ j=1,...,n\\ x_i=1\\ x_j\geq 0\ i\neq j,\ j=1,...,n\end{vmatrix}\end{cases}$$

Par définition, $\bar{\beta}_i(\delta^{n-1})=V(\overline{P_i(\delta^{n-1})})$.
De plus comme δ_{ij}^k par construction, nous savons que $\delta_{ji}^{n-1}=\delta_{ji}^{n-2}=\cdots=\delta_{ji}^i,\ j<i\ \forall i=2,...,n$, $\forall j=1,...,n-1$ et $\delta_{ij}^{n-1}=\delta_{ij}^{n-2}=\cdots=\delta_{ij}^i,\ i<j\ \forall j=2,...,n,\ \forall i=1,...,n-1$.

Donc $V(\overline{P_i(\delta^{n-1})})=V(\overline{P_i(\delta^i)})$.

Maintenant, examinons la fonction objectif de $(\overline{P_i(\delta^{i-1})})$

$$\sum_{j=1}^{i-1}(q_{ji}-\delta_{ji}^{i-1})x_j+\sum_{j=i+1}^{n}\delta_{ij}^{i-1}x_j$$

Nous avons ensuite

- *if $j=i-1$, $\delta_{ji}^{i-1}=\delta_{i-1,i}^{i-1}=-\overline{c}_i^{i-1}(\delta^{i-1})\geq 0$*
- *if $j=i-2$, $\delta_{ji}^{i-1}=\delta_{i-2,i}^{i-1}=\delta_{i-2,i}^{i-2}=-\overline{c}_i^{i-2}(\delta^{i-2})\geq 0$*

$\vdots$

- *if $j=1$, $\delta_{ji}^{i-1}=\delta_{1i}^{i-1}=\delta_{1i}^{i-2}=\delta_{1i}^1=-\overline{c}_i^1(\delta^1)\geq 0$*

et

$$\delta_{ij}^{i-1}=\delta_{ij}^{i-2}=...=\delta_{ij}^0=0.$$

Comme les valeurs non-négatives sont soustraites à q_{ji}, la valeur objectif de $(\overline{P_i(\delta^{i-1})})$ vérifie

$$V(\overline{P_i(\delta^{i-1})})\leq V(\overline{P_i(\mathbf{0})}).$$

Comme $\delta^i_{ji} = \delta^{i-1}_{ji}$ *(*$j < i$*) et* $\delta^i_{ij} = -\overline{c}^i_j(\delta^{i-1})$ *(*$j > i$*), provenant de la remarque 4.20, nous avons que* $V(\overline{P_i(\delta^{i-1})}) = V(\overline{P_i(\delta^i)})$*. Et, finalement*

$$\bar{\beta}_i(\delta^{n-1}) = V(\overline{P_i(\delta^{n-1})}) = V(\overline{P_i(\delta^i)}) = V(\overline{P_i(\delta^{i-1})}) \leq V(\overline{P_i(0)}) = \bar{\beta}_i(\mathbf{0}).$$

D'après le théorème 4.21, si nous définissons $\beta^\star$ as $\bar{\beta}(\delta^{n-1})$, alors nous pouvons trouver de meilleurs coefficients pour une contrainte valide. Ainsi nous pouvons calculer une meilleure borne pour (QKP) que celle fournie par $V(\overline{LP_{\bar{\beta}}})$. L'algorithme 3 résume la procédure pour calculer $\beta^\star$.

1. Données : q, c, a, b;
2. Résultats : δ, $\beta^\star_i$, $i = 2, 3, ..., n$;
3. $\delta = \delta^{pred} = \mathbf{0}$;
for Pour k = 1 à n-1 **do**
  4. Résoudre $(\overline{P_k(\delta^{pred})})$;
  5. $\beta^\star_{k+1} = V(\overline{P_k(\delta^{pred})})$;
  6. $\delta_{kj} = -\overline{c}^k_j(\delta^{pred})$, $j = k+1, ..., n$;
  7. $\delta^{pred} = \delta$;
end

Algorithm 3: Calcul de $\beta^\star$

4.6 L'ALGORITHME DE *branch-and-bound* PROPOSÉ

L'algorithme de *branch-and-bound* que nous proposons pour résoudre le problème linéarisé (LP_{β^*}), et ainsi (QKP), débute par la solution admissible x_H. Nous avons choisi d'utiliser la solution admissible fournie par l'heuristique de Billionnet et Calmels [56]. La solution x_H nous permet de construire la permutation π_H obtenue en prenant dans l'ordre décroissant les composantes du vecteur solution x_H.
A chaque noeud de l'arborescence, un majorant U est calculé par l'emploi de l'algorithme de génération de contraintes décrit lors de la Section 4.3 . Ainsi, partant de $(\overline{LP_{\beta^*}})$, de la solution $\overline{x}$ (et des coûts réduits $\overline{c}$), une phase de fixation commence. Durant cette phase, nous utilisons la procédure de fixation des variables par coûts réduits des variables proposée par Oliva at al. [58]. Si l'on considère un problème en variables $0-1$ et une borne inférieure associée L, on peut simplifier cette procédure comme suit

$$\text{Si } |\overline{c_i}| > U - L \text{ alors fixer } x_i \text{ à } \overline{x_i}.$$

Avant la phase de branchement, comme la procédure de génération de contraintes peut en générer un grand nombre, nous appliquons une relaxation agrégée des contraintes (*surrogate relaxation*) à chaque noeud afin de ne transférer au noeud fils qu'un problème soumis à une seule contrainte (plus simple à résoudre en pratique).
Cette relaxation agrégée (pour un ensemble K de contraintes) consiste à substituer les contraintes générées par une seule contrainte $t \leq \tilde{\beta^*_K} x$ pour laquelle

$$\tilde{\beta^*_K}(j) = \sum_{i \in K} \frac{\overline{\mu_i} \cdot \beta^*_{\pi_i}(j)}{\sum_{k \in K} \overline{\mu_k}}$$

où $\overline{\mu_i}$ est la valeur duale dérivée de la solution optimale de la relaxation continue associée à la contrainte $t \leq \beta^*_{\pi_i} x$. Soit $K^\star$ l'ensemble des indices qui inclut les indices des contraintes générées. Nous pouvons montrer que $\sum_{k \in K^\star} \overline{\mu_k} = 1$, ce qui simplifie l'expression pour chaque $\tilde{\beta^*_{K^\star}}(j)$

$$\tilde{\beta^*_{K^\star}}(j) = \sum_{i \in K^\star} \overline{\mu_i} \cdot \beta^*_{\pi_i}(j).$$

Toutefois, nous pouvons montrer que le programme linéaire relâché pour lequel toutes les contraintes $t \leq \beta^*_{\pi_i} x$ pour $i \in K^\star$ sont substituées à $t \leq \tilde{\beta^*}_{K^\star} x$ ont la même valeur objectif que le modèle avec toutes les contraintes de $K^\star$. Pour prouver cela, considérons le problème où les variables x sont fixées à $\overline{x}$, solution du problème linéaire. puis, nous avons $\bar{t} = \{\max t : t \leq \beta^*_{\pi_i} \cdot \overline{x}, \forall i \in K^\star\}$. D'après le théorème de forte dualité, nous avons

$$\sum_{i \in K^\star} (\beta^*_{\pi_i} \cdot \overline{x}) \cdot \overline{\mu_i} = \bar{t}.$$

Ainsi il est évident que $c^T \overline{x} + \bar{t}$ est valeur optimale du modèle "t-relaxation".

Ainsi, la contrainte $t \leq \tilde{\beta^*_{K^\star}} x$ est seulement passée aux noeuds fils dans notre algorithme de *branch-and-bound*. La procédure de branchement suit les critères classiques de variables la plus fractionnaire, celle dont la valeur est la plus proche de 0.5.

4.7 RÉSULTATS NUMÉRIQUES

Nous avons mené des expérimentations numériques en suivant le schéma proposé par Billionnet et al. [54] et par Pisinger et al. [19], et ce, pour tester les performances de nos méthodes de calcul de majorant et de résolution exacte de (QKP).
Plus précisemment, nous reportons les résultats numériques concernant : (i) la qualité de notre majorant et son temps CPU d'obtention en comparaison avec celui fourni par la méthode basée sur une décomposition lagrangienne [54] (cf. sous-section 4.7.1) ; (ii) le temps CPU requis par notre *branch-and-bound* en comparaison avec l'algorithme de *branch-and-bound* fourni par Pisinger et al. [19] (cf. subsection 4.7.2). Ces deux méthodes fournissent à ce jour le meilleur majorant et la meilleure valeur optimale pour (QKP), respectivement.
Toutes les instances ont été générées comme dans [17], [16], [57] et [19]. Les coefficients c_i et q_{ij} de la fonction objectif sont des entiers générés aléatoirement entre 0 et 100, les poids a_i des contraintes sont uniformément distribués entree 0 et 50. Le membre de droite des contraintes, b, est un entier généré aléatoirement entre 50 et $max\{50, \sum_{i=1}^{n} a_i\}$. Les temps CPU sont présentés en secondes (s) et représentent un moyenne de 10 instances résolues à l'optimum.

Nous avons codé nos méthodes de calcul de majorant et de résolution exacte de (QKP) en langage C++ et toutes les expérimentations ont été menées sur une machine Pentium 4 2.66 GHz Intel processor avec 1024 MB de RAM, sauf les résultats correspondant à [54] et [19].

4.7.1 Comparaison des majorants

La qualité de notre majorant obtenu *via* la t-relaxation est comparée à : (i) le majorant suggéré par Billionnet et al. [54] (basé sur une décomposition lagrangienne) ; (ii) la t-relaxation en nombres entiers (Integral t-relaxation), que nous avons développée (qui consiste à résoudre le problème final t-linéarisé avec les variables $0-1$ et non en variables continues).
La comparaison des trois méthodes de calcul des bornes supérieures est reportée dans le tableau 4.2. Le temps CPU pour obtenir la borne issue de la t-relaxation est comparée à la t-relaxation en nombres entiers (Integral t-relaxation) est présenté dans le tableau 4.3. En raison de la différence des machines sur lesquelles

ont été menées les expérimentations fournies par Billionnet et al. [54] et notre calcul de major ant, les temps de calcul de [54] sont omis puisque la comparaison ne serait pas significative. Les résultats se trouvent ici [59].
Les trois méthodes de calcul des bornes supérieures (cf. tableaux 4.2 et 4.3) ont été testées sur des instances de 100 variables avec une densité de 25% concernant le membre quadratique, comme proposées dans [54].
Le tableau 4.2 présente les différentes valeurs de majorant (colonne UB) ainsi que leurs écarts relatifs (notés gap en $\%$) par rapport à la valeur optimale (reportée dans la colonne Opt). De plus, le nombre de contraintes générées (colonne NGC) est mentionné concernant la procédure de t-linearisation. La t-relaxation en nombres entiers fournit une borne obtenue avec les mêmes contraintes générées par la t-relaxation mais en tenant compte de l'intégrité des composantes du vecteur de décision x.

TAB. 4.2 – Comparaison de la qualité des majorants

Méthode	Opt	[54]		t-relaxation ($\overline{LP_{\beta^*}}$)		Integral t-relaxation		
No.		UB	Gap(%)	UB	Gap(%)	UB	Gap(%)	NGC
1	18558	18910.56	1.90	19124.1	3.05	18865	1.65	109
2	56525	56574.63	0.09	56576	0.09	56525	0.00	32
3	3752	3807.68	1.48	3900.53	3.96	3785	0.88	312
4	50382	50448.08	0.13	51064.5	1.35	50589	0.41	580
5	61494	61623.22	0.21	61621	0.21	61494	0.00	6
6	36360	36464.87	0.29	36654.9	0.81	36399	0.11	283
7	14657	14749.58	0.63	14853.5	1.34	14657	0.00	445
8	20452	20525.15	0.36	20528.5	0.37	20452	0.00	46
9	35438	35485.16	0.13	35487	0.14	35438	0.00	361
10	24930	25191.5	1.05	25496.9	2.27	20190	1.04	401
Moyenne	**32255**	**32378.04**	**0.63**	**32530.7**	**0.85**	**32339.6**	**0.26**	**257.5**

Le majorant fourni par [54] est de meilleure qualité que celui obtenu en appliquant la t-linearisation. Toutefois, le dernier reste en moyenne à moins de 1% de l'optimum. Quand les variables considérées sont entières, la t-linearisation en nombres entiers, fournit un majorant clairement meilleure (le gap est à moins de 0.5%). Nous pouvons également établir que le nombre de contraintes générées est plus petit que le nombre total de permutations ($O(n!)$). A 0.26% de l'optimum, nous pouvons dire que notre algorithme permet de décrire un voisinage de la solution optimale avec peu de contraintes en comparaison aux $O(n^2)$ contraintes générées par la linéarisation classique.

TAB. 4.3 – Comparaison des temps CPU requis par les techniques de t-relaxation et t-linéarisation

Méthode	t-relaxation ($\overline{LP_{\beta^*}}$)		Integral t-relaxation		
Densité	Tps CPU (s)	Gap(%)	Tps CPU (s)	Gap(%)	NGC
25	48.04	0.85	2087.98	0.26	284.5
50	78.56	0.93	3326.05	0.41	375.1
75	179.37	1.71	5744.43	1.09	478.1
100	11.90	0.89	5202.76	0.56	302.7
Moyenne	**79.47**	**1.10**	**4090.31**	**0.58**	**360.1**

Le tableau 4.3 montre que la t-relaxation en nombres entiers demande beaucoup de temps CPU. Ce tableau expose les moyennes de temps CPU et l'écart relatif à l'optimum sur 10 instances de densités (25%, 50%, 75% et 100%). En effet, l'avantage du calcul de la t-relaxation en utilisant les variables en continu au lieu de résoudre le problème final en nombres entiers (Integral t-relaxation) apparaît très clairement : les résultats sont obtenus quasi instantanément (No 2, 5, 8), de même pour la t-relaxation en nombres entiers ; alors que

TAB. 4.4 – Temps CPU moyens (en s.) pour des instances 100 et 200 variables

	t-linéarisation		[19]	
Instance	Tps	Opt	Tps	Opt
GHS100.25	0.78	10	210.7	10
GHS100.50	1.54	10	54.2	10
GHS100.75	0.41	10	6.7	10
GHS100.100	0.2	10	2.7	10
Moyenne	**0.73**	100%	**68.56**	100%

	t-linéarisation		[19]	
Instance	Tps	Opt	Tps	Opt
GHS200.25	42.32	10	860	9
GHS200.50	13.09	10	168.9	10
GHS200.75	27.84	10	23	10
GHS200.100	397.11	10	76.5	10
Moyenne	**120.09**	100.0%	**267.28**	97.5%

les temps CPU sont souvent divisés par 3 ou 100 pour toutes les autres instances. Les temps CPU requis pour la t-relaxation en nombres entiers peuvent s'expliquer par le nombre de contraintes du problème résultant. Or, nous savons que le nombre de contraintes est un frein dans la résolution en pratique des problèmes de multi-sac-à-dos en variables $0-1$.

4.7.2 Comparaison des méthodes de résolution exacte

Nous comparons dans cette section les performances de notre algorithme de *branch-and-bound* basé sur la t-linéarisation avec celui proposé par Pisinger et al.[19] (qui est l'algorithme état de l'art pour (QKP)).

Un temps limite de 3h a été fixé pour chaque instance.

Notons que les ensembles d'instances utilisées pour faire la comparaison ne sont pas exactement les mêmes, mais elles sont générées de façon identique. Les machines utilisées sont également différentes. Pisinger et al. [19] ont employé un Pentium IV 2.4Ghz avec 1GB RAM. Toutefois, les configurations utilisées pour nos expérimentations sont très proches.

