Relaxations et calculs de bornes inférieures pour le QAP

W. Benajam[1], A. Lisser[1], M. Minoux[2]
1.Université de Paris Sud XI, Laboratoire de Recherche en Informatique,
Bâtiment 490, 91405 ORSAY Cedex.
benajam,lisser@lri.fr
2.Université de Paris VI, Laboratoire d'Informatique de Paris 6,
4, place jussieu 75251 Parix cedex05.
Michel.Minoux@lip6.fr

27 septembre 2005

Résumé : Nous présentons un algorithme de plans coupants pour le calcul des bornes inférieures pour le problème de l'affectation quadratique. Ce problème est un problème classique d'optimisation combinatoire dans lequel il convient de trouver le placement optimal de n unités sur n sites de façon à minimiser un coût quadratique dépendant à la fois des distances inter-sites et des flux inter-unités. L'affectation quadratique se modélise par un problème quadratique en nombres entiers. Elle a de très nombreuses applications pratiques mais reste extrêmement difficile à résoudre.
Nous proposons un algorithme de plans coupants. Celui-ci ajoute itérativement à une relaxation SDP des inégalités valides induisant des facettes du polytope quadrique. Nous verrons à travers les résultats numériques l'intérêt de cet algorithme.

Mots clés : Optimisation combinatoire, Programmation semidéfinie, Affectation quadratique, Algorithme de coupes.

Abstract : In this paper, we present a new cutting plane algorithm for solving quadratic assignment problem, hereby called QAP. This problem has been widely studied in the literature for the last decades. It consists in solving the location problem of n units among n given sites in order to minimize quadratic costs composed by both inter-sites costs and units flows. QAP is known to be hard problem to solve. However, it has several practical applications. We propose a new cutting plane algorithm for solving QAP. We use different cuts coupled with SDP relaxations for solving the master program. Among all valid inequalities of the Quadric polytope are used. Our competitive numerical results show the efficiency of our algorithm.

Keywords : Combinatorial Optimization, Semidefinite Programming, quadratic assignment problem, Cutting Plane Algorithm.

1 Introduction

1.1 Énoncé du problème

Le Problème d'Affectation Quadratique (QAP) est l'un des problèmes d'optimisation combinatoire les plus connus et qui a suscité un très grand nombre de travaux. Il a été introduit pour la première fois par Koopmans et Beckmann [25] en 1957 afin de déterminer la meilleure implantation de n unités (usines, services hospitaliers...) sur n sites connus. Il s'agit d'affecter une et une seule unité sur chaque site en minimisant le coût global de transit plus le coût d'installation. Le coût d'un transit est ici supposé égal au produit de la distance entre les sites multiplié par le flux de matière transportée. Pour exprimer le problème, on définit :

- $F = (f_{ij})$, la matrice des flux entre les unités,
- $D = (d_{ij})$, la matrice des distances entre les sites,
- $C = (c_{ij})$, le coût d'installation de l'unité i sur le site j,
- Π, l'ensemble des permutations de $\{1, 2, ..., n\}$ dans $\{1, 2, ..., n\}$.

Le problème d'affectation quadratique s'écrit alors :

$$\left\{ \min_{p \in \Pi} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{i,j} d_{p(i),p(j)} + \sum_{i=1}^{n} c_{i,p(i)} \right. \tag{1}$$

Dans ce contexte, $p(i) = j$ signifie que l'unité i est affectée au site j. Il n'est pas nécessaire que les matrices F et D soient symétriques ou carrées, même si elles le sont en pratique dans la majorité des cas. Notons que s'il y a plus de sites que d'unités, alors on peut ajouter des unités fictives qui ont un flot d'échange nul avec toutes les unités.
Il existe d'autres formalisations mathématiques, qui sont équivalentes pour ce problème, et qui permettent d'appliquer différentes approches de résolution. Toutes ces formulations sont décrites par Burkard et al. [32].

1.2 Les applications du QAP

Initialement conçu pour traiter un problème économique qui consiste à déterminer la meilleure implantation d'usines à des sites pré-déterminés, le problème d'affectation quadratique trouve de nombreuses applications concrètes et variées. Citons par exemple :

- Le placement de circuits VLSI[1] sur une micro plaquette [39] : il s'agit de déterminer le schéma de câblage qui minimise la longueur des fils utilisés ;
- Le placement des touches sur une machine à écrire : il s'agit de déterminer la place des touches sur un clavier qui minimisera les distances parcourues pour la saisie d'un texte défini ;
- Le placement de services dans un hôpital [14] et [26] : on cherche à minimiser la distance parcourue par les malades entre les différents services ;
- L'organisation dans une équipe de course de relais [20] ;
- L'ordonnancement de tâches dans une chaîne de production [17] ;
- D'autre types d'applications très variées existent comme l'ergonomie, la planification ou l'analyse des réactions chimiques.

Du point de vue théorique, de nombreux problèmes NP-complets sont des cas particuliers du problème d'affectation quadratique. Par exemple :

- Le problème du voyageur du commerce : la matrice des distances D est celle des distances entre les villes, et la matrice des flux F correspond à la matrice d'adjacence d'un cycle de longueur n.
- Le problème de la bipartition d'un graphe : la matrice D correspond à la matrice d'adjacence du graphe, la matrice F correspond à la matrice d'adjacence de deux sous-graphes complets disjoints de taille $\frac{n}{2}$.

2 Les méthodes de résolution

Plusieurs travaux de recherche ont été menés sur ce problème complexe. Il s'agit d'un problème NP-difficile [16] non approximable [37] sous l'hypothèse largement admise que $P \neq NP$.
Résoudre un problème NP-difficile comme le problème d'affectation quadra-

[1] En anglais "Very Large Scale Integration"

tique peut se faire de deux façons différentes :

- Méthodes exactes : dans ce cas, on cherche la meilleure solution possible et on prouve qu'elle est bien optimale.
- Méthodes approchées : dans ce cas, on cherche à obtenir une "bonne" solution, sans aucune garantie qu'elle soit la meilleure.

La première façon de procéder répond exactement au problème posé mais ne permet le plus souvent de résoudre que des instances de petite taille.
La seconde voie est donc très utile pour pouvoir aborder des problèmes de taille plus importante.
Nous allons examiner les principales méthodes de résolution qui ont été utilisées pour le problème de l'affectation quadratique. Pour plus de détails, le lecteur pourra se référer à [13, 2, 3].

2.1 Les méthodes approchées

De nombreuses méthodes ont été proposées pour résoudre le QAP de manière approchée. Pour les méthodes constructives, nous citons par exemple l'heuristique gloutonne proposée par Heragu et Kusiak [22]. Pour les méthodes itératives qui se sont développées depuis les années 60, citons par exemple les méthodes de Hiller H63 [23] et H63-66 [24] ainsi que la méthode CRAFT (Computerized Relative Allocation of Facilities Technique) [9]. Ces méthodes sont en grande majorité à base de recherche locale.
Des métaheuristiques ont été aussi appliquées sur le QAP. Elles ont donné de bons résultats. Parmi ces métaheuristiques, nous citons les méthodes de recherche Tabou [38, 40, 5, 41], le recuit simulé [10, 42]. Les algorithmes génétiques ont été également appliqués aussi au QAP, soit d'une manière autonome (voir [28, 15, 29]), soit en conjonction avec le recuit simulé [43]. On trouve aussi des méthodes hybrides qui combinent le recuit simulé avec la recherche Tabou, voir [7]. Par ailleurs, des heuristiques basées sur la méthode GRASP (greedy randomized search procedure) et des méthodes à base de colonies de fourmis ont été proposées pour la résolution du QAP dans [11]

2.2 Les méthodes exactes

Les méthodes de résolution exactes les plus connues pour le problème de l'affectation quadratique sont de type Branch-and-Bound et sont donc basées sur l'énumération implicite. Ces méthodes nécessitent le calcule de bornes inférieures de bonne qualité au préalable. Étant donné que le QAP est quadratique, de nombreuses linéarisations ont été proposées. La toute première a été proposées par Lawler en 1963 [27]. Depuis, d'autres linéarisations ont été proposées (Bazaraa et Sherali 1980 [6], Burkard et al 1987 [31], Adams et Johnson 1994 [1]. Elles diffèrent par leur nombre de contraintes et par leur nombre de variables.

En 1995, Resende, Ramakrishnan et Drezner ont proposé dans [35] une linéarisation du QAP basée sur la linéarisation d'Adams et Johnson [1]. Cette relaxation est celle qui donne les meilleures résultats, nous allons alors la détailler dans la section 3.2 afin de l'utiliser par la suite.
D'autres techniques pour calculer des bornes inférieures ont été appliquée au QAP. Citons par exemple, les bornes inférieures basées soit sur la projection des valeurs propres [4], soit sur la programmation semidéfinie [30], [34] et [36]. Ces relaxation feront l'objet du paragraphe 3.3.
Comme nous l'avons déjà vu, la première modélisation du QAP a été faite par Koopmans et Beckmann, dans cette version, la fonction objectif comporte un terme quadratique ainsi qu'un terme linéaire qui correspond au coût d'implantation. Une version simplifiée largement étudiée consiste à considérer le coût d'implantation nul ($c_{ij} = 0$). Dans cette version, la formule (1) devient alors :

$$\left\{ \min_{p \in \Pi} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{i,j} d_{p(i),p(j)} \right. \tag{2}$$

Nous allons donc considérer cette version afin de calculer des bornes inférieures de notre problème.

3 Calcul de bornes inférieures

Afin de calculer des bornes inférieures de notre problème, nous allons d'abord en présenter deux formulations l'une quadratique et l'autre linéaire. D'autres formulations ont été développées dans la littérature, voir par exemple [31].

3.1 Formulation en programme quadratique à variables entières

On associe à chaque permutation p une matrice booléenne $X = (x_{ij})$, avec $x_{ij} = 1$ si $j = p(i)$. Les éléments x_{ij} peuvent être interprétés comme suit :

$$x_{ij} = \begin{cases} 1 & \text{si l'unité } i \text{ est affectée au site } j \\ 0 & \text{sinon} \end{cases}$$

Le problème (2) peut se mettre sous la forme suivante :

$$(QAP)\begin{cases} \min & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ij}d_{kl}x_{ik}x_{jl} \\ \text{s.c.} & \\ & (1) \quad \displaystyle\sum_{i=1}^{n} x_{ik} = 1 \qquad \forall k \in \{1,...,n\} \\ & (2) \quad \displaystyle\sum_{k=1}^{n} x_{ik} = 1 \qquad \forall i \in \{1,...,n\} \\ & (d1) \quad x_{ik} \in \{0,1\} \qquad \forall i,k \in \{1,...,n\} \end{cases}$$

où les contraintes (1) et (2) sont les contraintes dites d'affectation. Elles permettent d'assurer que chaque unité soit placée sur un site et que chaque site accueille une et une seule unité.

3.2 Formulation en programme linéaire à variables entières

Il est possible de linéariser la formulation précédente (QAP) du problème de l'affectation quadratique. En 1995, Resende, Ramakrishnan et Drezner ont proposé dans [35] une telle linéarisation. Nous nous intéressons particulièrement à cette linéarisation car il est montré qu'elle donne des bornes inférieures de meilleure qualité que les bornes classiques de Gilmore-Lawler [18].
Cette linéarisation consiste à poser :

$$y_{ikjl} = x_{ik}x_{jl}$$
$$c_{ij}^{kl} = f_{ij}d_{kl}$$

et à ajouter les contraintes suivantes :

$$(3) \quad \sum_{i=1}^{n} y_{ikjl} = x_{jl} \qquad \forall k,j,l \in \{1,...,n\}$$

$$(4) \quad \sum_{k=1}^{n} y_{ikjl} = x_{jl} \qquad \forall i,j,l \in \{1,...,n\}$$

On obtient alors le programme linéaire en nombres entiers suivant :

$$(QAP_L)\begin{cases} \min & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} c_{ij}^{kl} y_{ikjl} & \\ \text{s.c} & & \\ & (1) \quad \displaystyle\sum_{i=1}^{n} x_{ik} = 1 & \forall k \in \{1,...,n\} \\ & (2) \quad \displaystyle\sum_{k=1}^{n} x_{ik} = 1 & \forall i \in \{1,...,n\} \\ & (3) \quad \displaystyle\sum_{i=1}^{n} y_{ikjl} = x_{jl} & \forall k,j,l \in \{1,...,n\} \\ & (4) \quad \displaystyle\sum_{i=1} y_{ikjl} = x_{jl} & \forall k,j,l \in \{1,...,n\} \\ & (d1) \quad x_{ik} \in \{0,1\} & \forall i,k \in \{1,...,n\} \\ & (d2) \quad y_{ikjl} \in \{0,1\} & \forall i,j,k,l \in \{1,...,n\} \end{cases}$$

Le modèle mathématique (QAP_L) est un programme linéaire en nombres entiers. Ce programme comporte $\frac{n^2(n^2+1)}{2}$ variables et $2n^3+2n$ contraintes. On peut diminuer sensiblement le nombre de variables et de contraintes, en utilisant d'une part le fait que $y_{ikjl}=0$ pour $i=j$ et $k\neq l$ et aussi $y_{ikjl}=0$ pour $k=l$ et $i\neq j$. Et d'autre part, $y_{ikjl}=y_{jlik}$.

Notons que la relaxation linéaire continue $(Q\bar{A}P_L)$ se déduit de (QAP_L) en remplaçant d'une part $x_{ik}\in\{0,1\}\quad \forall i,k\in\{1,...,n\}$ par $x_{ik}\geq 0\quad \forall i,k\in\{1,...,n\}$ et d'autre part $y_{ikjl}\in\{0,1\}\quad \forall i,j,k,l\in\{1,...,n\}$ par $y_{ikjl}\geq 0,\quad \forall i,j,k,l\in\{1,...,n\}$.

Maintenant que nous avons détaillé les formulations du QAP, nous allons les utiliser pour donner deux relaxations SDP différentes.

3.3 Programmation semidéfinie pour le QAP

Afin de calculer les bornes inférieures pour le problème de l'affectation quadratique, nous avons développé des méthodes basées sur la relaxation semidéfinie positive des programmes mathématiques (QAP) et (QAP_L).