Les tableaux 4.4 et 4.5 montrent clairement que notre *branch-and-bound* est plus performant que celui proposé par Pisinger et al. [19] pour les instances de petites tailles ($n = 100$) quelque soit la densité. Pour les densités 75% et 100% avec 200 variables, le *branch-and-bound* dévelopé par Pisinger et al. [19] se comporte mieux que notre algorithme. Toutefois, en moyenne notre approche est approximativement 2 fois plus rapide que celle de [19] pour $n = 200$.

4.8 CONCLUSION

Nous avons développé un nouveau cadre de linéarisation pour les problèmes quadratiques en variables $0-1$ (testé numériquement sur (QKP)) ; et ce, en ajoutant

1. une unique variable de décision à la fonction objectif

TAB. 4.5 – Temps CPU moyens (en s.) pour des instances 300 et 400 variables

	t-linéarisation		[19]	
Instance	Tps	Opt	Tps	Opt
GHS300.25	74.92	10	4031	10
GHS300.50	1480.18	9	556.8	10
GHS300.75	2134.21	10	94.7	10
GHS300.100	2163.94	9	90.3	10
Moyenne	**1444.43**	95%	**1193.2**	100%

	t-linéarisation		[19]	
Instance	Tps	Opt	Tps	Opt
GHS400.25	87.97	10	1190.1	9
GHS400.50	1468.31	10	978.3	10
GHS400.75	2698.21	9	275.0	10
GHS400.100	516.45	9	173.2	10
Moyenne	**1170.92**	95%	**640.41**	97.5%

2. et un ensemble de contraintes linéaires à l'ensemble des solutions admissibles initial.

La linéarisation offre la possibilité de calculer trois majorants. Ainsi, nous avons dans un premier temps proposé une étude théorique et expérimentale permettant de sélectionner la plus adpatée des trois possibilités dans le but de l'intégrer par la suite dans un algorithme de *branch-and-bound*. Puis, nous avons établi une technique pour renforcer les inégalités valides : calcul de coefficients des contraintes plus fins. Une procéduce de : *branch-and-bound* est ensuite suggérée basée sur : le majorant, une solution admissible fournie par Billionnet et Calmels [56] et une procédure de fixation de variables en chaque noeud de l'arbre de recherche. Enfin, des résultats numériques sont présentés. Notre borne supérieure est comparée à celle de Billionnet et al. [54] (meilleure borne connue à ce jour pour (QKP)) et notre *branch-and-bound* est comparé à celui de Pisinger et al. [19] (meilleure méthode exacte connue pour (QKP) à ce jour). Notre majorant est compétitif (à moins de1% de l'optimum) et notre *branch-and-bound* est clairement plus efficace que celui de [19] pour les instances de faible densité et comptant jusqu'à 400 variables.

Deuxième partie

Applications

Organisation de la partie II

Cette seconde partie est consacrée au traitement des applications issues du monde réel modélisées puis résolues (de façon exacte ou approchée) au moyen de la programmation mathématique : linéaire en nombres entiers (cf. **chapitre 5**) ou non linéaire et non convexe (**chapitre 6**).

CHAPITRE 5

Modélisation à l'aide de la programmation linéaire en nombres entiers

5.1 INTRODUCTION

Ce chapitre présente deux des applications que nous avons eu l'opportunité de traiter au moyen de la programmation linéaire en nombres entiers : un problème de combat contre la dengue (section 5.2) et un problème de détection de cible intelligente (section 5.3). Pour chacune des deux applications nous mentionnons : le contexte, la définition formelle du problème, les éléments clés qui font l'originalité de la modélisation proposée, et les étapes essentielles des techniques de résolution proposées. L'accent est donc mis sur l'aspect "modélisation", surtout concernant le problème de détection de cible intelligente. En effet, dans ce cas, contrairement au combat contre la dengue, pour lequel un problème de recherche opérationnelle connu permet une modélisation immédiate (en un problème d'ordonnancement périodique), il faut conceptualiser l'ensemble du problème.

5.2 COMBAT CONTRE LA DENGUE

Ce travail a été réalisé en collaboration avec P. Michelon (Pr. université d'Avignon) et M. Negrieros (Pr. unersidade estatual do Ceara) et a donné lieu à une publication dans une conférence internationale avec actes (ref. 7 dans le chapitre "Liste des publications").

5.2.1 Contexte de l'étude

Ce travail constitue l'une de mes premières études appliquées. Philippe Michelon lors de son séjour à l'*unersidade estatual do Ceara* en 2007, s'est vu confier en collaboration avec M. Negrieros, par le ministère de la santé du Brésil, une mission relative au problème du combat de la dengue.

La dengue est un moustique dont la piqure peut s'avérer mortelle chez l'homme et aucun traitement spécifique n'est encore à disposition. En conséquence, la seule voie de prévention ou de combat contre la dengue est l'élimination des moustiques vecteurs qui sont réperés dans des zones où ils se reproduisent. Plus précisemment, des véhicules équipés en insecticides sont envoyés dans ces zones infestées (et localisées) afin d'y pulvériser les insecticides.
En pratique, ces zones sont divisées en sous-zones. Ce découpage (qui a fait l'objet d'un travail préliminaire réalisé par P. Michelon et M. Negrieros [60]) est déterminé de telle sorte qu'une journée de travail d'un véhicule équipé soit suffisante pour nettoyer une sous-zone (formellement, chaque tâche dure une unité de temps). Cependant les pesticides tuent les moustiques mais pas les larves. Les larves mettent entre 7 et 9 jours à éclore. Il est donc primordial de repasser dans les sous-zones déjà traitées au moment où les larves éclosent tout en minimisant le nombre de véhicules à utiliser. Nous invitons le lecteur à se reporter à [60] pour obtenir plus de détails sur les aspects logistiques de la prévention et du combat contre la dengue.

Plus formellement, il s'agit d'un problème de minimisation du nombre de véhicules nécessaires pour effectuer une visite périodique des sous zones infectées. C'est un problème de recherche opérationnelle connu : le problème d'ordonnancement périodique (dont la formulation classique est NP-difficile [61]) où la périodicité n'est pas stricte mais représentée par un délai minimum et un délai maximum (respectivement dans notre cas 7 et 9). Nous avons montré que si toutes les périodes maximales sont égales pour toutes les tâches alors le problème associé à notre contexte du combat contre la dengue peut être résolu en temps polynomial en la taille de l'instance. Une simple utilisation d'un *solveur* de programmation linéaire suffit pour résoudre le problème initial. Mais, nous avons poussé l'étude à un cadre plus complexe, celui où les périodes maximales ne sont pas toutes égales. Une étude analytique du problème associé a été alors menée. Nous exposons

donc, dans ce document la modéslisation en nombres entiers ainsi que des heuristiques de résolution pour le problème généralisé.

5.2.2 Définition formelle du problème traité

Nous considérons ici un problème que nous notons $(GSPS)$ qui est une généralisation du problème d'ordonnancement périodique stricte [61] et du problème du combat contre la dengue.

Formellement, nous considérons J types d'activités et associons à chaque type j ($j = 1, .., J$) les paramètres suivants :
– n_j, le nombre d'activités de type j ;
– $\underline{F}_j$ le délai minimum entre deux exécutions d'une activité de type j ;
– $\overline{F}_j$ le délai maximum entre deux exécutions d'une activité de type j.

Chaque activité de chaque type nécessite une ressource unitaire et une durée d'une unité de temps. Les ressources sont identiques et utilisables à nouveau pour une autre tâche, une fois libres.

Le but du problème est de trouver un ordonnancement des tâches, réalisable, respectant les délais minimum et maximum entre deux éxécutions d'une même tâche, à travers un horizon de temps H, tout en minimisant le nombre de ressources employées. Les activités de type j doivent être exécutées au moins une fois au cours des premières $\overline{F}_j$ unités de temps.

Voici un exemple de solution admissible (qui s'avère être optimale dans ce cas particulier) illustrée par la Figure 5.1. La solution utilise 4 unités de ressources ou machines (ou véhicules si nous nous référons à l'application du combat contre la dengue) pour un horizon de 20 unités de temps, pour le problème correspondant aux données suivantes :

Type	n	$\underline{F}$	$\overline{F}$	activités
1	3	0	2	1,2,3
2	2	2	3	4,5
3	1	2	4	6
4	2	3	4	7,8
5	1	6	6	9

où chaque colonne correspond au type identifié, la seconde colonne reprend le nombre d'activités de chaque type, les troisième et quatrième colonnes mentionnent les délais minimum et maximum entre deux exéxcutions d'une activité de ce type, la dernière colonne reporte les activités.

5.2.3 Modélisation proposée

Nous avons donc modélisé ce problème, d'une part en un problème d'ordonnancement périodique généralisé (problème prouvé NP-difficile), puis *via* un programme linéaire en nombres entiers. Nous avons établi des propriétés ainsi que des remarques. Une première borne inférieure (dans un contexte de minimisation), que nous avons appelée *Trivial Lower Bound*, est calculée par une simple formule arithmétique. Elle est de qualité équivalente à la borne fournie par la relaxation continue du programme linéaire en variables entières modélisant le problème. Cette borne est ainsi utilisée. Aussi trois heuristiques sont proposées afin d'obtenir

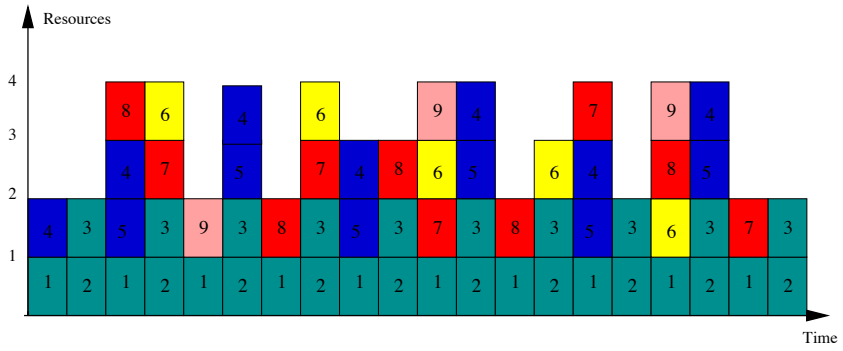

FIG. 5.1 – Une solution admissible

une solution admissible. L'une d'entre elles, plus efficace, consiste à résoudre exactement des sous problèmes du problème initial.
Nous proposons deux modélisations sous forme de programmes linéaires en nombres entiers pour $(GSPS)$: une formulation "faible" et une formulation "forte". Nous menons une étude comparative (théorique) de ces deux formulations *via* le corollaire 5.3.

Introduisons tout d'abord des notations nécessaires à l'écriture des modèles. Soient $J(i) \in \{1, 2, ..., J\}$ les types de chaque tâche i et $n = \sum_{j=1}^{J} n_j$ le nombre total de tâches du processus considéré.

Définissons dans un premier temps les variables de décision :

$$x_{it} = \begin{cases} 1 & \text{si la tâche } i \text{ est réalisée au temps } t \\ 0 & \text{sinon} \end{cases}$$

où $i = 1, ..., n$ et $t = 1, ..., H$.

$$R = \text{le nombre d'unités de ressources que nous souhaitons minimiser}$$

Alors, la formulation "faible" peut s'écrire comme :

$$GSPS \begin{cases} \min\ R \\ s.c. \left| \begin{array}{l} (5.2.1), (5.2.2)\ or\ (5.2.4), (5.2.3) \\ x_{it} \in \{0, 1\},\ \forall i = 1, ..., n,\ \forall t = 1, ..., H \\ R \geq 0 \end{array} \right. \end{cases}$$

où les contraintes (5.2.1), (5.2.2), (5.2.3) et (5.2.4) sont définies respectivement comme suit.

Pour chaque instant de temps t, le nombre de ressources utilisées est supérieur ou égal au nombre total de tâches à ordonnancer. Ainsi, nous avons la contrainte (5.2.1) :

$$R \geq \sum_{i=1}^{n} x_{it} \quad \forall t = 1, ..., H \tag{5.2.1}$$

Si la tâche i est exécutée à l'instant t, alors elle doit aussi l'être entre $t+1+\underline{F}_{j(i)}$ et $\overline{F}_{j(i)}$. Nous en déduisons alors la contrainte (5.2.2) :

$$\sum_{l=t+1+\underline{F}_{j(i)}}^{t+\overline{F}_{j(i)}} x_{il} \geq x_{it} \quad \forall i = 1, ..., n \ \ \forall t = 1, ..., H - \overline{F}_{j(i)} \tag{5.2.2}$$

Enfin, chaque tâche doit être ordonnancée au moins une fois sur chaque période et ce pour $\underline{F}_{j(i)}$ jours consécutifs. Nous obtenons la contrainte (5.2.3) :

$$\sum_{l=t}^{t+\underline{F}_{j(i)}} x_{il} \leq 1 \quad \forall i = 1, ..., n \ \ \forall t = 1, ..., H - \underline{F}_{j(i)} \tag{5.2.3}$$

La formulation "forte" est obtenue en remplaçant la contrainte (5.2.2) par :

$$\sum_{l=t+}^{t+\overline{F}_{j(i)}} x_{il} \geq 1 \quad \forall i = 1, ..., n \ \ \forall t = 1, ..., H - \overline{F}_{j(i)} \tag{5.2.4}$$

Cette contrainte signifie que sur chaque période comptant $\overline{F}_{j(i)}$ unités de temps, les tâches doivent être exécutées au moins une fois.

Notons $\lceil Z[\overline{GSPS}] \rceil$ le plus petit entier supérieur ou égal à la valeur optimale de la relaxation continue de $(GSPS)$ (formulation "forte").
La proposition suivante établit la borne inférieure fournie par la relaxation continue de $(GSPS)$ (formulation "forte").

Proposition 5.1 *Si l'horizon de temps H est supérieur ou égal au plus petit multiple commun de $\overline{F}_j$ ($j = 1, ..., J$), alors $\overline{x}$ définie par $\overline{x_{it}} = \dfrac{1}{\overline{F}_{j(i)}} \quad (\forall i = 1, ..., n, \ t = 1, ..., H)$, est une solution optimale pour la relaxation continue de de $(GSPS)$ (formulation "forte"), et sa valeur optimale vaut $\overline{R} = \sum_{j=1}^{J} \dfrac{n_j}{\overline{F}_j}$.*

Preuve 5.2 *Montrons que $(\overline{R}, \overline{x})$ est admissible pour la relaxation continue du problème initial.*

- $\sum_{i=1}^{n} \overline{x}_{it} = \sum_{j=1}^{J} \dfrac{n_j}{\overline{F}_j}$ *et ainsi la contrainte (5.2.1) est satisfaite par $(\overline{R}, \overline{x})$.*
- $\sum_{l=t}^{t+\overline{F}_{j(i)}} \overline{x}_{il} = \dfrac{\overline{F}_{j(i)}}{\overline{F}_{j(i)}} = 1$ *et la contrainte (5.2.4) est ainsi vérifiée.*

– $\sum_{l=t}^{t+\underline{F}_{j(i)}} \overline{x}_{il} = \frac{\underline{F}_{j(i)}}{\overline{F}_{j(i)}} \leq 1$ *et la contrainte (5.2.3) est ainsi vérifiée.*

Donc, $(\overline{R}, \overline{x})$ *est admissible pour la relaxation continue.*

Montrons maintenant qu'elle est optimale. En ce sens, considérons pour chaque solution admissible continue (R, x) *(de la relaxation linéaire) la quantité suivante et chaque tâche* i *:* $\sum_{t=1}^{F} x_{it}$ *où* F *est le plus petit multiple commun de* $\overline{F}_j$*. Alors, d'après la contrainte 5.2.4, nous avons :*

$$\sum_{t=1}^{F} x_{it} = x_{i1} + \ldots x_{i\overline{F}_{j(i)}} + x_{i\overline{F}_{j(i)}+1} \ldots + x_{iF} \geq \frac{F}{\overline{F}_{j(i)}} \tag{5.2.5}$$

car (5.2.4).