3.3.1 Relaxation semidéfinie SDP_0

Pour écrire les différentes relaxations semidéfinies du (QAP), on utilisera les notations suivantes :

On définit la matrice Z dans $\Re^{n\times n}$ par :

$$Z = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix}$$

Les contraintes d'affectation sont équivalentes à $Ze = e$ et à $Z^T e = e$ où e est le vecteur unitaire dans $\Re^n$.
On pose $X_{ik}^{jl} = x_{ij}x_{kl}$, $x = Vect(Z)$ et la matrice X de taille $n^2 \times n^2$ définie par $X = x^T x$ s'écrit :

$$X = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1n} \\ X_{21} & Y_{22} & \cdots & X_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{nn} \end{bmatrix}$$

où

$$X_{ij} = \begin{bmatrix} X_{ij}^{11} & X_{ij}^{12} & \cdots & X_{ij}^{1n} \\ X_{ij}^{21} & X_{ij}^{22} & \cdots & X_{ij}^{2n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{ij}^{n1} & X_{ij}^{n2} & \cdots & X_{ij}^{nn} \end{bmatrix}$$

Pour construire une première relaxation semidéfinie du programme (QAP), nous réecrivons les contraintes (1) et (2) en utilisant la matrice Y définie par :

$$Y = \begin{bmatrix} X & x \\ x^T & 1 \end{bmatrix}$$

Comme nous avons vu dans le chapitre 1 :

$$Y \succeq 0 \text{ est équivalent à } X - diag(X)diag(X)^T \succeq 0$$

avec $diag(X) = x$, on note ces contraintes (d).
Afin de simplifier l'écriture du programme mathématique, nous allons avoir besoin des définitions suivantes.

Définitions 1 *Soit A une matrice $(m \times p)$ d'élément a_{ij} et B une matrice $(s \times t)$.*
Le produit de Kronecker : $A \otimes B$ est une matrice $(ms \times pt)$ telle que :

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1p}B \\ a_{21}B & a_B & \cdots & a_{2p}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{n2}B & \cdots & a_{mp}B \end{bmatrix}$$

En utilisant cette définition et la matrice X, la fonction objectif du modèle (QAP) s'écrit :

$$\min \quad (D \otimes F) \bullet X$$

Définitions 2 *Soit X la matrice par bloc définit ci-dessus. On définit les deux opérateurs linéaires $bdiag(X)$ et $odiag(X)$ de $\Re^{n^2 \times n^2}$ à $\Re^{n \times n}$ comme suit :*

$$bdiag(X) = \sum_{i=1}^{n} X_{ii}$$

$$(odiag(X))_{ij} = Tr(X_{ij}), \quad i, j = 1, ..., n.$$

En utilisant ces deux opérateurs, les contraintes (1) et (2) s'écrivent :

$$\begin{aligned}
Z^T e &= \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1n} & x_{2n} & \cdots & x_{nn} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} \sum_{l=1}^{n} x_{l1} \\ \sum_{l=1}^{n} x_{l2} \\ \vdots \\ \sum_{l=1}^{n} x_{ln} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{l=1}^{n} X_{ll}^{11} \\ \sum_{l=1}^{n} X_{ll}^{22} \\ \vdots \\ \sum_{l=1}^{n} X_{ll}^{nn} \end{bmatrix} \quad \text{puisque } X_{ii}^{jj} = x_{ij} x_{ij} = x_{ij} \\
&= diag(\sum_{l=1}^{n} X_{ll}) \\
&= diag(bdiag(X)) \\
&= e
\end{aligned}$$

De la même façon, on réécrit les contraintes (2) sous la forme :

$$diag(odiag(X)) = e$$

Une première relaxation semidéfinie du programme (QAP) peut donc s'écrire :

$$(SDP_0) \begin{cases} \min & (D \otimes F) \bullet X \\ \text{s.c.} & \\ & (1) \quad diag(bdiag(X)) = e \\ & (2) \quad diag(odiag(X)) = e \\ & (d) \quad X - diag(X) diag(X)^T \succeq 0 \end{cases}$$

Nous avons remarqué lors de premiers tests numériques que la relaxation (SDP_0) donne des bornes inférieures très éloignées de l'optimum, voire des bornes inférieures négatives pour plusieurs jeux de données. Afin d'améliorer la qualité de cette borne, nous proposons d'autres relaxations semidéfinies du programme (QAP) en introduisant des contraintes d'orthogonalité sur la matrice variable.

3.3.2 Relaxation semidéfinie SDP_1

A partir des contraintes d'affectation suivantes $Ze = e$ et $Z^T e = e$, en multipliant la première égalité par la matrice Z^T et la deuxième égalité par Z, on obtient :

$$Z^T Ze = Z^T e = e \quad \text{et} \quad ZZ^T e = Ze = e$$

ce qui implique $ZZ^T = I_n \quad (1')$ et $Z^T Z = I_n \quad (2')$. En d'autres termes la matrice Z est une matrice orthogonale.
On réécrit les contraintes $(1')$ et $(2')$ en utilisant les opérateurs *bdiag* et *odiag* définis précédemment :

$$\begin{aligned}
Z^T Z &= \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1n} & x_{2n} & \cdots & x_{nn} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{l=1}^n x_{l1}x_{l1} & \sum_{l=1}^n x_{l1}x_{l2} & \cdots & \sum_{l=1}^n x_{l1}x_{ln} \\ \sum_{l=1}^n x_{l2}x_{l1} & \sum_{l=1}^n x_{l2}x_{l2} & \cdots & \sum_{l=1}^n x_{l2}x_{ln} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{l=1}^n x_{ln}x_{l1} & \sum_{l=1}^n x_{ln}x_{l2} & \cdots & \sum_{l=1}^n x_{ln}x_{ln} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{l=1}^n X_{ll}^{11} & \sum_{l=1}^n X_{ll}^{12} & \cdots & \sum_{l=1}^n X_{ll}^{1n} \\ \sum_{l=1}^n X_{ll}^{21} & \sum_{l=1}^n X_{ll}^{22} & \cdots & \sum_{l=1}^n X_{ll}^{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{l=1}^n X_{ll}^{n1} & \sum_{l=1}^n X_{ll}^{n2} & \cdots & \sum_{l=1}^n X_{ll}^{nn} \end{bmatrix} \\
&= \sum_{l=1}^n \begin{bmatrix} X_{ll}^{11} & X_{ll}^{12} & \cdots & X_{ll}^{1n} \\ X_{ll}^{21} & X_{ll}^{22} & \cdots & X_{ll}^{2n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{ll}^{n1} & X_{ll}^{n2} & \cdots & X_{ll}^{nn} \end{bmatrix} \\
&= \sum_{l=1}^n X_{ll} \\
&= bdiag(X) = I_n
\end{aligned}$$

De la même façon, on démontre que les contraintes $(2')$ s'écrivent :

$$odiag(X) = I_n.$$

En ajoutant donc ces contraintes dans le programme (SDP_0), on obtient le programme (SDP_1) suivant :

$$(SDP_1)\begin{cases} \min & (D \otimes F) \bullet X \\ \text{s.c.} & \\ & (1') \quad bdiag(X) = I_n \\ & (2') \quad odiag(X) = I_n \\ & (d) \quad X - diag(X)diag(X)^T \succeq 0 \end{cases}$$

Nous avons remarqué lors des tests numériques que la relaxation (SDP_1) sont meilleurs que ceux de (SDP_0). Cependant, ils restent assez éloignés de l'optimum. Ceci s'explique par la présence d'éléments négatifs dans la matrice Y. En effet, la valeur de la fonction objectif $((D \otimes F) \bullet X)$ se voit abaissée par la somme de ces termes. Ce phénomène est le même que celui évoqué pour le problème du FAP. Pour y palier, il convient d'ajouter des contraintes de positivité. On obtient alors le modèle (SDP_{Re}) suivant (qui est présenté dans [34]) :

$$(SDP_{Re})\begin{cases} \min & (D \otimes F) \bullet X \\ \text{s.c} & \\ & (1') \quad bdiag(X) = I \\ & (2') \quad odiag(X) = I \\ & \\ & (d) \quad X - diag(X)diag(X)^T \succeq 0 \\ & (P) \quad X_{ikjl} \geq 0 \quad \forall i,j,k,l \in \{1,...,n\} \end{cases}$$

Les résultats numériques du modèle (SDP_{Re}) sont présentés dans les tableaux (2) et (5). Nous allons revenir sur l'analyse des résultats de ce modèle ainsi que sa comparaison avec notre propre méthodes dans le paragraphe 4.

3.3.3 Relaxation semidéfinie SDP_2

Motivés par les résultats de Drezner dans [12], nous nous sommes également penchés sur la formulation (QAP_L) du paragraphe 3.2 afin d'écrire une relaxation semidéfinie.
Nous considérons les contraintes (3) et (4) du programme(QAP_L).
Si la contrainte $\sum_{i=1}^{n} X_{ikjl} = x_{jl}$ porte l'indice s, elle s'écrit dans le programme (SDP_2) sous la forme $B_s \bullet Y = 0$, où B_s est la matrice telle que $B_s \bullet Y = \sum_{i=1}^{n} X_{ikjl} - x_{jl}$.

De même, si la contrainte $\sum_{k=1}^{n} X_{ikjl} = x_{jl}$ porte l'indice r, elle s'écrit dans le programme $(SDP2)$ sous la forme $B_r \bullet Y = 0$, où B_r est la matrice telle que $B_r \bullet Y = \sum_{k=1}^{n} X_{ikjl} - x_{jl}$.
Toutes ces contraintes sont regroupées dans le programme $(SDP2)$ sous la contrainte $B(Y) = 0$. On a $2nn^2$ matrices B pour représenter l'ensemble des contraintes (3) et (4).
Nous pouvons finalement écrire la relaxation semidéfinie du (QAP_L), notée (SDP_2) :

$$(SDP_2)\begin{cases} \min & (D \otimes F) \bullet Y \\ \text{s.c} & \\ & (1) \quad diag(bdiag(Y)) = e \\ & (2) \quad diag(odiag(Y)) = e \\ & (5) \quad B(Y) = 0 \\ & (d) \quad Y \succeq 0 \end{cases}$$

Dans le programme (SDP_2), on a $2n$ contraintes de type (1) et (2), $2nn^2$ contraintes de type (5) et n^2+1 contraintes pour exprimer que $diag(X) = x$ et que le dernier élément dans la matrice Y vaut 1 (les contraintes (d)). Ce qui donne un nombre de contraintes de $2n^3 + n^2 + 2n + 1$.
Afin d'éviter le phénomène mis en évidence dans la section précédente, nous renforçons cette relaxation par l'ajout de coupes de non négativité pour les éléments de la matrice Y :

$$(P) \qquad Y_{ikjl} \geq 0 \qquad \forall i, j, k, l \in \{1, ..., n\}.$$

Une deuxième façon d'écrire les contraintes linéaires d'affectation dans un programme semidéfini est d'élever au carré les deux membres de la contrainte. On obtient donc pour les contraintes (1), les contraintes (1") suivantes :

$$(\sum_{i=1}^{n} x_{ik})^2 = 1 \quad \forall k \in \{1, ..., n\}$$

Et pour les contraintes (2), les contraintes (2") suivantes :

$$(\sum_{k=1}^{n} x_{ik})^2 = 1 \quad \forall i \in \{1, ..., n\}$$

En ajoutant ces contraintes au programme semidéfini, on obtient la formu-

lation suivante qui est celle présentée dans [36] :

$$(SDP_{Ro}) \begin{cases} \min & (D \otimes F) \bullet Y \\ \text{s.c} & \\ & (1) \quad diag(bdiag(Y)) = e \\ & (1'') \quad (\sum_{i=1}^{n} x_{ik})^2 = 1 \quad \forall k \in \{1, ..., n\} \\ & (2) \quad diag(bdiag(Y)) = e \\ & (2'') \quad (\sum_{k=1}^{n} x_{ik})^2 = 1 \quad \forall i \in \{1, ..., n\} \\ & (5) \quad B(Y) = 0 \\ & (d) \quad Y \succeq 0 \\ & (P) \quad X_{ikjl} \geq 0 \quad \forall i, j, k, l \in \{1, ..., n\} \end{cases}$$

Les résultats numériques de la relaxation (SDP_{Ro}) sont de bonne qualité mais les instances testées ne dépassent pas $n = 30$. Ceci est dû au nombre très important de contraintes . Celui-ci est de l'ordre de n^4 pour les contraintes de type (P) et de l'ordre de n^3 pour les contraintes de type (5). Dans le paragraphe suivant, nous avons adapté développé l'algorithme de plans coupants developpé précedement pour la résolution du FAP. Cet algorithme ne génère que les contraintes violées pour alléger le programme semidéfini.

3.4 Plans Coupants pour le QAP

Soit (SDP_{QAP}) la formulation semidéfinie renforcée du problème d'affectation quadratique :

$$(SDP_{QAP}) \begin{cases} \min & (D \otimes F) \bullet Y \\ \text{sous les contraintes} & \\ & (1') \quad bdiag(Y) = I_n \\ & (2') \quad odiag(Y) = I_n \\ & (5) \quad B(Y) = 0 \\ & \\ & (d) \quad Y \succeq 0 \\ & (P) \quad Y_{ikjl} \geq 0 \forall i, j, k, l \in \{1, ..., n \end{cases}$$

Pour déterminer la borne, la formulation (SDP_{QAP}) ne comporte au départ que les contraintes (1'),(2') et (d). Du fait de leur nombre important, les contraintes (5) et (P) sont ensuite générées par l'algorithme de plans coupants présenté ci-dessous.
Le principe général des algorithmes de génération de coupes est de résoudre progressivement un problème en ajoutant un certain nombre de coupes (voir

[19], [8]).

Afin de déterminer dans quel ordre ces contraintes seront ajoutées, nous allons réutiliser la notion de **degré de violation** :

- le degré de violation d'une coupe de positivité est donnée par $d_i(c) = Y_{sr}$ pour chaque variable Y_{sr}. Notre algorithme génère une coupe définie par la contrainte $Y_{sr} \geq 0$ si $d(c_{sr}) \leq \epsilon$, où ϵ est un seuil donné.

- les degrés de violation des coupes de type (5) sont données par

$$d_{e1}(c) = |\sum_{i=1}^{n} y_{ikjl} - x_{jl}|.$$

  et

$$d_{e2}(c) = |\sum_{k=1}^{n} y_{ikjl} - x_{jl}|.$$

  Notre algorithme génère une coupe $\sum_{i=1}^{n} y_{ikjl} - x_{jl} = 0$ (respectivement la coupe $\sum_{k=1}^{n} y_{ikjl} - x_{jl}$) si $d_{e1}(c) \geq \epsilon_i$ (respectivement $d_{e2}(c) \geq \epsilon_i$), où ϵ_i est un seuil donné.

Précisons que dans l'implémentation de notre algorithme, nous stockons toutes les contraintes violées de type (P) dans un ensemble V_P et les contraintes violées de type 5 dans un ensemble V_S . Nous trions ces ensembles et nous considérons uniquement un pourcentage des contraintes les plus violées. Ce sont donc ces contraintes qui détermineront les coupes à considérer à chaque itération. Le processus se termine lorsqu'il n'y a plus de contraintes violées. Notons que l'ordre dans lequel on génère les contraintes violées est très important.En effet, en pratique, nous constatons que, si l'on génère toutes les familles de contraintes en même temps, alors la qualité de la borne inférieure est dégradé par rapport au résultats obtenus avec les contraintes de type P uniquement. Ainsi, nous générons d'abord les contraintes de type (P). Quand il ne reste plus de contrainte de positivité violée, nous générons alors les autres contraintes de type (5). Nous détaillons ce procédé par l'algorithme de plans coupants ci dessous.