D'autre part, la contrainte (5.2.1) est vérifiée pour tout t *de 1 à* F*. Ainsi :*

$$F \times R \geq \sum_{t=1}^{F} \sum_{i=1}^{n} x_{it} \tag{5.2.6}$$

et donc (cf. (5.2.5) :

$$F \times R \geq \sum_{j=1}^{J} F \frac{n_j}{\overline{F}_j} \tag{5.2.7}$$

i.e.

$$R \geq \sum_{j=1}^{J} \frac{n_j}{\overline{F}_j} = \overline{R} \tag{5.2.8}$$

Il en découle que la valeur obtenue par la fonction objectif pour une solution admissible est au moins égale à la valeur donnée par $\overline{x}$*. Donc,* $(\overline{R}, \overline{x})$ *est optimale pour la relaxation continue de la formulation "forte" quelque soit un horizon plus grand ou égal à* F*.*

Corollaire 5.3 *Si l'horizon de temps* H *est plus grand ou égal que le plus petit multiple commun de* $\overline{F}_j$*, alors les deux formulations portent bien leurs noms. En effet, la borne inférieure fournie par la formulation "faible" est plus petite ou égale à celle fournie par la relaxation continue de la formulation "forte".*

Preuve 5.4 *Il suffit de montrer que* $(\overline{R}, \overline{x})$ *est admissible pour la formulation "faible", ou en d'autres termes, que* $\overline{x}$ *vérifie la contrainte (5.2.2), en utilisant un simple calcul.*

Corollaire 5.5 *Si l'horizon de temps* H *est plus grand ou égal que le plus petit multiple commun de* $\overline{F}_j$*,* $(j = 1, ..., J)$*, alors la borne de la relaxation continue de la formulation "forte" est égale à la borne inférieure triviale (BInfT) établie ci après et obtenue via une simple formule.*

où

$$BInfT = \max \left\{ \left\lceil \frac{\sum_{j=1}^{J} n_j \lfloor \frac{t}{\overline{F}_j} \rfloor}{t} \right\rceil / \quad t = 1, ..., H \right\} \tag{5.2.9}$$

Remarque 5.6 *Si l'horizon de temps H est plus grand ou égal que le plus petit multiple commun de $\overline{F}_j$, $(j = 1, ..., J)$, alors BInfT est égale à :*

$$BInfT = \left\lceil \sum_{j=1}^{J} \frac{n_j}{\overline{F}_j} \right\rceil \tag{5.2.10}$$

Preuve 5.7 *La démonstration du corollaire découle directement de la remarque 5.6 et de 5.1.*

5.2.4 Technique de résolution adoptée

Nous avons proposé trois heuristiques gloutonnes pour calculer une bonne solution admissible pour $(GSPS)$. La principale idée de chacune d'entre elles est basée sur l'utilisation de la borne inférieure triviale (BInfT) et sur la possibilité d'ordonnancer une tâche le plus tard possible. Nous présentons successivement les étapes principales des trois heuristiques gloutonnes.

Heuristiques I et II. Nous présentons les heuristiques I et II en même temps parce qu'elles ne diffèrent l'une de l'autre que par rapport au critère de tri des périodes maximales. L'idée consiste à ordonnancer le type le plus compliqué en premier. Ainsi, les deux heuristiques seront différentes, selon la mesure appliquée pour déterminer ce que signifie "type le plus compliqué".
Concernant l'heuristique I, un type est considéré comme "compliqué", s'il semble induire la consommation d'une grande quantité de ressources, qui peut être calculée par la remarque 5.6, *via* $\frac{n_j}{\overline{F}_j}$. En conséquence, pour l'heuristique I, les types sont triés par ordre décroissant des $\frac{n_j}{\overline{F}_j}$ alors que pour l'heuristique II la "complication" d'un type est repérée par le fait que nous avons peu de possibilités d'exécuter la tâche s'y rapportant, après avoir fixé sa première exécution, i.e. $\overline{F_j} - \underline{F_j}$ est "petit". Ainsi, dans l'heuristique II, les types sont triés par ordre croissant selon $\overline{F_j} - \underline{F_j}$.

Les grandes lignes de ces deux heuristiques sont présentées ci dessous.

Trier les types par ordre décroissant en fonction de $\frac{n_j}{\overline{F}_j}$
for $j = 1$ à J **do**
  Calculer BInfT pour les j premiers types
  if possible **then**
    Ordonnancer successivement les tâches de type j le plus tard possible sur les ressources disponibles déjà en usage
  else
    prendre une nouvelle ressource
  end if
end for

Algorithm 4: Heuristique I

Heuristique III. Nous conservons ici le critère du tri des types de l'heuristique II. Nous plaçons d'abord les tâches relatives au types "simples" et résolvons exactement le sous-problème sur l'horizon dans sa globalité, en utilisant le *branch-and-bound* fourni par un *solveur* de programmation linéaire (ici à ce moment là CPLEX10.0). Nous répétons ceci pour chaque type. Les principales étapes de l'heuristique III sont établies ci dessous.

Trier les types par ordre croissant en fonction de $\overline{F_j} - \underline{F_j}$
for $j = 1$ à J **do**
  Calculer BInfT pour les j premiers types
  if possible **then**
    Ordonnancer successivement les tâches de type j le plus tard possible sur les ressources disponibles déjà en usage
  else
    prendre une nouvelle ressource
  end if
end for

Algorithm 5: Heuristique II

Trier les types par ordre croissant en fonction de $\overline{F_j} - \underline{F_j}$
for $j = 1$ à J **do**
  Successivement ordonnancer sur tout l'horizon de temps donné, chaque tâche en minimisant R en prenant en compte les tâches déjà placées.
end for

Algorithm 6: Heuristique III

En pratique, Heuristique III fournit de meilleures solutions admissibles mais demande un temps CPU plus conséquent que les heuristiques I et II (qui sont équivalentes en terme de qualité de solution admissible et de temps CPU). Il s'agit alors d'arbitrer selon le temps imparti au calcul de la solution admissible.

Des expérimentations numériques ont été menées et ont montré que :
– la formulation "forte" permet de résoudre à l'aide de l'utilisation de ILOG-Cplex10.0 à l'optimum des instances de grandes tailles. Cette formulation est la plus appropriée pour traiter $(GSPS)$;
– la borne inférieure ($\lceil Z[\overline{GSPS}] \rceil$) fournie par l'utilisation de la formulation "forte" est de bonne qualité et obtenue dans des temps CPU très rapides. Celle-ci est égale à la valeur donnée par $BInfT$ si H est supérieur ou égal au plus petit commun multiplicateur des délais maximum ;
– Les heuristiques I et II produisent des bornes de moins qualité que celle fournie par l'heuristique III. Toutefois, celle-ci consomme un temps CPU très important.

5.3 DÉTECTION D'UNE CIBLE INTELLIGENTE

Ce chapitre fait l'objet d'une partie de la thèse de Carlos Diego Rodrigues (actuellement maître de conférences à l'université de Fortaleza au Brésil), que j'ai co-encadré avec P. Michelon (Pr. à l'université d'Avignon). Ce travail a été mené en collaboration avec B. Detienne (actuellement McF à l'université de Bordeaux I, Institut de Mathématiques de Bordeaux (IMB)) et a donné lieu à la publication d'un article (ref. 3 dans le chapitre "Liste de publications").

5.3.1 Contexte de l'étude

Cette étude a été menée dans le cadre d'un contrat des marchés publics avec la Direction Générale des Armées Techniques et navales (DGATn) ; contrat obtenu après avoir répondu à un appel d'offre.

Le travail que nous avons réalisé dans le cadre de ce contrat se situe dans le domaine de la théorie de la recherche "*search theory*". Cette dernière a vu le jour au cours de la deuxième guerre mondiale dans un contexte militaire particulier, à savoir la recherche d'un sous-marin dans une zone de recherche donnée. Deux contextes sont alors essentiellement étudiés : (i) la cible (ou le sous-marin) se cache, c'est-à-dire reste immobile dans une position inconnue : c'est le problème de la recherche d'une cible statique dans un espace de recherche donné ; (ii) le sous-marin est repéré dans une certaine position à un moment donné et le chercheur doit le retrouver quelques minutes plus tard alors que le sous-marin a changé de position : ce problème est connu sous le nom de *Flamming Datum*. La cible est recherchée par un porteur qui possède un nombre limité de capteurs. Ces capteurs devront être déposés, activés, ré-activés selon leur durée de vie, leur nombre d'émissions possibles...

Le problème que nous devons traiter consiste alors à maximiser la probabilité de détection de la cible tout en minimisant le nombre de capteurs mis à l'eau. Il faut également fournir un plan de déploiement des capteurs *via* le porteur, et les différentes actions relatives aux capteurs.

De nombreuses variantes de ces deux problèmes peuvent être considérées et ce, en fonction des données relatives au chercheur et à la cible. Par exemple, le sous-marin peut avoir la capacité d'employer différentes vitesses, différentes manoeuvres... Toutes ces informations relatives aux comportements d'un sous-marin durant la deuxième guerre mondiale sont reportées dans [62]. Dans la littérature trois approches apparaissent afin de formaliser le problème de la recherche d'une cible (statique, mobile, intelligente...) : la programmation stochastique, la théorie des jeux et la théorie des graphes. Notons que, le problème tel que nous devons le traiter, comprenant toutes les contraintes imposées par la DGATn, n'a pas été étudié jusqu'à aujourd'hui.

Nous avons proposé un modèle sous forme de programme mathématique, que nous avons appelé "ensembliste" pour formuler ce problème de détection de cibles. Le terme "ensembliste" provient de l'idée que nous ne considérons pas la cible comme unique (même si c'est le cas) mais nous simulons un nombre très grand de cibles pour avoir une vue d'ensemble et permettant de simuler le comportement de la cible de façon déterministe alors que celui ci ne l'est pas bien entendu. Cette idée nous permet de proposer un programme linéaire en variables entières au lieu d'un programme stochastique permettant de simuler le comportement de la cible. Il correspond à un simulateur qui fait intervenir un nombre important de cibles dans une même zone pour une mission d'une durée donnée. La probabilité de détection de la cible est alors calculée en fonction du nombre de cibles localisées durant la mission. Le nombre de cibles est un élément important quant à la fiabilité du calcul (de l'ordre de 150000 cibles pour que le modèle soit fiable, en général mais dans notre cas ce nombre est moins important tout en restant grand).

Le modèle ensembliste proposé se formule en un programme linéaire en nombre entiers de très grande taille. Cette taille est donc fonction du nombre de cibles. Ainsi pour faire face à cette difficulté nous avons choisi de résoudre le problème global en utilisant une méthode dite des fenêtres glissantes (fenêtre fonction du temps choisi, par exemple si la durée de la mission est de 60 minutes, on peut découper ces 60 minutes en 6 fenêtres de 10 minutes). A l'intérieur de chaque fenêtre le programme linéaire en variables entières est résolu exactement avec le logiciel libre LPSolve (demande de la DGATn que le logiciel d'optimisation soit gratuit).

Bien entendu, la principale difficulté concernant cette application réside dans la modélisation du problème réel. La difficulté provient en grande partie du nombre très important de données et aussi de décisions à prendre.

5.3.2 Définition formelle du problème

Nous présentons ici une définition formelle du problème de détection de la cible intelligente.

Remarquons que la définition formelle proposée comporte déjà une modélisation que nous proposons, et ce, à travers le choix dans un premier temps de la discrétisation du temps et de l'espace, puis dans la modélisation des trajectoires des cibles fictives. La modélisation du problème débute dès la définition formelle de ce dernier. Ces modélisations induiront un programme linéaire en nombres entiers associé.
Nous considérons un problème d'optimisation pour lequel le temps et l'espace sont discrétisés. Un unique chercheur et une cible intelligente se déplacent dans cet environnement constitué d'un ensemble fini de cellules. Le chercheur (3 types de chercheurs sont considérés : un bateau, un avion ou un hélicoptère), a pour objectif de maximiser la probabilité de détection de la cible pour un horizon de temps fixé. Ce dernier est soumis à des contraintes : de continuité de route empruntée (il doit pouvoir se déplacer d'une cellule vers une autre) et de distance nécessaire au delà de laquelle le capteur ne peut être activé. La cible est intelligente (i.e. elle a de la mémoire) et apprend le comportement du chercheur (i.e. elle est capable de savoir à quel type de chercheur elle est confrontée). Ainsi, elle est capable de tirer avantage de telles informations. Elle doit accomplir l'une des trois missions suivantes : elle se cache dans une zone donnée (m1), elle a été détectée dans une zone donnée et fuit (m2) ou elle doit traverser une zone surveillée (un détroit par exemple) (m3).
Les principales "règles du jeu" sont les suivantes. Une unique cible intelligente est recherchée par un unique chercheur. Le temps et l'espace sont finis et discrets. Plus précisemment le temps est un ensemble T de périodes consécutives $\{1, 2, ..., \tau, ..., \mathcal{T}\}$. Un ensemble fini de cellules représente l'espace de recherche noté (AOI). Le chercheur et la cible se déplacent de cellule en cellule tant que la recherche est active (i.e. tant que l'horizon de temps n'est pas atteint). Enfin, le but est de fournir un plan de recherche, spécifiant où et quand les capteurs doivent être placés et activés, de telle sorte à maximiser la probabilité de détection de la cible durant l'horizon de temps imparti à la mission.
Pour plus de clareté, nous avons regroupé les notations utilisées dans cette section dans le tableau 5.1.

5.3.2.a Caractéristiques de la cible

Afin de traduire formellement les missions que la cible doit suivre, nous avons introduit une simple donnée générique dans la structure du problème. Comme donnée d'entrée, deux sous-ensembles de cellules provenant de l'ensemble complet de cellules K sont définis : un espace initial "Initial area", ensemble de cellules à partir desquelles la cible peut démarrer sa mission, et une zone d'arrivée "Arrival zone", ensemble de cellules que la cible doit atteindre si elle veut accomplir sa mission. Comme l'illustre la Figure 5.2, ceci permet de modéliser facilement les trois missions (m1), (m2) et (m3). Pour la mission (m1), l'espace de recherche complet est à la fois zone de départ et d'arrivée de la cible, puisqu'elle s'y cache et que le chercheur n'a aucune idée de l'endroit où elle se trouve, si elle se trouve bien dans l'espace de recherche. Concernant (m2), nous voyons bien sur la figure l'endroit où la cible a été repérée, et les bords de la figure représentent les zones d'arrivées de la cible puisqu'elle veut fuir le chercheur, donc sortir de l'espace de recherche. Enfin concernant la mission (m3), la cible doit traverser une zone, arbitrairement nous avons placé la zone de départ à gauche et la zone d'arrivée à droite de la figure.
Par ailleurs, lorsque la cible se trouve à la cellule $k \in K$, elle est capable de contre-détecter l'activation d'un capteur localisé dans l'ensemble des cellules $CD(k)$. Ceci implique qu'elle connaît alors immédiatement la position, le rayon de détection et le temps de vie du capteur ainsi repréré (en fait, elle connaît la sorte de capteur auquel elle est confrontée). De plus, la cible a une mémoire illimitée et ne pénètrera pas dans une zone qu'elle sait potentiellement couverte par un capteur à moins qu'elle n'y soit obligée. Les données d'entrée incluent aussi la vitesse de la cible. Nous avons modélisé ceci en associant à chaque cellule $k \in K$,

TAB. 5.1 – Notations des données du problème

Données	Notations
Données Générales	
Ensemble de cellules	K
Indice des cellules	k
Horizon de temps	$\mathcal{T}$
Ensemble des périodes de temps	$T = \{0, ..., \mathcal{T}\}$
Données relatives au chercheur et aux capteurs	
Nombre de capteurs	C
Durée de vie d'un capteur	D
Nombre maximal d'activations d'un capteur	L
Temps de déploiement d'un capteur	I
Temps d'installation d'un capteur	a
Ensemble de cellule que le chercheur peut atteindre s'il est en k	$Adj_S(k)$
Espace de détection pour un capteur installé en k	$CS(k)$
Ensemble de cellules qui sont à portée radio du chercheur	$Radio(k)$
Données relatives à la cible	
Indice de la cible	Tg
Ensemble de cibles fictives	M
Transitions de chaque cible pour une période	$\tau^{Tg}_{k',k',\tau}$
Ensemble de cellules que peut atteindre la cible située en k	$Adj_T(k)$
Espace de contre détection pour une cible située en k	$CD(k)$

un ensemble de cellules adjacentes $Adj_T(k)$, que la cible peut atteindre en un instant de temps.