Algorithme 1 Plans Coupants pour le QAP

#nbrIter est le nombre d'itérations
#nbrIterMax est le nombre d'itérations maximum
violee est un variable booléenne qui indique l'existence ou non de contraintes violées

Résoudre $SDP2$
$violee = VRAI$
Tant que $(\#nbrIter < \#nbrIterMax)$ et $(violee = VRAI)$
répéter
  si $d_i(c) \leq \epsilon_i$
    alors stocker la contrainte c dans un ensemble V_P
  fin si
  Ajouter les p contraintes de V_P les plus violées au SDP_{k-1}
  Résoudre SDP_k
  $k \leftarrow k + 1$
jusqu'à ce que $d_i(c) \geq \epsilon_i$
A l'étape l
répéter
  si $d(c) \leq \epsilon_i$
    alors stocker la contrainte c dans un ensemble V_S
  fin si
  Ajouter les q contraintes de V_S les plus violées au SDP_{l-1}
  Résoudre SDP_l
  $l \leftarrow l + 1$
jusqu'à ce que $d_e(c) \geq \epsilon_e$
$violee = FAUX$

A chaque itération, on utilise le logiciel *SB* de Helmberg [21] pour résoudre les programmes semidéfinis obtenus par l'ajout des coupes. Ce solveur est basé sur la méthode des faisceaux.
Nous avons pensé à rajouter des contraintes de linéarisation et de triangle qui peuvent généralement améliorer la qualité des bornes inférieures. Toutefois, nous avons testé notre méthode avec et sans ces contraintes et nous n'avons pas constaté d'amélioration. Nous allons maintenant présenter les résultats numériques obtenus avec notre méthode et les comparer aux autres méthodes existantes.

4 Résultats numériques

Commençons tout d'abord par la description des instances sur lesquelles nous avons effectué les tests.

4.1 Instances du QAP

Burkard, Karisch et Rendl [33] ont regroupé plusieurs instances dans QAPLIB[2]. Cette librairie permet d'avoir les informations suivantes sur chacune de ces instances :

- Nom de l'instance : ce nom se décompose sous la forme d'un radical de trois lettres mis pour les trois premières lettres du nom du premier

[2] http ://www.opt.math.tu-graz.ac.at/qaplib/

auteur. Le suffixe est constitué d'une valeur qui représente la taille du problème. Par exemple, l'instance *had*20 a été crée par S.W. Hadley, F. Rendl et H. Wolkowicz, et correspond à un problème contenant 20 usines et 20 emplacements.

- La solution optimale quand elle est connue.

- Pour les instances dont on ne connaît pas la solution optimale, on donne la meilleure borne inférieure et la meilleure solution approchée.

Afin d'évaluer les performances de notre approche, nous avons effectué des tests numériques sur différentes instances du QAPLIB 4.1. Nous avons implémenté notre algorithme de plans coupants en langage C++ sur un Athlon 2800+ avec 512Mo du Ram.

4.2 Comparaison du nombre de contraintes des différentes relaxations SDP

Nous comparons ici le nombre de contraintes générées par notre algorithme avec ceux des formulations SDP_{Ro} [36] et SDP_{Re} [34]. L'importance de cette analyse réside dans le fait qu'un nombre de contraintes réduit permet de résoudre des instances de grande taille. Nous désignons le nombre de contraintes de positivité par ($\#cont_P$), le nombre de contraintes de Sherali-Adams par ($\#cont_{Sh}$) et plus généralement le nombre de contraintes globales par ($\#cont_G$).

On constate sur le tableau 1 que l'on réduit visiblement le nombre de contraintes globales par rapport à celle de la formulation SDP_{Ro}. Sur les instances Esc16, par exemple, on peut réduire de moitié le nombre de ces contraintes. Cette réduction est surtout dûe à celle du nombre de contraintes de positivité qui est du même ordre de grandeur.
Le nombre de contraintes de Sherali-Adams de notre formulation, bien que plus petit, reste assez proche de celui de celle de SDP_{Ro}. Ceci s'explique par le fait que le seuil d'acceptation de ces contraintes dans notre formulation est élevé ($\epsilon_e = 10^{-5}$). Ainsi, en augmentant ce seuil, on peut réduire considérablement le nombre de ces contraintes. C'est d'ailleurs ce que l'on propose pour les instances de grande taille ($n > 30$).

De même, on peut voir sur le tableau (1) que notre formulation réduit le nombre de contraintes globales par rapport à celui de la formulation SDP_{Re} sauf pour les instances Nug12 et Tai12.
Ce résultat est d'autant plus intéressant que notre formulation génère les contraintes de Sherali-Adams qui ne sont pas générées dans la formulation SDP_{Re}. En effet, la réduction importante du nombre de contraintes de po-

TAB. 1 – Comparaison entre le nombre de contraintes de la formulation (SDP_{Ro}) et SDP_{PC}

Data	SDP_{Ro}			SDP_{PC}		
	$\#cont_P$	$\#cont_{Sh}$	$\#cont_G$	$\#cont_P$	$\#cont_{Sh}$	$\#cont_G$
Esc16a	32640	8192	41137	13441	8192	22162
Esc16b	32640	8192	41137	9968	8192	18689
Esc16c	32640	8192	41137	14875	8192	23596
Esc16d	32640	8192	41137	15574	8177	24280
Esc16e	32640	8192	41137	11958	8190	20677
Esc16g	32640	8192	41137	10561	8186	19276
Esc16h	32640	8192	41137	11824	8192	20545
Esc16i	32640	8192	41137	14182	8177	22888
Esc16j	32640	8192	41137	8868	8192	17589
Had12	10296	3456	13933	7973	3206	11480
Had14	19110	5488	24837	13796	5337	19540
Had16	32640	8192	41137	19521	8179	28229
Had18	52326	11664	64369	33754	11600	46021
Had20	79800	16000	96261	47364	16000	64185
Nug12	10296	3456	13933	7644	3419	11364
Nug14	19110	5488	24837	14192	5464	20063
Nug15	25200	6750	32221	18187	6739	25392
Nug20	79800	16000	96261	52112	15965	68898
Rou12	10296	3456	13933	7709	3414	11424
Rou15	25200	6750	32221	16584	6742	23792
Rou20	79800	16000	96261	50421	15980	67222
Scr12	10296	3456	13933	8142	3452	11895
Scr15	25200	6750	32221	17508	6747	24721
Scr20	79800	16000	96261	57864	15992	74677

TAB. 2 – Comparaison entre le nombre de contraintes de la formulation (SDP_{Re}) et SDP_{PC}

Data	SDP_{Re}		SDP_{PC}		
	$\#cont_P$	$\#cont_G$	$\#cont_P$	$\#cont_{Sh}$	$\#cont_G$
Esc16a	32640	33169	13441	8192	22162
Esc16b	32640	33169	9968	8192	18689
Esc16c	32640	33169	14875	8192	23596
Esc16d	32640	33169	15574	8177	24280
Esc16e	32640	33169	11958	8190	20677
Esc16g	32640	33169	10561	8186	19276
Esc16h	32640	33169	11824	8192	20545
Esc16i	32640	33169	14182	8177	22888
Esc16j	32640	33169	8868	8192	17589
Scr20	79800	80621	57864	15992	74677
Rou20	79800	80621	50421	15980	67222
Nug12	10296	10597	7456	3419	11176
Nug14	19110	19517	13597	5440	19444
Nug15	25200	25666	18187	6739	25392
Nug16a	32640	33169	22611	8180	31320
Nug16b	32640	33169	22800	8178	31507
Nug17	41616	42212	28974	9790	39360
Nug18	52326	52993	38604	11566	50837
Nug20	79800	80621	52112	15965	68898
Nug21	97020	97924	63219	19502	83625
Nug22	116886	117877	77388	21089	99468
Nug24	165600	166777	111607	27513	140297
Nug25	195000	196276	135361	31248	167885
Nug27	265356	266842	184688	39366	225540
Nug30	404550	406381	192060	54000	247891
Kra30a	404550	406381	174993	53967	230791
Kra30b	404550	406381	138648	54000	194479
Tai12a	10296	10597	8006	2710	11017
Tai15a	25200	25666	17203	6747	24416
Tai17a	41616	42212	25187	9824	35607
Tai20a	79800	80621	50170	15989	66980
Tai25a	195000	196276	114388	31243	146907
Tai30a	404550	406381	191847	54000	247678
Tho30	404550	406381	171467	54000	227298

TAB. 3 – Comparaison entre le nombre de contraintes de la version Explicite et de notre algorithme de plans coupants SDP_{PC}

Data	$SDP_{Explicite}$			SDP_{PC}		
	$\#cont_P$	$\#cont_{Sh}$	$\#cont_G$	$\#cont_P$	$\#cont_{Sh}$	$\#cont_G$
Esc32a	523776	65536	525857	190884	31298	224263
Esc32b	523776	65536	525857	195751	65535	263367
Esc32c	523776	65536	525857	151959	65536	219576
Esc32d	523776	65536	525857	173994	65536	241611
Esc32e	523776	65536	525857	136668	65536	204285
Esc32f	523776	65536	525857	201655	56444	260180
Esc32g	523776	65536	525857	133413	65536	201030
Esc32h	523776	65536	525857	140472	65536	208089
Tai35a	749700	85750	752186	162929	85750	251165
Tai40a	1279200	128000	1282441	200049	127993	331283
Tai50a	3123750	250000	3128801	260969	88214	354234
Tai60a	6478200	432000	6485461	272155	108238	387654
Sko42	1554966	148176	1558537	232006	20296	255873
Sko49	2881200	235298	2886052	294136	46243	345231
Sko56	4915680	351232	4922009	306627	74698	387654
Sko64	8386560	524288	8394817	338528	53215	400000
Sko72	13434336	746496	13444777	346337	43222	400000
Tho40	1279200	128000	1282441	178885	128000	310126

sitivité compense largement la prise en compte de ce type de contraintes.
Pour les instances de plus grande taille (tableau 3), on peut voir que le nombre de contraintes dans une formulation explicite est très important et rend donc une telle résolution impossible. Nous montrons grâce à notre formulation, que ce nombre peut être sensiblement réduit (sauf pour Sko64 et Sko72 où nous limitons le nombre de contraintes gérées) et permet donc une résolution de ces instances. Notre méthode est d'ailleurs, à notre connaissance, la première méthode basée sur la relaxation semidéfinie à résoudre des instances de cette taille.

Ainsi grâce à ce gain, on a réussi à calculer des bornes inférieures pour des instances de taille allant jusqu'à ($n = 72$). Nous résumont les résultats dans les tableaux 4, 5 et 6.

4.3 Comparaison avec les résultats de Roupin [36]

Le tableau 4 expose les résultats des calculs de la borne inférieure de la relaxation semidéfinie SDP_{Ro} et notre relaxation SDP_{PC}.
Dans ce tableau : *Data* est le nom du jeu de données résolu. *Opt* est la solution optimale lorsqu'elle est connue. Si celle-ci n'est pas connue, alors on reporte la meilleure solution entière approchée.
Pour chaque méthode, $B.inf$ représente la borne inférieure obtenue par le programme correspondant. CPU représente le temps de calcul. Il est exprimé par le format *heures* : *minutes* : *secondes*. *Gap* en % représente l'écart relatif de la borne inférieure avec l'optimum. Cet écart est calculé à l'aide de la formule :

$$Gap = \frac{Opt - B.inf}{Opt} \times 100$$

Pour notre méthode SDP_{PC}, $\#iter_P$ (respectivement $\#iter_{Sh}$) représente le nombre d'itérations nécessaires pour générer les contraintes de positivités (respectivement les contraintes de Sherali-Adams). Le nombre total d'itérations de l'algorithme de plans coupants se déduit par la somme de $\#iter_P$ et $\#iter_{Sh}$.
Gain en % représente l'écart entre la borne inférieure obtenue par SDP_{Ro} et celle obtenue par SDP_{PC}. Cet écart est calculé comme suit :

$$Gain = \frac{(B.inf)_{PC} - (B.inf)_{Ro}}{(B.inf)_{PC}} \times 100.$$

D'après cette formule, on peut dire que quand *Gain* est positif, alors cela signifie que notre borne inférieure est de meilleure qualité que celle obtenue par SDP_{Ro}.

D'après le tableau 4, l'écart relatif (*Gap*) obtenu avec notre méthode est toujours inférieur à 5% sauf pour les instances Esc16a, Esc16d et Esc16i.

TAB. 4 – Comparaison entre les résultats de la formulation SDP_{Ro} et notre algorithme de plans coupants

Data	*Opt*	SDP_{Ro}			SDP_{PC}					*Gain*
		B.inf	*CPU*	*Gap*	*B.inf*	*CPU*	*itP*	*itSh*	*Gap*	
Esc16a	68	63	02 :41 :00	7,35	63	02 :51 :34	5	2	7,35	0,00
Esc16b	292	290	04 :16 :00	0,68	290	01 :53 :49	4	2	0,68	0,00
Esc16c	160	154	13 :36 :00	3,75	154	09 :26 :46	5	2	3,75	0,00
Esc16d	16	13	05 :25 :00	18,75	13	01 :33 :00	5	2	18,75	0,00
Esc16e	28	27	14 :59 :00	3,57	27	10 :50 :06	5	2	3,57	0,00
Esc16g	26	25	09 :01 :00	3,85	25	10 :38 :14	5	2	3,85	0,00
Esc16h	996	972	08 :23 :00	2,41	976	08 :45 :21	5	2	2,01	0,41
Esc16i	14	12	16 :05 :00	14,29	12	11 :29 :19	5	2	14,29	0,00
Esc16j	8	8	02 :03 :00	0,00	8	01 :44 :28	5	3	0,00	0,00
Had12	1652	1652	00 :10 :09	0,00	1652	00 :03 :52	3	2	0,00	0,00
Had14	2724	2724	00 :56 :52	0,00	2724	04 :22 :51	4	1	0,00	0,00
Had16	3720	3720	02 :37 :00	0,00	3720	00 :59 :05	5	2	0,00	0,00
Had18	5358	5356	08 :42 :00	0,04	5358	03 :18 :08	8	2	0,00	0,04
Had20	6922	6920	13 :05 :00	0,03	6920	09 :16 :35	11	3	0,03	0,00
Nug12	578	567	02 :01 :00	1,90	568	01 :49 :32	4	2	1,73	0,18
Nug14	1014	1009	05 :55 :00	0,49	1010	03 :19 :43	5	2	0,39	0,10
Nug15	1150	1138	05 :49 :00	1,04	1140	03 :24 :49	6	3	0,87	0,18
Nug20	2570	2494	12 :08 :00	2,96	2503	11 :44 :22	8	4	2,61	0,36
Rou12	235528	235402	08 :32 :00	0,05	235496	07 :21 :55	3	2	0,01	0,04
Rou15	354210	348838	08 :44 :00	1,52	349780	06 :14 :27	5	2	1,25	0,27
Rou20	725522	691124	17 :10 :00	4,74	693791	15 :51 :18	8	3	4,37	0,38
Scr12	31410	31391	04 :24 :00	0,06	31410	03 :20 :26	4	3	0,00	0,06
Scr15	51140	51136	12 :39 :00	0,01	51140	08 :19 :34	6	3	0,00	0,01
Scr20	110030	105054	13 :13 :00	4,52	105112	12 :25 :13	9	4	4,47	0,06

Sur la moité des instances testées, l'écart est inférieur à 1%. En comparaison avec les bornes inférieures obtenues avec (SDP_{Ro}), nos bornes inférieures sont toujours de qualité égale voire meilleures. Le temps de calcul avec notre formulation est toujours inférieur à celui nécessaire pour (SDP_{Ro}), sauf pour les instances Esc16a, Esc16g, Esc16h et Had14.