Notre approche consiste à simuler certaines routes possibles pour les cibles et à considérer chacune de ces possibilités comme une entité indépandante et déterministe. Cette entité est ce que nous appelons une "cible fictive". Chaque cible a sa propre trajectoire (une seule). Ainsi, maximiser la probabilité de capturer la cible réelle revient à trouver le nombre maximum de cibles fictives.

Modélisation de la trajectoire d'une cible fictive.

Afin de caractériser l'intelligence et le comportement réactif de la cible, nous avons pris un échantillon des chemins possibles. Cet échantillon suit l'idée proposée par [63], où l'auteur considère tous les chemins possibles pour la cible pour un nombre fini de cellules et un horizon de temps fini. Il met en exergue dans ce travail qu'il n'est pas possible de visualiser toutes les possibilités, sauf pour des problèmes de toute petite taille. Nous devons alors considérer seulement un sous ensemble de cas pour pouvoir mettre en oeuvre en pratique la résolution. Par ailleurs, concernant notre étude, seules nous intéressent les trajectoires pour lesquelles la cible est en mesure d'atteindre son objectif, *a priori*.

Dans notre contexte, l'ensemble des trajectoires pour chaque cible $Tg \in M$ peut être formulé comme un problème d'arbre couvrant du graphe dont les sommets seraient les cellules (cf. Figure 5.3) pour chaque

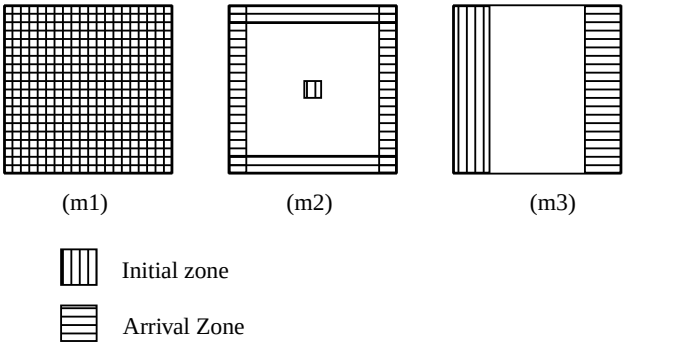

FIG. 5.2 – Zones de la cible pour chacune des trois missions classiques.

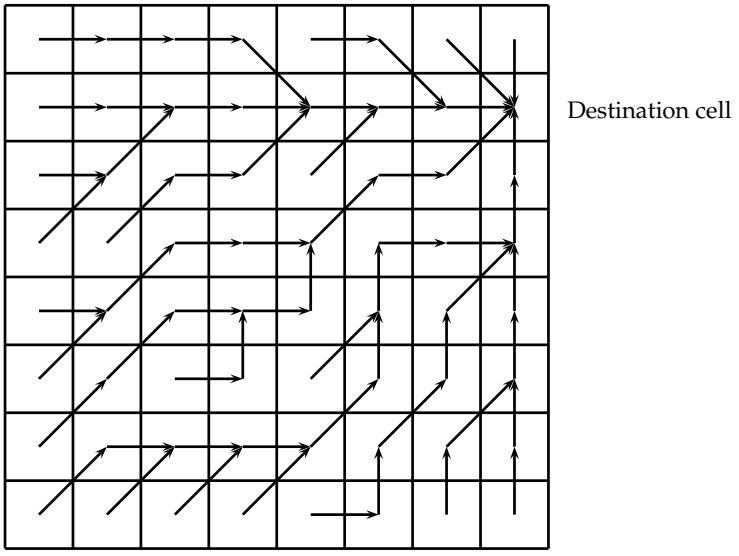

FIG. 5.3 – Trajectoires des cibles fictives : arbre couvrant.

instant de temps τ. Ces trajectoires peuvent être décrites comme :

$$t^{Tg}_{k,k',\tau} = \begin{cases} 1, & \text{si la cible } Tg \in M \text{ préfère aller de la} \\ & \text{cellule } k \in K \text{ vers la cellule } k' \text{ à l'instant } \tau \in [0, \mathcal{T}]. \\ 0, & \text{sinon.} \end{cases}$$

Cette variable détermine une route prioritaire pour la cible Tg, qui peut être suivie tant qu'elle ne mène pas à une cellule couverte par un capteur. Pour établir un vecteur de transitions pour chaque cible, nous déterminons dans un premier temps un point de départ aléatoirement dans la zone d'arrivée de la cible. Ce point identifie où se situe la cellule de destination pour la cible. Puis, nous définissons un coût aléatoire de transition d'une cellule à une autre pour la cible. Nous utilisons ensuite un algorithme de chemin de coût minimal d'un point de départ vers la cellule destination. Ce résultat est transformé en un vecteur de transition t.

5.3.2.b Caractéristiques du chercheur

Le chercheur est caractérisé par une vitesse, qui est représentée par l'ensemble $Adj_S(k)$ de cellules qui peuvent être atteintes durant un instant de temps à partir de la cellule k. Nous avons aussi défini une zone de départ et d'arrivée du chercheur. Le chercheur est équipé de capteurs au nombre de C. Un capteur comprend : un temps de déploiement I, durant lequel le chercheur doit rester à l'emplacement où le capteur est installé ; et un temps d'installation a (i.e. le capteur ne peut être utilisé qu'après ce temps a). De plus, le chercheur ne peut activer un capteur que s'il est à portée radio. Ceci est représenté par l'ensemble de cellules $radio(k)$ à partir desquelles le chercheur peut activer le capteur situé en cellule k. Chaque capteur a un temps de vie ainsi qu'un nombre d'activations limités. Les capteurs ont un rayon de détection formulé à travers l'ensemble de cellules $CS(k)$ couvertes par le capteur placé à la cellule k. Plus précisemment, si le capteur k est activé, toutes les cibles fictives de l'ensemble $CS(k)$ sont considérées comme détectées.

5.3.3 Modélisation proposée

Nous présentons dans cette sous-section la formulation du problème sous forme d'un programme linéaire en nombres entiers (PLNE). Ce dernier compte un nombre de variables de décision et de contraintes importants. Ainsi, nous présentons dans un premier temps les variables de décision (cf. Tableau 5.2) et les contraintes (cf. Tableau 5.3). Dans un second temps nous établissons le PLNE résultant.

Variables de décision	Signification
Chercheur	
$x_{k\tau}$	$x_{k\tau} = 1$, si le chercheur est en cellule $k \in K$ à l'instant $\tau \in \{0, \mathcal{T}\}$ $x_{k\tau} = 0$ sinon
Capteurs	
$d_{k\tau}$	$d_{k\tau} = 1$ si le capteur est déposé en cellule $k \in K$ à l'instant $\tau \in \{0, \mathcal{T}\}$ $d_{k\tau} = 0$ sinon
$y_{k\tau}$	$y_{k\tau} = 1$ si le capteur est activé en cellule $k \in K$ à l'instant $\tau \in \{0, \mathcal{T}\}$ $y_{k\tau} = 0$ sinon
Cibles	
$z_{k\tau}^{Tg}$	$z_{k\tau}^{Tg} = 1$ si la cible $Tg \in M$ est détectée en cellule $k \in \{l \in K \vert w_{l\tau}^{Tg} = 1\}$ à l'instant $\tau \in \{0, \mathcal{T}\}$ $z_{k\tau}^{Tg} = 0$ sinon
v^{Tg}	$v^{Tg} = 1$ si la cible $Tg \in M$ est détectée durant l'horizon de temps $v^{Tg} = 0$ sinon

TAB. 5.2 – Variables de décision

L'ensemble du programme linéaire s'écrit de la façon suivante :

$$(PLNE) \begin{cases} \max \sum_{Tg \in M} v^{Tg} = f \\ s.c. \left| \begin{array}{l} (C1), (C2), (C3), (C4), (C5), (C6), \\ (C7), (C8), (C9) \\ x_{k\tau}, y_{k\tau}, d_{k\tau}, z_{k\tau}^{Tg}, v^{Tg} \in \{0,1\} \\ \tau \in \mathcal{T}, Tg \in M, k \in K \end{array} \right. \end{cases} \tag{5.3.1}$$

Contraintes	Signification
Chercheur	
$(C1)$ $\sum_{k\in K} x_{k\tau} = 1$, $\tau \in T$	Le chercheur a une unique position pour chaque période de temps τ
$(C2)$ $x_{k,\tau+1} \leq \sum_{k'\|k\in Adj_S(k')} x_{k'\tau}$, $\forall k \in K, \tau \in \{0, \mathcal{T}-1\}$	Le chercheur peut passer de la cellule vers une cellule adjacente
Capteurs	
$(C3)$ $\sum_{k\in K} \sum_{\tau\in\{0,\mathcal{T}\}} d_{k\tau} \leq C$	Le nombre de capteurs est limité
$(C4)$ $\sum_{\tau\in[\tau',\tau'+D]} y_{k\tau} \leq L$, $\forall k \in K, \tau' \in \{0, \mathcal{T}\}$	Le nombre d'activations des capteurs est limité
$(C5)$ $y_{k\tau'} \leq \sum_{\tau\in[\tau'-D-a,\tau'-a]} d_{k\tau}$, $\forall k \in K, \tau' \in \{0, \mathcal{T}-1\}$	Une activation n'a lieu qu'après installation et durant le temps de vie du capteur
$(C6)$ $x_{k\tau} \geq d_{k\tau'}, \forall k \in K, \tau'$ $\in \{0, \mathcal{T}\}, \tau \in [\tau', \tau'+I]$	Le chercheur doit rester durant le temps de déploiement du capteur
$(C7)$ $y_{k\tau} \leq \sum_{k'\in radio(k)} x_{k'\tau}$, $\forall k \in K, \tau \in \{0, \mathcal{T}\}$	Le chercheur doit être à portée radio pour activer le capteur
Cibles	
$(C8)$ $z_{k\tau}^{Tg} \leq y_{k'\tau}$, $\forall Tg \in M, \tau \in T, k \in K \| w_{k\tau}^{Tg} = 1$, $k' \| k \in CS(k')$	Une cible est détectée à la cellule seulement si le capteur couvrant cette cellule est activé
$(C9)$ $v^{Tg} \leq \sum_{k\in K\|w_{k\tau}^{Tg}=1} \sum_{\tau\in T} z_{k,\tau}^{Tg}$, $\forall Tg \in M$	Une cible est détectée si elle est repérée dans une cellule à un instant donné

TAB. 5.3 – Formulations et significations des contraintes

5.3.4 Technique de résolution adoptée

La méthode de la Fenêtre Glissante [64] est couramment utilisée pour les problèmes d'ordonnancement et de planification de la production où les opérations à planifier doivent suivre une chronologie, ce qui constitue une caractéristique commune avec le problème qui nous intéresse ici. Le principe de la méthode est relativement simple mais son efficacité n'est plus à démontrer. Nous allons l'expliciter dans le cas de la détection de cible.
Le problème d'optimisation inhérent à la détection de cible est donc à la fois un problème de grande taille (au moins pour certaines instances) et à la structure complexe. Pour résoudre ses plus grandes instances, il est donc nécessaire de proposer une méthode de type "diviser pour régner", c'est à dire consistant à partitioner le problèle initial en sous-problèmes déterminés de telle sorte à être abordable par une méthode exacte. Pour ce faire, nous avons choisi d'exploiter l'aspect chronologique du problème. Le découpage en sous-problèmes se fait sous forme de "fenêtres glissantes" correspondant au découpage du temps. Si le nombre de fenêtres est égale à un, on traite globalement le problème initial et on obtient donc une solution optimale. Ainsi, le rôle de la paramétrisation prend tout son sens. Plus la taille de la fenêtre sera grande, plus le sous- problème correspondant sera difficile à résoudre (ou, plus exactement, plus il faudra de temps pour le résoudre) car plus grande sera sa taille. Chaque sous-problème est résolu exactement au moyen d'une méthode de *branch-and-bound*.
Ainsi, la méthode comporte deux paramètres, désignés par s et p dans ce qui suit. s désigne la largeur (en termes d'unité de temps) de la fenêtre et définit donc la taille de chacun des sous-problèmes qui sont résolus.

Si $s = \mathcal{T}$ ($\mathcal{T}$ étant la durée de la mission), alors le problème initial sera résolu. Or comme mentionné, plus haut, cela n'est envisageable que lorsque le problème n'est pas de grande taille. Dans l'implantation de la méthode, ce paramètre est calibré.
Le second paramètre, p, correspond au nombre d'unités de temps pour lesquelles nous faisons glisser la fenêtre entre deux itérations. C'est le pas de déplacement de la fenêtre. Naturellement, $p \leq s$. C'est un paramètre beaucoup moins sensible que le précédent.
Ainsi la méthode consiste à définir la première fenêtre de temps entre les instants 1 et $1 + s$, à résoudre à l'optimum le sous-problème ainsi défini et à fixer les décisions prises entre 1 et $1 + p$. La fenêtre glisse alors jusqu'à l'instant $1 + p$ et un second sous-problème, correspondant à optimiser des décisions entre les instants $1 + p$ et $1 + p + s$, est résolu et le processus est itéré.
Dans un algorithme de Fenêtre Glissante, nous considérons habituellement que la fenêtre peut glisser de manière unidirectionnelle, selon le temps. Dans notre méthode, inspirée par l'algorithme *"Forward and Backward"* proposée par Brown [65], nous avons fait une rémise en cause des décisions précedemments faites pour améliorer la solution, évitant alors une solution gloutonne. En effet, lorsque la dernière fenêtre est résolue (correspondant aux décision de fin de mission), nous sauvegardons la solution obtenue et recommençons le processus à partir de cette solution, en faisant glisser la fenêtre dans l'ordre anti-chronologique (en remontant le temps). Les "allées et venues" de la fenêtre se poursuivent tant que la solution s'améliore (à une marge près).

Ainsi, la méthode comporte 3 niveaux algorithmiques. Il faut gérer le sens de glissement de la fenêtre (chronologique ou anti-chronologique) et, pour un sens donné, définir les fenêtres qui seront successivement traitées. Un troisième niveau concerne la résolution du sous-problème de façon exacte *via* un *branch-and-bound*.
La méthode démarre sans solution connue ou, de manière équivalente, par un plan de déploiement "vide'" et prend en entrée les paramètres et données du problème. Le premier niveau (gestion du sens et du critère d'arrêt) est régi par le pseudo-code suivant qui prend en entrée les différents paramètres.