4.4 Comparaison avec les résultats de Rendl [34]

Dans ce paragraphe, nous comparons les résultats de notre méthode (SDP_{PC}) avec (SDP_{Re}). Les résultats sont reportés dans le tableau (5).

Nous constatons un gain important en terme de qualité des bornes inférieures par rapport à la formulation SDP_{Re} pour tous les jeux de données testés. En effet, le gain est supérieur à 4% pour la moité des instances, voire supérieur à 12% pour les instances Esc16d, Esc16e, Esc16g, Esc16i et Esc16j. L'écart relatif (*Gap*) obtenu avec notre méthode est toujours inférieur à 7% sauf pour les instances Esc16a, Esc16d, Esc16i et Tai30a.
Le temps de calcul nécessaire pour résoudre les instances avec SDP_{PC} reste raisonnable mais nous ne pouvons pas faire de comparaison avec les temps de calcul de SDP_{Re} car ils ne sont pas fournis.

4.5 Instances de grande taille

Le fait d'utiliser les plans coupants au lieu d'expliciter toutes les contraintes nous permet de résoudre des instances de grande taille qui ne sont traitées ni dans [34] ni dans [36].
Dans le tableau (6), on représente les résultats de SDP_{PC} appliqués à des jeux de données de taille supérieure à 30. Notons que la borne inférieure pour ces instances est calculée par des méthodes différentes. Pour les connaître, le lecteur peut se référer à QAPLIB (paragraphe 4.1).

Nous constatons un gain en qualité de la borne inférieure par rapport aux meilleures solutions connues dans littérature sauf pour les instances Sko [38]. Ces instances ont comme particularité que la matrice de distances entre les sites vérifie l'inégalité triangulaire et que les valeurs de la matrice des flux entre les unités sont pseudo-aléatoires.
Pour le reste des instances, le gain est supérieur à $9,5\%$ pour Esc32a, Esc32b, Esc32c, Esc32d, Esc32h et Tai35a. Il est compris entre $0,54\%$ et 5% pour Tai40a, Tai40a, Tai60a, Sko42 et Tho40.

5 Conclusion

Nous avons proposé une nouvelle méthode SDP_{PC} basée sur un algorithme de génération de coupes. Cette méthode fournit globalement de meilleurs résultats que ceux obtenus par d'autres méthodes SDP, à savoir

TAB. 5 – Comparaison entre les résultats de la formulation SDP_{Re} et notre algorithme de plans coupants

Data	*Opt*	SDP_{Re}		SDP_{PC}				*Gain*
		B.inf	*gap*	*B.inf*	*#itP*	*#itSh*	*gap*	
Esc16a	68	59	13,24	63	15	2	7,35	6,35
Esc16b	292	288	1,37	290	4	2	0,68	0,69
Esc16c	160	142	11,25	154	5	2	3,75	7,79
Esc16d	16	8	50,00	13	5	2	18,75	38,46
Esc16e	28	23	17,86	27	5	2	3,57	14,81
Esc16g	26	20	23,08	25	5	2	3,85	20,00
Esc16h	996	970	2,61	976	5	2	2,01	0,61
Esc16i	14	9	35,71	12	5	2	14,29	25,00
Esc16j	8	7	12,50	8	5	3	0,00	12,50
Scr20	110030	94998	13,66	105112	9	4	4,47	9,62
Rou20	725522	663833	8,50	693791	8	3	4,37	4,32
Nug12	578	557	3,63	568	4	2	1,73	1,94
Nug14	1014	992	2,17	1010	5	2	0,39	1,78
Nug15	1150	1122	2,43	1140	6	3	0,87	1,58
Nug16a	1610	1570	2,48	1596	6	3	0,87	1,63
Nug16b	1240	1188	4,19	1231	6	3	0,73	3,49
Nug17	1732	1669	3,64	1703	7	2	1,67	2,00
Nug18	1930	1852	4,04	1892	8	4	1,97	2,11
Nug20	2570	2451	4,63	2503	8	4	2,61	2,08
Nug21	2438	2323	4,72	2371	9	4	2,75	2,02
Nug22	3596	3440	4,34	3519	11	5	2,14	2,24
Nug24	3488	3310	5,10	3386	12	5	2,92	2,24
Nug25	3744	3535	5,58	3604	12	5	3,74	1,91
Nug27	5234	4965	5,14	5107	14	5	2,43	2,78
Nug30	6124	5803	5,24	5922	20	7	3,30	2,01
Kra30a	88900	77647	12,66	86052	26	5	3,20	9,77
Kra30b	91420	81156	11,23	86294	25	5	5,61	5,95
Tai12a	224416	222784	0,73	224416	3	2	0,00	0,73
Tai15a	388214	364761	6,04	376800	5	3	2,94	3,20
Tai17a	491812	451317	8,23	475832	6	2	3,25	5,15
Tai20a	703482	637300	9,41	669667	7	3	4,81	4,83
Tai25a	1167256	1041337	10,79	1092433	11	5	6,41	4,68
Tai30a	1818146∗	1652186	9,13	1686466	26	4	7,24	2,03
Tho30	149936	136059	9,26	142005	29	5	5,29	4,19

∗ meilleure solution approchée

TAB. 6 – Résultats de l'algorithme de plans coupants pour des instances de grande taille.

Data	*MB.Sup*	Littérature		SDP_{PC}				*Gain*
		B.inf	*gap*	*B.inf*	*itP*	*itSh*	*gap*	
Esc32a	130	88	32,31	100	9	4	23,08	12,00
Esc32b	168	100	40,48	127	13	5	24,40	21,26
Esc32c	642	506	21,18	612	16	6	4,67	17,32
Esc32d	200	152	24,00	180	12	5	10,00	15,56
Esc32e	2⋆	±	±	2	8	4	0,00	±
Esc32f	2⋆	±	±	1	9	5	50,00	±
Esc32g	6⋆	±	±	6	8	5	0,00	±
Esc32h	438	352	19,63	417	7	5	4,79	15,59
Tai35a	2422002	1951207	19,44	2162738	8	7	10,70	9,78
Tai40a	3139370	2492850	20,59	2619146	4	9	16,57	4,82
Tai50a	4941410	3854359	22,00	3900924	12	3	21,06	1,19
Tai60a	7208572	5555095	22,94	5675384	4	3	21,27	2,12
Sko42	15812	14934	5,55	15046	9	4	4,84	0,74
Sko49	23386	22004	5,91	21198	12	6	9,36	-3,80
Sko56	34458	32610	5,36	31074	13	6	9,82	-4,94
Sko64	48498	45736	5,70	43644	8	4	10,01	-4,79
Sko72	66256	62691	5,38	62522	8	4	5,64	-0,27
Tho40	240516	214218	10,93	215387	4	10	10,45	0,54

$*$ solution optimale
$\pm$ valeur non fourni

SDP_{Re} et SDP_{Ro}. De plus, le fait que le nombre de contraintes à gérer soit relativement petit par rapport à celui de ces méthodes permet de résoudre des instances de grande taille. Ceci n'était pas possible avec ces autres méthodes de SDP. Pour certaines de ces instances, nous avons montré que l'on peut améliorer les bornes inférieures déjà connues par d'autres méthodes de résolution.