Données : Taille de la fenêtre $taille$, Pas pas, Données du modèle Mod (cf. sous-section pécédente)
Résultat : P le plan de recherche
$P \leftarrow \emptyset$;
$sens \leftarrow chronologique$;
repeat
  $P' \leftarrow P$
  $P \leftarrow$Fenêtre-Glissante($taille$, pas, Mod, P', $sens$)
  if $sens = chronologique$ **then**
    $sens \leftarrow antichronologique$
  else
    $sens \leftarrow chronologique$
  end if
until $f(P) - f(P') < \epsilon$
return P

Algorithm 7: FAB-Glissant

où f est la fonction objectif du modèle considéré et ϵ la marge au delà de laquelle il est souhaitable de poursuivre la recherche. À chaque itération, une fonction "Fenêtre-Glissante($taille$, pas, Mod, P', $sens$)" est donc appelée : c'est elle qui est chargée de faire glisser la fenêtre et de résoudre les sous-problèmes. Tout au

long du déroulement de cet algorithme, un plan de déploiement P est donc maintenu.

Le plan de déploiement crée par "Fenêtre-glissante", comme son nom l'indique est construit à partir du plan courant selon la procédure de fenêtre glissante dans le temps (soit dans l'ordre chronologique si $sens = 1$, soit dans l'ordre anti-chronologique si $sens = -1$). En quelque sorte "Fenêtre-glissante" déplace la fenêtre d'une extrémité à l'autre de l'intervalle de temps correspondant à la mission, en passant par les positions intermédiaires où, à chaque fois un sous-problème est résolu à l'optimum.
En sortie de "Fenêtre-glissante", nous obtenons donc un plan de déploiement nouveau qui va devenir le plan courant, tant que le processus de "va-et-vient" est poursuivi. Il ne s'arrête en effet, que lorsque le nouveau plan courant n'est pas meilleur que l'ancien (à une constante près ϵ) ; "meilleur" étant pris au sens de la valeur donnée à la fonction objectif du modèle considéré.
L'essentiel de la méthode se situe donc dans la fonction "Fenêtre-glissante" dont le pseudo-code est donné par :

Données : Taille de la fenêtre $taille$, Pas pas, Données du modèle Mod, Plan de recherche P_I, sens $sens$
Résultat : Plan de recherche P
$P \leftarrow P_I$
if $sens = chronologique$ **then**
  $t \leftarrow 0$
else
  $t \leftarrow \mathcal{T} - taille$
end if
$extreme \leftarrow faux$
repeat
  for $t^{hors} \notin [t, t + taille]$ **do**
    Affecter toutes les variables indexées par t^{hors} en Mod selon P;
  end for
  for $t^{dans} \in [t, t + taille]$ **do**
    Libérer toutes les variables indexées par t^{dans} en Mod;
  end for
  $(v, P) \leftarrow$ Branch-and-Bound(Mod, f);
  if $sens = chronologique$ **then**
    if $t < \mathcal{T} - taille$ **then**
      $t \leftarrow min(t + pas, \mathcal{T} - taille)$;
    else
      $extreme \leftarrow vrai$;
    end if
    if $t > 0$ **then**
      $t \leftarrow max(t - pas, 0)$;
    else
      $extreme \leftarrow vrai$;
    end if
  end if
until $extreme = vrai$
return $\underline{P}$

Algorithm 8: Fenêtre-Glissante

L'algorithme "Fenêtre glissante" constiste à faire glisser de pas unités de temps une fenêtre, après que le sous-problème correspondant ait été optimisé, soit dans le sens chronologique, soit dans le sens anti-chronologique, selon la valeur de $sens$.
A partir d'un plan existant P_I (partiellement vide lors de la première itération de FAB mais complet lors des itérations sub-séquentes), "Fenêtre glissante" appelle donc la méthode "Branch-and-Bound" pour résoudre le sous-problème associé de façon exacte. Celui-ci correspond à $taille$ unités de temps sur lesquelles les décisions (trajectoire du porteur, dépôt et activations des capteurs) seront optimisées.
Selon le sens, la première fenêtre sera donc comprise entre 0 et $taille$ (sens chronologique) ou entre $T-taille$ et T (sens anti-chronologique). Toutes les variables correspondant aux décisions à prendre en dehors de la fenêtre de temps courante sont fixées à leurs valeurs courantes tandis que celles correspondant aux décisions à prendre dans la fenêtre courante sont libérées. Celles-ci seront alors optimisées compte tenu des décisions prises (et momentanément fixées) avant et après la fenêtre courante. Le plan courant est donc modifié par les nouvelles affectations des variables de décision concernées par la fenêtre courante. Cette dernière est alors déplacée de pas unités de temps jusqu'à ce que la fenêtre arrive à une extrémité (0 ou $\mathcal{T}$ selon $sens$).

Des expérimentations numériques ont été menées sur des instances les plus réalistes possibles fournies par la DGATn. Notre méthode de fenêtre glissante nous permet de traiter la plupart des instances mais pour que la résolution puisse fournir une solution proche de l'optimum, la fenêtre doit être la plus grande possible. De ce fait, il est nécessaire de réduire les informations à prendre en compte pour fournir un plan de déploiement, c'est à dire, que certaines contraintes on été omises afin de réduire la taille du problème initial. Par ailleurs, même si la demande de la DGATn était de traiter les expérimentations au moyen de LPSolve, nous avons traité les instances avec ILOG-CPLEX, ce qui nous a permis d'en traiter beaucoup plus.

5.4 CONCLUSION

Dans cette section nous avons présenté deux applications que nous avons eu l'occasion de traiter au moyen de la programmation linéaire en nombre entiers. Le choix de ces deux applications s'explique par la complexité de la modélisation associée. C'est pourquoi l'accent est mis sur les modèles et non sur les résultats numériques des méthodes proposées (résultats disponibles dans les articles relatifs à ces travaux). D'autres applications, comme par exemple le transport de personnes handicapées ou le traitement automatique des langues ont été traitées mais non mentionnées dans ce manuscrit par souci d'équilibre des parties de ce dernier.

CHAPITRE 6

Programmation bi-niveaux et non linéaire fractionnaire en variables mixtes

6.1 INTRODUCTION

Ce chapitre comporte deux sections, chacune d'elles est associée à une application issue du domaine des réseaux. La première (cf. section 6.2) traite un problème de tarification optimale pour un service de livraison pour un produit vendu par *e*-commerce. Dans ce cas, c'est un réseau de transport qui est concerné. Les outils de théorie des jeux employés dans ce contexte sont fréquemment utilisés dans le domaine des réseaux sans fils pour lesquels des problèmes de congestion apparaissent bien souvent. C'est un programme mathématique bi-niveaux qui est considéré (plus précisemment un programme mathématique avec contraintes d'équilibres stochastiques). La seconde application (cf. section 6.3) concerne un problème d'optimisation de l'énergie, dans des réseaux hétérogènes, traité au moyen de la programmation mathématique (non linéaire, non convexe et fractionnaire en variables mixtes). Pour chacune des deux applications nous mentionnons : le contexte, la définition formelle du problème, les éléments clés qui font l'originalité de la modélisation proposée, et les étapes essentielles des techniques de résolution proposées. Ainsi, comme dans le chapitre précédent nous mettons l'accent sur l'aspect modélisation.

6.2 TARIFICATION OPTIMALE D'UN SERVICE DE LIVRAISON

Ce travail a été réalisé en collaboration avec Y. Hayel (McF Université d'Avignon), T. Jimenez (Ingénieur de recherche Université d'Avignon, L. Brotcorne (Chargée de recherche, INRIA Lille Nord Europe) et B. Tousni (Doctorant en co-tutelle Université d'Avignon et INRIA Lille Nord Europe), et ce, dans le cadre de l'ANR RESPET (janvier 2012 - juillet 2015). Un travail préliminaire a été publié dans une revue internationale (cf. ref. 1 dans le chapitre "Liste des publications"). Ce travail portait sur l'étude du problème de e-commerce simplifié et proposait une étude théorique du modèle de théorie des jeux proposé. Une étude approfondie de l'existence et de l'unicité de l'équilibre a été menée. Le travail présenté dans cette section est donc une poursuite de ref. 1, établissant une modélisation sous forme de programme mathématique.

6.2.1 Contexte

Les entreprises de livraison à domicile ainsi que le secteur d'activités liées à la logistique urbaine vivent un bouleversement avec l'essor important du commerce en ligne [66]. En effet, depuis quelques années, de nombreux acteurs proposent de nouveaux services venant faire concurrence aux entreprises historiques. Afin de rester donc compétitif, les entreprises de services de livraison doivent adapter leurs offres en tenant compte des usages des consommateurs. Notre travail se situe dans ce contexte, appelé $B2C$ ("*business to consumer*" en anglais) et particulièrement lorsque le nombre de clients est très grand. Le cas que nous traitons ici est fourni par DHL qui effectue les livraisons pour *Toys "R" us*.
Nous nous intéressons au problème d'optimisation bien connu de tarification optimale avec un modèle bi-niveau [67] dans lequel un fournisseur (le *leader*) fixe les tarifs de ses services de livraison, les clients (les *followers*) réagissent en minimisant leur fonction d'utilité. Outre le tarif du *leader*, cette fonction inclut un terme non linéaire exprimant la congestion du service/la qualité du service.

Nous proposons donc ici d'étudier un modèle de programmation mathématique avec contraintes d'équilibres stochastiques qui est une variante des problèmes de programmation mathématique avec contraintes d'équilibres (MPEC) et de la programmation bi-niveau [68]. Pour plus de réalisme, les équilibres considérés sont stochastiques. Nous proposons une méthode pour la détermination de l'équilibre des *followers*, pour des tarifs donnés. Nous introduisons par la suite une nouvelle heuristique basée sur l'analyse de la sensibilité de l'équilibre des *followers*) par rapport au tarif pour résoudre le problème du *leader*.

6.2.2 Définition formelle

Nous supposons que chaque client choisit un service parmi l'ensemble possible des services $\mathcal{J}$. Le nombre total de services est $|\mathcal{J}| = J$. Nous considérons deux types de livraison : livraison à domicile (DH) et livraison dans un point relais colis (WH). Le coût du service j perçu par le client dépend du tarif A_j fixé par le fournisseur, et de la qualité de service (évaluée en fonction de l'effet de congestion induit par les choix des autres clients). La demande des clients suit une loi de Poisson de taux λ. Chaque nouveau client choisit son service de livraison, où la proportion de choisir le service $j \in \mathcal{J}$ est notée p_j, avec $\sum_{j\in\mathcal{J}} p_j = 1$. La forme générale du coût c_j du service j est donnée par la fonction :

$$c_j(p_j) = A_j + \alpha_j f_a(p_j)$$

où $f_a(p_j)$ représente le niveau de congestion du service j de type $a \in \{DH, WH\}$ et α_j un coefficient de conversion monétaire. La fonction représentant le niveau de congestion du réseau dépend du type de service. En effet, l'aspect congestion est ressenti en terme de temps d'attente avant livraison concernant le service DH (livraison à domicile), mais correspond à un rejet ou non du colis lorsque le service correspondant est WH (relais colis). Afin de formaliser le temps d'attente de livraison d'un colis $f_D(p_j)$ pour le service j de DH, nous considérons un modèle général de file d'attente M/G/1. Ainsi, le taux d'arrivée des colis suit une loi de Poisson avec un taux λp_j (du fait que nous supposons que les décisions de chaque client sont inobservables). Les colis sont distribués un à un, et le temps de service pour chaque colis est une variable aléatoire positive. Le temps moyen de distribution d'un colis est donné par la formule de Pollaczek-Khinchin [69]. Cette fonction de coût $f_W(p_i)$ pour chaque client qui choisit d'être livré *via* le WH (relais colis) i ne dépend donc pas du délai d'attente mais du fait que le colis peut être ou non refusé si le relais colis est saturé. Ce problème peut bien entendu se produire, du fait que de nombreux relais colis se situent en ville et de ce fait leur capacité d'accueil est très limitée. Nous supposons que pour un colis non rejeté, le temps durant lequel le colis reste dans le relais est aléatoire et suit une loi de distribution exponentielle de taux μ. Ainsi, le modèle du service i de type WH est une file d'attente $M/M/K_i/K_i$ où K_i correspond à la capacité du relais i. La fonction de congestion $f_W(p_i)$ s'écrit alors à l'aide d'une probabilité de blocage π, et donnée par la formule de Erlang-B [69]. Maintenant que nous avons défini le coût perçu par chaque client choisissant DH ou WH, nous allons chercher une situation stable de laquelle le client n'a pas d'intérêt à s'en écarter. Nous recherchons cette situation d'équilibre du fait des outils que nous utilisons dans le cadre de cette étude, à savoir la théorie des jeux. Cette dernière propose plusieurs modèles afin de formuler la décision des clients. Le modèle qui nous parait le plus adapté à notre contexte est celui, dit à choix discret (DCM) [70], du fait qu'il offre la possibilité de prendre en compte une certaine "incertitude" concernant les décisions des clients (par exemple manque d'information, erreur par manque d'attention lors de la prise de décision, ...) Ces modèles sont très souvent utilisés pour des problèmes d'affectation de trafic [71]. Nous aboutissons alors au concept d'Equilibre Stochastique pour l'utilisateur (en anglais *Stochastic User Equilibrium* (SUE)) [72] et plus généralement à ce qui se nomme en anglais *quantal-response equilibrium* (QRE) [73].
En général, une QRE est un vecteur de distribution $p^* = (p_1^*, \ldots, p_J^*)$ qui vérifie le système de point fixe suivant :

$$\forall j \in \mathcal{J}, \quad p_j^* = F_j(p^*), \tag{6.2.1}$$

où la fonction F_j est dite fonction de réponse du client choisissant le service j. La QRE sera dépendante du concept d'équilibre qui sera choisi pour décrire le système (par exemple équilibre de Wardrop, de Logit, de Probit...). Par ailleurs, dans notre contexte, nous considérons que les services de même type sont corrélés entre eux, i.e. leurs probabilités d'être choisi par un client ne sont pas indépendantes. Nous nous sommes alors basés sur les modèles de choix discrets avec des nids de type Logit (en anglais *nested Logit discrete*

choice model) [70]. Les services de même type sont regroupés dans un nid, et le processus de choix peut être considéré comme un choix en deux temps par le client. Dans un premier temps, le client choisit entre les nids DH et WH ; dans un second temps, le client choisit un service à l'intérieur même du nid.
La proposition suivante montre que le système que nous obtenons peut être résolu par un problème de minimisation. Nous formulons la proposition pour un DCM avec nid Logit général pour lequel J choix s'offrent au client et où chaque choix j est associé à un nid $n(j)$, avec $\mathcal{N}$ l'ensemble de N nids. Notons θ un coefficient positif associé à la dispersion perçue dans la fonction de coût, et notons ϕ_n le coefficient de corrélation du nid $n \in \mathcal{N}$.

Proposition 6.1 *Les conditions de Karush-Kuhn-Tucker du problème de minimisation suivant :*

$$min_p Z = Z_1 + Z_2 + Z_3, \tag{6.2.2}$$

avec :

$$Z_1 = \sum_{n\in\mathcal{N}} \sum_{j\in n} \int_0^{p_j} c_j(s)ds, \quad Z_2 = \frac{1}{\theta} \sum_{n\in\mathcal{N}} \sum_{j\in n} p_j \ln(p_j), \quad Z_3 = \sum_{n\in\mathcal{N}} \frac{1-\phi_n}{\theta\phi_n} \left((\sum_{j\in n} p_j) \ln(\sum_{j\in n} p_j) \right),$$

soumis aux contraintes :

$$\sum_{n\in\mathcal{N}} \sum_{j\in n} p_j = 1 \quad \textit{et} \quad p_j \geq 0, \quad \forall j \in \mathcal{J},$$

sont les conditions pour trouver une solution pour le stochastic user equilibrium (6.2.1).

L'existence et l'unicité de la solution pour ce problème de minimisation est démontrée en prouvant la convexité de la fonction objectif Z.

6.2.3 Modélisation proposée

Nous avons donc choisi de modéliser le comportement des clients au moyen des outils de la théorie des jeux et en particulier en utilisant le modèle SUE. Reste donc maintenant à modéliser au moyen de la programmation mathématique la tarification que souhaite réaliser le fournisseur. Bien entendu, dès la définition du problème, et du fait même de la prise en considération du client et du fournisseur (respectivement *follower* et *leader*), nous sommes face à une programme mathématique particulier, i.e. comportant deux niveaux. Nous parlons donc de programmation bi-niveaux, et plus particulièrement, dans ce contexte de théorie des jeux de Programme Mathématique avec Contraintes d'Equilibre (MPEC), avec un ajout dans notre cadre d'étude : l'aspect stochastique (au niveau de l'équilibre).
Afin de modéliser notre problème sous cette forme, nous supposons que le fournisseur (situé au niveau haut du programme, c'est le *leader*), contrôle ses tarifs A_j pour tout service $j \in \mathcal{J}$. Pour un ensemble fixé de variables du fournisseur, les clients réagissent en choisissant le service qui minimise leur propre fonction de coût. En notant u les variables du *leader*, le problème MPEC associé s'écrit :

$$(MPEC) \begin{cases} max_u & F(u,p), & (C_1) \\ s.c. & L_j \leq u_j \leq U_j \quad \forall j \in \mathcal{J}, & (C_2) \\ & p = p^*(u). & (C_3) \end{cases}$$

où les contraintes (C_2) imposent des bornes sur les variables du *leader*, et les contraintes (C_3) établissent que p est solution du problème d'équilibre stochastique paramétrisé par u.
Trouver une solution pour ce programme est un défi du fait que l'équilibre stochastique dépend du paramètre u. L'utilisation de l'analyse de sensibilité de l'équilibre des *followers* en fonction des tarifs du *leader*, va nous permettre de passer outre cette difficulté. Cette analyse nous fournit une approximation des dérivées

de l'équilibre p^* par rapport au contrôle du leader u. Ces informations nous permettent de construire deux heuristiques pour obtenir les tarifs optimaux du *leader*. La première utilise une approximation du gradient de la fonction objectif du *leader*. La deuxième fait appel a une recherche d'optimisation locale. Nous présentons ces voies de résolution dans la sous-section suivante.

6.2.4 Technique de résolution adoptée

6.2.4.a Analyse de sensibilité de l'équilibre stochastique SUE

L'équilibre $p^*(u)$ est solution du problème non linéaire paramétrisée par u. Pour un tel problème, l'analyse de sensibilité de l'équilibre [74] est un outil précieux pour calculer les variations de la solution quand le paramètre est perturbé. Ceci peut être fait en calculant une estimation du gradient de $p^*(u)$ par rapport à u. Suivant cette idée, nous avons dérivé l'expression du gradient de notre *nested Logit* SUE par rapport aux variables du fournisseur. Plus précisemment, pour le point courant $u^n = (.., u_i^n, ..)$, nous calculons les dérivées $[\frac{\partial p_j}{\partial u_i}]_{u=u^n}, \forall i, j = 1, .., J$.
Nous présentons maintenant brièvement trois heuristiques pour résoudre notre problème de tarification. Les deux premières (GDA et SLS) utilisent l'analyse de sensibilité, la troisème (BLS) est implémentée comme point de repère.

6.2.4.b Algorithme de descente de gradient (GDA)

Dans cette première heuristique l'analyse de sensibilité est utilisée pour calculer le gradient de la fonction objectif $F(u, p)$ en appliquant les étapes successives suivantes :

$$\frac{\partial F(u, p(u))}{\partial u_j} = \frac{\partial F}{\partial u_j} + \sum_{i \in \mathcal{J}} \frac{\partial F}{\partial p_i} \frac{\partial p_i}{\partial u_j}, \forall j \in \mathcal{J}.$$

Le gradient résultant est utilisé comme direction de descente pour mettre à jour les tarifs des différents services. Les étapes sont les suivantes :
– Etape 0. *Initialisation* : les variables du fournisseurs sont initialisées u^0, et soit $n = 0$, le nombre d'itérations.
– Etape 1. Résoudre le problème SUE paramétrisé par u^n en utilisant l'analyse de sensibilité.
– Etape 2. Calculer les dérivées de $\frac{\partial p^\star}{\partial u}$, en utilisant l'analyse de sensibilité.
– Etape 3. *Trouver la direction* : Calculer la direction de descente d^n (gradient de F).
– Etape 4. *Mise à jour* : $u^{n+1} = u^n + \sigma^n d^n$, σ^n est la taille du pas de descente.
– Etape 5. *Convergence* : Si $|u^{n+1} - u^n| < \epsilon$ alors Stop, sinon retourner à l'étape 1.
A l'étape 4, nous utilisons un pas pré-définit $\sigma^n = 1/n$, et ϵ est une valeur petite pour évaluer la convergence.

6.2.4.c Recherche locale bi-niveaux (BLS)

Cet algorithme est une recherche locale ayant comme point de départ u_0 (variables du fournisseur). Dans le voisinage V_0 de u_0, la fonction objectif $F(u, p(u))$ est évaluée $\forall u \in \mathcal{V}_0$. Le meilleur voisinage $\bar{u}$ est sélectionné et amélioré $F(u, p(u))$ mettant à jour le point courant. Une nouvelle recherche est effectuée dans le voisinnage $\bar{u}$. Cette étape est répétée tant qu'une amélioration est constatée.

6.2.4.d Analyse de sensibilité basée sur de la recherche locale (SLS)

SLS est également une recherche locale utilisant le même voisinage que (BLS). La différence réside dans l'étape d'évaluation. Au lieu de calculer un SUE pour chaque voisin, basée sur l'analyse de sensibilité, nous

utilisons l'approximation suivante : $p_j^\star(u^{n+1}) = p_j^\star(u^n) + \sum_{i\in\mathcal{J}} \frac{\partial p^\star}{\partial u_i}(u_i^{n+1} - u_i^n)$.

Voici des résultats préliminaires et succincts, dans le but de fournir une idée sur les heuristiques. Nous les comparons sur deux exemples pour lesquels le fournisseur maximise son revenu $F(u,p) = \lambda \sum_{j\in\mathcal{J}} u_j p_j$. Le premier exemple, comprend 3 services : un DH, deux DW (formant le nid). Dans le second exemple le hombre de services est de 6, répartis dans deux nids.

	Ex 1 : 3 services			Ex 2 : 6 services		
	GDA	SLS	BLS	GDA	SLS	BLS
Revenue	99.05	99.06	99.20	109.02	109.02	109.78
Nb iter	354	15	15	818	50	168
Time (s)	3.88	0.20	0.99	16	1.57	198

Sur ces deux très simples exemples, (BLS) fournit la meilleure solution. Aucune conclusion générale ne peut bien entendu être tirée. (SLS) présente un intéressant compromis entre qualité de la solution et temps de calcul pour l'obtenir. Ce travail est toujours en cours de traitement.

6.3 EFFICACITÉ ÉNERGÉTIQUE DANS LES RÉSEAUX DE TÉLÉPHONIE MOBILE

Travail réalisé en collaboration avec Kai YANG (post-doctorant au LRI, université Paris Sud XI) et Steven MARTIN (professeur au LRI, université Paris Sud XI) et qui a donné lieu à la soumission d'un article dans une revue internationale (noté [s2] dans le chapitre "Liste des publications").

6.3.1 Contexte

Cette étude concerne les réseaux de téléphonie mobile. Ces dix dernières années, de nombreuses recherches ont été menées concernant l'Efficacité Spectrale (ES) (en anglais *Spectral Efficiency* (SE)), et ce pour fournir une meilleure qualité de service face à une demande toujours plus importante. En parallèle, par souci des coûts engendrés par l'utilisation de l'énergie mais aussi par la prise en compte de considérations environnementales, l'Efficacité Energétique (EE) (en anglais *Energy Efficiency* (EE)) a retenu l'attention des chercheurs dans le domaine des réseaux sans fil. Malheureusement, ces deux objectifs sont contradictoires en terme d'optimisation. En effet, l'amélioration de (ES) implique une augmentation de la consommation de l'énergie (voir [75], [76] et [77]). Ce travail est axé sur l'optimisation de l'Efficacité Energétique (EE). En générale, celle-ci est définie comme le rapport du débit consommé par un utilisateur et de la puissance globale du réseau. Le problème d'optimisation s'y rapportant possède un unique optimum global [78]. Isheden *et al.* [79] proposent de formuler le problème de maximisation de l'Efficacité Energétique (EE) comme un programme fractionnaire (PF) [80]. Nous sommes donc partis de cette formulation pour traiter le problème d'allocation de ressources et de maximisation de l'Efficacité Energétique (EE) dans un réseau hétérogène. Le modèle initial est alors un programme fractionnaire non concave en variables mixtes ($MINLFP$). Ce problème est NP-difficile et un défi quant à sa résolution pratique, du fait de difficultés additionnées : la fonction objectif fractionnaire, non concave, comprenant des logarithmes au numérateur et au dénominateur, et les variables de décision sont binaires et continues. Nous avons alors proposé de transformer ($MINLFP$) en un programme convexe non linéaire en variables mixtes ($MINLP$) dont la relaxation continue est bi-concave puis en un programme convexe non linéaire en variables mixtes ($MINLP_{sc}$) pour lequel la relaxation continue est simplement concave en substituant les termes croisés quadratiques présents à la fois dans la fonction objectif et dans les contraintes.

6.3.2 Définition formelle

Nous considérons un réseau hétérogène muni d'une technique dite OFDMA (en anglais *Orthogonal Frequency Division Multiple Access*) ; technique de multiplexage et de codage des données. Ce réseau comprend une station de base principale notée MeNB, N stations secondaires notées SeNBs, M utilisateurs et K ressources notées RB . Soient $\mathcal{N} = 0, 1, 2, ..., N$, $\mathcal{M} = \{1, 2, ..., M\}$, et $\mathcal{K} = \{1, 2, ..., K\}$ les ensembles de bases secondaires, d'utilisateurs et des ressources.

Nous mentionnons maintenant les formules des composantes nécessaires à l'écriture de l'expression de l'Efficacité Energétique.

Le débit global du réseau est donnée par l'expression :

$$R = \sum_{m=1}^{M} R_m \tag{6.3.1}$$

où

$$R_m = B \sum_{k=1}^{K} log_2(1 + \sum_{n=0}^{N} \frac{\delta_{n,m,k} p_{n,m,k} |h_{n,m,k}|^2}{N_0 B}) \tag{6.3.2}$$

avec B la bande passante de la ressource RB, $\delta_{n,m,k} \in \{0; 1\}$ inconnue du système qui vaut 1 si la station de base n affecte la ressource RB k au client m, 0 sinon, $p_{n,m,k}$ représente la puissance de transmission de la station n vers le client m pour la ressource RB k (inconnue du système également) ; $h_{n,m,k}$ est un coefficient fixé par avance, coefficient du canal entre la station n vers le client m pour la ressource RB k ; enfin N_0 est la variance du canal à bruit blanc additif Gaussien.
La consommation de la puissance sur le réseau s'exprime comme suit :

$$P_c = P_s + \xi R \tag{6.3.3}$$

avec P_s une partie statique et la seconde partie ξR est une partie dynamique proportionnelle à la puissance globale du réseau, où ξ représente la consommation de la puissance dynamique par unité.
Ainsi, la puissance totale est donnée par :

$$P = \varsigma P_t + P_s + \xi R \tag{6.3.4}$$

où $P_t = \sum_{n=0}^{N} \sum_{m=1}^{M} \sum_{k=1}^{K} \delta_{n,m,k} p_{n,m,k}$ est la puissance total de transmission et ς le coefficient inverse de l'amplificateur de puissance.

Nous pouvons maintenant établir l'expression de l'Efficacité Energétique (EE) :

$$\eta_{EE} = \frac{R}{\varsigma P_t + P_s + \xi R} \tag{6.3.5}$$

Notre objectif est alors de maximiser EE (i.e. η_{EE}) soumise à des contraintes de structure du problème. Nous établissons donc dans la prochaine sous section le programme mathématique associé.

6.3.3 Modélisation proposée

Le programme mathématique correspondant à la maximisation de l'EE sous contraintes de structure du réseau est le suivant :

$$(MINLFP)\begin{cases}\max\ \eta_{EE} & & \\ s.c \begin{cases} (C_1): \sum_{m=1}^{M}\delta_{n,m,k}\leq 1 & \forall n\in\mathcal{N},\forall k\in\mathcal{K} \\ (C_2): \sum_{m=1}^{M}\sum_{k=1}^{K}\delta_{n,m,k}p_{n,m,k}\leq p_n^{max} & \forall n\in\mathcal{N} \\ (C_3): -R_m\leq -R_m^{th} & \forall m\in\mathcal{M} \\ 0\leq p_{n,m,k}\leq p_n^{max} & \\ \delta_{n,m,k}\in\{0;1\} & \\ p_{n,m,k}\in\mathbb{R} & \end{cases}\end{cases}$$

où R_m^{th} et p_n^{max} représentent respectivement le débit minimum nécessaire pour un client m et la puissance totale maximale de transmission pour la station n. La contrainte (C_1) signifie qu'une seule ressource est affectée à un client. Les contraintes (C_2) et (C_3) imposent une puissance totale de transmission pour chaque station et un débit nécessaire pour chaque client, respectivement.

$\delta_{n,m,k}$ et $p_{n,m,k}$ sont les variables de décision du modèle, respectivement $0-1$ et réelle positive. Nous sommes donc confrontés à un programme non linéaire, fractionnaire, non concave en variables mixtes $(MINLFP)$.

Remarque 6.2 *Bien entendu, il est possible que $(MINLFP)$ n'ait pas de solution admissible, et ce en raison des contraintes (C_2) et (C_3) qui peuvent être contradictoires. Pour éviter cet écueil nous avons ajusté ces contraintes, par exemple en diminuant le débit de certains utilisateurs pour rendre réalisable le problème.*

Nous présentons maintenant lors de la prochaine sous section la voie de résolution que nous suggérons.

6.3.4 Technique de résolution adoptée

Nous proposons de résoudre $(MINLFP)$ en le transformant dans un premier temps en un programme équivalent concave (*via* l'algorithme fourni par Dinkelbach [80]) fournissant une relaxation de $(MINLFP)$ puis ce programme concave est transformé en un programme non fractionnaire sans les produits des variables binaires par les variables réelles en un programme concave $(MINLP)$.

Ré-écrivons $(MINLFP)$ comme un programme non linéaire concave $(MINLP)$:

$$(MINLP)\begin{cases}\max\ F(\lambda)=R-\lambda(\varsigma P_t+P_s+\xi R) & & \\ s.c \begin{cases} (C_1): \sum_{m=1}^{M}\delta_{n,m,k}\leq 1 & \forall n\in\mathcal{N},\forall k\in\mathcal{K} \\ (C_2): \sum_{m=1}^{M}\sum_{k=1}^{K}\delta_{n,m,k}p_{n,m,k}\leq p_n^{max} & \forall n\in\mathcal{N} \\ (C_3): -R_m\leq -R_m^{th} & \forall m\in\mathcal{M} \\ 0\leq p_{n,m,k}\leq p_n^{max} & \\ \delta_{n,m,k}\in\{0;1\} & \\ p_{n,m,k}\in\mathbb{R} & \end{cases}\end{cases}$$

où $F(\lambda)$ est continue et monotone décroissante, en fonction de λ et possède une unique solution optimale λ^* (voir [80]).