Références

[1] W.P. Adams and T.A. Johnson. Improved linear programming-based lower bounds for the quadratic assignment problems. *DIMACS Series in Discrete Mathematics and Theorical Computer Science*, 16 :43–75, 1994.

[2] E. Angel. Le problème d'affectation quadratique et la recherche locale. Technical report, Laboratoire de Recherche en Informatique, Université de Paris-Sud, 1995.

[3] E. Angel. *La rugosité des paysages : une théorie pour la difficulté des problèmes d'optimisation combinatoire relativement aux méta-heuristiques.* PhD thesis, Université de Paris-Sud, 1998.

[4] K.M. Anstreicher and N.W. Brixius. A new bound for the quadratic assignment problems based on convex quadratic programming. *Math programs. Ser A*, 89 :341–357, 2001.

[5] R. Battiti and G. Tecchiolli. The reative tabu search. *ORSA Journal on Computing*, 6(2) :126–140, 1994.

[6] M.S. Bazaraa and H.D.Sherali. Benders' partitioning scheme applied to a new formulation of the quadratic assignment problem. *Naval Research Logistics Quaterly*, 27 :29–41, 1980.

[7] J.A. Bland and G.P. Dawson. Large-scale layout of facilities using heuristic hybrid algorithm. *Applied Mathematical Modelling*, 18 :500–503, 1994.

[8] N. Boissin. *Optimisation des fonctions quadratiques en variables bivalentes.* PhD thesis, CNAM, 1994.

[9] G.C. Armour E.S. Buffa and T.E. Vollmann. Allocating facilities with craft. *Harvard Business Review*, 42 :136–158, 1964.

[10] R.E. Burkard and F. Rendl. A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operations Research*, 17 :169–174, 1984.

[11] A. Colorni and V. Maniezzo. The ant system applied to the quadratic assignment problem. *nowledge and Data Engineering*, 11(5) :769–778, 1999.

[12] Z. Drezner. Lower bounds based on linear programming for the quadratic assignment. *Comput. Optim & Applic*, 4 :159–165, 1995.

[13] E. Çela. *The Quadratic Assignment Problem.* 1998.

[14] A.N. Elshafei. Hospital layout as a quadratic assignment problem. *Operations Research Quarterly*, 28 :167–179, 1977.

[15] C. Fleurent and J. Ferland. Genetic hybrids for the quadratic assignment problem. Technical Report IRO-870, 1993.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability, A guide to the Theory of NP-Completeness.* Freeman & Co., San Francisco, 1979.

[17] A.M. Geoffrion and G.W. Grave. Scheduling parallel production lines with changeover costs : Pratical applications of a quadratic assignment / linear programming approach. *Operations Research*, 24 :595–610, 1976.

[18] P. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *SIAM*, 10 :305–313, 1962.

[19] S.A. Gueye. *Linéarisation et relaxation lagrangienne pour problèmes quadratiques en variables binaires.* PhD thesis, Université d'avignon et des Pays de Vaucluse, 2002.

[20] D.R. Heffley. Assigning runners to a relay team. *Optimal Strategies in Sports*, pages 169–171, 1977.

[21] C. Helmberg and F. Rendl. "aspectral bundle method for semidefinite programming". *ZIB Preprint SC-97-37, Konrad-Zuse-Zentrum fur Informationstechnik Berlin*, 1997.

[22] SS. Heragu and A. Kusiak. Machine layout problem in flexible manufacturing systems. *Operations Research*, 36(2) :258–267, 1988.

[23] F.S. Hiller. Quantitative tools for plant layout analysis. *Journal of Industrial Engineering*, 14 :33–40, 1963.

[24] F.S. Hiller and M.M. Conners. The quadratic assignment problem algorithms and the location of indivisible facilities. *Management Science*, 13 :42–57, 1966.

[25] T.C. Koopmans and M.J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25 :53–76, 1957.

[26] J. Krarup and P.M. Pruzan. Computer-aided layout design. *Mathematical Programming Study*, 9 :75–94, 1978.

[27] E.L. Lawler. The quadratic assignment problem. *Management Science*, 9 :586–599, 1963.

[28] P. Merz and B. Freisleben. A genetic local search approach to the quadratic assignment problem. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, San Francisco, CA, 1997. Morgan Kaufmann.

[29] T. Ostrowski and V.T. Ruoppila. Genetic annealing search for index assignment in vector quantization. *Pattern Recognition Letters*, 18(4) :311–318, 1997.

[30] F.Rendl P.M Pardalos and H. Wolkowicz. The quadratic assignment problem : A survey and recent developments. *DIMACS Series in Discrete Mathematics and Theorical Computer Science*, 16 :1–32, 1994.

[31] G. Finke R.E Burkard and F. Rendl. Quadratic assignment problems. *Annals of Discrete Mathematics*, 31 :61–82, 1987.

[32] P.M. Pardalos R.E. Burkard, E. Çela and L. Pitsoulis. The quadratic assignment problem. In P.P. Pardalos and M.G.C. Resende, editors, *Handbook of Combinatorial Optimization*, pages 241–238. Kluwer Academic Publishers, Dordrecht, 1998.

[33] S.E. Karisch R.E. Burkard and F. Rendl. Qaplib - a quadratic assignment problem library. *Journal of Global Optimization*, 10 :391–403, 1997.

[34] F. Rendl and R. Sotirov. Bounds for the quadratic assignment problem using the bundle method. Technical report, University of Klagenfurt, Universitaetsstrasse 65-67, Austria, 2003.

[35] M.G.C. Resende, K.G. Ramakrishnan, and Z. Drezner. Computing lower bounds for the quadratic assignment problem with an interior point algorithm for linear programming. *Operations Research*, 43(5) :781–791, 1995.

[36] F. Roupin. From linear to semidefinite programming : an algorithm to obtain semidefinite relaxations for bivalent quadratic problems. *Jounral of Combinatorial Optimization*, 8(4) :469–493, 2004.

[37] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23 :555–565, 1976.

[38] J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, 2 :33–45, 1990.

[39] L. Steinberg. The backboard wiring problem : a placement algorithm. *SIAM Review*, 3 :37–50, 1961.

[40] E.D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17 :443–455, 1991.

[41] E.D. Taillard. Comparaison of iterative searches for the quadratic assignment problem. *Location Science*, 3 :87–105, 1995.

[42] U.W. Thonemann and A. Bölte. An improved simulated annealing algorithm for the quadratic assignment problem. Technical report, School of Business, Department of Production and Operations Research, University of Paderborn, Germany, 1994.

[43] P.C. Yip and Y. Pao. Aguided evolutionary simulated annealing approach to lied to the quadratic assignment problem. *IEEE transactions on systems, manand cybernetics*, 24(9) :1383–1387, septembre 1994.