$(MINLP)$ peut être résolu directement *via* un algorithme *Branch-and-Bound* [81]. Ainsi, nous pourrions directement traiter $(MINLP)$ et en rester à ce stade du raisonnement. Toutefois, il est bien connu que dans un *Branch-and-Bound* classique, le calcul de la relaxation continue du problème initial en chaque noeud de l'arborescence joue un rôle important. Or dans ce cas, celle-ci est bi-concave et non simplement concave (en raison des deux types de variables de décision, et surtout des termes quadratiques $\delta_{n,m,k}p_{n,m,k}$ présents dans les contraintes de type (C_2) et dans la fonction objectif. Nous avons alors linéarisé ces termes produits en utilisant une technique classique proposée par Glover [20].

Le problème équivalent $(MINLP_{sc})$, toujours non linéaire, mais dont la relaxation continue est simplement concave, est alors le suivant :

$$(MINLP_{sc}) \begin{cases} \max\ F(\lambda) = R_2 - \lambda(\varsigma P_{2_t} + P_s + \xi R_2) & \\ s.c \left| \begin{array}{ll} (C_1) : \sum_{m=1}^{M} \delta_{n,m,k} \leq 1 & \forall n \in \mathcal{N}, \forall k \in \mathcal{K} \\ (C_2) : \sum_{m=1}^{M} \sum_{k=1}^{K} x_{n,m,k} \leq p_n^{max} & \forall n \in \mathcal{N} \\ (C_3) : -R_{2_m} \leq -R_m^{th} & \forall m \in \mathcal{M} \\ (C_4) : x_{n,m,k} \leq \min\{p_{n,m,k}, p_n^{max}\delta_{n,m,k}\} & \\ (C_5) : x_{n,m,k} \geq (p_{n,m,k} + p_n^{max}\delta_{n,m,k} - p_n^{max})^+ & \\ 0 \leq p_{n,m,k} \leq p_n^{max} & \\ \delta_{n,m,k} \in \{0;1\} & \\ p_{n,m,k} \in \mathbb{R} & \\ x_{n,m,k} \in \mathbb{R}^+ & \end{array} \right. \end{cases}$$

avec

$$R_{2_m} = B \sum_{k=1}^{K} log_2(1 + \sum_{n=0}^{N} \frac{x_{n,m,k}|h_{n,m,k}|^2}{N_0 B}) \tag{6.3.6}$$

$$R_2 = \sum_{m=1}^{M} R_{2_m} \tag{6.3.7}$$

$$P_{2_t} = \sum_{n=0}^{N} \sum_{m=1}^{M} \sum_{k=1}^{K} x_{n,m,k} \tag{6.3.8}$$

et où $(a)^+ = \max\{0; a\}$.

Une étude expérimentale préliminaire a été menée. Bien entendu, $(MINLP_{sc})$ compte un nombre très important de variables et contraintes et sa résolution exacte demande un temps CPU (en secondes) bien plus important que si l'on accepte une solution approchée. Bonmin (solveur libre de programmes non linéaires en variables entières, développé par Bonami *et al.* [82]) a été employé dans les deux configurations. Les premiers résultats sont reportés dans le tableau 6.1 avec une station de base, 2 utilisateurs et le nombre de ressources qui varie de 2 à 8.

6.4 CONCLUSION

Ce chapitre a été consacré à l'exposé de deux applications nécessitant inévitablement des modélisations encore plus complexes que celles suggérées dans le chapitre précédent, à savoir des modélisations sous forme de programmes non linéaires (bi-niveaux stochastique d'une part, non convexe non linéaire fractionnaire en

TAB. 6.1 – Temps CPU (en sec.)

Nbre ressources K	$K=2$	$K=3$	$K=4$	$K=5$	$K=6$	$K=7$	$K=8$
EE Optimale	6.05	11.23	17.86	49.88	87.82	309.87	969.41
EE approchée	0.0540	0.0546	0.0590	0.0606	0.0621	0.0631	0.0650

variables mixtes, d'autre part). Ces deux applications nécessitent encore du travail, surtout concernant la gestion efficace de la non linéarité et de la non convexité des programmes mathématiques établis. Ce constat est le point de départ des perspectives exposées dans le chapitre suivant.

CHAPITRE 7

Conclusion générale et perspectives

7.1 CONCLUSION GÉNÉRALE

Le travail réalisé dans le cadre de cette thèse d'habilitation est centré autours de la programmation mathématique en variables entières, d'une part d'un point de vue théorique, d'autre part d'un point de vue appliqué. Ces deux aspects ont constitué les deux parties de ce manuscrit. Nous nous sommes attachés pour chacune de ses deux parties à aborder les problèmes dans un souci de mise en oeuvre des techniques de résolution proposées.

Théorie. La partie théorique aborde deux problèmes académiques de la recherche opérationnelle : le problème du multi-sac-à-dos quadratique en variables entières (pour lequel la fonction objectif est non séparable) $(QMKP)$ et le problème du sac-à-dos quadratique en variables $0-1$, (QKP). Pour ces deux problèmes nous avons proposé une méthode de résolution exacte basée sur une linéarisation originale soit du problème initial (pour (QKP)) soit d'un problème équivalent séparable (pour $(QMKP)$).
Chacune des deux transformations est étudiée analytiquement dans un premier temps. Des méthodes exactes sont ensuite détaillées, mettant en exergue les aspects importants quant à leur mise en oeuvre en pratique, afin d'être compétitives et de dépasser les méthodes état de l'art pour certaines instances.
L'originalité des linéarisations repose sur les idées suivantes.
- Concernant $(QMKP)$: l'idée consiste à linéariser le problème initial (parce que nous avions constaté lors de ma thèse de doctorat que bien souvent le problème linéaire se comporte mieux en pratique), en passant par l'intermédiaire d'un problème séparable (obtenu via une décomposition de Gauss).
- Concernant (QKP) : l'idée consiste à linéariser le problème initial en n'ajoutant qu'une seule variable de décision (et un ensemble de contraintes en nombre exponentiel). Cette linéarisation est basée sur la caractérisation de l'enveloppe convexe du problème initial.

Applications. La partie appliquée expose le traitement de quatre problèmes, tous issus du monde réel. Le combat contre la dengue et la détection de cibles intelligentes ont été regroupés dans un même chapitre parce que nous les avons modélisé toutes deux sous forme d'un programme linéaire en variables entières. La tarification optimale d'un service de livraison en ligne ainsi que le problème d'efficacité énergétique dans les réseaux de téléphonie mobile ont été regroupés au sein d'un même chapitre nécessitant une modélisation sous forme de programmes non linéaires. Pour chacun des chapitres nous avons pris soin de définir : le contexte de l'étude, la définition formelle du problème traité et surtout la modélisation proposée avant d'établir les grandes lignes de techniques de résolution développées. Effectivement, l'accent est mis dans ce manuscrit sur l'aspect modélisation. Une idée des résultats de l'implémentation des méthodes est donnée.

7.2 PERSPECTIVES

Les travaux présentés dans ce manuscrit offrent des perspectives de recherche en lien avec la programmation non linéaire convexe ou non convexe en variables entières (ou mixtes). En effet, cette formulation permettrait le traitement d'une large gamme d'applications (cf. dernier chapitre de ce manuscrit), trop souvent abordées aujourd'hui de façon approchée dès l'étape de la modélisation *via* des programmes linéaires au lieu de non linéaires. Il serait alors intéressant de développer un outil informatique qui regrouperait le traitement des différents problèmes énoncés ci-après. Ces derniers, sont exposés du plus particulier au plus général.

- ***Programmes quadratiques convexes en variables entières avec contraintes quadratiques.***
  - *Contraintes séparables*. Dans un contexte où les contraintes sont séparables et que les variables de décision sont entières, nous pourrions simplement appliquer une linéarisation par morceaux, précédemment utilisées pour linéariser la fonction objectif de $(QMKP)$ séparable [37].
  - *Contraintes non séparables*. Dans ce cas, il s'agirait soit d'appliquer directement la transformation du séparable vers le non séparable puis la linéarisation proposée au chapitre 3 ; soit de traiter le problème

à l'aide d'une adaptation de la méthode proposée par Billionnet *et al.* [83], qui convexifie un problème initial non convexe mais peut s'appliquer à un problème déjà convexe.

– ***Programmes non linéaires convexes séparables.*** Il s'agirait ici de traiter en particulier le problème du multi-sac-à-dos non linéaire séparable en variables entières, (NLP), qui consiste à maximiser la somme de n fonctions concaves et différentiables $f_i(x_i)$, soumise à m contraintes linéaires séparables du type $\sum_{i=1}^{n} g_i(x_i)$ (où chaque g_i est concave et différentiable) et à l'intégrité des variables x_i. Le problème (NLP) trouve en particulier une application dans le problème de l'allocation de ressources. Le degré de chaque partie, $f_i(x_i)$, de la fonction économique est alors égale à 3. Bretthauer et Shetty [84] proposent une méthode de type *branch-and-bound* reposant sur des procédures de ré-optimisation du problème. Cette méthode s'avère très performante en comparaison d'une méthode classique comme celle utilisée par un logiciel commercial. L'adaptation de notre algorithme de branch-and-bound développé pour $(QMKP)$ séparable [37] afin de résoudre ce problème (NLP) nous semble donc une perspective particulièrement intéressante.

– ***Programmes fortement non linéaires non convexes.***
  – *Programme quadratique*. Billionnet *et al.* ont réalisé l'un des rares travaux concernant la classe la plus générale des programmes quadratiques en variables entières. Ces derniers, proposent une première étape consistant à rendre convexe le programme non convexe *via* la programmation semi-définie [47]. Il serait sans doute intéressant de trouver un moyen d'adapter la linéarisation que nous avons développée pour $(QMKP)$ convexe non séparable (cf. chapitre 3 au cas non convexe, et nous comparer à [47].
  – *Programme fortement non linéaire non convexe*. Le traitement de cette classe de problème découle directement de l'application exposée lors de la deuxième section du chapitre 6. En effet, le programme mathématique relatif à ce dernier présente : une fonction objectif fractionnaire (comprenant au numérateur et au dénominateur un logarithme) non convexe, des contraintes non linéaires et des variables de décision mixtes (0-1 et réelles). Pour le moment nous avons géré la non convexité du problème en appliquant une technique classique en lien avec la programmation fractionnaire. Toutefois, nous souhaiterions tenter de travailler sur les logarithmes dans un premier temps, qui sont des fonctions concaves. Dans un second temps, l'aspect fractionnaire serait contourné par la convexification globale de la fonction objectif. Enfin, la nature des variables de décision devrait être traitée en même temps que le traitement des fonctions logarithmiques (si une linéarisation des celles-ci est possible). Liberti expose de nombreuses méthodes de reformulations de programmes non linéaires dans [85] ; il serait très intéressant de nous en inspirer.
  – *Programme Bi-niveaux avec contraintes d'équilibre stochastique*. Le traitement de cette classe de problème provient de l'application présentée au chapitre 6. Il serait intéressant dans ce contexte de poursuivre les efforts concernant le développement d'une heuristique qui fournirait une solution admissible pour le problème initial, basée sur l'analyse de sensibilité et la recherche locale.

Liste des publications

REVUES INTERNATIONALES

1. Y. Hayel (40%), D. Quadri (40%), T. Jimenez (15%) and L. Brotcorne (5%) : A queueing model for last mile delivery service with noncooperative customers (**Annals of Operations Research**, $link.springer.com/article/10.1007/s10479-014-1647-x$)

2. D. Quadri (70%) and E.Soutil (30%) : Reformulation and solution approach for nonseparable integer quadratic programs (**Journal of Operational Research Society**, $doi10.1057/jors.2014.76, http://www.palgrave-journals.com/jors/journal/vaop/ncurrent/full/jors201476a.html$)

3. B. Detienne (40%), D. Quadri (40%), CD. Rodrigues (20%) : Two phase resolution for the general moving target search problem based on a 0-1 linear model, **International Journal of Production Research** 52(24), 7177-7192 (2014)

4. D. Feillet (20%), T. Garaix (20%), F. Lehuede (20%), O. Peton (20%) and D. Quadri (20%) : A new consistent vehicle routing problem for the transportation of handicaped persons, **Networks** 63(3), 211-224 (2014)

5. N. Camelin (20%), B. Detienne (20%), S. Huet (20%), D. Quadri (20%) and F. Lefevre (20%) , Concept discovery and automatic semantic annotation for language understanding in an information ? query dialogue system using Latent Dirichlet Allocation and segmental methods, **Communications in Computer and Information Science Series**, Springer 348, 45-59 (2013)

6. C-D. Rodrigues (45%), D. Quadri (45%), P. Michelon (5%)and S. Gueye (5%), A t-linearization for QKP, **SIAM Journal on Optimization** 22(4), 1449-1468 (2012)

7. F. Della Croce (20%) and D. Quadri (80%), Improving an exact approach for solving separable integer quadratic knapsack problems, **Journal of Combinatorial Optimization** 23(1), 21-28 (2012)

8. D. Quadri (70%) and E. Soutif (30%), A roof linearization algorithm to obtain a tight upper bound for integer nonseparable quadratic programming, **Electronic Notes in Discrete Mathematics** 36, 271-278 (2010)

9. D. Quadri (70%), E. Soutif (25%) and P. Tolla (5%), , Exact solution method to solve large scale integer quadratic multidimensional knapsack problems, **Journal of Combinatorial Optimization** 17 (2), 157-167 (2009)

10. D. Quadri (70%) and E. Soutif (30%), Rewriting integer variables into zero-one variables : some guidelines for the integer quadratic multi-knapsack problem, **Operational Research** 7 (2), 299-314 (2007)

11. D. Quadri (70%), E. Soutif (25%) and P. Tolla (5%), Upper bounds for large scale integer quadratic multidimensional knapsack, **International Journal of Operations Research** 4 (3), 146-154 (2007)

ARTICLES SOUMIS DANS UNE REVUE INTERNATIONALE

1. K. Yang, S. Martin, D. Quadri, J. Wu and G. Feng, Energy-Efficient Downlink Resource Allocation in LTE Heterogeneous Networks

2. T. Garaix, D. Feillet, D. Quadri, F. Lehuede, O. Peton, A branch-and-price algorithm for the Consistent Vehicle Routing Problem

CONFÉRENCES INTERNATIONALES AVEC ACTES

1. B. Tousni, Y. Hayel, D. Quadri, L. Brotcorne : Mathematical Programming with Stochastic Equilibrium Constraints applied to Last-mile Delivery Services, **International Network Optimization Conference 2015**

2. B. Tousni, Y. Hayel, D. Quadri, L. Brotcorne, Last-mile delivery services design under stochastic user, **INFORMS Transportation Science and Logistics Society Workshop 2015**

3. Y. Hayel, D. Quadri, T. Jimenez, L. Brotcorne A Game Theoretical Model for Freight Transportation invited paper, **International Conference on Analytic and Stochastic Modelling Techniques and Applications (ASMTA) 2012**

4. N. Camelin, B. Detienne, S. Huet, D. Quadri and F. Lefevre, Unsupervised Concept Annotation using Latent Dirichlet Allocation and Segmental Methods, **EMNLP 2011**

5. N. Camelin, B. Detienne, S. Huet, D. Quadri and F. Lefevre, Concept Discovery for Language Understanding in an Information-Query Dialogue, **KDIR 2011**

6. CD Rodrigues, D. Quadri, P. Michelon, S. Gueye, A t-linearization to exactly solve 0-1 quadratic knapsack problems, **EWMNLP 2010**

7. P. Michelon, D. Quadri and M. Neigreiros, On a class of periodic scheduling problems : models, lower bounds and heuristics, **IMCSIT 2008 Conference**, IEEE Catalog Number CFP0864E-CDR, 899-906 (2008)

8. D.Quadri, E. Soutif et P. Tolla, A branch-and-bound algorithm to solve large scale integer quadratic multidimensional knapsack problems, **SOFSEM 2007**, Springer LNCS 4362, 456-464 (2007)

CHAPITRES DE LIVRES

1. D. Quadri, E. Soutif et P. Tolla , Integer quadratic knapsack problems, chapter 9 of **Combinatorial Optimization and Theoretical Computer Science**, 267-295, Wiley InterSience (2010)

2. D. Quadri, E. Soutif et P. Tolla, An Upper Bound for the Integer Quadratic Multi-knapsack Problem, chapter 19 of **Combinatorial optimization - Theoretical computer science : interfaces and perspectives**, 495-505, Wiley InterScience (2008)

3. D. Quadri, E. Soutif et P. Tolla, Les problèmes de sac-à-dos quadratiques en variables entières, **Optimisation combinatoire**, volume 4, 191-211, Hermès-Sciences (2007)

4. D. Quadri, E. Soutif et P. Tolla, Programmation Quadratique en Entiers : un majorant pour le multi-sac-à-dos quadratique séparable entier, **Annales du Lamsade No.4 "30 ans du Lamsade"**, 411-422 (2007)

Bibliographie

[1] G. S. Lueker. Two np-complete problems in nonnegative integer programming. *Computer Science Labatory, Princenton, NJ*, Report No. 178 (A6), 1975.

[2] P. Chaillou, P. Hansen, and Y. Mathieu. Best network flow bounds for the quadratic knapsack problem. *Lecture Notes in Mathematics*, 1403 :226–235, 1986.

[3] S. Martello and P. Toth. *Knapsack Problems : Algorithms and Computer Implementations.* John Wiley & Sons, Inc., New York, NY, USA, 1990.

[4] M. Weingartner. Capital budgeting of interrelated projects : survey and synthesis. *Management Science*, 12 (7) :485–516, 1966.

[5] T.L. Morin and R.E. Martsen. Branch and bound strategies for dynamic programming. *Operations Research*, 24 (4) :611–627, 1976.

[6] T.L. Morin and R.E. Martsen. An algorithm for nonlinear knapsack problems. *Management Science*, 22 (10) :1147–1158, 1976.

[7] B. Faaland. An integer programming algorithm for portfolio selection. *Management Science*, 20 (10) :1376–1384, 1974.

[8] H.M. Markowitz. Portfolio selection. *Journal of Finance*, 7 (1) :77–91, 1952.

[9] D. Li, Sun X.L., and J. Wang. Optimal lot solution to cardinality constrained mean-variance formulation for portfolio selection. *Mathematical Finance*, 16 :83–101, 2006.

[10] T. Van Zandt. Heuristic and exact solution method for convex nonlinear knapsack problem. *CEPR Discussion Pape*, 4022, 2003.

[11] M-C. Plateau and Y-A. Rios-Solis. Optimal solutions for unrelated parallel machines scheduling problems using convex quadratic reformulations. *European Journal of Operational Research*, 201(3) :729–736, 2010.

[12] M. Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *Journal of ACM*, 48(2) :206–242, 2001.

[13] W.D. Pisinger. The quadratic knapsack problem - a survey. *Discrete Applied Mathematics*, 155 :623–648, 2007.

[14] G. Gallo, P.L. Hammer, and B. Simeone. Quadratic knapsack problem. *Mathematical Programming Study*, 12 :132–149, 1980.

[15] A. Caprara, D. Pisinger, and P Toth. Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing*, 11 :125–137, 1999.

[16] P. Michelon and L. Veilleux. Lagrangean methods for the 0-1 quadratic knapsack problem. *European Journal of Operational Research*, 92 :326–341, 1996.

[17] A. Billionnet and E. Soutif. An exact method for the 0-1 quadratic knapsack problem based on lagrangian decomposition. *European Journal of Operational Research*, 157(3) :565–575, 2004.

[18] Létocart, A. L., Nagih, and G. Plateau. Reoptimization in lagrangian methods for the quadratic knapsack problem. *Computers & Operations research*, 39 :12–18, 2012.

[19] D. Pisinger, A. Bo Rasmussen, and R. Sandvick. Solution of large-sized quadratic knapsack problems through agrgessive reduction. *INFORMS Journal on Computing*, 19(2) :280–290, 2007.

[20] F. Glover. Improved linear integer progamming formulations of nonlinear integer problems. *Management Science*, 22 (4) :455–460, 1975.

[21] H. Sherali and W. Adams. *A reformulation technique for solving discrete and continuous non convex problems*. Kluwer Academic Publishers, 1999.

[22] S. Elloumi, A. Faye, and E. Soutif. Decomposition and linearization for 0-1 quadratic programming. *Annals of Operations Research*, 99 :79–93, 2000.

[23] W. Chaovalitwongse, P.M Pardalos, and O.A. Prokopyev. New linearization technique for multi-quadratic 0-1 programming problems. *Operations Research Letters*, 32 :517–522, 2004.

[24] N. Krislock, J. Malick, and F. Roupin. Nonstandard semidefinite bounds for solving exactly 0-1 quadratic problems. In *EURO XXV*, 2012.

[25] F. Roupin. Algorithmes combinatoires et relaxations par programmation linéaire et semidéfinie. application à la résolution de problèmes quadratiques et d'optimisation dans les graphes. *Habilitation à Diriger des Recherches*, 2006.

[26] E. Balas and J.B. Mazzola. Nonlinear 0-1 programming : linearization techniques. *Mathematical Programming*, 30 :1–21, 1984.

[27] E. Balas and J.B. Mazzola. Nonlinear 0-1 programming : dominance relations and algorithms. *Mathematical Programming*, 30 :22–45, 1984.

[28] S. Gueye and P. Michelon. Linearization for quadratic 0-1 problems. *Annals of Operations Research*, 140 :235–261, 2005.

[29] S. Gueye and P. Michelon. A linearization framework for unconstrained quadratic (0-1) problems. *Discrete Applied Mathematics*, 15 :1255–1266, 2009.

[30] R. Fortet. Application de l'algèbre de boole en recherche opérationelle. *Revue Française de la Recherche Opérationelle*, 4 :17–25, 1960.

[31] P. Hansen and Meyer C. Improved compact linearization for the unconstrained quadratic 0-1 minimization problem. *Discrete Applied Mathematics*, 157 :1267–1290, 2009.

[32] W.P. Adams, R.J. Forrester, and F. Glover. Comparisons and enhancement strategies for linearizing mixed 0-1 quadratic programs. *Discrete Applied Mathematics*, 1 :99–120, 2004.

[33] A. Billionnet, S. Elloumi, and M-C. Plateau. Improving the performance of standard solvers for quadratic 0-1 programs by a tight convex reformulation : The qcr method. *Discrete Applied Mathematics*, 157(6) :1185–1197, 2009.

[34] L. Liberti. Compact linearization for binary quadratic problems. *4OR*, 5(3) :231–245, 2007.

[35] K. Bretthauer and B. Shetty. A branch and bound algorithm for integer quadratic knapsack problems. *INFORMS Journal on Computing*, 7(1) :109–116, 1995.

[36] M. Djerdjour, K. Mathur, and H. Salkin. A surrogate-based algorithm for the general quadratic multidimensional knapsack. *Operation Research Letters*, 7 (5) :253–257, 1988.

[37] D. Quadri, E. Soutif, and P Tolla. Exact solution method to solve large scale integer quadratic multidimensional knapsack problems. *Journal of Combinatorial Optimization*, 17(2) :157–167, 2009.

[38] B. Zhang and B. Chen. Heuristic and exact solution method for convex nonlinear knapsack problem. *Asia-Pacific Journal of Operations Research*, 29(5) :1250031, 2012.

[39] Z. Hua, B. Zhang, and L. Liang. An approximate dynamic programming approach to convex quadratic knapsack problems. *Computers & Operations Research*, 33 :660–673, 2006.

[40] B. Zhang and Z. Hua. Simple solution methods for separable mixed linear and quadratic knapsack problem. *Applied Mathematical Modelling*, 36 :3245–3526, 2012.

[41] M. Djerdjour. An enumerative algorithm framework for a class of nonlinear integer programming problems. *European Journal of Operational Research*, 101 :104–121, 1997.

[42] J.P. Dussault, J.A. Ferland, and B. Lemaire. Convex quadratic programming with one constraint and bounded variables. *Mathematical Programming*, 36 :90–104, 1986.

[43] J.J. More and S.A Vavasis. On the solution of concave knapsack problems. *Mathematical Programming*, 49 :397–411, 1991.

[44] H.P. Benson, S.S. Erenguc, and R. Horst. A note on adapting methods for continuous global optimization to the discrete case. *Annals of Operations Research*, 25 :243–252, 1990.

[45] S.S. Erenguc and H.P. Benson. An algorithm for indefinite integer quadratic programming. *Computational Mathematics and Applications*, 21 :99–106, 1991.

[46] R. Horst and H. Tuy. *Global Optimization : Deterministic Approaches.* Springer, Berlin, 1990.

[47] A. Billionnet, S. Elloumi, and A. Lambert. An efficient compact quadratic convex reformulation for general integer quadratic programs. *Computational Optimization and Applications*, 54 :141–162, 2013.

[48] X.J. Zheng, X.L. Sun, and D Li. Separable relaxation for nonconvex quadratic integer programming : integer diagonalisation approach. *Journal of Optimization Theory and Applications*, 146 :463–489, 2010.

[49] Z. Minjin and C. Wei. Reducing the number of variables in integer quadratic programming problem. *Applied Mathematical Modelling*, 34 :424–436, 2010.

[50] O.Y. Ozaltin, O.A. Prokopyev, and A.J Schaefer. Two-stage quadratic integer programs with stochastic right-hand sides. *Mathematical Programming Series A*, 133 :121–158, 2012.

[51] J.H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, England, 1965.

[52] D. Quadri, E. Soutif, and P Tolla. Upper bounds for large scale integer quadratic multidimensional knapsack. *International Journal of Operations Research*, 4(3) :146–154, 2007.

[53] P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4 :63–86, 1998.

[54] A. Billionnet, A. Faye, and E. Soutif. A new upper bound for the 0-1 quadratic knapsack problem. *European Journal of Operational Research*, 112 (3) :664–672, 1999.

[55] W. D. Pisinger, A. Rasmussen, and R. Sandvik. Solution of large quadratic knapsack problems through agressive reduction. *INFORMS Journal on Computing (to appear)*, 2006.

[56] A. Billionnet and F. Calmels. Linear programming for the 0-1 knpasack problem. *European Journal of Operational Research*, 92 :310–325, 1996.

[57] A. Caprara, D. Pisinger, and P Toth. Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing*, 11 :125–137, 1999.

[58] C. Oliva, P. Michelon, and C. Artigues. Constraint and linear programming : Using reduced costs for solving the zero/one multiple knapsack problem. In *International Conference on Constraint Programming, Proceedings of the workshop on Cooperative Solvers in Constraint Programming (CoSolv 2001)*, pages 87–98, 2001.

[59] E. Soutif. Expreriments for qkp. *website of the CEDRIC CNAM Paris, http ://cedric.cnam.fr/ soutif/*, 2001.

[60] M. Negreiros, A.E. Xavier, A.F. Xavier, and P. Michelon. A framework of computational systems and optimization models for the prevention and combat of dengue. In *17th IFORS Triennial Conference, Hawai*, 2005.

[61] A.M. Campbell and J.R. Hardin. Vehicle minimization for periodic deliveries. *European Journal of Operational Research*, 165 :668–684, 2005.

[62] B.O. Koopman. *Search and screening.* Pergamon, 1980.

[63] R. Hohzaki. Search allocation game. *European Journal of Operational Research*, 172 :101–119, 2006.

[64] P. Zhendong, T. Jiafu, and R.Y.K Fung. Synchronization of inventory and transportation under flexible vehicle constraint : A heuristics approach using sliding windows and hierarchical tree structure. *Operations Research*, 28(6) :1275–1289, 1980.

[65] S.S. Brown. Optimal search for a moving target in discrete time and space. *European Journal of Operational Research*, 192(3) :824–836, 2009.

[66] R. Ducret and L. Delaitre. Parcel delivery and urban logistics- changes in urban courier, express and parcel services : the french case. In *13th World Conference on Transport Research*, 2013.

[67] E. Verhoef. Economic efficiency and social feasibility in the regulation of road transport externalities. *PhD Theis*, 1996.

[68] B. Colson, P. Marcotte, and G. Savard. Bilevel programming : A survey. *4OR*, 3 :87–107, 2005.

[69] Y. Hayel, D. Quadri, T. Jimenez, and L. Brotcorne. Decentralized optimization of last-mile delivery services with non-cooperative bounded rational customers. *Annals of Operations Research*, 2015.

[70] M. Ben-Akiva and S. Lerman. *Discrete choice analysis : Theory and application to travel demand.* MA :MIT Press, Cambridge, 1985.

[71] Y. Sheffi. Urban transportation networks : Equilibrium analysis with mathematical programming methods. *Prentice-Hall*, 1985.

[72] S Anderson, A. de Palma, and J. Thissei. *Discrete choice theory of product differentiation.* MA :MIT Press, Cambridge, 1992.

[73] D. MacFadden. Quantal choice analysis : A survey. *Annals of Economic and Social Measurement*, 5 :363–370, 1976.

[74] H. Yang. Sensitivity analysis for queuing equilibrium network flow and its application to traffic control. *Mathematical and computer modeling*, 22(4) :247–258, 1995.

[75] Y. Chen, S. Zhang, S. Xu, and G. Y. Li. Fundamental trade-offs on green wireless networks. *IEEE Commun. Mag.*, 40(6) :30–37, 2011.

[76] C. Xiong, G.Y. Li, S. Zhang, Y. Chen, and S. Xu. Energy and spectral-efficiency tradeoff in downlink ofdma networks. *IEEE Trans. Wireless Commun.*, 10(10) :3874–3886, 2011.

[77] C. He, B. Sheng, P. Zhu, X. You, and G. Y. Li. Energy- and spectral-efficiency tradeoff for distributed antenna systems with proportional fairness. *IEEE J. Sel. Areas Commun.*, 31(5) :894–902, 2013.

[78] G. Miao, N. Himayat, and G. Y. Li. Energy-efficient link adaptation in frequency-selective channels. *IEEE Trans. Commun.*, 58(2) :545–554, 2010.

[79] C. Isheden, Z. Chong, E.A. Jorswieck, and G. Fettweis. Framework for link-level energy efficiency optimization with informed transmitter. *IEEE Trans. Wireless Commun.*, 11(8) :2946–2957, 2012.

[80] W. Dinkelbach. On nonlinear fractional programming. *Management Sci.*, 13(7) :492–498, 1967.

[81] S. Burer and A. N. Letchford. Non-convex mixed-integer nonlinear programming : a survey. *Operations Research and Management Science*, 17(2) :97–106, 2012.

[82] P. Bonami, L.T. Biegler, Conn A.R., G. Cornujols, I.E. Grossmann, C.D. Laird, J. Lee, A. Lodi, F. Margot, and N. Sawaya. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5 :186–204, 2012.

[83] A. Billionnet, S. Elloumi, and A. Lambert. Convex reformulations of integer quadratically constrained problems. In *ISMP (21th International Symposium of Mathematical programming)*, page 1 page, Berlin, Germany, August 2012.

[84] K. Bretthauer and B. Shetty. Quadratic resource allocation with generalized upper bounds. *Operations Research Letters*, 20 :51–57, 1997.

[85] L. Liberti. Reformulation techniques in mathematical programming. *Habilitation à Diriger des Recherches*, 2007.